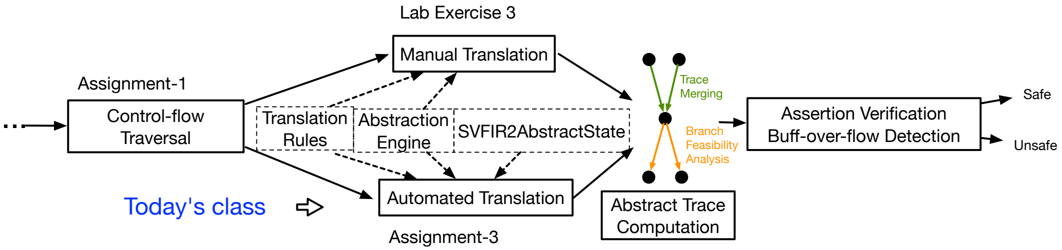
Abstract Interpretation and its Applications

Yulei Sui

School of Computer Science and Engineering

University of New South Wales, Australia

Today’s class



Abstract Execution on Pointer-Free SVFIR

- For simplicity, let's first consider abstract execution on a pointer-free language.
- This means there are no operations for memory allocation (like $\texttt{p} = \texttt{alloc}_o$) or for indirect memory accesses (such as `p = *q` or `*p = q`).
- Here are the pointer-free SVFStmts and their C-like forms:

SVFStmt	**C-Like form**
ConsStmt	$\ell : \texttt{p} = \texttt{c}$
CopyStmt	$\ell : \texttt{p} = \texttt{q}$
BinaryStmt	$\ell : \texttt{r} = \texttt{p} \otimes \texttt{q}$
PhiStmt	$\ell : \texttt{r} = \texttt{phi}(\texttt{p}_1, \texttt{p}_2, \ldots, \texttt{p}_n)$
Sequence	$\ell_1 ; \ell_2$
BranchStmt	ℓ_1 : if$(x < c)$ then ℓ_2 else ℓ_3

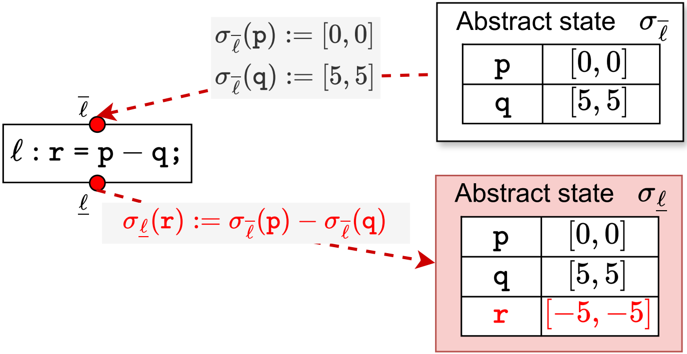
Abstract Execution Rules on Pointer-Free SVFIR

Let's use the *Interval* abstract domain to update σ based on the following rules for different SVFSTMT:

SVFSTMT	**C-Like form**	**Abstract Execution Rule**
CONSSTMT	$\ell : \mathtt{p} = \mathtt{c}$	$\sigma_{\underline{\ell}}(\mathrm{p}) := [\mathrm{c}, \mathrm{c}]$
COPYSTMT	$\ell : \mathtt{p} = \mathtt{q}$	$\sigma_{\underline{\ell}}(\mathrm{p}) := \sigma_{\overline{\ell}}(\mathrm{q})$
BINARYSTMT	$\ell : \mathtt{r} = \mathtt{p} \otimes \mathtt{q}$	$\sigma_{\underline{\ell}}(r) := \sigma_{\overline{\ell}}(p) \hat{\otimes} \sigma_{\overline{\ell}}(q)$
PHISTMT	$\ell : \mathtt{r} = \mathtt{phi}(\mathtt{p_1}, \mathtt{p_2}, \ldots, \mathtt{p_n})$	$\sigma_{\underline{\ell}}(r) := \bigsqcup_{i=1}^{n} \sigma_{\overline{\ell}}(p_i)$
SEQUENCE	$\ell_1 ; \ell_2$	$\forall v \in \mathcal{V}, \sigma_{\overline{\ell_2}}(v) \sqsupseteq \sigma_{\underline{\ell_1}}(v)$
BRANCHSTMT	ℓ_1 : if$(x < c)$ then ℓ_2 else ℓ_3	$\sigma_{\overline{\ell_2}}(x) := \sigma_{\underline{\ell_1}}(x) \sqcap [-\infty, c-1]$, if $\sigma_{\underline{\ell_1}}(x) \sqcap [-\infty, c-1] \neq \bot$ $\sigma_{\overline{\ell_3}}(x) := \sigma_{\underline{\ell_1}}(x) \sqcap [c, +\infty]$, if $\sigma_{\underline{\ell_1}}(x) \sqcap [c, +\infty] \neq \bot$

An Example: Abstract Execution on BinaryStmt

SVFStmt	C-Like form	Abstract Execution Rule
BinaryStmt	$\ell : \mathtt{r} = \mathtt{p} \otimes \mathtt{q}$	$\sigma_{\underline{\ell}}(r) := \sigma_{\overline{\ell}}(p) \hat{\otimes} \sigma_{\overline{\ell}}(q)$



Abstract Execution on SVFIR in the Presence of Pointers

- SVFIR in the presence of pointers contain pointer-related statements including AddrStmt, GepStmt, LoadStmt and StoreStmt.

SVFStmt	**C-Like form**
ConsStmt	$\ell : \mathtt{p} = \mathtt{c}$
CopyStmt	$\ell : \mathtt{p} = \mathtt{q}$
BinaryStmt	$\ell : \mathtt{r} = \mathtt{p} \otimes \mathtt{q}$
PhiStmt	$\ell : \mathtt{r} = \mathtt{phi}(\mathtt{p_1}, \mathtt{p_2}, \ldots, \mathtt{p_n})$
Sequence	$\ell_1 ; \ell_2$
BranchStmt	ℓ_1 : if($x < c$) then ℓ_2 else ℓ_3
AddrStmt	$\ell : \mathtt{p} = \mathtt{alloc}$
GepStmt	$\ell : \mathtt{p} = \&(\mathtt{q} \rightarrow \mathtt{i})$ or $\mathtt{p} = \&\mathtt{q}[\mathtt{i}]$
LoadStmt	$\ell : \mathtt{p} = *\mathtt{q}$
StoreStmt	$\ell : *\mathtt{p} = \mathtt{q}$

Abstract Execution on SVFIR in the Presence of Pointers

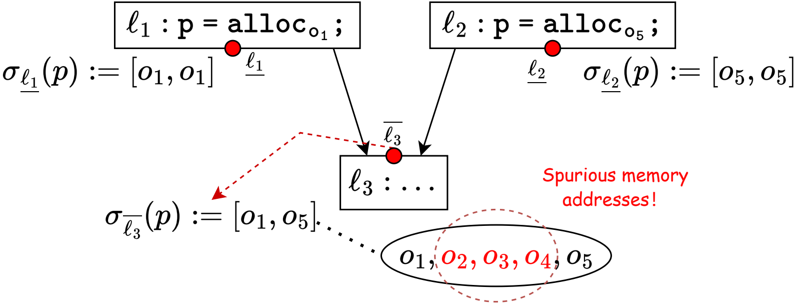
An Example

Let's try analyzing this kind of SVFIR using the same way as we did for pointer-free SVFIR based on a single interval domain.

Abstract Execution on SVFIR in the Presence of Pointers

An Example

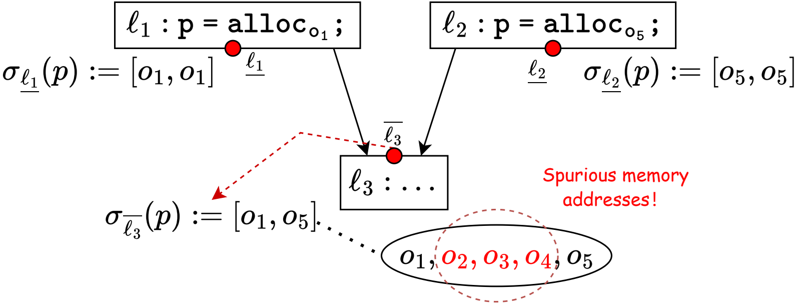
Let's try analyzing this kind of SVFIR using the same way as we did for pointer-free SVFIR based on a single interval domain.



Abstract Execution on SVFIR in the Presence of Pointers

An Example

Let's try analyzing this kind of SVFIR using the same way as we did for pointer-free SVFIR based on a single interval domain.



❌ Using intervals to represent discrete memory address value is imprecise.

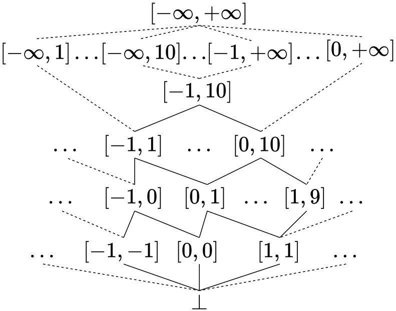
Abstract Execution on SVFIR in the Presence of Pointers

✔ We require **a combination of memory address and interval domains** to precisely and efficiently perform abstract execution on SVFIR in the presence of pointers.

Abstract Execution over Memory Address and Interval Domains

Interval and Memory Address Domains

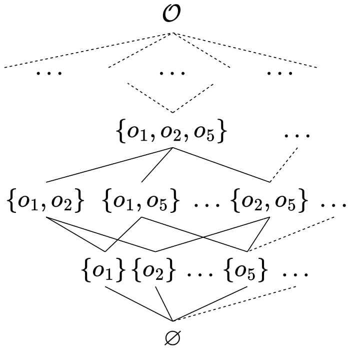
Interval abstraction (*Interval* domain) for scalar variables.



Abstract Execution over Memory Address and Interval Domains

Interval and Memory Address Domains

Discrete values (*MemAddress* domain) for memory addresses.



Abstract Trace for Memory Address and Interval Domains

- The abstract trace for memory address and interval domains is defined as:

	Notation	**Domain**	**Implementation**
Abstract trace	σ	$\mathbb{L} \times \mathcal{V} \rightarrow \mathit{Interval} \times \mathit{MemAddress}$	*preAbstractTrace*, *postAbstractTrace*
Abstract state	σ_L	$\mathcal{V} \rightarrow \mathit{Interval} \times \mathit{MemAddress}$	*AbstractState.varToAbsVal*
Abstract value	$\sigma_L(p)$	$\mathit{Interval} \times \mathit{MemAddress}$	*AbstractValue*

Abstract Trace for Memory Address and Interval Domains

- The abstract trace for memory address and interval domains is defined as:

	Notation	Domain	Implementation
Abstract trace	σ	$\mathbb{L} \times \mathcal{V} \to \mathit{Interval} \times \mathit{MemAddress}$	*preAbstractTrace*, *postAbstractTrace*
Abstract state	σ_L	$\mathcal{V} \to \mathit{Interval} \times \mathit{MemAddress}$	*AbstractState.varToAbsVal*
Abstract value	$\sigma_L(p)$	$\mathit{Interval} \times \mathit{MemAddress}$	*AbstractValue*

- *Interval* is used for tracking the interval value of **scalar variables.**

Abstract Trace for Memory Address and Interval Domains

- The abstract trace for memory address and interval domains is defined as:

	Notation	Domain	Implementation
Abstract trace	σ	$\mathbb{L} \times \mathcal{V} \rightarrow \mathit{Interval} \times \mathit{MemAddress}$	$\mathit{preAbstractTrace}, \mathit{postAbstractTrace}$
Abstract state	σ_L	$\mathcal{V} \rightarrow \mathit{Interval} \times \mathit{MemAddress}$	$\mathit{AbstractState.varToAbsVal}$
Abstract value	$\sigma_L(p)$	$\mathit{Interval} \times \mathit{MemAddress}$	$\mathit{AbstractValue}$

- *Interval* is used for tracking the interval value of **scalar variables.**
- *MemAddress* is used for tracking the memory addresses of **memory address variables.**

Abstract Trace for Memory Address and Interval Domains

Cross-Domain Interaction

- During abstract execution, the memory address domain and the interval domain interact with each other.

Abstract Trace for Memory Address and Interval Domains

Cross-Domain Interaction

- During abstract execution, the memory address domain and the interval domain interact with each other.
- To track the value to value correlation at each program point, we define: $\delta \in \mathbb{L} \times \mathit{MemAddress} \rightarrow \mathit{Interval} \times \mathit{MemAddress}$.

Abstract Trace for Memory Address and Interval Domains

Cross-Domain Interaction

- During abstract execution, the memory address domain and the interval domain interact with each other.
- To track the value to value correlation at each program point, we define: $\delta \in \mathbb{L} \times \mathit{MemAddress} \rightarrow \mathit{Interval} \times \mathit{MemAddress}$.
- For top-level variables, we still use $\sigma \in \mathbb{L} \times \mathcal{P} \rightarrow \mathit{Interval} \times \mathit{MemAddress}$ to track the memory addresses or interval values of these variables.

Abstract Trace for Memory Address and Interval Domains

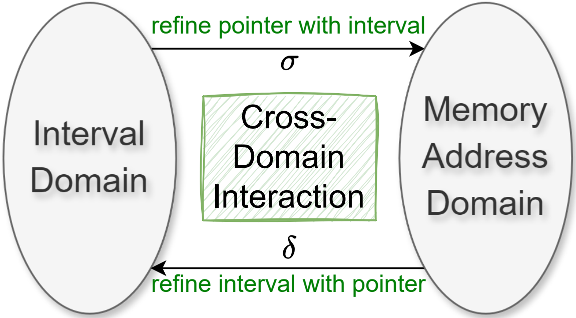
Cross-Domain Interaction

- During abstract execution, the memory address domain and the interval domain interact with each other.
- To track the value to value correlation at each program point, we define: $\delta \in \mathbb{L} \times \textit{MemAddress} \rightarrow \textit{Interval} \times \textit{MemAddress}$.
- For top-level variables, we still use $\sigma \in \mathbb{L} \times \mathcal{P} \rightarrow \textit{Interval} \times \textit{MemAddress}$ to track the memory addresses or interval values of these variables.

	Notation	Domain	Implementation
Abstract trace	σ	$\mathbb{L} \times \mathcal{P} \rightarrow \textit{Interval} \times \textit{MemAddress}$	*preAbstractTrace*, *postAbstractTrace*
	δ	$\mathbb{L} \times \textit{MemAddress} \rightarrow \textit{Interval} \times \textit{MemAddress}$	
Abstract state	σ_L	$\mathcal{P} \rightarrow \textit{Interval} \times \textit{MemAddress}$	*AbstractState.varToAbsVal*
	δ_L	$\textit{MemAddress} \rightarrow \textit{Interval} \times \textit{MemAddress}$	*AbstractState.locToAbsVal*
Abstract value	$\sigma_L(p)$	$\textit{Interval} \times \textit{MemAddress}$	*AbstractValue*
	$\delta_L(o)$		

Abstract Trace for Memory Address and Interval Domains

Cross-Domain Interaction



Abstract Execution Rules on SVFIR in the Presence of Pointers

Now let's use the *Interval* $\times$ *MemAddress* abstract domain to update σ and δ based on the following rules for different SVFSTMT:

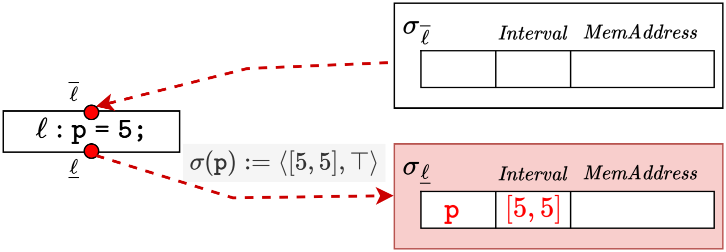
SVFSTMT	**C-Like form**	**Abstract Execution Rule**
CONSSTMT	$\ell : \mathtt{p} = \mathtt{c}$	$\sigma_{\underline{\ell}}(\mathtt{p}) := \langle[\mathtt{c}, \mathtt{c}], \top\rangle$
COPYSTMT	$\ell : \mathtt{p} = \mathtt{q}$	$\sigma_{\underline{\ell}}(\mathtt{p}) := \sigma_{\overline{\ell}}(\mathtt{q})$
BINARYSTMT	$\ell : \mathtt{r} = \mathtt{p} \otimes \mathtt{q}$	$\sigma_{\underline{\ell}}(r) := \sigma_{\overline{\ell}}(p) \hat{\otimes} \sigma_{\overline{\ell}}(q)$
PHISTMT	$\ell : \mathtt{r} = \mathtt{phi}(\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_n)$	$\sigma_{\underline{\ell}}(r) := \bigsqcup_{i=1}^{n} \sigma_{\overline{\ell}}(p_i)$
BRANCHSTMT	ℓ_1 : if$(x < c)$ then ℓ_2 else ℓ_3	$\sigma_{\overline{\ell_2}}(x) := \sigma_{\underline{\ell_1}}(x) \sqcap [-\infty, c-1]$, if $\sigma_{\underline{\ell_1}}(x) \sqcap [-\infty, c-1] \neq \perp$ $\sigma_{\overline{\ell_3}}(x) := \sigma_{\underline{\ell_1}}(x) \sqcap [c, +\infty]$, if $\sigma_{\underline{\ell_1}}(x) \sqcap [c, +\infty] \neq \perp$
SEQUENCE	$\ell_1; \ell_2$	$\delta_{\overline{\ell_2}} \sqsupseteq \delta_{\underline{\ell_1}}, \sigma_{\overline{\ell_2}} \sqsupseteq \sigma_{\underline{\ell_1}}$
ADDRSTMT	$\ell : \mathtt{p} = \mathtt{alloc}_{\mathtt{o_i}}$	$\sigma_{\underline{\ell}}(\mathtt{p}) := \langle\top, \{o_i\}\rangle$
GEPSTMT	$\ell : \mathtt{p} = \&(\mathtt{q} \rightarrow \mathtt{i})$ or $\mathtt{p} = \&\mathtt{q}[\mathtt{i}]$	$\sigma_{\underline{\ell}}(\mathtt{p}) := \bigsqcup_{\mathtt{o} \in \gamma(\sigma_{\overline{\ell}}(\mathtt{q}))} \bigsqcup_{j \in \gamma(\sigma_{\overline{\ell}}(\mathtt{i}))} \langle\top, \{\mathtt{o.fld}_j\}\rangle$
LOADSTMT	$\ell : \mathtt{p} = *\mathtt{q}$	$\sigma_{\underline{\ell}}(\mathtt{p}) := \bigsqcup_{o \in \{o \mid (o \mapsto \_) \in \delta_{\overline{\ell}}\}} \delta_{\overline{\ell}}(o)$
STORESTMT	$\ell : *\mathtt{p} = \mathtt{q}$	$\delta_{\underline{\ell}} := (\{o \mapsto \sigma_{\overline{\ell}}(\mathtt{q}) \mid o \in \gamma(\sigma_{\overline{\ell}}(\mathtt{p}))\} \sqcup \delta_{\overline{\ell}})$

Implementation of Abstract State and Abstract Trace

- For a program point L, the abstract state AS is an instance of the class named *AbstractState*, consisting of:
  - *varToAbsVal* : $\sigma_L \in \mathcal{P} \rightarrow$ *Interval* $\times$ *MemAddress*
  - *locToAbsVal* : $\delta_L \in$ *MemAddress* $\rightarrow$ *Interval* $\times$ *MemAddress*
- The abstract trace is divided into two maps, *preAbstractTrace* and *postAbstractTrace*, which record the abstract states before and after each control flow point respectively.
  - For example, for a control flow node ℓ, *preAbstractTrace*(ℓ) includes $\sigma_{\overline{\ell}}$ and $\delta_{\overline{\ell}}$, and *postAbstractTrace*(ℓ) represents $\sigma_{\underline{\ell}}$ and $\delta_{\underline{\ell}}$.

An Example: Abstract Execution on ConsStmt

SVFStmt	C-Like form	Abstract Execution Rule
ConsStmt	$\ell : \mathtt{p} = \mathtt{c}$	$\sigma_{\underline{\ell}}(\mathtt{p}) := \langle[\mathtt{c}, \mathtt{c}], \top\rangle$

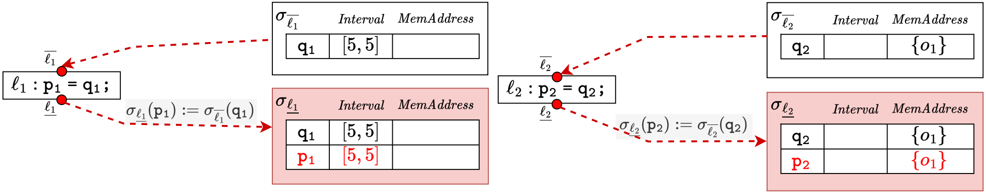


Algorithm 1: Abstract Execution Rule for ConsStmt

postAbstractTrace$[\ell][\ell.lhs]$:= *AbstractValue*$([c, c])$

An Example: Abstract Execution on CopyStmt

SVFStmt	C-Like form	Abstract Execution Rule
CopyStmt	$\ell : p = q$	$\sigma_{\underline{\ell}}(p) := \sigma_{\overline{\ell}}(q)$

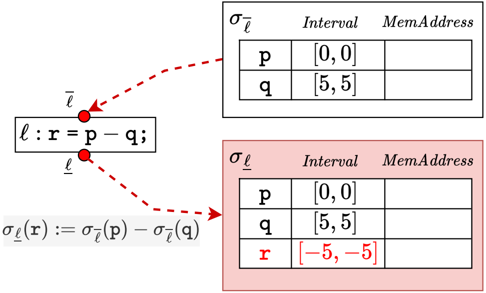


Algorithm 2: Abstract Execution Rule for CopyStmt

postAbstractTrace$[\ell][\ell.$*lhs*$]$:= *preAbstractTrace*$[\ell][\ell.$*rhs*$]$

An Example: Abstract Execution on BINARYSTMT

SVFSTMT	C-Like form	Abstract Execution Rule
BINARYSTMT	$\ell : \mathtt{r} = \mathtt{p} \otimes \mathtt{q}$	$\sigma_{\underline{\ell}}(r) := \sigma_{\overline{\ell}}(p) \hat{\otimes} \sigma_{\overline{\ell}}(q)$

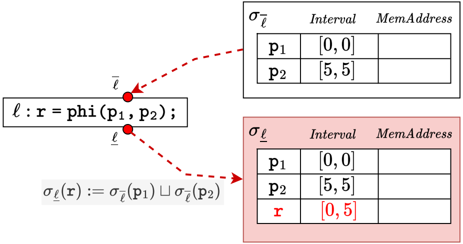


Algorithm 3: Abstract Execution Rule for BINARYSTMT

$postAbstractTrace[\ell][\ell.res] := preAbstractTrace[\ell][\ell.op1] \hat{\otimes} preAbstractTrace[\ell][\ell.op2]$

An Example: Abstract Execution on PhiStmt

SVFStmt	C-Like form	Abstract Execution Rule
PhiStmt	$\ell : \mathtt{r} = \mathtt{phi}(\mathtt{p_1}, \mathtt{p_2}, \ldots, \mathtt{p_n})$	$\sigma_{\underline{\ell}}(r) := \bigsqcup_{i=1}^{n} \sigma_{\overline{\ell}}(p_i)$

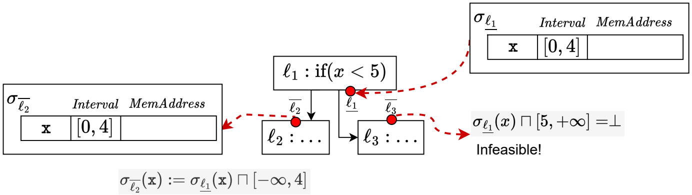


Algorithm 4: Abstract Execution Rule for PhiStmt

rhsVal := *UnknownAbsVal*
for *op* $\in \ell$.*ops* **do**
  rhsVal.join_with(*preAbstractTrace*[ℓ][*op*])
postAbstractTrace[ℓ][ℓ.*res*] := *rhsVal*

An Example: Abstract Execution on BRANCHSTMT

SVFSTMT	C-Like form	Abstract Execution Rule
BRANCHSTMT	ℓ_1 : if($x < c$) then ℓ_2 else ℓ_3	$\sigma_{\overline{\ell_2}}(x) := \sigma_{\underline{\ell_1}}(x) \sqcap [-\infty, c-1]$, if $\sigma_{\underline{\ell_1}}(x) \sqcap [-\infty, c-1] \neq \perp$ $\sigma_{\overline{\ell_3}}(x) := \sigma_{\underline{\ell_1}}(x) \sqcap [c, +\infty]$, if $\sigma_{\underline{\ell_1}}(x) \sqcap [c, +\infty] \neq \perp$

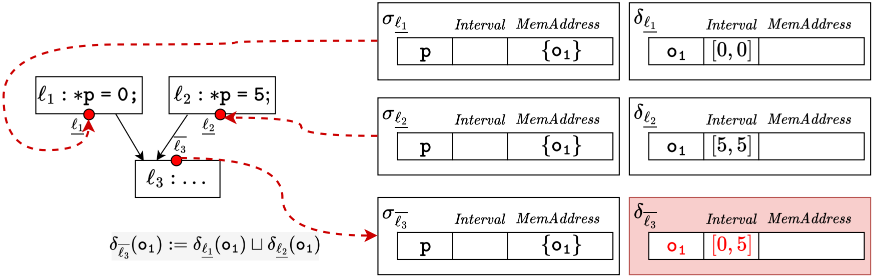


Algorithm 5: Abstract Execution Rule for BRANCHSTMT

$trueCond := postAbstractTrace[\ell_1][x] \sqcap AbstractValue([-\infty, c-1])$
if $\neg trueCond.is\_bottom()$ **then**
  $preAbstractTrace[\ell_2][x] := postAbstractTrace[\ell_1][x] \sqcap AbstractValue([-\infty, c-1])$
$falseCond := postAbstractTrace[\ell_1][x] \sqcap AbstractValue([c, +\infty])$
if $\neg falseCond.is\_bottom()$ **then**
  $preAbstractTrace[\ell_3][x] := postAbstractTrace[\ell_1][x] \sqcap AbstractValue([c, +\infty])$

An Example: Abstract Execution on SEQUENCE

SVFSTMT	C-Like form	Abstract Execution Rule
SEQUENCE	$\ell_1; \ell_2$	$\delta_{\overline{\ell_2}} \sqsupseteq \delta_{\underline{\ell_1}}, \sigma_{\overline{\ell_2}} \sqsupseteq \sigma_{\underline{\ell_1}}$

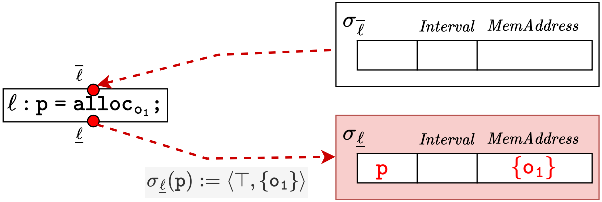


Algorithm 6: Abstract Execution Rule for SEQUENCE

for $\ell' \in$ *predecessors*(ℓ) **do**
  preAbstractTrace[ℓ].*joinWith*(*postAbstractTrace*[ℓ'])

An Example: Abstract Execution on ADDRSTMT

SVFSTMT	C-Like form	Abstract Execution Rule
ADDRSTMT	$\ell : \mathtt{p} = \mathtt{alloc}_{\mathtt{o_i}}$	$\sigma_{\underline{\ell}}(\mathtt{p}) := \langle \top, \{o_i\} \rangle$



Algorithm 7: Abstract Execution Rule for ADDRSTMT

1 *postAbstractTrace*$[\ell][\ell.lhs] :=$ *AbstractValue*$(\{o_i\})$

An Example: Abstract Execution on GEPSTMT

SVFSTMT	C-Like form	Abstract Execution Rule
GEPSTMT	ℓ : `p = &(q → i)` or `p = &q[i]`	$\sigma_{\underline{\ell}}(\mathtt{p}) := \bigsqcup_{o \in \gamma(\sigma_{\overline{\ell}}(\mathtt{q}))} \bigsqcup_{j \in \gamma(\sigma_{\overline{\ell}}(\mathtt{i}))} \langle \top, \{\mathtt{o.fld}_j\} \rangle$

$\overline{\ell}$

ℓ : `p = &(q[i]);`

$\underline{\ell}$

$\sigma_{\underline{\ell}}(\mathtt{p}) := \langle \top, \{\mathtt{o_1.fld_1}\} \rangle \sqcup \langle \top, \{\mathtt{o_1.fld_2}\} \rangle$

$\sigma_{\overline{\ell}}$

	Interval	*MemAddress*
q		$\{\mathtt{o_1}\}$
i	$[1, 2]$	

$\sigma_{\underline{\ell}}$

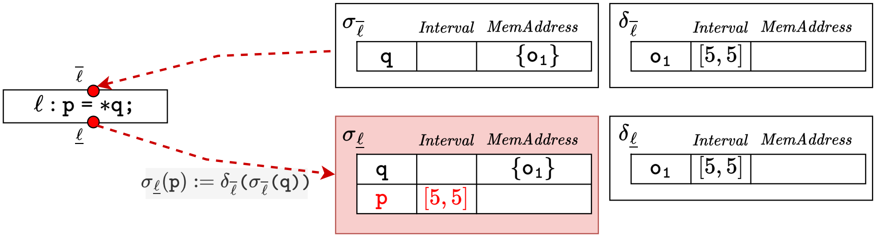
	Interval	*MemAddress*
q		$\{\mathtt{o_1}\}$
i	$[1, 2]$	
p		$\{\mathtt{o_1.fld_1}, \mathtt{o_1.fld_2}\}$

Algorithm 8: Abstract Execution Rule for GEPSTMT

gepAbsVal := *UnknownAbsVal*
offsetAbsVal := *preAbstractTrace*[ℓ][ℓ.*offset*]
for *idx* $\in$ [*offsetAbsVal*.*lb*(), *offsetAbsVal*.*ub*()] **do**
    gepAbsVal.*join_with*(*getGepObjAddress*(ℓ.*base*, *idx*))
postAbstractTrace[ℓ][ℓ.*res*] := *gepAbsVal*

An Example: Abstract Execution on LOADSTMT

SVFSTMT	C-Like form	Abstract Execution Rule
LOADSTMT	$\ell : \texttt{p} = *\texttt{q}$	$\sigma_{\ell}(\texttt{p}) := \bigsqcup_{o \in \{o \mid (o \mapsto \_) \in \delta_{\overline{\ell}}\}} \delta_{\overline{\ell}}(o)$

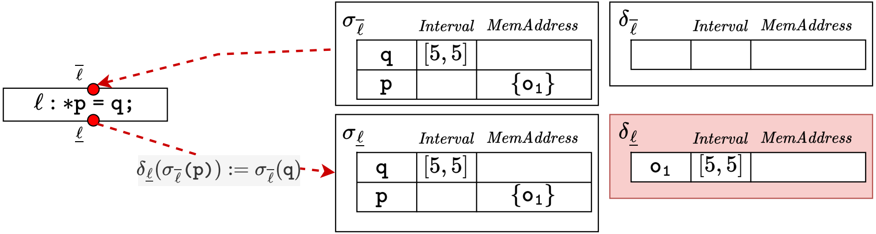


Algorithm 9: Abstract Execution Rule for LOADSTMT

rhsVal := *UnknownAbsVal*
tmpAS := *preAbstractTrace*[ℓ]
for *addr* $\in$ *tmpAS*[ℓ.*rhs*] **do**
    rhsVal.*join_with*(*tmpAS*.*load*(*addr*))
postAbstractTrace[ℓ][ℓ.*lhs*] := *rhsVal*

An Example: Abstract Execution on StoreStmt

SVFStmt	C-Like form	Abstract Execution Rule
StoreStmt	$\ell : *\mathtt{p} = \mathtt{q}$	$\delta_{\underline{\ell}} := (\{o \mapsto \sigma_{\overline{\ell}}(\mathtt{q}) \mid o \in \gamma(\sigma_{\overline{\ell}}(\mathtt{p}))\} \sqcup \delta_{\overline{\ell}})$



Algorithm 10: Abstract Execution Rule for StoreStmt

tmpAS := *preAbstractTrace*[ℓ]
for *addr* $\in$ *tmpAS*[ℓ.*lhs*] **do**
  tmpAS.store(*addr*, *tmpAS*[ℓ.*rhs*])
postAbstractTrace[ℓ] := *tmpAS*