Modularity in Lattices: A Case Study on the Correspondence between Top-Down and Bottom-Up Analyses

Ghila Castelnuovo	Tel Aviv University
Mayur Naik	Georgia Institute of Technology
Noam Rinetzky	**Tel Aviv University**
Mooly Sagiv	Tel Aviv University
Hongseok Yang	University of Oxford

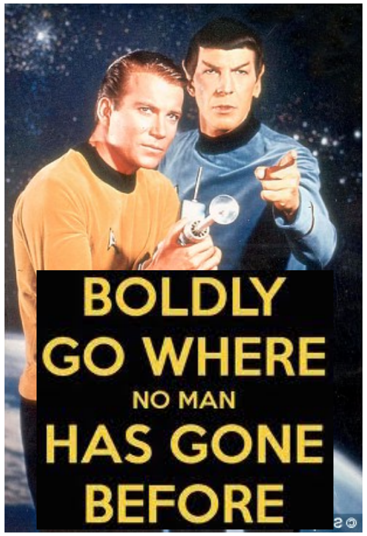
SAS'15 10-September-2015

Research problem

- A precise compositional (heap) analysis

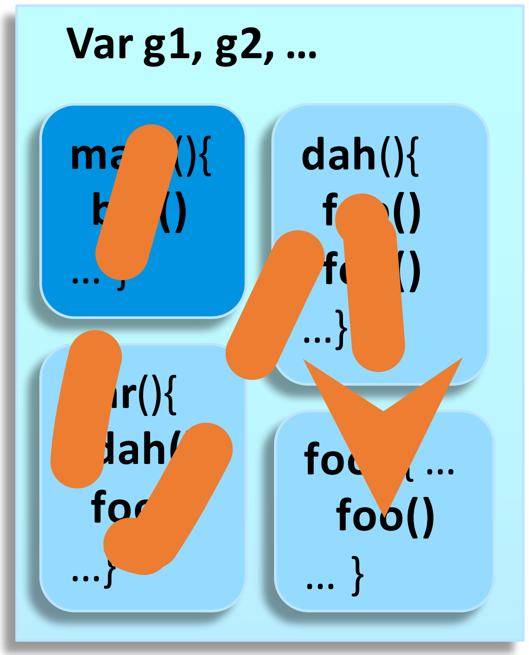
Research problem

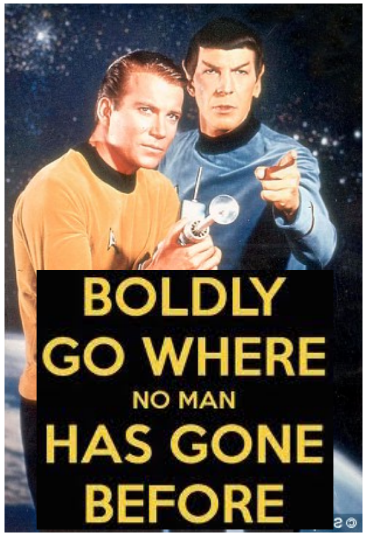
- A precise compositional (heap) analysis
- Compositional?
  - Bottom-Up: Context-independent
  - Top-Down: Context-dependent



Research problem

- A precise compositional (heap) analysis
- Compositional?
  - Bottom-Up: Context-independent
  - Top-Down: Context-dependent



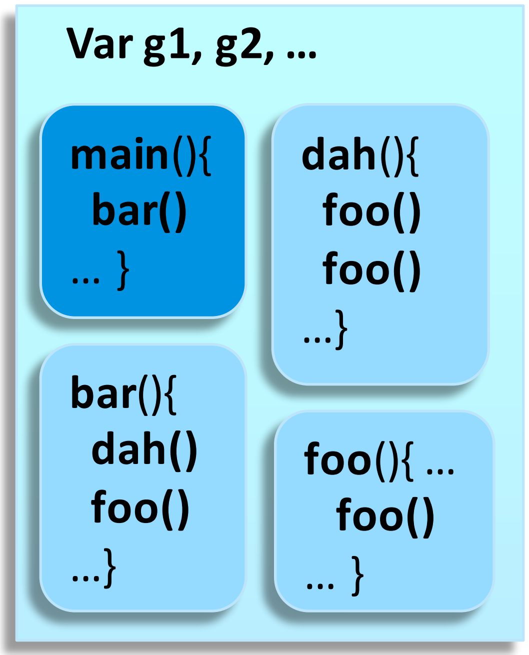


Research problem

- A precise compositional (heap) analysis

✓Compositional

- Bottom-Up: Context-independent
- Top-Down: Context-dependent



```
Var g1, g2, …

main(){        dah(){
  bar()          foo()
… }              foo()
               …}

bar(){         foo(){ …
  dah()          foo()
  foo()        … }
…}
```

Research problem

- A precise compositional (heap) analysis
- Precise?

Research problem

- A precise compositional (heap) analysis
- Precise?
  - Precise enough for a particular client?

Challenges

- Accounting for **all** calling contexts
  - Soundness
  - Precision
  - Scalability
    - Size of procedure summaries
    - Cost of summary instantiation

Contributions

- Modular connection analysis [Ghiya & Hendren, '96]*
  - Lightweight heap analysis
  - Used for parallelization
- Provably as precise as the top-down version
  - Top-down analysis sound (by abstract interpretation)
  - Implies soundness
- Experimental evaluation
  - Bottom-up scales much better than the top-down
  - Little loss of precision compared to original analysis

*Slightly modified version of the original analysis

This paper is a mere glimpse …

- Ghila Castelnuovo's Master Thesis:

  Modular lattices for compositional Interprocedural Analysis

- **Framework of compositional analysis**

- Guaranteed precision relative to top-down analysis

Available at: http://www.cs.tau.ac.il/Castelnuovo.Ghila-MSc.Thesis.pdf

⊔-based compositional analysis

" … Mission: To explore strange new worlds, to seek out new life and new civilizations, to boldly go where no one has gone before."

(Starting in baby steps…)

$\sqcup$-based compositional analysis

$$\llbracket \mathbf{st} \rrbracket(\mathbf{d}) = \mathbf{d} \sqcup \mathbf{C_{st}}$$

- Transformers defined using $\sqcup$
  - $\mathbf{C_{st}}$ is an element in the domain

Composition by adaptation

$$[\![st1; st2]\!](d) = (d \sqcup C_1) \sqcup C_2$$

- Transformers defined using $\sqcup$
  - C_i is an element in the domain
  - Recall: $\sqcup$ is commutative, associative, idempotent

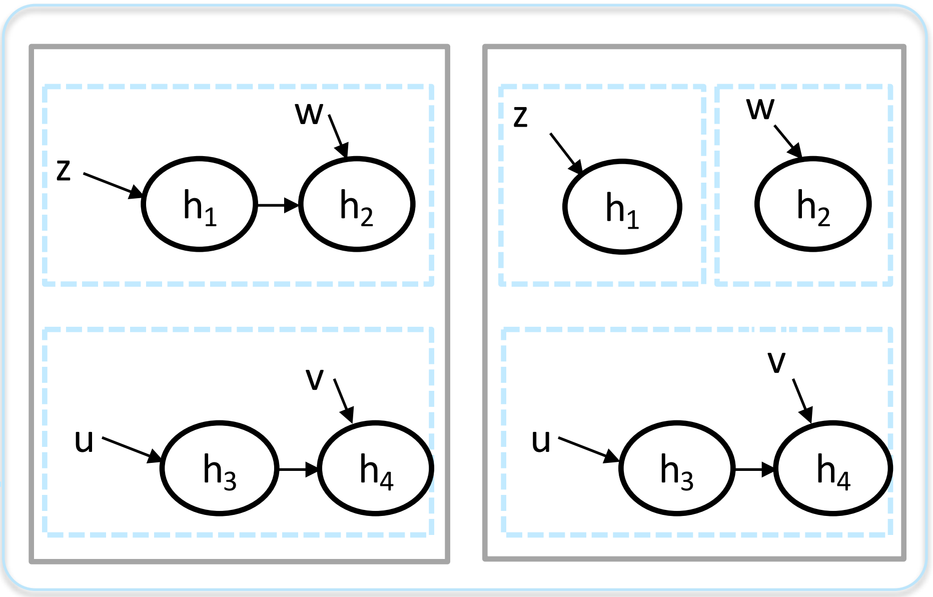
Adapt the result of analyzing d instead of analyzing $d \sqcup d'$!

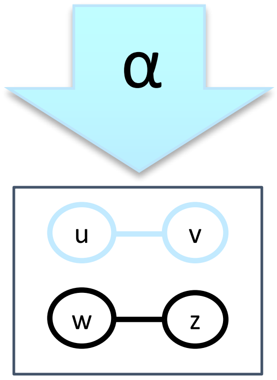
Compositional analysis:

$$[\![p()]\!]^{\#}(d \sqcup d') = [\![p()]\!]^{\#}(d) \sqcup d'$$

Connection analysis (CA)

```
static main(){
 l1   z = new h1
 l2   w = new h2
 l3   u = new h3
 l4   v = new h4
 l5   u.f = v
 l6   if(…) z.f = w

}
```





$\{\{u,v\},\{w,z\}\}$

Are **z** and **w** **connected**?

Maybe

Are **z** and **u** **connected**?

No!

Interprocedural CA can be expensive…

# of calling contexts: 1

```
main(){
 X = new h₁
 Y = new h₂
 p1()
}
```

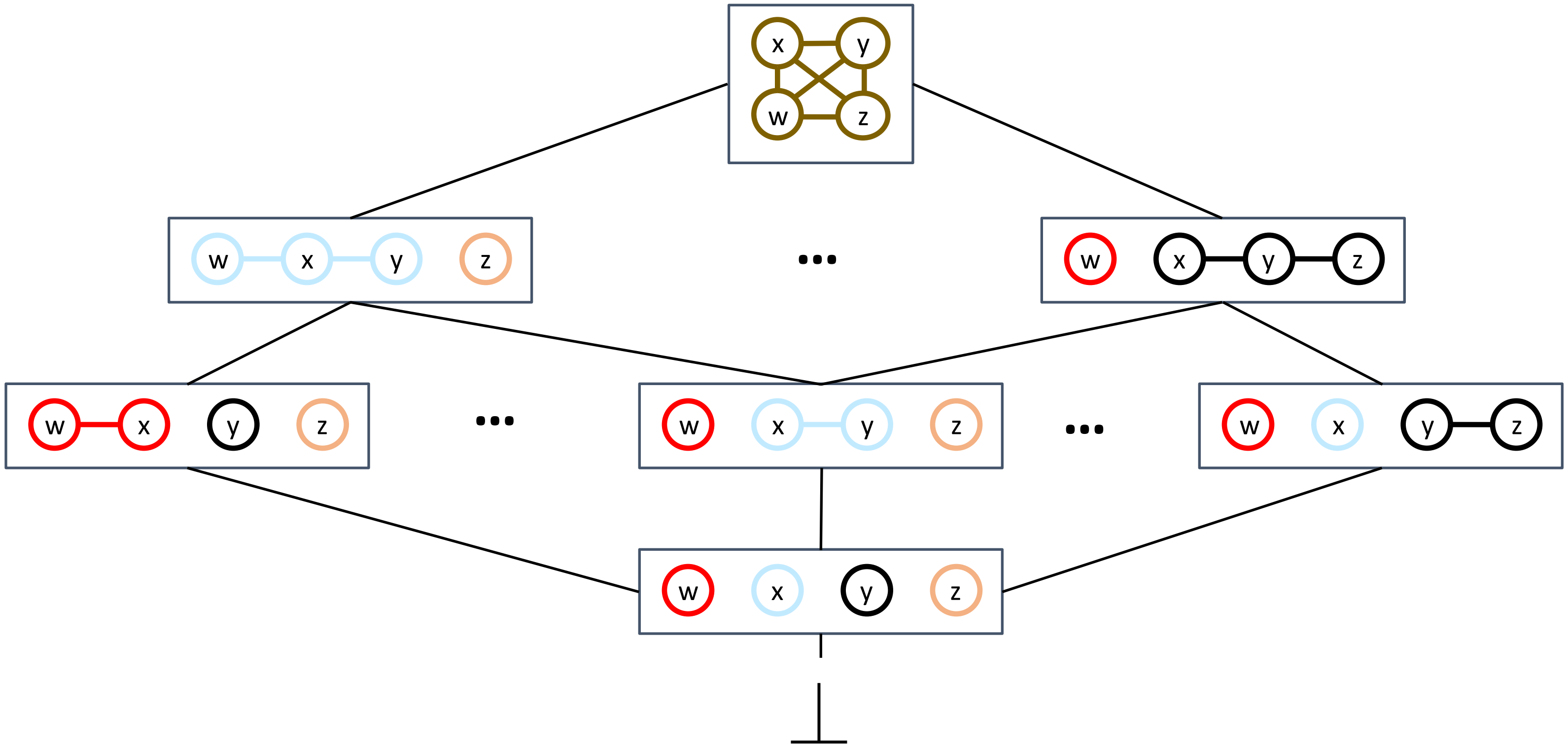
Compositional CA (simplified)

- Partition abstract domain
- $\sqcup$-based transformers

$\Rightarrow$ Compositionality by adaptation

CA: Partition abstract domain

- **D = (Partition(V), $\sqsubseteq$) $\simeq$ (Equiv(V), $\sqsubseteq$)**
  - **Partition(V)** Set of partitioning of V
    - **Equiv(V)** Set of equivalence relations over V
  - $\sqsubseteq$ Refinement



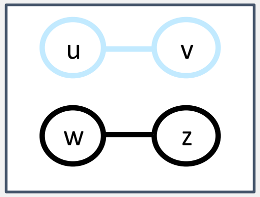
CA: Abstract transformers (simplified)

$$[\![st]\!](d) = d \sqcup C_{st}$$

- Transformers defined using $\sqcup$
  - C_{st} is a constant partition, e.g., $C_{w.f=u} = U_{w,u} = \{\{w,u\}, \{z\}, \{v$

- $[\![x = null]\!]^{\#}(d) = d$
- $[\![x = new]\!]^{\#}(d)= d$
- $[\![x.f = y]\!]^{\#}(d) \quad = d \sqcup U_{xy}$
- $[\![x = y]\!]^{\#}(d) \quad = d \sqcup U_{xy}$
- $[\![x = y.f]\!]^{\#}(d) \quad = d \sqcup U_{xy}$

$[\![\mathbf{w.f = u}]\!]^{\#}$



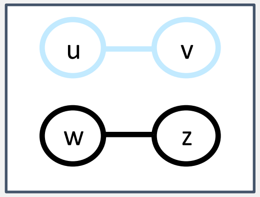
CA: Abstract transformers (simplified)

$$[\![st]\!](d) = d \sqcup C$$

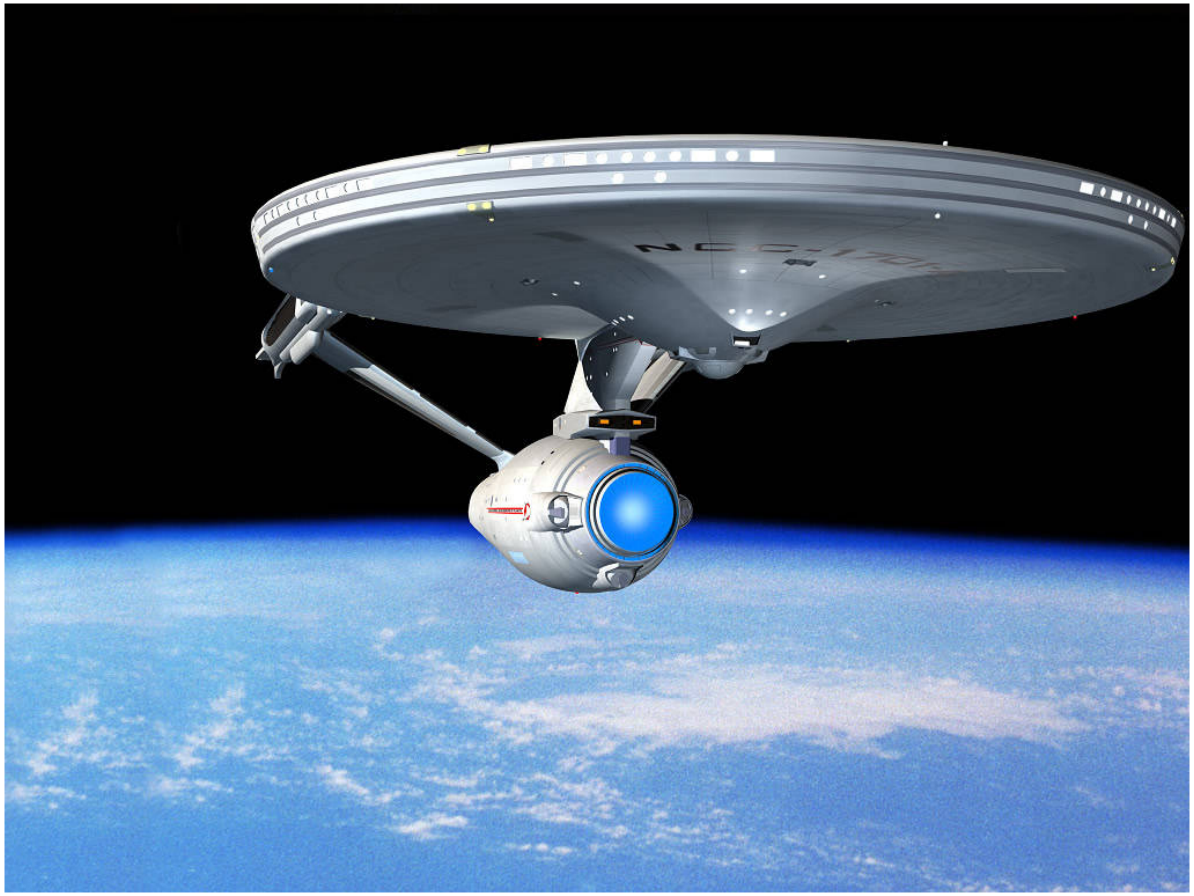
- Transformers defined using $\sqcup$
  - C_{st} is a constant partition, e.g., $C_{w=u} = U_{w,u} = \{\{w,u\}, \{z\}, \{v\}\}$

- $[\![x = null]\!]^\#(d) = d$
- $[\![x = new]\!]^\#(d)= d$
- $[\![x.f\ \ y]\!]^\#(d) \quad = d \sqcup U_{xy}$
- $[\![x = y]\!]^\#(d) \quad\quad = d \sqcup U_{xy}$
- $[\![x = y.f]\!]^\#(d) \quad = d \sqcup U_{xy}$

$[\![\mathbf{w = u}]\!]^\#$



CA: Abstract transformers

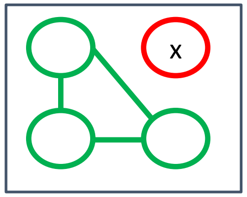


(Moving on towards the real thing …)

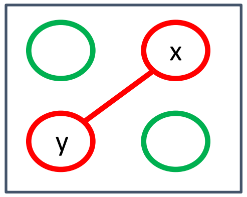
CA: Abstract transformers

- Transformers defined using $\sqcup$ and $\sqcap$
  - $U_{x,y} = \{ \{x,y\}, \{z\}, \{w\}\}$
  - $S_x = \{ \{x\}, \{y,z,w\} \}$
- $[\![x = null]\!]^{\#}(d) = d \sqcap S_x$
- $[\![x = new]\!]^{\#}(d) = d \sqcap S_x$
- $[\![x.f = y]\!]^{\#}(d) = d \sqcup U_{xy}$
- $[\![x = y]\!]^{\#}(d) = (d \sqcap S_x) \sqcup U_{xy}$
- $[\![x = y.f]\!]^{\#}(d) = (d \sqcap S_x) \sqcup U_{xy}$

S_x : Separation



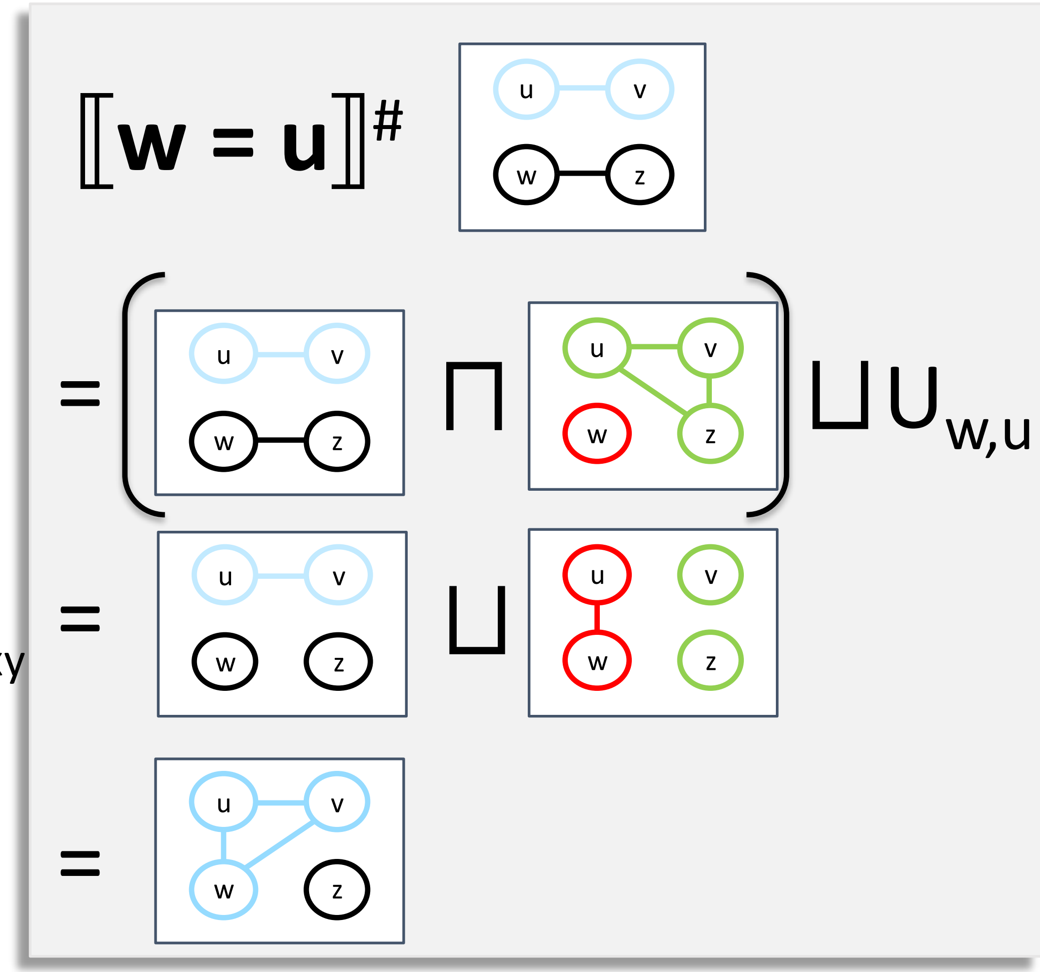
$U_{x,y}$: Unification



CA: Abstract transformers

- Transformers defined using $\sqcup$ and $\sqcap$
  - $U_{x,y} = \{ \{x,y\}, \{z\}, \{w\}\}$
  - $S_x = \{ \{x\}, \{y,z,w\} \}$

- $[\![x = \text{null}]\!]^\#(d) = d \sqcap S_x$
- $[\![x = \text{new}]\!]^\#(d) = d \sqcap S_x$
- $[\![x.f = y]\!]^\#(d) = d \sqcup U_{xy}$
- $[\![x = y]\!]^\#(d) = (d \sqcap S_x) \sqcup U_{xy}$
- $[\![x = y.f]\!]^\#(d) = (d \sqcap S_x) \sqcup U_{xy}$



Can we use adaptation?

- Transformers defined using ⊔ and ⊓
  - $U_{x,y}$ = { {x,y}, {z}, {w}}
  - S_x = { {x}, {y,z,w} }

?

~~**Goal: composition by adaptation**~~

~~$⟦p()⟧^{\#}(d \sqcup d') = ⟦p()⟧^{\#}(d) \sqcup d'$~~

Modularity in Lattices

- For adaptation: $(d \sqcup d') \sqcap d_p = (d \sqcap d_p) \sqcup d'$

Modularity in Lattices

- For adaptation: $(d \sqcup d') \sqcap d_p = (d \sqcap d_p) \sqcup d'$
- An element ***d_p*** in a lattice ***D*** is **right modular** iff

  $\forall d,d' \in D.$ *if* $d' \sqsubseteq d_p$

  then $(d \sqcup d') \sqcap d_p = (d \sqcap d_p) \sqcup d'$

  - D is **modular** if all its elements are right modular

- The partition domain is **NOT** a modular lattice
- But it is modular enough …

Conditionally adaptable transformers

- CA Transformers: $[\![st]\!]^{\#}(d) = (d \sqcap S_x) \sqcup U_{x,y}$
- $U_{x,y}$ and S_x are **right-modular**

$\Rightarrow$ **Conditionally adaptable transformers**

- $\forall d, d' \in D.$ if $d' \sqsubseteq U_{x,y}$ … and $d' \sqsubseteq S_x$ …

  then $[\![st]\!]^{\#}(d \sqcup d') = [\![st]\!]^{\#}(d) \sqcup d'$

Compositional connection analysis

- Intra-procedural analysis is conditionally adaptable
  - Delay the operation of a join (**d** $\sqcup$ **d'**)
  - Adapt the result

- Inter-procedural analysis is unconditionally adaptable!

  $\Longrightarrow$Hence, compositional

Compositional connection analysis

(We are now at warp 7)

Compositional connection analysis

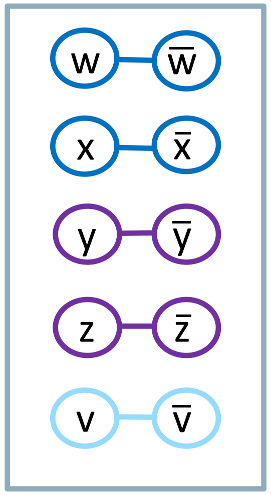
- Procedure call are **conditionally adaptable**
  - Represent any procedure inputs as $d = \iota \sqcup d'$
  - $\forall st.\ st = (\ldots \sqcap d_p \ldots) \Rightarrow d' \sqsubseteq d_p$
  - ι is a particular element in the **Triad Domain**

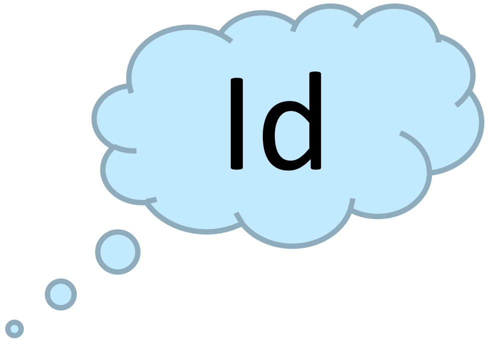
- **Phase I** Analyze every procedure **once** on ι

- **Phase II** Instantiate $p(\iota)$ with information from call context

Who is ι ?

$D[w,\overline{w},x,\overline{x},y,\overline{y},z,\overline{z}]$





Triad domain

- Partition domain comprised of
  - **G** current values: x, y, x
  - $\bar{\mathbf{G}}$ input values: $\bar{x}$, $\bar{y}$, $\bar{z}$
  - $\dot{\mathbf{G}}$ auxiliary (temporary) values: $\dot{x}$, $\dot{y}$, $\dot{z}$
    - To compute effect of procedure calls using $\sqcup$
      - “Relational join”
  - **L** Current local variables

Globals

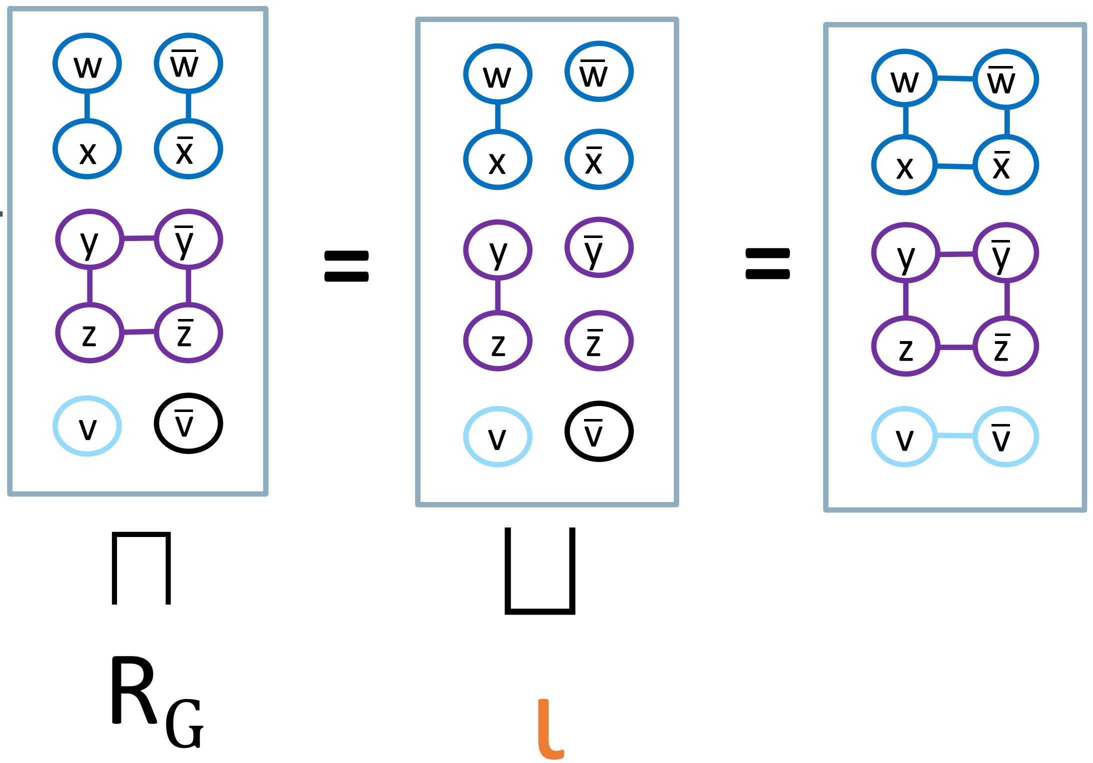
Locals

Entering a procedure (top-down)

$$[\![\text{entry}]\!](d) = (d \sqcap R_G) \sqcup \iota$$

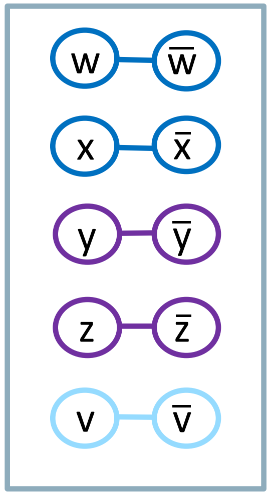
$$R_G = \{G, \{\bar{x}\}, \{\dot{x}\} \mid x \;\; G\}$$

```
p() {
   ...

   q();

}
```



“Entering a procedure” (bottom-up)

$$[\![\text{entry}]\!](d) = \iota$$



Returning from a procedure (TD & BU)

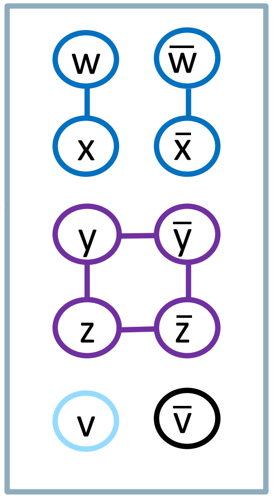
$$[\![\text{return}]\!](d_{\text{exit}}, d_{\text{call}}) =$$

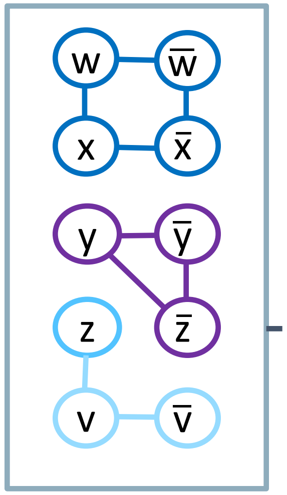
$$(\underset{\substack{\uparrow \\ \text{Rename} \\ \text{current to } \bullet}}{f_{call}}(d_{\text{call}}) \sqcup \underset{\substack{\uparrow \\ \text{Rename} \\ \text{input to } \bullet}}{f_{exit}}(d_{\text{exit}})) \sqcap R_{\bar{G} \cup G'}$$

Returning from a procedure (TD & BU)

$$[\![\text{return}]\!](d_{exit}, d_{call}) =$$
$$\left(\underset{\substack{\uparrow \\ \text{Rename current to } \bullet}}{f_{call}}(d_{call}) \sqcup \underset{\substack{\uparrow \\ \text{Rename input to } \bullet}}{f_{exit}}(d_{exit}) \right) \sqcap R_{\bar{G} \cup G'}$$

```
p() {
   ...

   q();

}
```



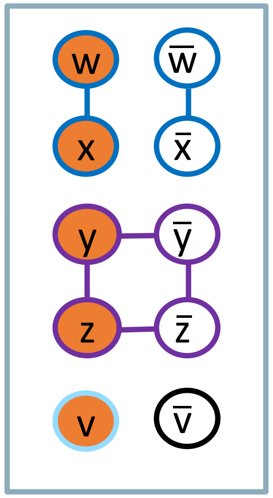


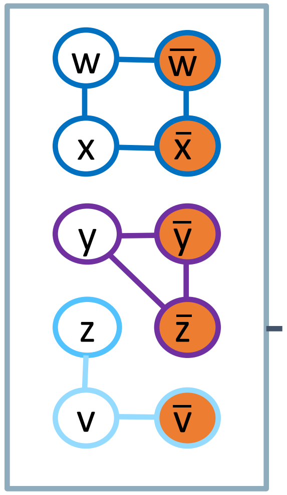
```
q() {
   z = new()
   z.f = v

}
```

Returning from a procedure (TD & BU)

$$[\![\text{return}]\!](d_{exit}, d_{call}) =$$
$$(\underset{\substack{\uparrow \\ \text{Rename current to} \bullet}}{f_{call}}(d_{call}) \sqcup \underset{\substack{\uparrow \\ \text{Rename input to} \bullet}}{f_{exit}}(d_{exit})) \sqcap R_{\bar{G} \cup G'}$$

```
p() {
   ...

   q();



}
```





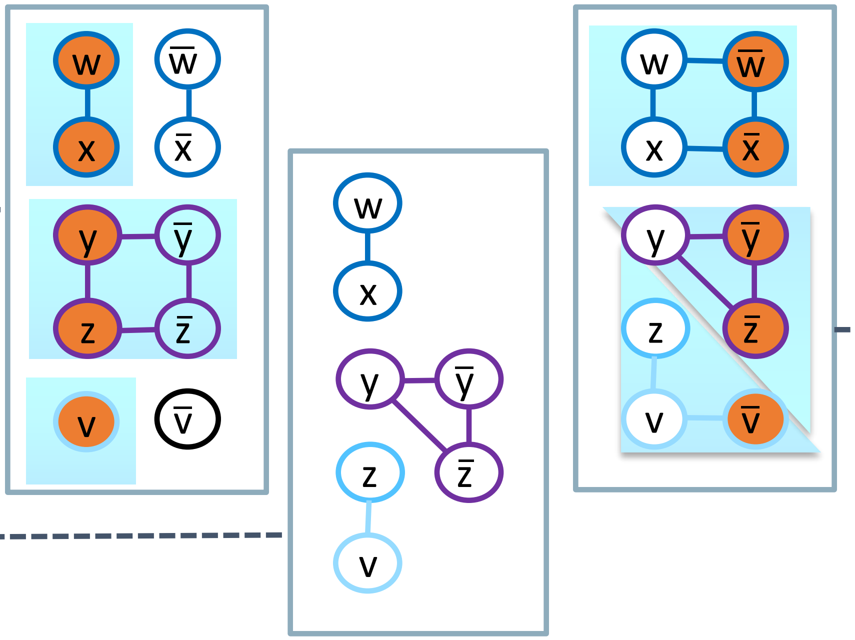
```
q() {
   z = new()
   z.f = v




}
```

Returning from a procedure (TD & BU)

$$[\![\text{return}]\!](d_{exit}, d_{call}) =$$

$$(\underset{\substack{\uparrow \\ \text{Rename current to } \bullet}}{f_{call}}(d_{call}) \sqcup \underset{\substack{\uparrow \\ \text{Rename input to } \bullet}}{f_{exit}}(d_{exit})) \sqcap R_{\bar{G} \cup G'}$$

```
p() {
   ...

   q();

}
```



```
q() {
   z = new()
   z.f = v

}
```

Coincidence Theorem

⟦p()⟧ bottom-up

$$[\![\text{return}]\!]([\![C_{\text{body}}]\!] \circ [\![\text{entry}]\!](\mathbf{d}), d) = [\![\text{return}]\!]([\![C_{\text{body}}]\!](\iota), d)$$

⟦p()⟧ top-down

Where is the magic?

- The magic is in the proof!

The magic is in the proof!

- Proof shows that effect of calling context can be delayed

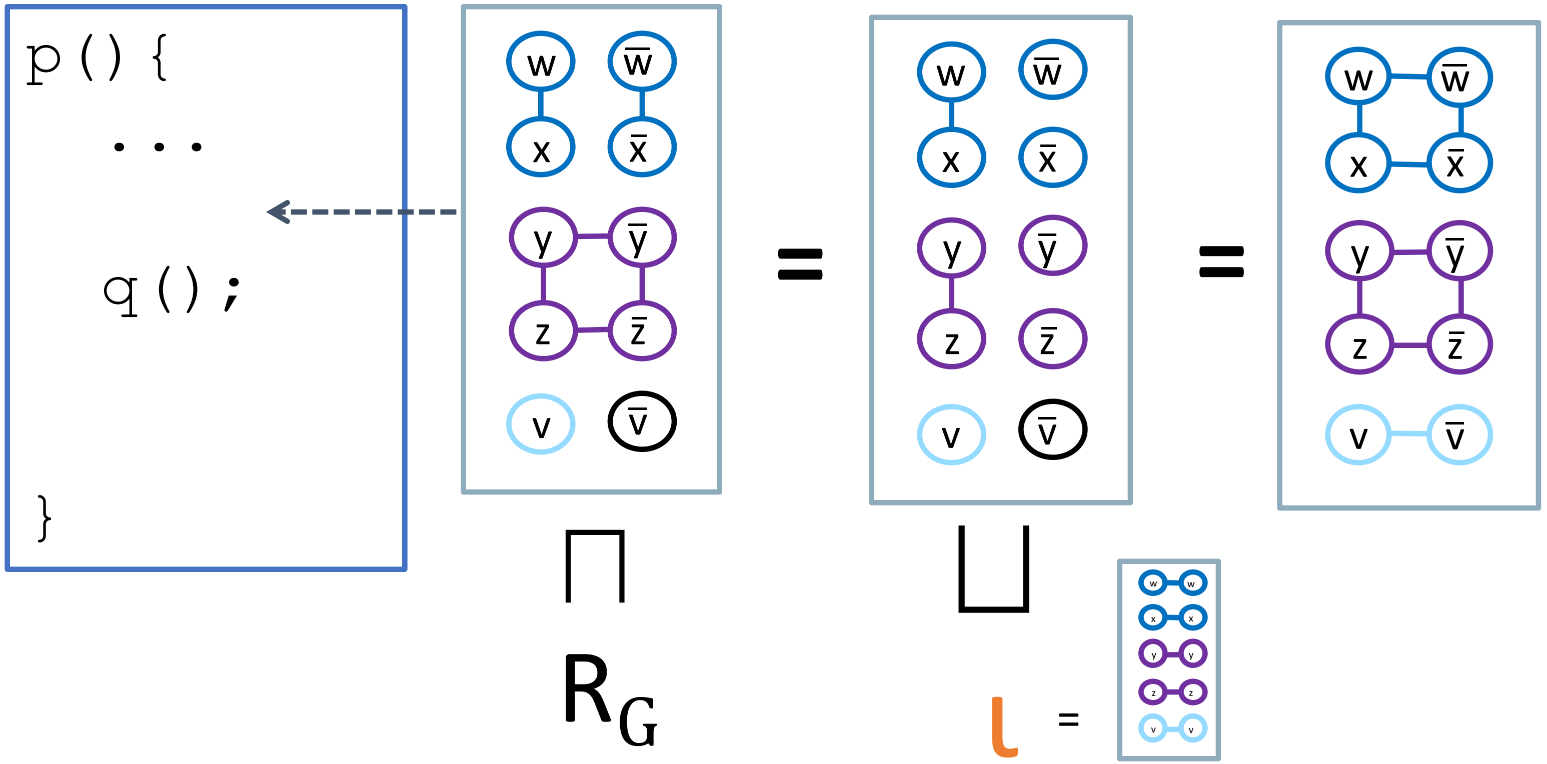
- Non-trivial
  - But rewarding

- Key observations
  - Uniform entry states
  - Counterpart representation
  - ….

Uniform entry states

$$[\![\mathrm{entry}]\!](d) = (d \sqcap R_G) \sqcup \iota \quad \textit{vs} \quad [\![\mathrm{entry}]\!](d) = \iota$$

$$R_G = \{G, \{\bar{x}\}, \{\dot{x}\} \mid x \;\; G\}$$

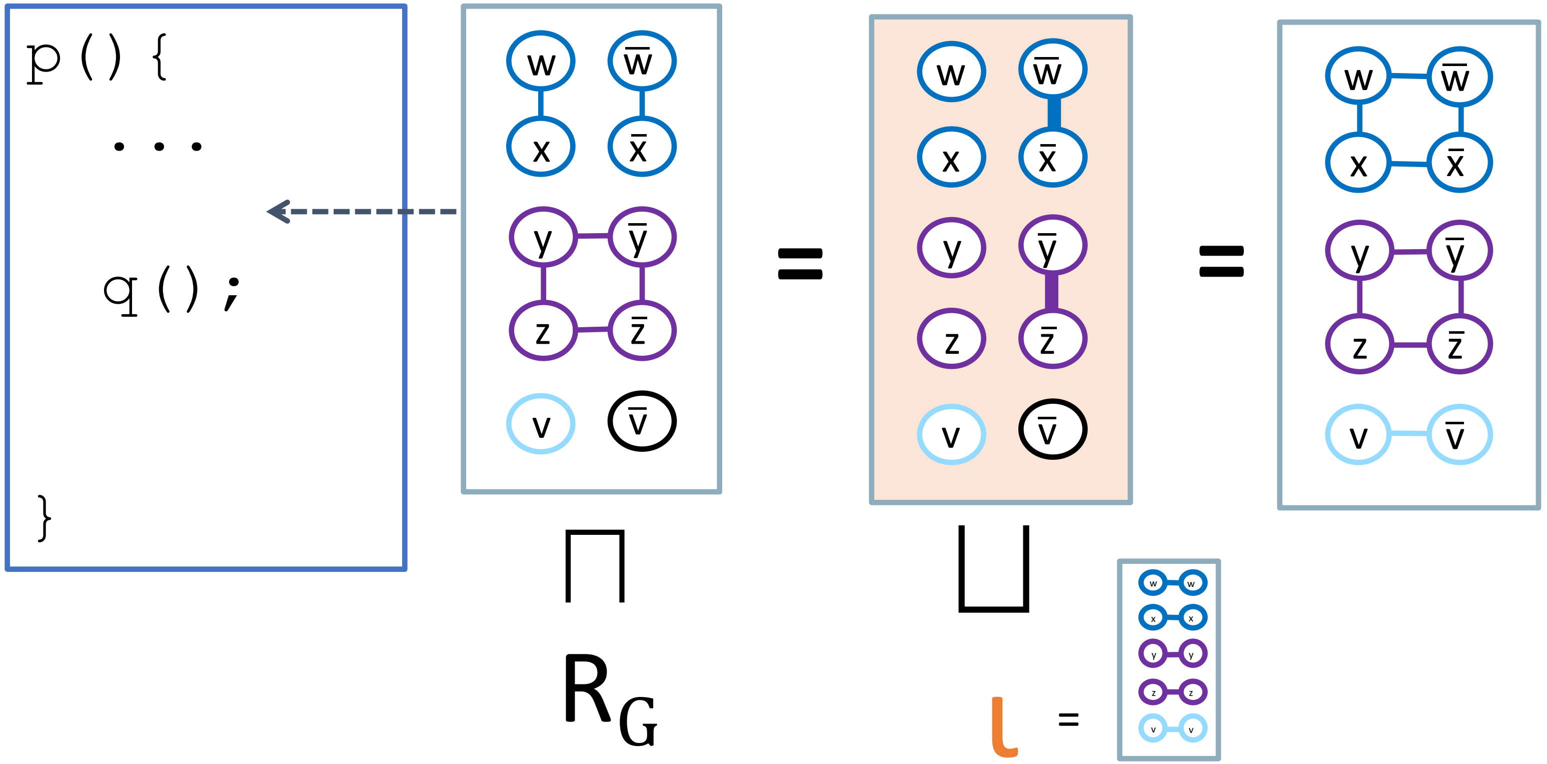


```
p() {
   ...

   q();

}
```

Uniform entry states

$[\![\text{entry}]\!](d) = (d \sqcap R_G) \sqcup \iota$ *vs* $[\![\text{entry}]\!](d) = \iota$

$R_G = \{G, \{\bar{x}\}, \{\dot{x}\} \mid x \;\; G\}$

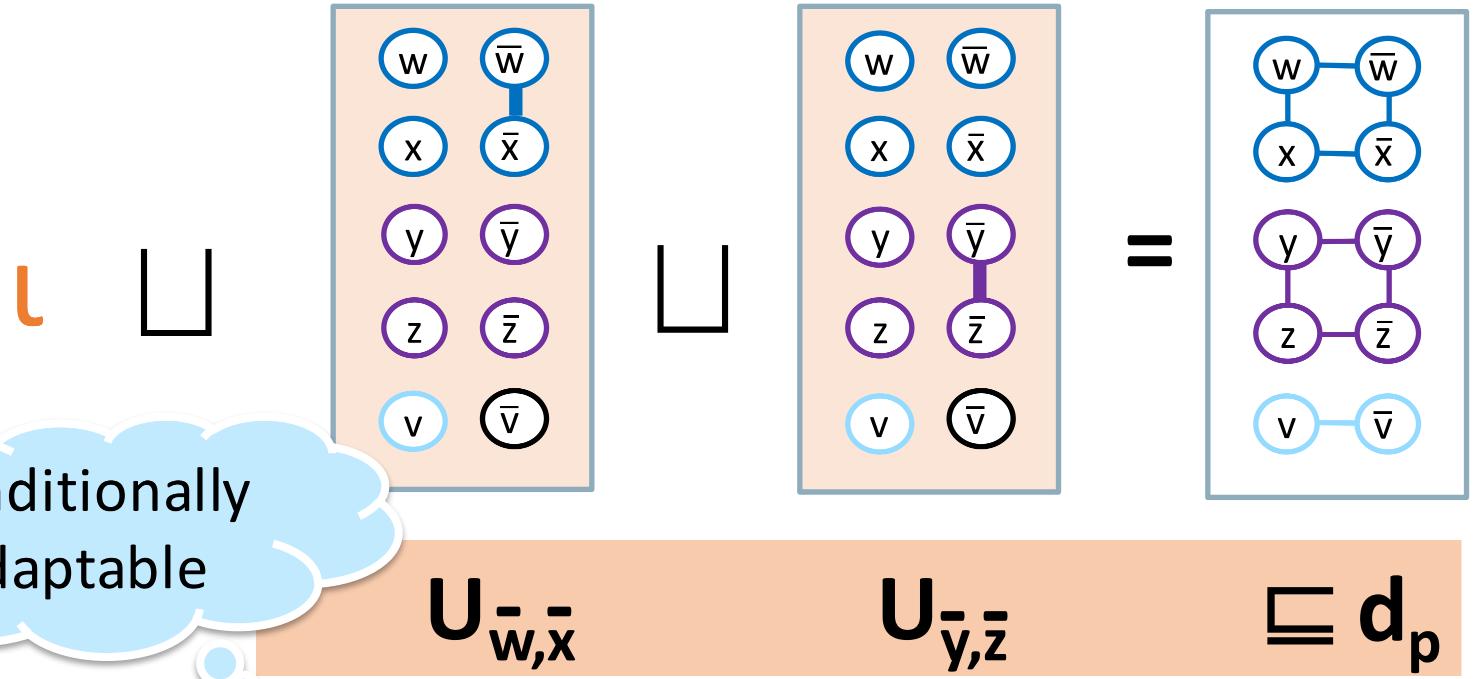


```
p() {
   ...

   q();

}
```

Uniform entry states

$$[\![\text{entry}]\!](d) = (d \sqcap R_G) \sqcup \iota \quad \textit{vs} \quad [\![\text{entry}]\!](d) = \iota$$

$R_G = \{G, \{\bar{x}\}, \{\dot{x}\} \mid x\ \ G\}$

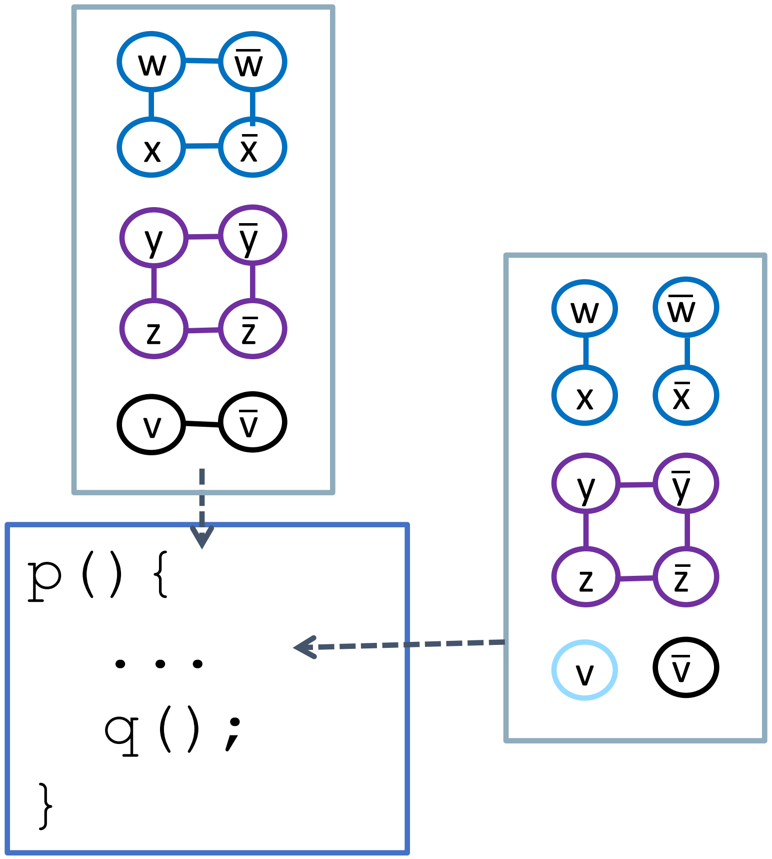


Conditionally adaptable

Counterpart representation

$[\![\text{entry}]\!](d) = (d \sqcap R_G) \sqcup \iota$ *vs* $[\![\text{entry}]\!](d) = \iota$

$R_G = \{G, \{\bar{x}\}, \{\dot{x}\} \mid x \;\; G\}$

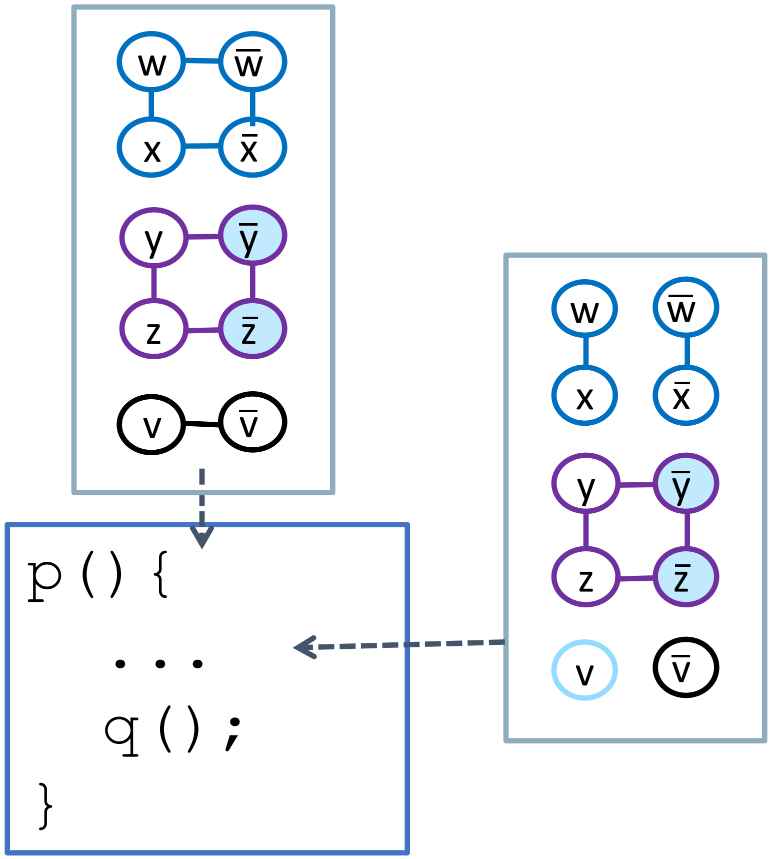


```
p() {
   ...
   q();
}
```

Counterpart representation

$[\![\mathrm{entry}]\!](d) = (d \sqcap R_G) \sqcup \iota$ *vs* $[\![\mathrm{entry}]\!](d) = \iota$

$R_G = \{G, \{\bar{x}\}, \{\dot{x}\} \mid x \;\; G\}$

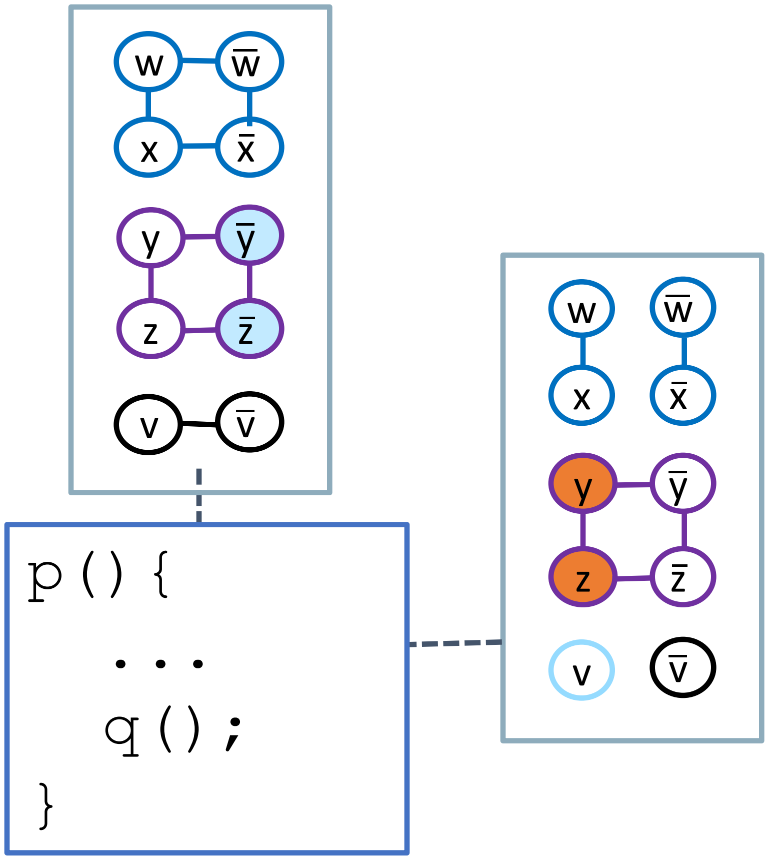


```
p() {
   ...
   q();
}
```

Counterpart representation

$[\![\text{entry}]\!](d) = (d \sqcap R_G) \sqcup \iota$ *vs* $[\![\text{entry}]\!](d) = \iota$

$R_G = \{G, \{\bar{x}\}, \{\dot{x}\} \mid x \;\; G\}$

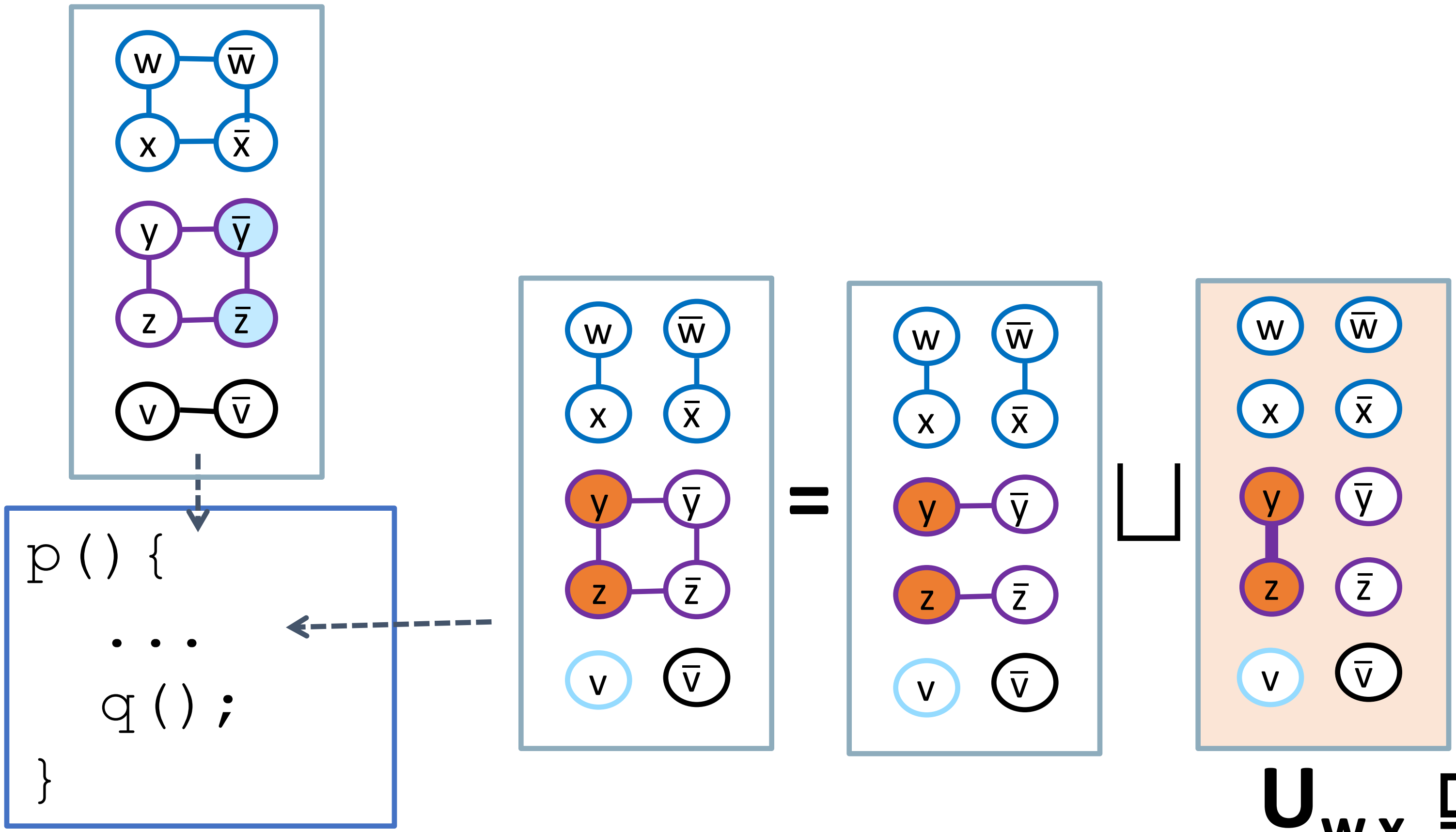


```
p() {
   ...
   q();
}
```

Counterpart representation

$[\![\text{entry}]\!](d) = (d \sqcap R_G) \sqcup \iota$ *vs* $[\![\text{entry}]\!](d) = \iota$

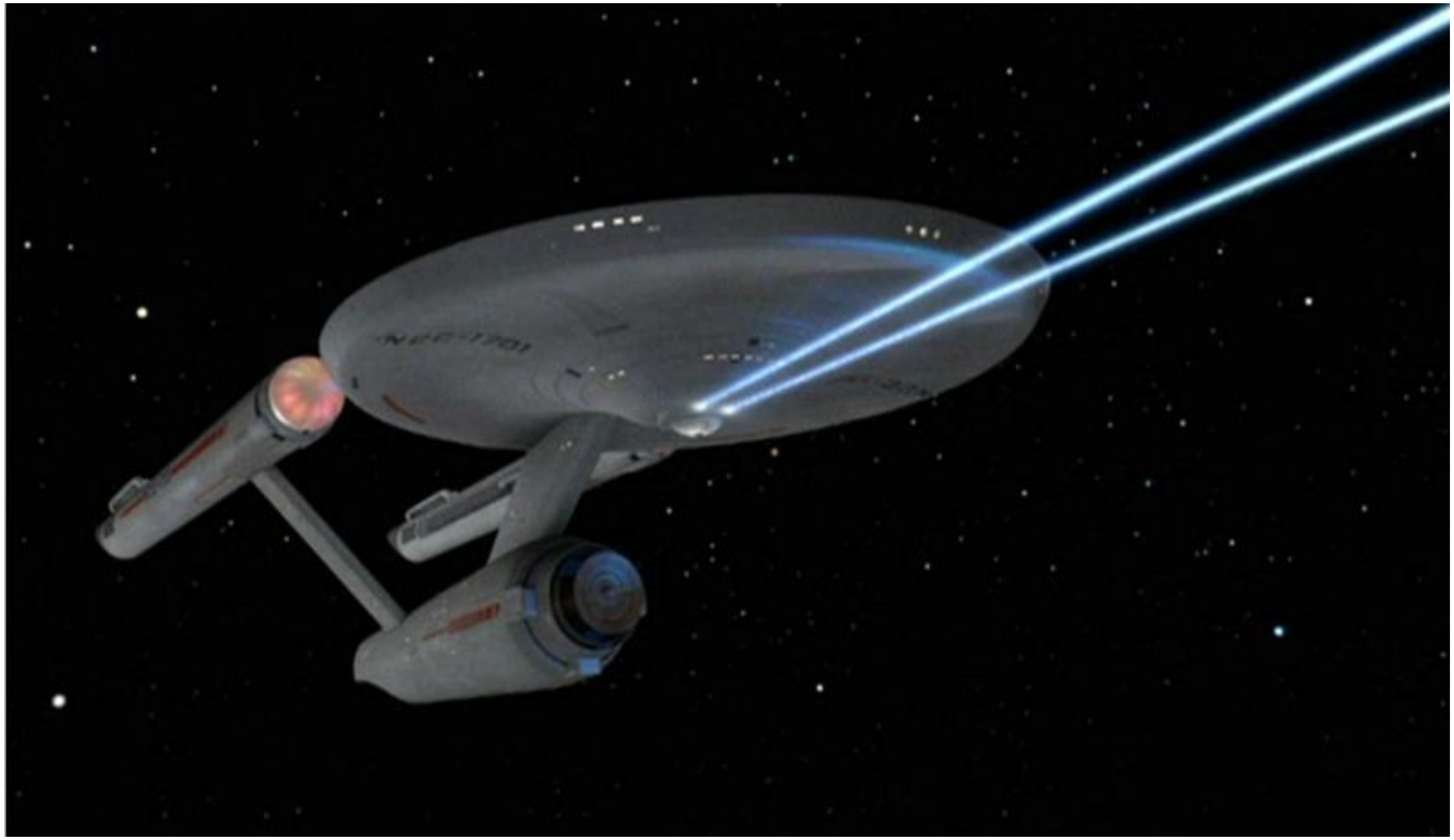
$R_G = \{G, \{\bar{x}\}, \{\dot{x}\} \mid x \ \ G\}$



```
p() {
   ...
   q();
}
```

$U_{w,x} \sqsubseteq R_G$

Experimental results



Experimental results

- Compared 3 versions of connection analysis
  - Original top-down
  - Triad top-down
  - Triad bottom-up (compositional)

Input-dependent transformers

Original

$$[\![x.f = y]\!] \begin{cases} \textbf{Merge} & x \neq null \wedge y \neq null \\ \textbf{Skip} & \text{otherwise} \end{cases}$$

Ours

$$[\![x.f = y]\!] \; \{ \; \textbf{Merge}$$

Experimental setup (DaCapo)

	description	methods	bytecodes
Grande2	Java Grande Kernels	237	13,724
Grande3	Java Grande Large apps	1,162	75,139
Antlr	Parser generator	2,400	128,684
Weka	Machine Learning Library	3,391	223,291
Bloat	Optimizations and Analysis tool	4,699	311,727

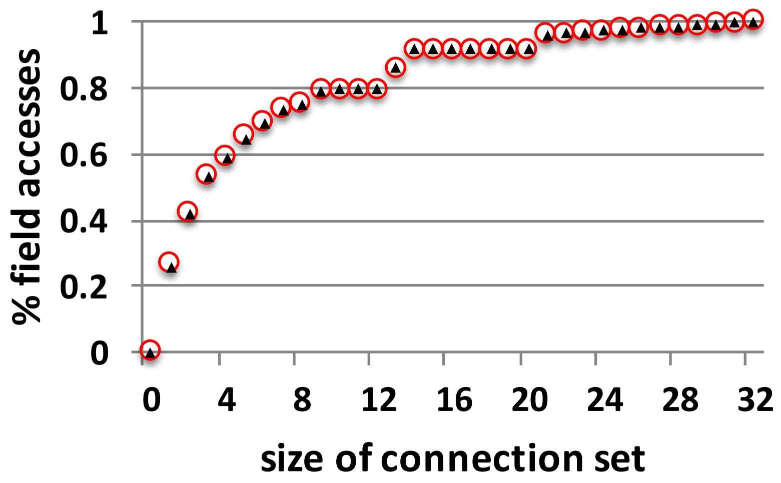
- JRE 1.6; Linux; Intel Xeon 2.13GHz; 123GB RAM
- Using Chord program analysis framework

Experimental evaluation

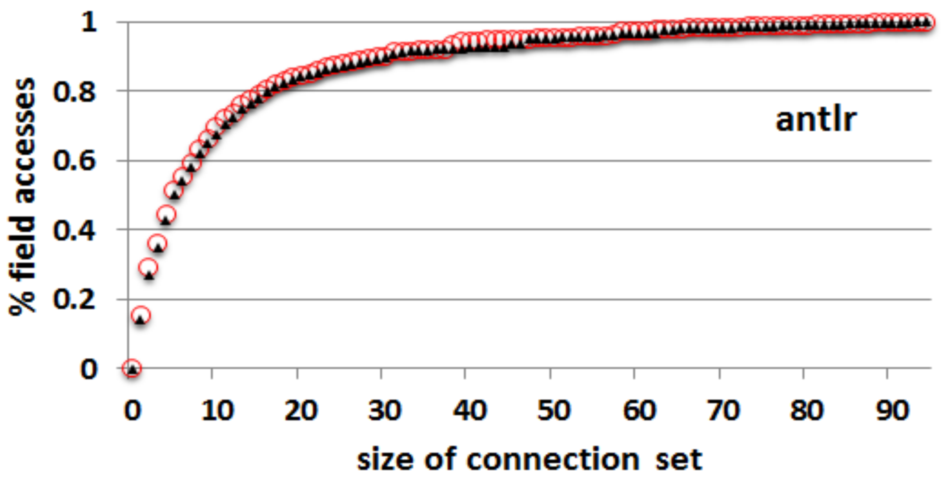
Precision

- Near perfect overlap
- Only 2-5% is lost

grande2



antlr



○ Original top-down ▲ bottom-up (= modified top-down)

Experimental evaluation

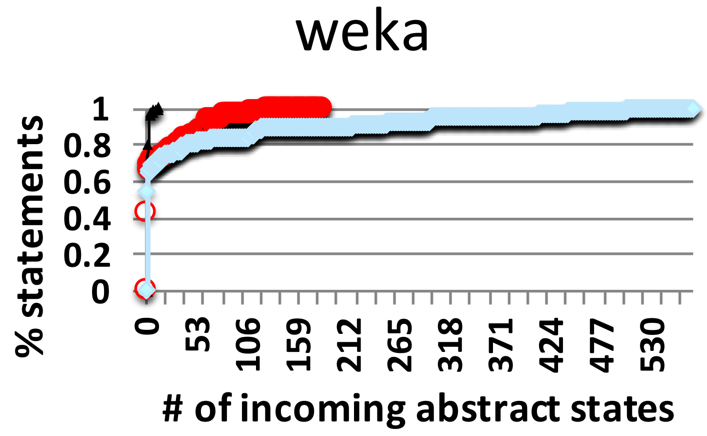
Scalability

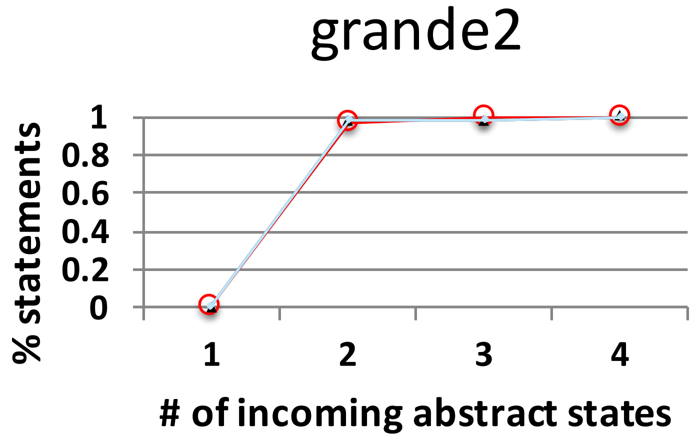
	Bottom-up		**Original Top-down**	**Triad Top-down**
	Summaries computation	instantiation		
Grande2	0.6 sec	0.9 sec	1 sec	0.9 sec
Grande3	43 sec	1:21 min	1:11 min	51 sec
Antlr	16 sec	30 sec	1:23 sec	25 sec
Weka	46 sec	2:48 min	**Timeout!**	**Timeout!**
Bloat	3:03 min	30 min	**Timeout!**	**Timeout!**

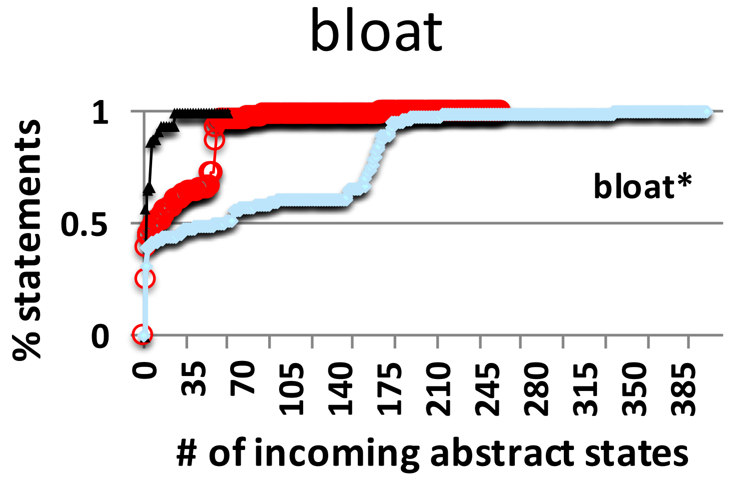
Experimental evaluation

Scalability

- Top down blows-up







● Modified top-down ○ Original top-down ▲bottom-up

Related work

- General theory [Cousot & Cousot, CC'02]
- Modular analysis for logical programs
  [Codish et al. POPL'03] [Giacoabazi, JLP'98]
- Abstract domain for modular analyses
  [Giacoabazi et al., TCS'99]
- **Condensation** and modular analyses
  [Giacoabazi et al. TOCL'05, TOPLAS'98]
  - Condensing abstract domains allow to derive bottom-up analyses with the same precision as top-down ones
  - Lattice-theoretic characterization:

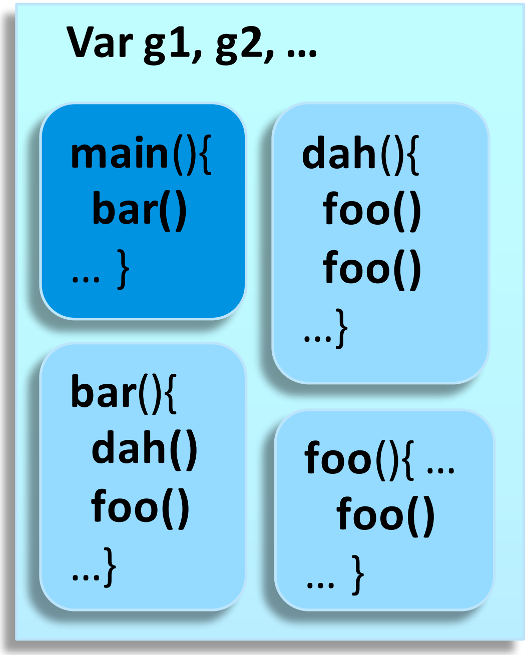
$$F(a \otimes b) = a \otimes F(b)$$

Limitations and future work

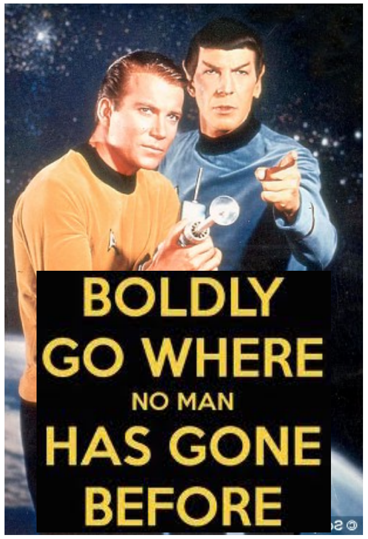
- Transfer functions are input-independent
  - Limited expressivity
- Generalize to other instances
  - Copy constant propagation
  - Taint analysis
  - Castelnuovo’s thesis has a general framework
    - But it is still rather restricted

Summary

- A precise scalable compositional heap analysis



```
Var g1, g2, …

main(){          dah(){
  bar()            foo()
… }                foo()
                 …}
bar(){
  dah()          foo(){ …
  foo()            foo()
…}               … }
```



Top Down

Bottom up

Thank you!