More on Lattices

Reading: NNH 2.3

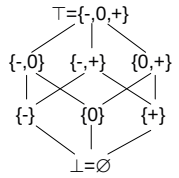
17-654/17-765
Analysis of Software Artifacts
Jonathan Aldrich

Announcements

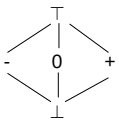
- Homework due at midnight
  - Any questions?

Sign Analysis

Subset Lattice



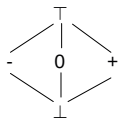
Custom Lattice



- Custom lattice exchanges precision for performance
  - Merges some subset lattice values into ⊤
  - Reduces lattice height => faster fixed point, less storage

Formal Definition

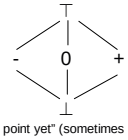
Custom Lattice



- Lattice is a tuple of custom lattices
  - One for each variable in the program
- Forward analysis
- Injected tuple $\iota = \langle \top, \ldots, \top \rangle$
  - Assume the worst of input variables
  - Could also assume $\iota = \langle 0, \ldots, 0 \rangle$ to model Java initialization
- Simple transfer functions *(σ is input data flow value)*
  - Can't use kill and gen sets, because *data flow values aren't sets!*
  - $f^{SA}([x := a],\sigma) = \sigma\,[x \mapsto SA(a,\sigma)]$
  - $f^{SA}([skip],\sigma) = \sigma$
  - $f^{SA}([b],\sigma) = \sigma$ *// could do better with separate T/F functions*
  - $SA(n,\sigma) = sign(n)$ *// returns sign of n*
  - $SA(x,\sigma) = \sigma(x)$
  - $SA(a_1\ op_a\ a_2,\sigma) = \top$ *// could do better by modeling each op*

Discussion

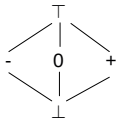
Custom Lattice



- Monotonicity
  - When a data flow value at a program point is recomputed, the new value is always ⊒ the old one
- ⊥
  - Most precise value possible
  - Always initial value at each program point
  - Typically means "haven't looked at this program point yet" (sometimes has domain semantics)
- ⊤
  - Least precise value (but always safe)
  - Typically means "don't know anything, and will never know more for this program point"
- +
  - Means "I believe this is +, but that might change to ⊤ as I iterate over different program paths"
  - Similar for -, 0
- Injected value ι
  - Depends on assumptions of analysis
    - sometimes ⊤, sometimes ⊥, sometimes another value

What if Initial Value were ⊤?

Custom Lattice



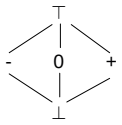
$[x := 5]^1$
while $[x < 10]^2$ do
$[x := x+1]^3$
$[y := x]^4$

Initialize everything to $\langle (x,\top), (y,\top) \rangle$

$SA_{enter}[1] = \iota = \langle (x,\top), (y,\top) \rangle$
$SA_{exit}[1] = \langle (x,+), (y,\top) \rangle$
$SA_{enter}[2] = SA_{exit}[1] \sqcup SA_{exit}[3]$
$= \langle (x,+), (y,\top) \rangle \sqcup \langle (x,\top), (y,\top) \rangle$
$= \langle (x,+ \sqcup \top), (y,\top \sqcup \top) \rangle$
$= \langle (x,\top), (y,\top) \rangle$
...
Everything else evaluates to $\langle (x,\top), (y,\top) \rangle$
This fixed point is safe,
but we know that $x > 0$!

Initial Value Should Be ⊥

Custom Lattice



[x := 5]^1
while [x < 10]^2 do
    [x := x+1]^3
[y := x]^4

Initialize everything to <(x,⊥), (y,⊥)>
$SA_{enter}[1] = \iota$ =<(x,⊤), (y,⊤)>
$SA_{exit}[1]$ = <(x,+), (y,⊤)>
$SA_{enter}[2] = SA_{exit}[1] \sqcup SA_{exit}[3]$
= <(x,+), (y,⊤)> ⊔ <(x,⊥), (y,⊥)>
= <(x,+ ⊔ ⊥), (y,⊤ ⊔ ⊥)>
= <(x,+), (y,⊤)>
…
Everything else evaluates to <(x,+), (y,⊤)>
Better result (and still safe!)

Analysis Correctness
(continued)

Reading: NNH 2.2 (optional)

17-654/17-765
Analysis of Software Artifacts
Jonathan Aldrich

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, $\alpha(T,t) \sqsubseteq F(pp(T,t))$

Proving Soundness

- Thus far:
  - Formally define analysis
    - Lattice framework
  - Define abstraction function
    - Maps (trace,time) to a data flow lattice value
  - Formalize execution
    - Structured operational semantics
    - Execution traces
- Prove *local soundness* for flow functions
- Apply *global soundness theorem*
- Examples

Abstraction

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

Semantics of Assignment in WHILE

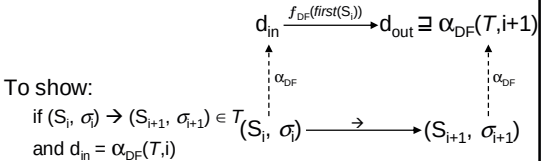
$$\frac{}{([x := a]^\ell, \sigma) \rightarrow ([], \sigma[x \mapsto \mathcal{A}(a, \sigma)])}\ [ass]$$

- Start with a program [x := a]$^\ell$ and a store σ
  - Goal: rewrite to a new program and new store
- We execute [x := a]$^\ell$ resulting in:
  - The empty program []
  - Evaluate a with store σ to get $\mathcal{A}$(a, σ)
  - Update x's value to be $\mathcal{A}$(a, σ)
- Example: a = x+3, σ = (x=5,y=7)
  - We get the pair ([], (x=8,y=7))

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 $(\mathit{first}(S_1), \sigma_1), \ldots, (\mathit{first}(S_n), \sigma_n)$
    - 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 $d_{out} = f_{DF}(\mathit{first}(S_i), d_{in})$
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

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 $d_{out} = f_{CP}(\mathit{first}(S_i), d_{in})$
then $\alpha_{CP}(T,i+1) \sqsubseteq d_{out}$

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

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

Global Soundness

- Theorem (Global Soundness)
  - Assume that $\forall T \in \mathit{traces}(S_*)\ \alpha_{DF}(T,0) \sqsubseteq \iota$ and that analysis DF is monotone and locally sound with respect to α_{DF}
  - Then for any fixed point DF_{fix} of DF on program S_*, $\forall T \in \mathit{traces}(S_*)$ $\forall t \in \mathit{times}(T)$ we have $\alpha_{DF}(T,t) \sqsubseteq DF_{fix}(pp(T,t))$
- Proof: For each trace T we do induction on t
  - Induction hypothesis: $\alpha_{DF}(T,t) \sqsubseteq DF_{fix}(pp(T,t))$
  - Base case: t=0
    - By assumption $\alpha_{DF}(T,0) \sqsubseteq \iota = DF_{fix}(pp(T,0))$
  - Inductive case: time t and statement S_t
    - *Simplifying assumption: straight-line control flow*
    - By induction hypothesis we have $\alpha_{DF}(T,t\text{-}1) \sqsubseteq DF_{fix}(pp(T,t\text{-}1))$
    - By monotonicity of DF we have: $f_{DF}(S_t, \alpha_{DF}(T,t\text{-}1)) \sqsubseteq f_{DF}(S_t, DF_{fix}(pp(T,t\text{-}1)))$
    - By local soundness we have $\alpha_{DF}(T,t) \sqsubseteq f_{DF}(S_t, \alpha_{DF}(T,t\text{-}1))$
    - By transitivity we get $\alpha_{DF}(T,t) \sqsubseteq f_{DF}(S_t, DF_{fix}(pp(T,t\text{-}1)))$
    - But $f_{DF}(S_t, DF_{fix}(pp(T,t\text{-}1))) = DF_{fix}(pp(T,t))$
    - So we have $\alpha_{DF}(T,t) \sqsubseteq DF_{fix}(pp(T,t))$

Abstraction for Reaching Definitions

- $\alpha_{RD}(<p_1,m_1>\ldots<p_n,m_n>,t)$ =
  $\{ (x, p_k) \mid x \in FV(S_*)$
  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

- $\alpha_{LV}(<p_1,m_1>\ldots<p_n,m_n>,t)$ =
  $\{ x \mid x \in FV(stmt(p_k))$ 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 $\alpha_{RD}(T,i) \sqsubseteq d_{out}$

Iterative Worklist Algorithm

Reading: NNH 2.4

17-654/17-765
Analysis of Software Artifacts
Jonathan Aldrich

Worklist Algorithm

```
worklist = new Stack();
∀ℓ ∈ labels(S*) do
    analysis[ℓ] = ⊥;
∀ℓ ∈ E do
    analysis[ℓ] = ι;
    worklist.addAll({(ℓ, ℓ2) | (ℓ, ℓ2) ∈ F});

while (!worklist.isEmpty()) do
    (ℓ1, ℓ2) = worklist.pop();
    if f_ℓ1(Analysis[ℓ1]) ⋢ Analysis[ℓ2] then
        Analysis[ℓ2] = Analysis[ℓ2] ⊔ f_ℓ1(Analysis[ℓ1])
        ∀ℓ3 where (ℓ2, ℓ3) ∈ F do
            worklist.add((ℓ2, ℓ3));
```

Example of Worklist

[a := 1]^1
[b := 2]^2
while [a < 2]^3 do
  [b := b * 1]^4;
  [a := a + 1]^5;
[a := b + 1]^6;

Iter	Edge	Worklist	$a_\circ$	$b_\circ$
0	-	1→2	⊤	⊤
1	1→2	2→3	1	⊤
2	2→3	3→4,3→6	1	2
3	3→4	4→5,3→6	1	2
4	4→5	5→3,3→6	1	2
5	5→3	3→4,3→6	⊤	2
6	3→4	4→5,3→6	⊤	2
7	4→5	5→3,3→6	⊤	2
8	5→3	3→6	⊤	2
9	3→6	-	⊤	2

Worklist: Properties

- Correctness
  - Implements chaotic iteration, therefore correct
- Performance
  - Visits each node only when data changes
    - up to h = height of lattice
  - Propagates data along control flow edges
    - up to e = max outbound edges per node
  - Assume lattice operation cost is o
  - Overall, O(h*e*o)

Worklist Algorithm

```
worklist = new Stack();                          h: height of lattice and max
∀ℓ ∈ labels(S,) do                                  times any node can change
    analysis[ℓ] = ⊥;                             n: number of nodes
∀ℓ ∈ E do                                        e: number of edges
    analysis[ℓ] = ι;                             o: cost of data flow operations
    worklist.addAll({(ℓ, ℓ2) | (ℓ, ℓ2) ∈ F});

while (!worklist.isEmpty()) do
    (ℓ1, ℓ2) = worklist.pop();                   may execute O(h*e) times
    if f_ℓ1(Analysis[ℓ1]) ⋢ Analysis[ℓ2] then     cost O(h*e*o)
                                                 may execute O(h*n) times
        Analysis[ℓ2] = Analysis[ℓ2] ⊔ f_ℓ1(Analysis[ℓ1])   cost O(h*n*o)
                                                 may execute O(h*e) times
        ∀ℓ3 where (ℓ2, ℓ3) ∈ F do                cost O(h*e)
            worklist.add((ℓ2, ℓ3));              cost O(h*e)
```


Performance

	h	o	O(h*e*o)
Reaching Definitions			
Live Variables			
Constant Propagation (sets)			
Constant Propagation (lattice)			
Sign Analysis			

Performance

	h	o	O(h*e*o)
Reaching Definitions	n	n	$n^2 * e$
Live Variables	v	v	$v^2 * e$
Constant Propagation (sets)	∞	∞	∞ (May not terminate)
Constant Propagation (lattice)	2*v	2*v	$v^2 * e$
Sign Analysis	2*v	2*v	$v^2 * e$

Nonterminating Analysis

(Moral: make your lattices finite height!)

[x := 0]^1
while [x < y]^2 do
  [x := x + 1]^3;
[x := 0]^4;

Iter	Position	x	y
0	--	∅	∅
1	entry(1)	**Z**	**Z**
2	exit(1)	{0}	**Z**
3	entry(2)	{0}	**Z**
4	exit(2)	{0}	**Z**
5	entry(3)	{0}	**Z**
6	exit(3)	{1}	**Z**
7	entry(2)	{0,1}	**Z**
8	exit(2)	{0,1}	**Z**
9	entry(3)	{0,1}	**Z**
10	exit(3)	{1,2}	**Z**
11	entry(2)	{0,1,2}	**Z**
12	exit(2)	{0,1,2}	**Z**
13	entry(3)	{0,1,2}	**Z**
14	exit(3)	{1,2,3}	**Z**
15	entry(2)	{0,1,2,4}	**Z**
…			