AiracV's Design [1]

Jaeho Shin ⟨netj@ropas.snu.ac.kr⟩

ROPAS Show&Tell

2005-12-02

[1] Many design choices originate from heavy discussions with Jaehwang Kim.

Today's Goal

- See what's inside AiracV

Overview

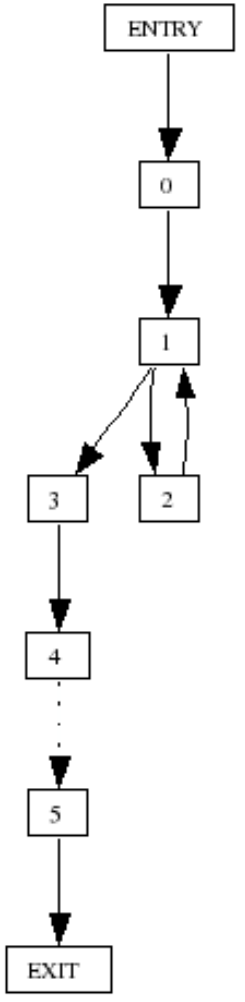
- Program as Graph
  - Control Flow Graph
  - Commands
- Concrete World
  - Concrete Domain
  - Concrete Semantics
- Abstract World
  - Abstract Domain
  - Trace Abstraction
  - Data State Abstraction
  - Partitioning
  - Abstract Semantics

Now,

- Program as Graph
  - Control Flow Graph
  - Commands
- Concrete World
  - Concrete Domain
  - Concrete Semantics
- Abstract World
  - Abstract Domain
  - Trace Abstraction
  - Data State Abstraction
  - Partitioning
  - Abstract Semantics

Control Flow Graph

- A directed, connected graph
- Node = basic block
- Flow edge
- Return edge



Why CFG?

- Inevitable due to `goto`s
  (need to know the target of them anyway)
- Natural to treat programs as graphs

Node

- Three types

$$\begin{aligned} n &: \mathit{Node} \\ n &= n \mid \texttt{ENTRY} \mid \texttt{EXIT} \end{aligned}$$

- Has associated basic block

$$\mathit{blkof} : \mathit{Node} \rightarrow \mathit{Block}$$

Flow Edges

- Successors = set of nodes control flow to

$$succ \quad : \quad Node \longrightarrow 2^{Node}$$

- Predecessors = set of nodes control flow from

$$pred \quad : \quad Node \longrightarrow 2^{Node}$$

Basic Block

- Two types: call or commands

$$\begin{array}{lcl} \mathit{blk} & : & \mathit{Block} \\ \mathit{blk} & ::= & \texttt{CALL} \mid \mathit{cmd}^* \end{array}$$

ENTRY and EXIT

- One CFG for each procedure
- Single ENTRY and single EXIT for a CFG
- ENTRY = where procedure begins
- EXIT = where procedure ends
- C program = set of CFGs

Calls and Returns

- Call node = whose basic block is `CALL`
- Two edges for a procedure call
  - call node to procedure's `ENTRY`
  - procedure's `EXIT` to return node
- Return edge = return node for a call

$$rtrn \quad : \quad Node \rightarrow Node$$

Now,

- Program as Graph
  - Control Flow Graph
  - Commands
- Concrete World
  - Concrete Domain
  - Concrete Semantics
- Abstract World
  - Abstract Domain
  - Trace Abstraction
  - Data State Abstraction
  - Partitioning
  - Abstract Semantics

Commands

block	*blk*	::=	cmd^* | `CALL`
command	*cmd*	::=	`SET(`*lv*`,`*e*`)`
		|	`ALLOC(`*lv*`,`*a*`)`
		|	`ASSERT(`*r*`)`
		|	`ESCAPE(`*e*`)`

Expressesions

expression	e	::=	$n \mid e + e \mid lv \mid \&lv$
lvalue	lv	::=	$x \mid *e \mid e[e] \mid e.x$
allocation	a	::=	$[e] \mid \{x^*\}$
relation	r	::=	$e{=}e \mid e{<}e \mid !r$

Now,

- Program as Graph
  - Control Flow Graph
  - Commands
- Concrete World
  - Concrete Domain
  - Concrete Semantics
- Abstract World
  - Abstract Domain
  - Trace Abstraction
  - Data State Abstraction
  - Partitioning
  - Abstract Semantics

State

$$State \quad = \quad Pos \times Mem \times Alloc$$

Control State

$$
\begin{aligned}
Pos &= Ctx \times Node \\
Ctx &= (ProcId \times RtPos \times EscAddr)^* \\
RtPos &= Pos \\
EscAddr &= Addr
\end{aligned}
$$

Data State

$$
\begin{aligned}
Mem &= Addr \xrightarrow{\text{fin}} Val \\
Addr &= Region \times SubAddr + Ctx \times Var \\
SubAddr &= Index + FieldName \\
Val &= Num + Addr + Proc \\
Proc &= ProcId \times Var^*
\end{aligned}
$$

Allocation State

$$
\begin{aligned}
\mathit{Alloc} &= \mathit{Region} \xrightarrow{\text{fin}} \mathit{Info} \\
\mathit{Region} &= \mathit{Ctx} \times \mathit{History} \times \mathit{AllocSite} \\
\mathit{History} &= \mathit{Pos}^* \\
\mathit{AllocSite} &= \mathit{Pos} \\
\mathit{Info} &= \mathit{Size} + \mathit{FieldName}^* \\
\mathit{Size} &= \mathbb{Z}
\end{aligned}
$$

Now,

- Program as Graph
  - Control Flow Graph
  - Commands
- Concrete World
  - Concrete Domain
  - Concrete Semantics
- Abstract World
  - Abstract Domain
  - Trace Abstraction
  - Data State Abstraction
  - Partitioning
  - Abstract Semantics

Semantics as Trace

- Standard semantics

$$Trace = State^*$$

- Collecting semantics

$$2^{Trace}$$

Transition

State after a single transition by $\rightsquigarrow$

$$\begin{array}{rcl} \tau & : & \mathit{Trace} \\ \sigma, \sigma' & : & \mathit{State} \\ \tau\sigma & \rightsquigarrow & \sigma' \end{array}$$

$$\ldots$$

- Transition depends on the *Node* of *Pos*
- Node of block is either cmd^* or `CALL`

Reaction

$$\mathcal{R}\ cmd^* \quad : \quad (Mem \times Alloc) \rightarrow (Mem \times Alloc)$$

$$\ldots$$

Values

$$
\begin{array}{rcl}
\mathcal{V}\ e & : & \mathit{Mem} \to \mathit{Val} \\
\mathcal{V}\ n\ m & = & n \\
\mathcal{V}\ e_1 + e_2\ m & = & (\mathcal{V}\ e_1\ m\ +\ \mathcal{V}\ e_2\ m) \\
\mathcal{V}\ \mathit{lv}\ m & = & m\ (\mathcal{L}\ \mathit{lv}\ m) \\
\mathcal{V}\ \&\mathit{lv}\ m & = & \mathcal{L}\ \mathit{lv}\ m
\end{array}
$$

Addresses

$$
\begin{array}{rcl}
\mathcal{L}\ \mathit{lv} & : & \mathit{Mem} \to \mathit{Addr} \\
\mathcal{L}\ x\ m & = & (\mathit{ctx}, x) \\
\mathcal{L}\ *e\ m & = & \mathcal{V}\ e\ m \\
\mathcal{L}\ e_1[e_2]\ m & = & (a, i + (\mathcal{V}\ e_2\ m)) \text{ where } (a, i) = \mathcal{V}\ e_1\ m \\
\mathcal{L}\ e.x\ m & = & (a, x) \text{ where } (a, 0) = \mathcal{V}\ e\ m
\end{array}
$$

Relations

$$\mathcal{P}\ r \quad : \quad Mem \rightarrow \{\text{true}, \text{false}\}$$

$$\ldots$$

Now,

- Program as Graph
  - Control Flow Graph
  - Commands
- Concrete World
  - Concrete Domain
  - Concrete Semantics
- Abstract World
  - Abstract Domain
  - Trace Abstraction
  - Data State Abstraction
  - Partitioning
  - Abstract Semantics

Control State

$$
\begin{aligned}
\hat{Graph} &= 2^{\hat{Pos} \times \hat{Pos}} \\
\hat{Pos} &= \hat{Ctx} \times Node \\
\hat{Ctx} &= ProcId
\end{aligned}
$$

$$
\begin{aligned}
\alpha_{Flow} &: 2^{Trace} \longrightarrow \hat{Graph} \\
pp &: Pos \longrightarrow \hat{Pos} \\
cp &: Ctx \longrightarrow \hat{Ctx}
\end{aligned}
$$

Data State

$$\begin{aligned} \hat{Table} &= \hat{Pos} \xrightarrow{\text{fin}} \hat{Mem} \\ \hat{Mem} &= \hat{Addr} \xrightarrow{\text{fin}} \hat{Val} \end{aligned}$$

$$\begin{aligned} \alpha_{Data} &: 2^{Trace} \longrightarrow \hat{Table} \\ \alpha_{Mem} &: 2^{Mem} \longrightarrow \hat{Mem} \end{aligned}$$

Abstract Values

$$
\begin{aligned}
\hat{Val} &= \hat{\mathbb{Z}} \times \hat{Addr} \times \hat{Proc} \\
\hat{Addr} &= 2^{\hat{Region} \times \hat{SubAddr} + \hat{Ctx} \times Var} \\
\hat{SubAddr} &= \hat{Index} + FieldName \\
\hat{Proc} &= 2^{Proc}
\end{aligned}
$$

$$
\begin{aligned}
\alpha_{Val} &: 2^{Val} \rightarrow \hat{Val} \\
\alpha_{Addr} &: 2^{Addr} \rightarrow \hat{Addr} \\
\alpha_{Num} &: 2^{Num} \rightarrow \hat{\mathbb{Z}}
\end{aligned}
$$

Allocation State

$$
\begin{aligned}
\hat{Alloc} &= \hat{Region} \xrightarrow{\text{fin}} \hat{Info} \\
\hat{Info} &= \hat{Size} \times 2^{FieldName^*} \\
\hat{Region} &= \hat{Ctx} \times AllocSite
\end{aligned}
$$

$$
\begin{aligned}
\alpha_{Alloc} &: 2^{Trace} \longrightarrow \hat{Alloc} \\
\alpha_{Info} &: 2^{Info} \longrightarrow \hat{Info}
\end{aligned}
$$

Escape Addresses

$$\hat{Dump} = \hat{Ctx} \xrightarrow{\text{fin}} \hat{Addr}$$

$$\alpha_{Dump} : 2^{Trace} \longrightarrow \hat{Dump}$$

Now,

- Program as Graph
  - Control Flow Graph
  - Commands
- Concrete World
  - Concrete Domain
  - Concrete Semantics
- Abstract World
  - Abstract Domain
  - Trace Abstraction
  - Data State Abstraction
  - Partitioning
  - Abstract Semantics

Trace Abstraction

Four components of abstract semantics

$$(\hat{Graph} \times \hat{Table} \times \hat{Alloc} \times \hat{Dump})$$

$$\begin{array}{lcl} \alpha & : & 2^{Trace} \rightarrow (\hat{Graph} \times \hat{Table} \times \hat{Alloc} \times \hat{Dump}) \\ \alpha & = & \langle \alpha_{Flow}, \alpha_{Data}, \alpha_{Alloc}, \alpha_{Dump} \rangle \end{array}$$

Control State Abstraction

$$\alpha_{\mathit{Flow}} \;:\; 2^{\mathit{Trace}} \to \hat{\mathit{Graph}}$$

$$\begin{aligned}\alpha_{\mathit{Flow}}\, T \;=\; \{&((\mathit{cp}\ c_1, n_1), (\mathit{cp}\ c_2, n_2)) \mid \\ &(\cdots, (p_1, m_1, a_1), (p_2, m_2, a_2), \cdots) \in T, \\ &(c_1, n_1) = p_1, (c_2, n_2) = p_2\}\end{aligned}$$

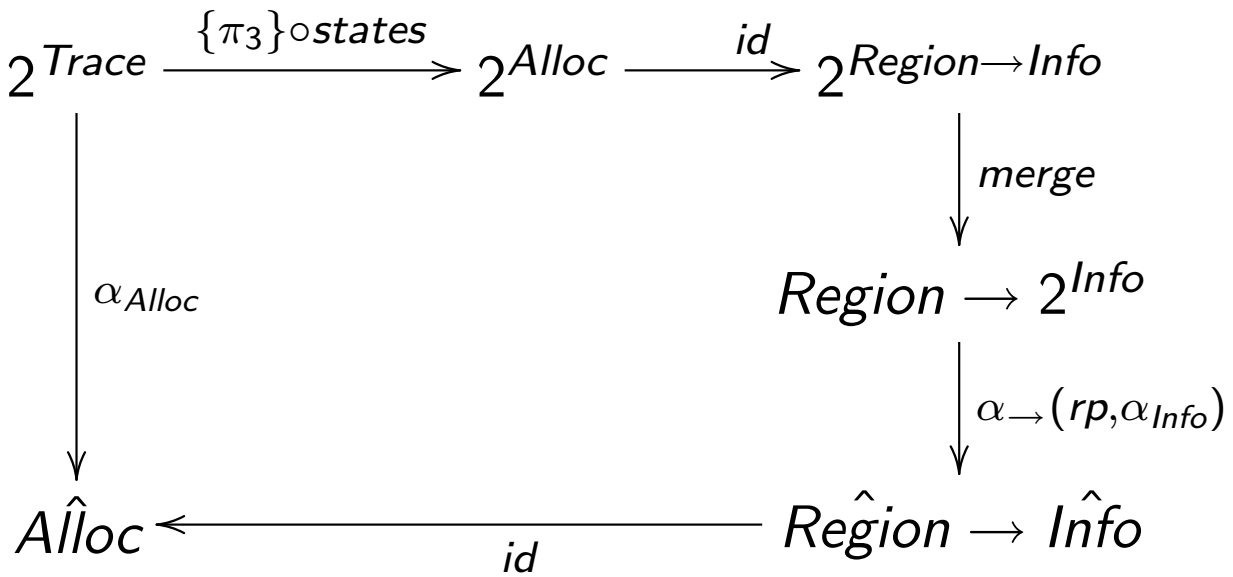
Slicing Traces

$$states \quad : \quad 2^{Trace} \longrightarrow 2^{Pos \times Mem \times Alloc}$$

$$\begin{array}{ccc} 2^{Trace} & \xrightarrow{\quad id \quad} & 2^{State^*} \\ \Big\downarrow{\scriptstyle states} & & \Big\downarrow{\scriptstyle \bigcup \circ \{slices\}} \\ 2^{Pos \times Mem \times Alloc} & \xleftarrow[\quad id \quad]{} & 2^{State} \end{array}$$

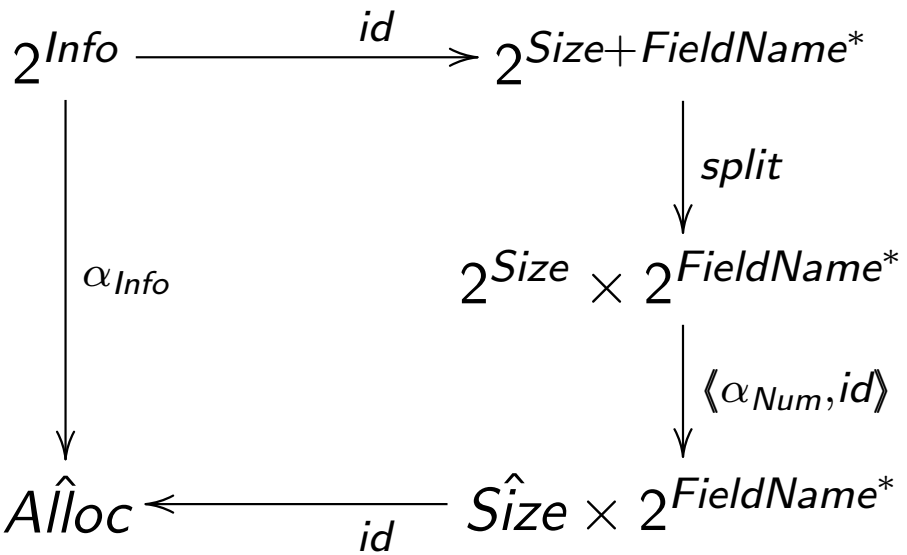
Allocation State Abstraction

$$\alpha_{Alloc} \quad : \quad 2^{Trace} \longrightarrow \hat{Alloc}$$



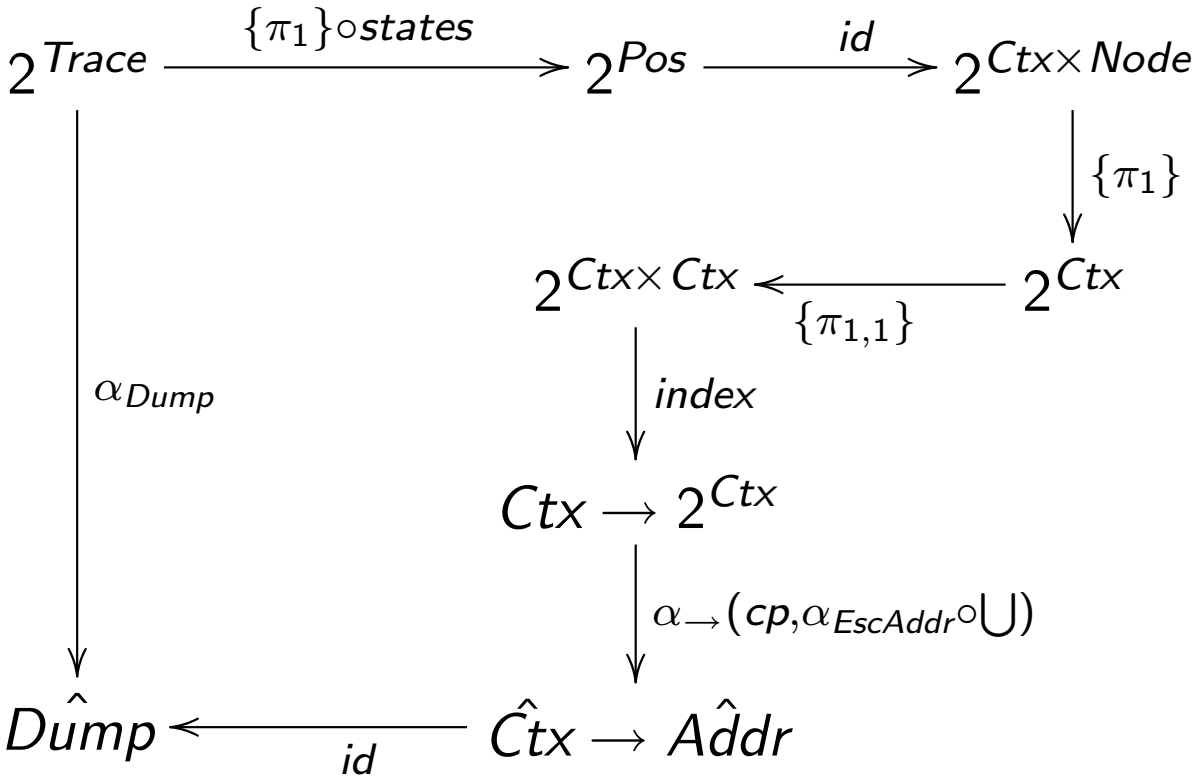
Allocation Info Abstraction

$$\alpha_{Info} \quad : \quad 2^{Info} \rightarrow A\hat{l}loc$$

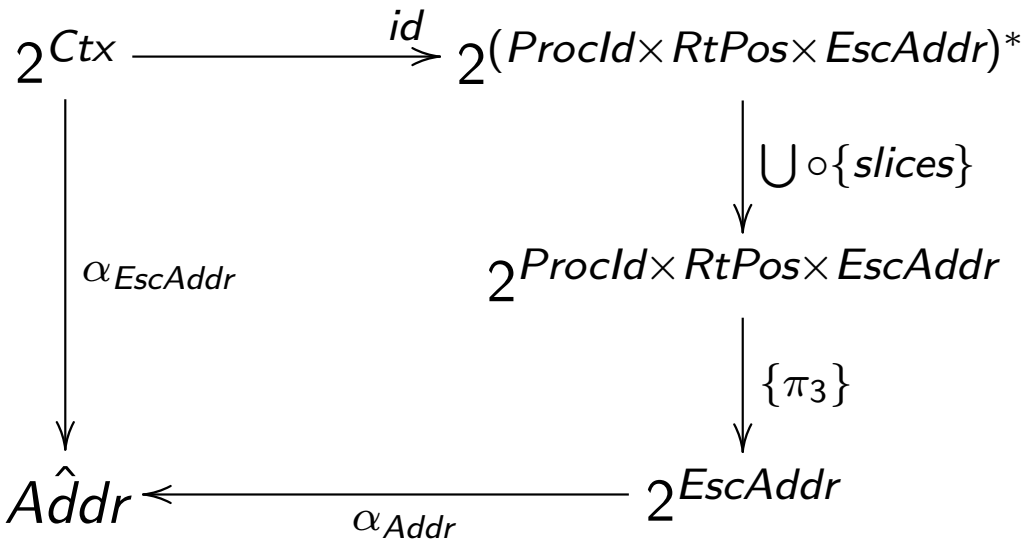


Dump Abstraction

$$\alpha_{Dump} : 2^{Trace} \rightarrow \hat{Dump}$$



Escape Address Abstraction

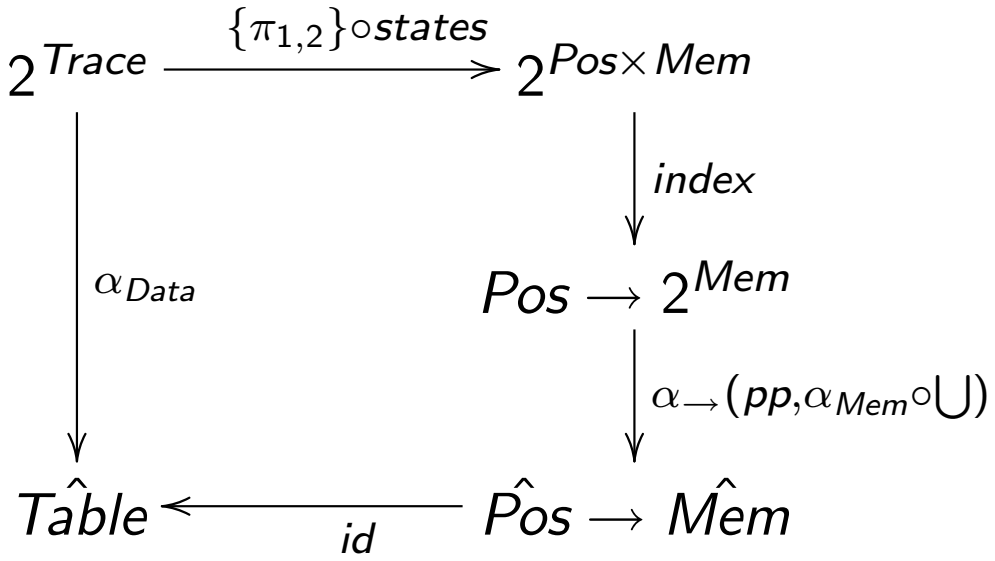


Now,

- Program as Graph
  - Control Flow Graph
  - Commands
- Concrete World
  - Concrete Domain
  - Concrete Semantics
- Abstract World
  - Abstract Domain
  - Trace Abstraction
  - Data State Abstraction
  - Partitioning
  - Abstract Semantics

Data State Abstraction

$$\alpha_{Data} \quad : \quad 2^{Trace} \longrightarrow \hat{Table}$$



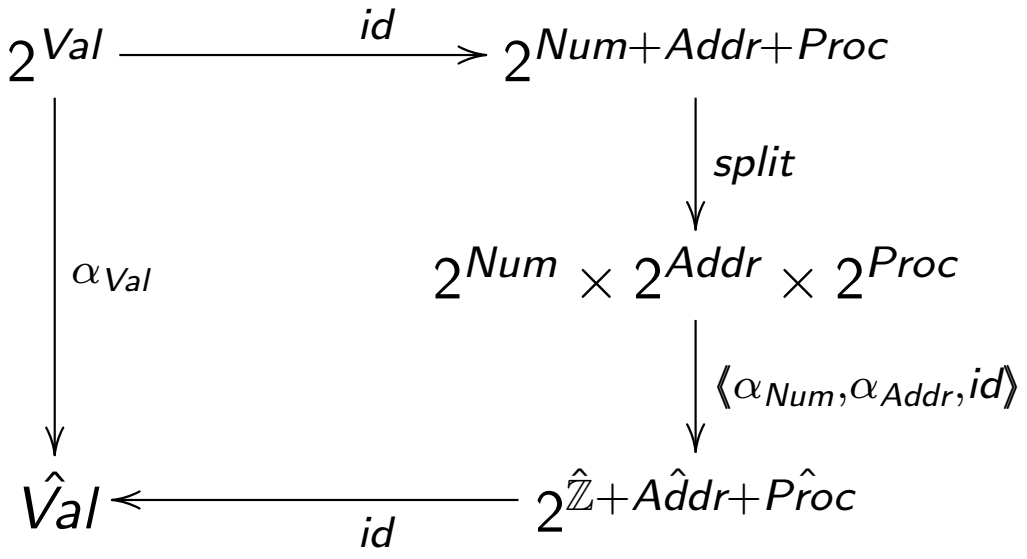
Memory Abstraction

$$\alpha_{Mem} \quad : \quad 2^{Mem} \longrightarrow \hat{Mem}$$

$$\begin{array}{ccc} 2^{Mem} & \xrightarrow{\quad id \quad} & 2^{Addr \rightarrow Val} \\ \Big\downarrow{\scriptstyle \alpha_{Mem}} & & \Big\downarrow{\scriptstyle merge} \\ & & Addr \longrightarrow 2^{Val} \\ & & \Big\downarrow{\scriptstyle \alpha_{\Rightarrow}(\alpha_{Addr}, \alpha_{Val} \circ \bigcup)} \\ \hat{Mem} & \xleftarrow[\quad id \quad]{} & \hat{Addr} \longrightarrow \hat{Val} \end{array}$$

Value Abstraction

$$\alpha_{Val} \quad : \quad Val \rightarrow \hat{Val}$$



Numeral Abstraction

Integer interval domain $\hat{\mathbb{Z}}$

$$\alpha_{Num} \quad : \quad 2^{Num} \rightarrow \hat{\mathbb{Z}}$$

$$\begin{array}{rcl}
\alpha_{Num}\ \emptyset & = & \bot \\
\alpha_{Num}\ N & = & [minN, maxN] \\
min\ N & = & \begin{cases} n' & \text{if } \exists n' \in Num \leq \forall n \in N \\ -\infty & \text{otherwise} \end{cases} \\
max\ N & = & \begin{cases} n' & \text{if } \exists n' \in Num \geq \forall n \in N \\ +\infty & \text{otherwise} \end{cases}
\end{array}$$

Address Abstraction

$$\alpha_{Addr} \quad : \quad 2^{Addr} \longrightarrow \hat{Addr}$$

$$\begin{array}{ccc}
2^{Addr} & \xrightarrow{\quad id \quad} & 2^{Region \times SubAddr + Ctx \times Var} \\
 & & \downarrow split \\
 & & 2^{Region \times SubAddr} \times 2^{Ctx \times Var} \\
\downarrow \alpha_{Addr} & & \downarrow \langle \alpha_{DynAddr}, \langle cp, id \rangle \rangle \\
 & & 2^{\hat{Region} \times \hat{SubAddr}} \times 2^{\hat{Ctx} \times Var} \\
 & & \downarrow mix \\
\hat{Addr} & \xleftarrow[\quad id \quad]{} & 2^{\hat{Region} \times \hat{SubAddr} + \hat{Ctx} \times Var}
\end{array}$$

Dynamic Address Abstraction

$$\alpha_{DynAddr} : 2^{Region \times SubAddr} \longrightarrow 2^{\hat{Region} \times \hat{SubAddr}}$$

$$\begin{array}{ccc} 2^{Region \times SubAddr} & \xrightarrow{index} & Region \longrightarrow 2^{SubAddr} \\ \downarrow \alpha_{DynAddr} & & \downarrow \alpha_{\longrightarrow}(rp, \alpha_{SubAddr}) \\ 2^{\hat{Region} \times \hat{SubAddr}} & \xleftarrow[couple]{} & \hat{Region} \longrightarrow 2^{\hat{SubAddr}} \end{array}$$

Sub-Address Abstraction

$$\alpha_{SubAddr} : 2^{SubAddr} \longrightarrow 2^{\hat{SubAddr}}$$

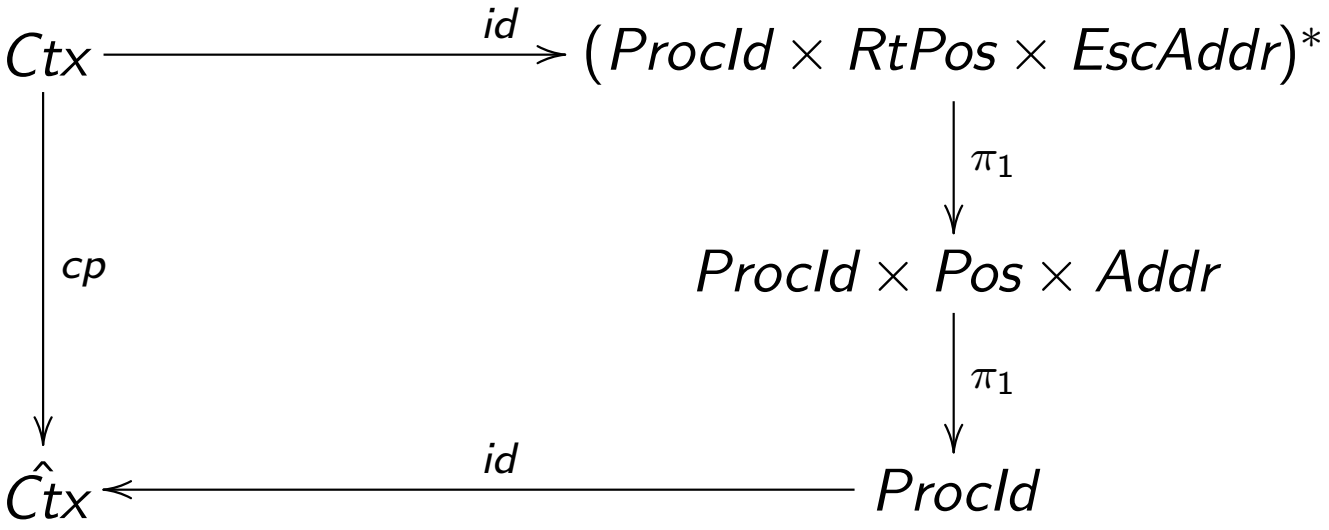
$$\begin{array}{ccc}
2^{SubAddr} & \xrightarrow{\quad id \quad} & 2^{Index+FieldName} \\
 & & \downarrow split \\
 & & 2^{Index} \times 2^{FieldName} \\
\downarrow \alpha_{SubAddr} & & \downarrow \langle \{\} \circ \alpha_{Num}, id \rangle \\
 & & 2^{\hat{Index}} \times 2^{FieldName} \\
 & & \downarrow mix \\
2^{\hat{SubAddr}} & \xleftarrow[id]{} & 2^{\hat{Index}+FieldName}
\end{array}$$

Now,

- Program as Graph
  - Control Flow Graph
  - Commands
- Concrete World
  - Concrete Domain
  - Concrete Semantics
- Abstract World
  - Abstract Domain
  - Trace Abstraction
  - Data State Abstraction
  - Partitioning
  - Abstract Semantics

Context Partitioning

$$cp \quad : \quad Ctx \rightarrow \hat{Ctx}$$



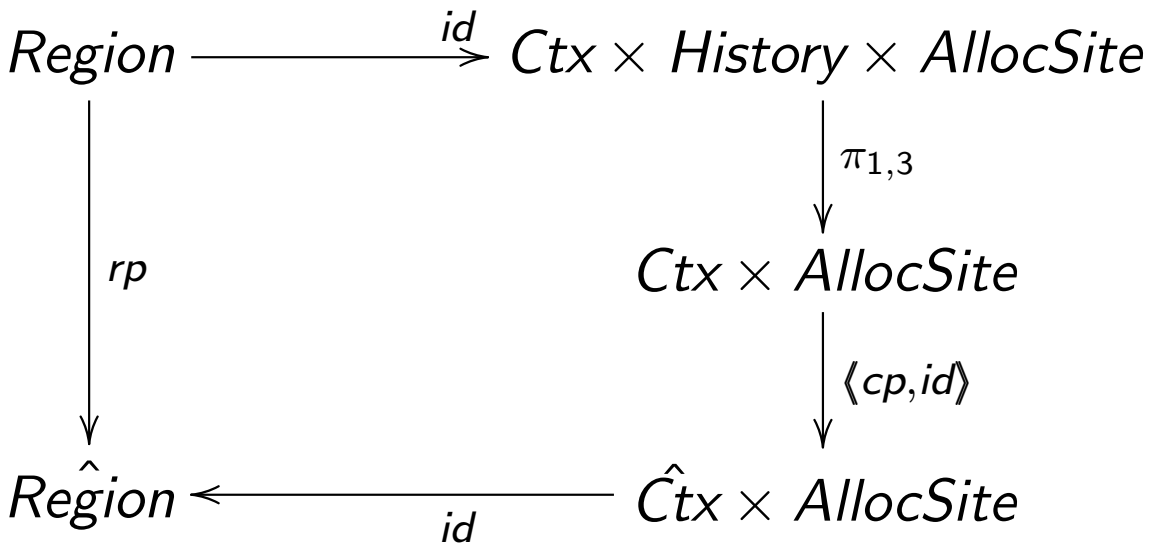
Position Partitioning

$$pp \quad : \quad Pos \longrightarrow \hat{Pos}$$

$$\begin{array}{ccc} Pos & \xrightarrow{id} & Ctx \times Node \\ \downarrow{\scriptstyle pp} & & \downarrow{\scriptstyle \langle cp, id \rangle} \\ \hat{Pos} & \xleftarrow[id]{} & \hat{Ctx} \times Node \end{array}$$

Region Partitioning

$$rp \quad : \quad Region \longrightarrow \hat{Region}$$



Now,

- Program as Graph
  - Control Flow Graph
  - Commands
- Concrete World
  - Concrete Domain
  - Concrete Semantics
- Abstract World
  - Abstract Domain
  - Trace Abstraction
  - Data State Abstraction
  - Partitioning
  - Abstract Semantics

The Equation

Abstract Semantics as fixed point

...