Precise Sparse Abstract Execution via Cross-Domain Interaction

ICSE 2024

Xiao Cheng, Jiawei Wang, Yulei Sui

xiao.cheng@unsw.edu.au

Computer Science and Engineering
UNSW Sydney

April 24, 2024

Contribution

- A precise **cross-domain abstract execution/interpretation** over a combined domain through **correlation tracking**.

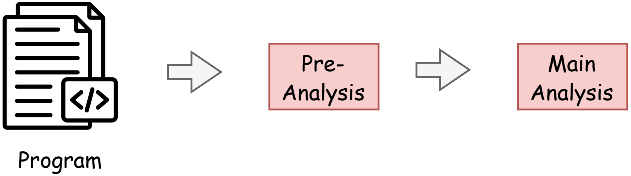
Contribution

- A precise **cross-domain abstract execution/interpretation** over a combined domain through **correlation tracking**.
- An **implication-equivalent (virtual) memory address grouping approach**.

Contribution

- A precise **cross-domain abstract execution/interpretation** over a combined domain through **correlation tracking**.
- An **implication-equivalent (virtual) memory address grouping approach**.
- Significantly **boost the precision and efficiency of assertion-checking clients**, e.g., buffer overflow and null dereference detection.

Staged Analysis

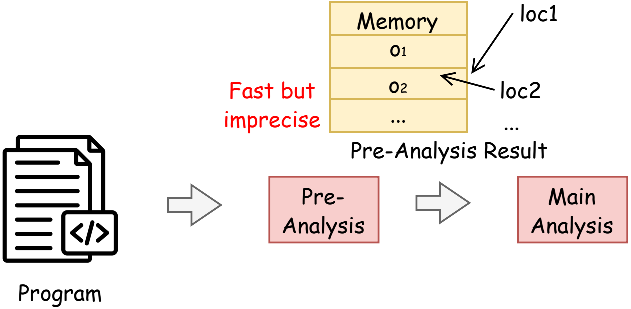


Program

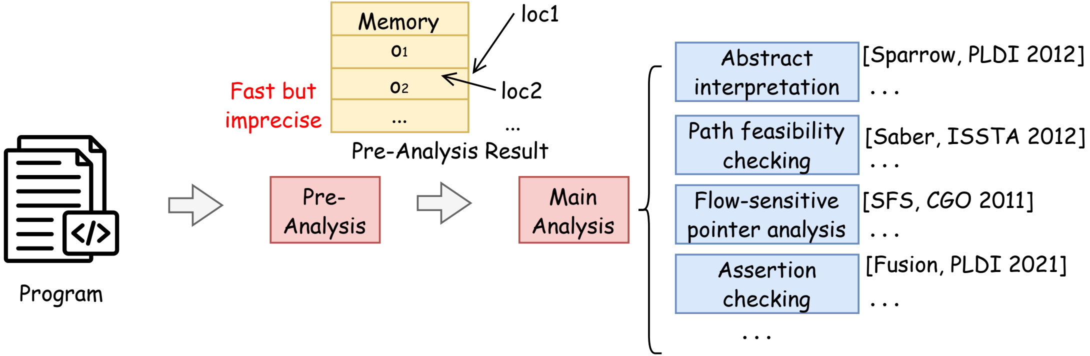
Pre-Analysis

Main Analysis

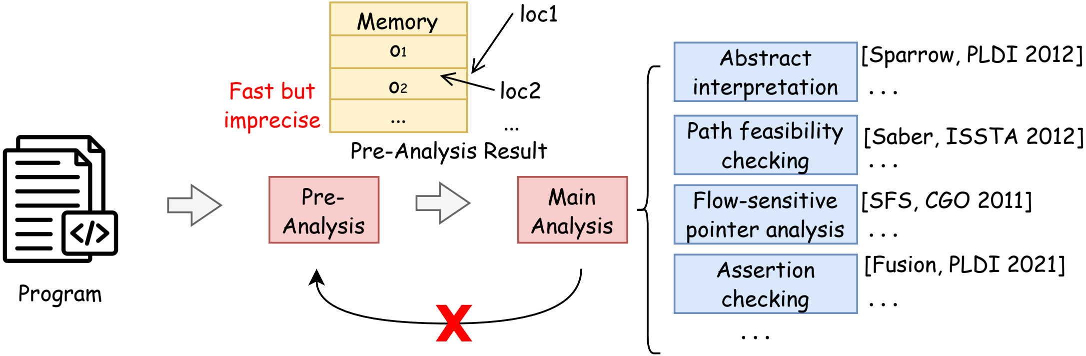
Staged Analysis



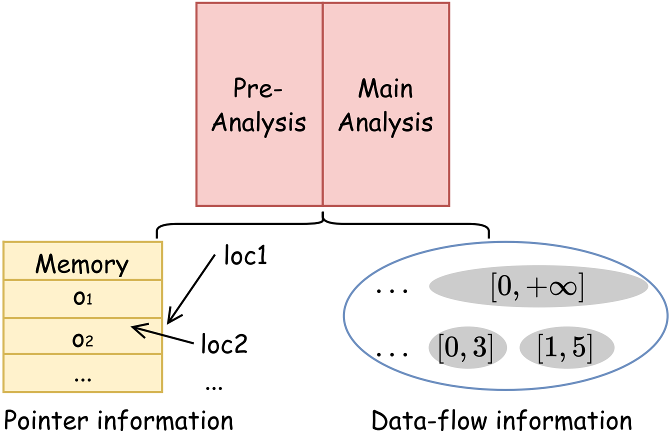
Staged Analysis



Staged Analysis



Combined Analysis



Pre-Analysis

Main Analysis

Memory

o_1

o_2

...

loc1

loc2

...

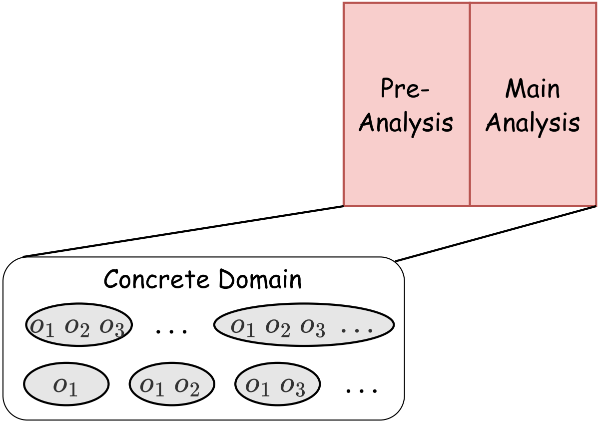
$\ldots$ $[0, +\infty]$

$\ldots$ $[0, 3]$ $[1, 5]$

Pointer information

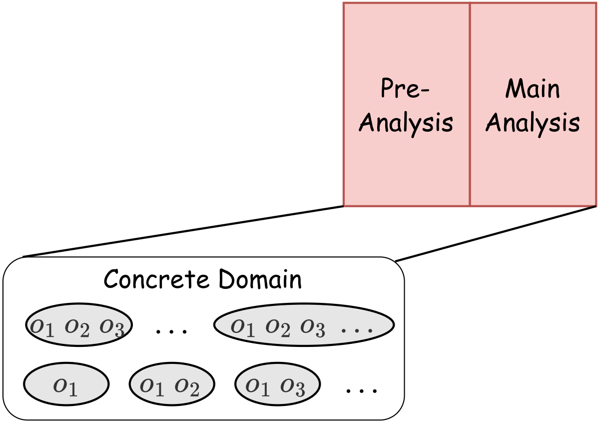
Data-flow information

Combined Analysis with Concrete Domain



One concrete domain for both analyses?

Combined Analysis with Concrete Domain



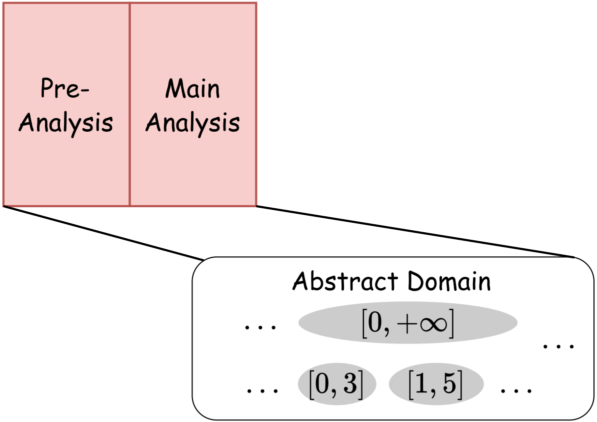
One concrete domain for both analyses?

{1, 2, 3, ..., infinit numbers}

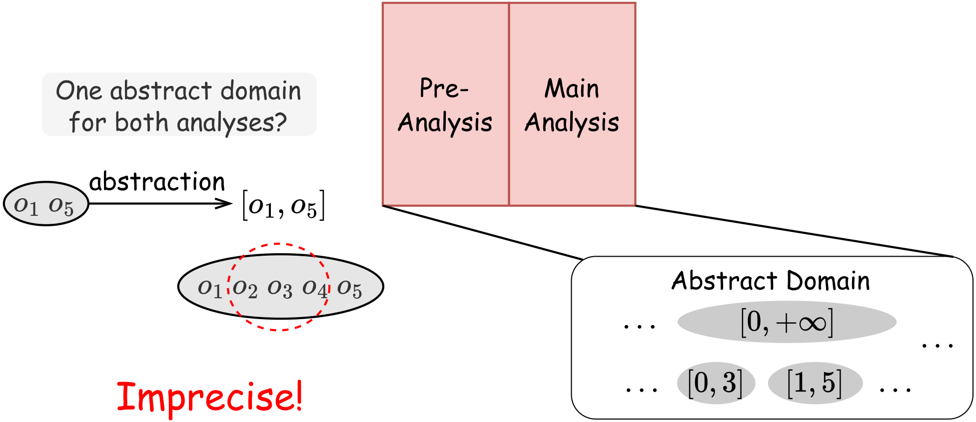
Unscalable!

Combined Analysis with Abstract Domain

One abstract domain for both analyses?



Combined Analysis with Abstract Domain



One abstract domain for both analyses?

Pre-Analysis

Main Analysis

$o_1 \ o_5 \xrightarrow{\text{abstraction}} [o_1, o_5]$

$o_1 \ o_2 \ o_3 \ o_4 \ o_5$

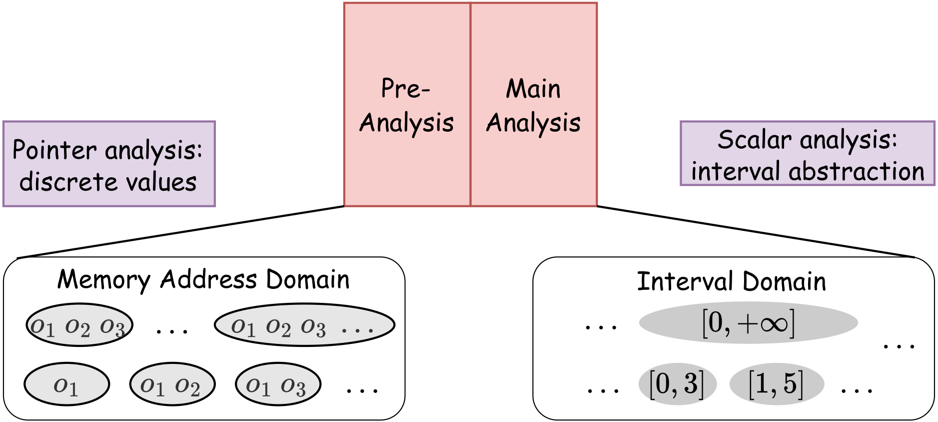
Imprecise!

Abstract Domain

$\ldots \quad [0, +\infty] \quad \ldots$

$\ldots \quad [0, 3] \quad [1, 5] \quad \ldots$

Combined Analysis with Combined Domain



Pre-Analysis

Main Analysis

Pointer analysis: discrete values

Scalar analysis: interval abstraction

Memory Address Domain

$o_1\ o_2\ o_3$ $\ldots$ $o_1\ o_2\ o_3\ \ldots$

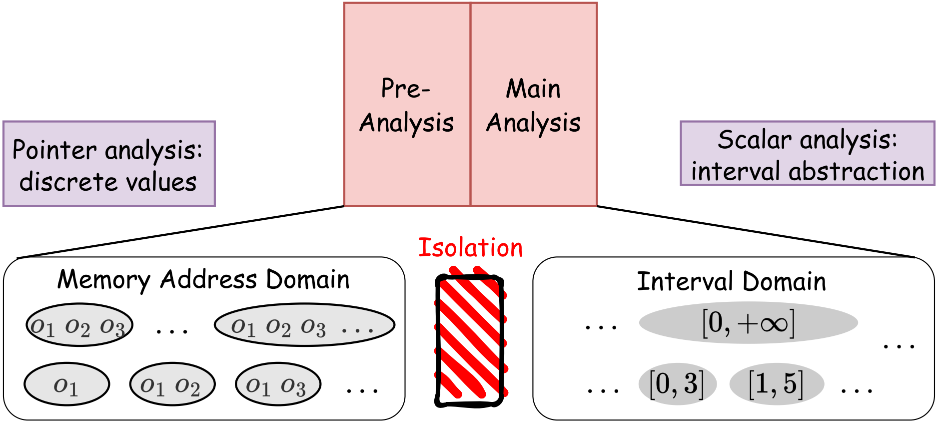
o_1 $o_1\ o_2$ $o_1\ o_3$ $\ldots$

Interval Domain

$\ldots$ $[0, +\infty]$ $\ldots$

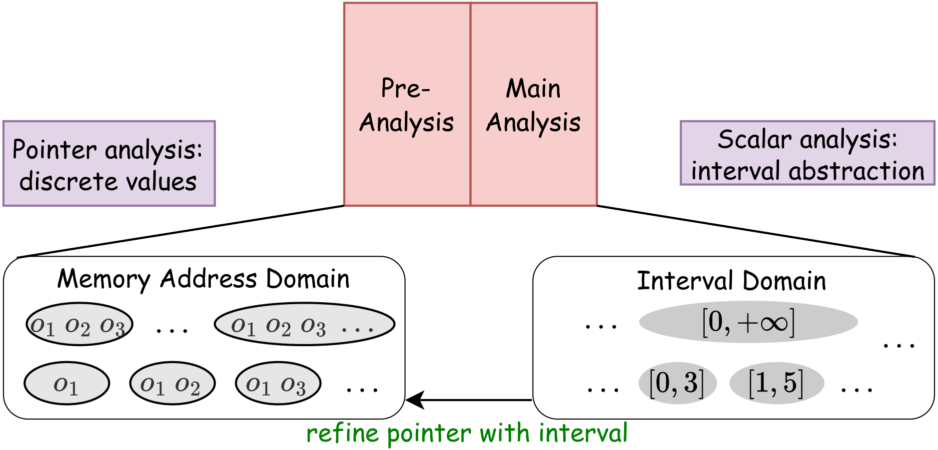
$\ldots$ $[0, 3]$ $[1, 5]$ $\ldots$

Combined Analysis with Combined Domain



Precision loss without cross-domain interaction

Cross-Domain Online Refinement



Pre-Analysis

Main Analysis

Pointer analysis: discrete values

Scalar analysis: interval abstraction

Memory Address Domain

$o_1\ o_2\ o_3$ $\dots$ $o_1\ o_2\ o_3\ \dots$

o_1 $o_1\ o_2$ $o_1\ o_3$ $\dots$

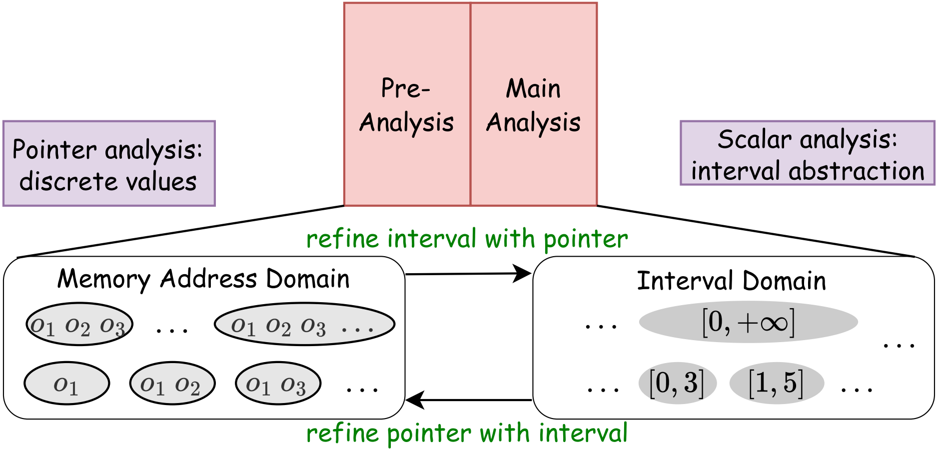
Interval Domain

$\dots$ $[0, +\infty]$ $\dots$

$\dots$ $[0, 3]$ $[1, 5]$ $\dots$

refine pointer with interval

Cross-Domain Online Refinement



Pre-Analysis

Main Analysis

Pointer analysis: discrete values

Scalar analysis: interval abstraction

refine interval with pointer

Memory Address Domain

$o_1\ o_2\ o_3$ $\dots$ $o_1\ o_2\ o_3\ \dots$

o_1 $o_1\ o_2$ $o_1\ o_3$ $\dots$

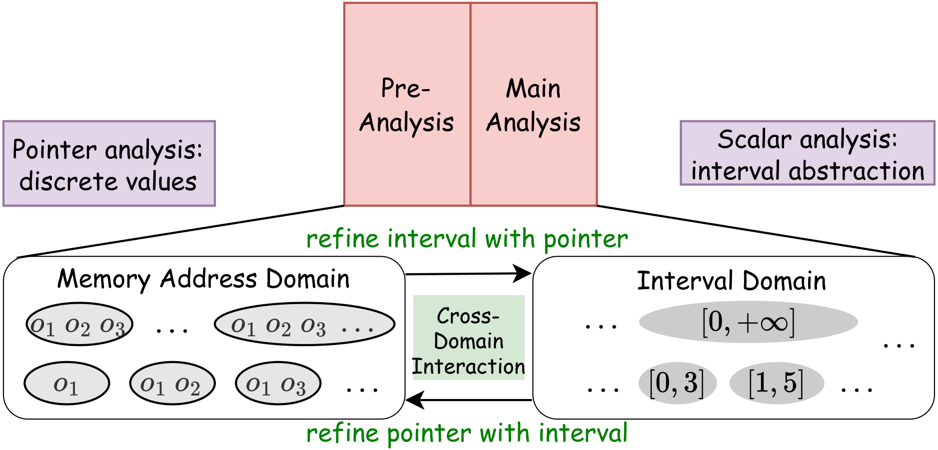
Interval Domain

$\dots$ $[0, +\infty]$ $\dots$

$\dots$ $[0, 3]$ $[1, 5]$ $\dots$

refine pointer with interval

Cross-Domain Online Refinement



Pre-Analysis

Main Analysis

Pointer analysis: discrete values

Scalar analysis: interval abstraction

refine interval with pointer

Memory Address Domain

$o_1\ o_2\ o_3$ . . . $o_1\ o_2\ o_3$ . . .

o_1 $o_1\ o_2$ $o_1\ o_3$ . . .

Cross-Domain Interaction

Interval Domain

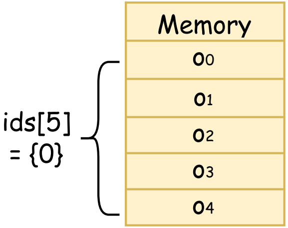
. . . $[0, +\infty]$ . . .

. . . $[0, 3]$ $[1, 5]$ . . .

refine pointer with interval

An Example

Analysis without Cross-Domain Refinement



ids[5] = {0}

Memory

o_0

o_1

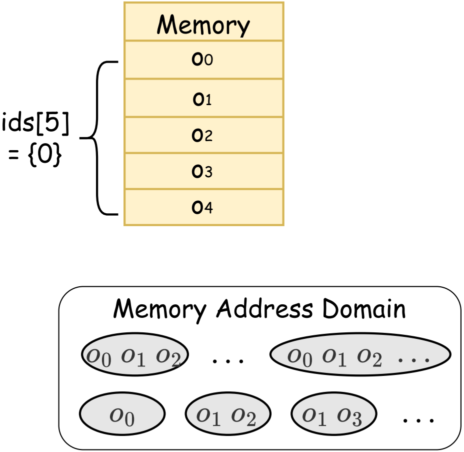
o_2

o_3

o_4

An Example

Analysis without Cross-Domain Refinement



Memory

o_0

o_1

o_2

o_3

o_4

ids[5] = {0}

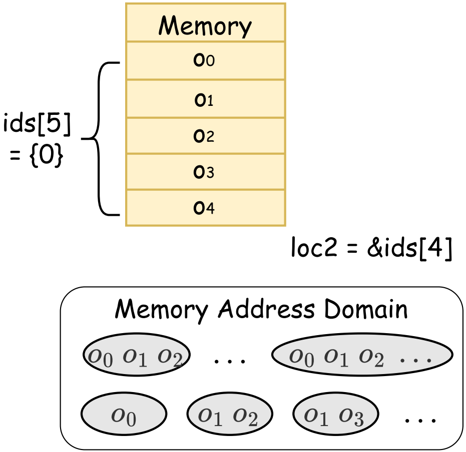
Memory Address Domain

$o_0\ o_1\ o_2$ … $o_0\ o_1\ o_2\ \ldots$

o_0 $o_1\ o_2$ $o_1\ o_3$ …

An Example

Analysis without Cross-Domain Refinement



Memory
o_0
o_1
o_2
o_3
o_4

ids[5] = {0}

loc2 = &ids[4]

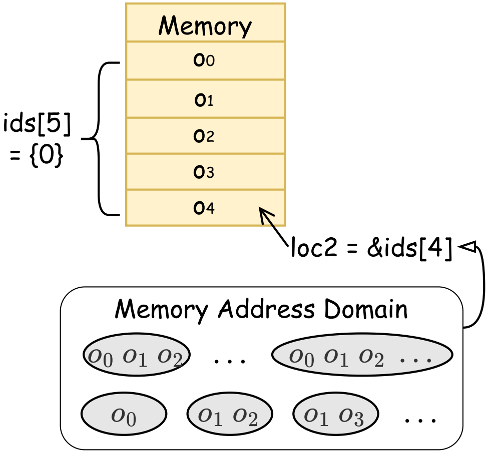
Memory Address Domain

o_0 o_1 o_2 ... o_0 o_1 o_2 ...

o_0 o_1 o_2 o_1 o_3 ...

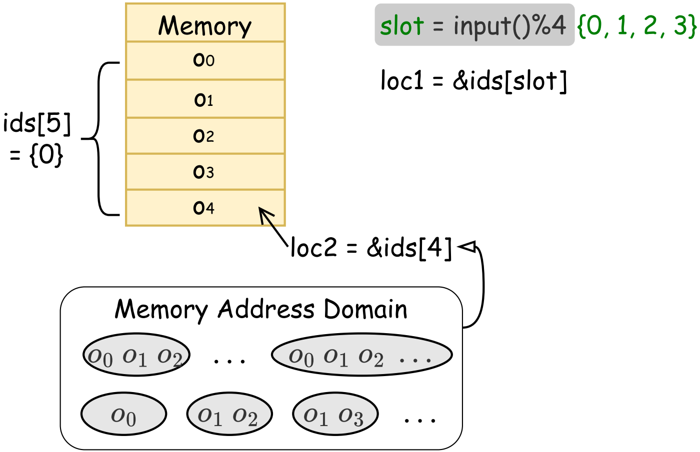
An Example

Analysis without Cross-Domain Refinement



An Example

Analysis without Cross-Domain Refinement



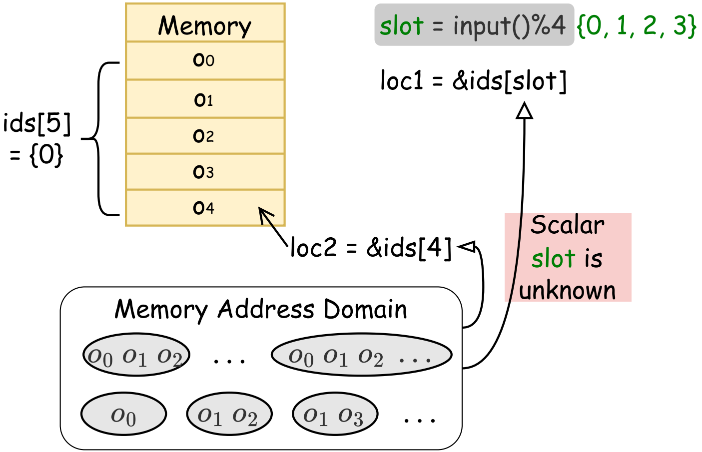
slot = input()%4 {0, 1, 2, 3}

loc1 = &ids[slot]

loc2 = &ids[4]

An Example

Analysis without Cross-Domain Refinement



slot = input()%4 {0, 1, 2, 3}

loc1 = &ids[slot]

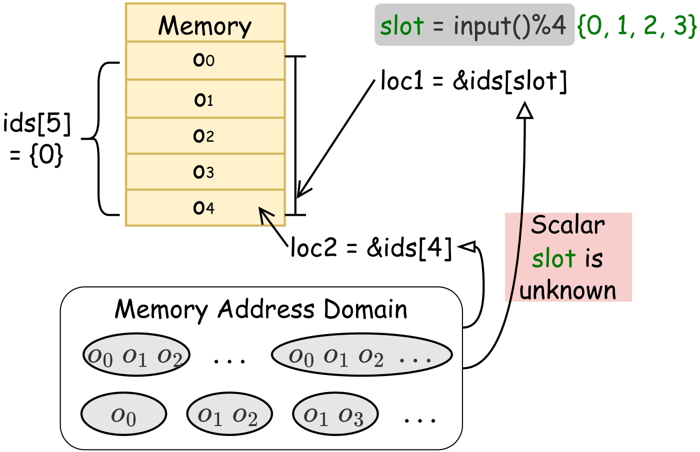
loc2 = &ids[4]

ids[5] = {0}

Scalar slot is unknown

An Example

Analysis without Cross-Domain Refinement



Memory

o_0

o_1

o_2

o_3

o_4

ids[5] = {0}

slot = input()%4 {0, 1, 2, 3}

loc1 = &ids[slot]

loc2 = &ids[4]

Scalar slot is unknown

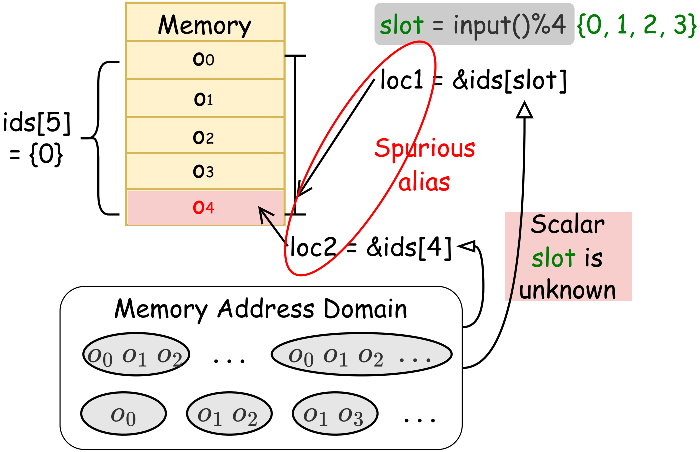
Memory Address Domain

$o_0\ o_1\ o_2$ . . . $o_0\ o_1\ o_2\ \ldots$

o_0 $o_1\ o_2$ $o_1\ o_3$ . . .

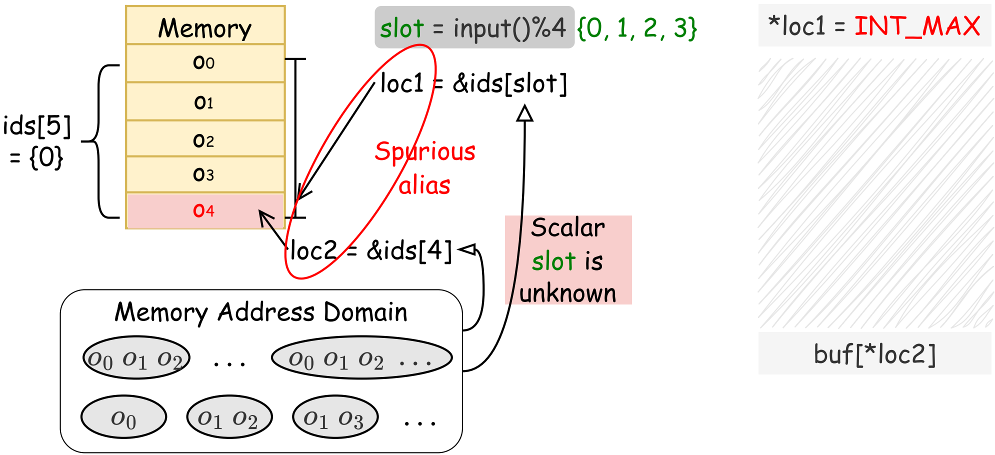
An Example

Analysis without Cross-Domain Refinement



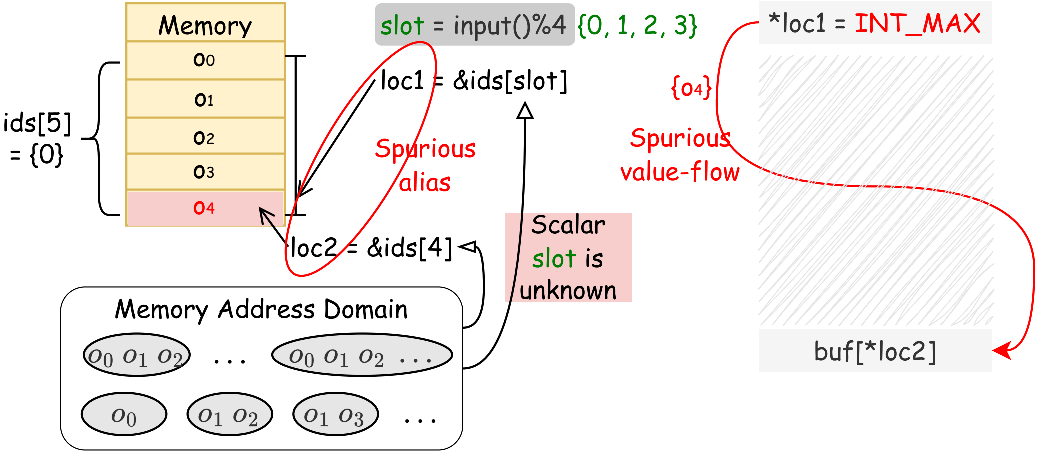
An Example

Analysis without Cross-Domain Refinement



An Example

Analysis without Cross-Domain Refinement



Memory

o_0

o_1

o_2

o_3

o_4

ids[5] = {0}

slot = input()%4 {0, 1, 2, 3}

loc1 = &ids[slot]

loc2 = &ids[4]

Spurious alias

Scalar slot is unknown

Memory Address Domain

$o_0\ o_1\ o_2$ . . . $o_0\ o_1\ o_2$ . . .

o_0 $o_1\ o_2$ $o_1\ o_3$ . . .

*loc1 = INT_MAX

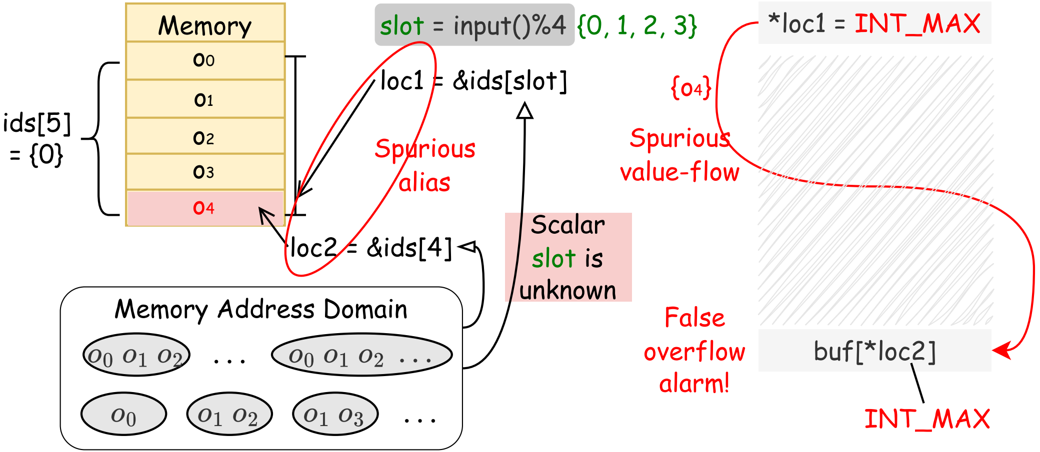
{o_4}

Spurious value-flow

buf[*loc2]

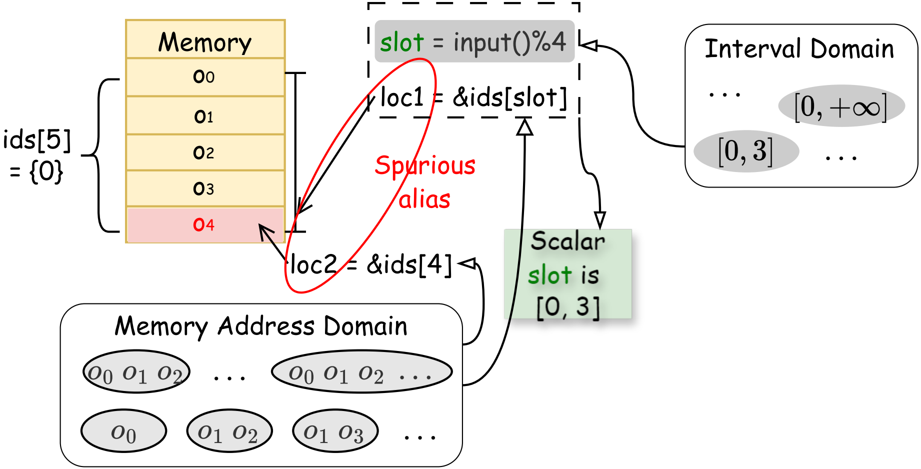
An Example

Analysis without Cross-Domain Refinement



An Example

Analysis with Cross-Domain Refinement



Memory

o_0

o_1

o_2

o_3

o_4

ids[5] = {0}

slot = input()%4

loc1 = &ids[slot]

Spurious alias

loc2 = &ids[4]

Interval Domain

$\dots$ $[0, +\infty]$

$[0, 3]$ $\dots$

Scalar slot is [0, 3]

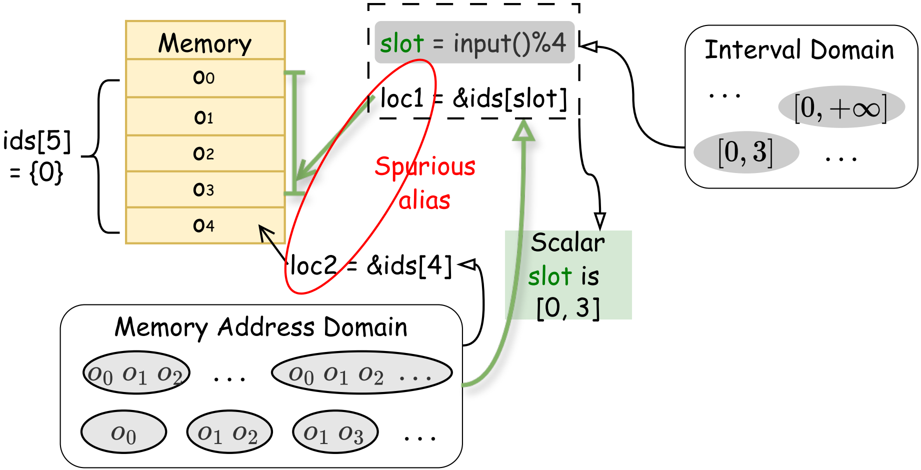
Memory Address Domain

$o_0\ o_1\ o_2$ $\dots$ $o_0\ o_1\ o_2\ \dots$

o_0 $o_1\ o_2$ $o_1\ o_3$ $\dots$

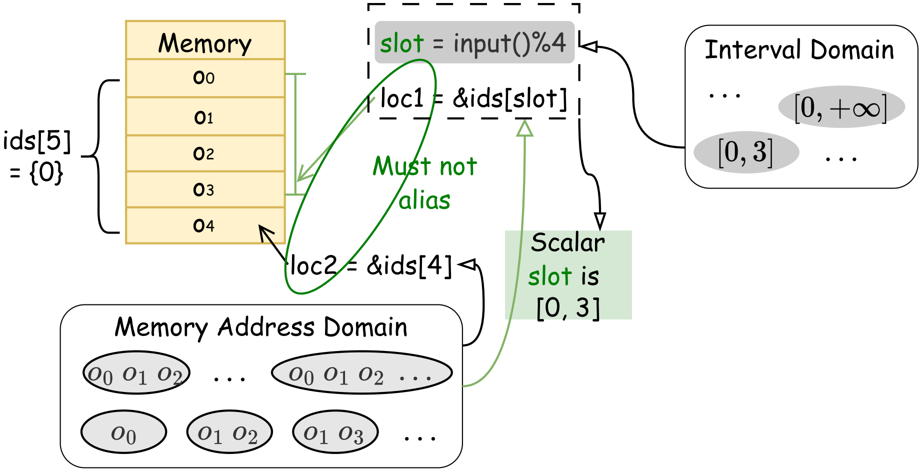
An Example

Analysis with Cross-Domain Refinement



An Example

Analysis with Cross-Domain Refinement



Memory

o0

o1

o2

o3

o4

ids[5] = {0}

slot = input()%4

loc1 = &ids[slot]

Must not alias

loc2 = &ids[4]

Interval Domain

$\dots$ $[0, +\infty]$

$[0, 3]$ $\dots$

Scalar slot is [0, 3]

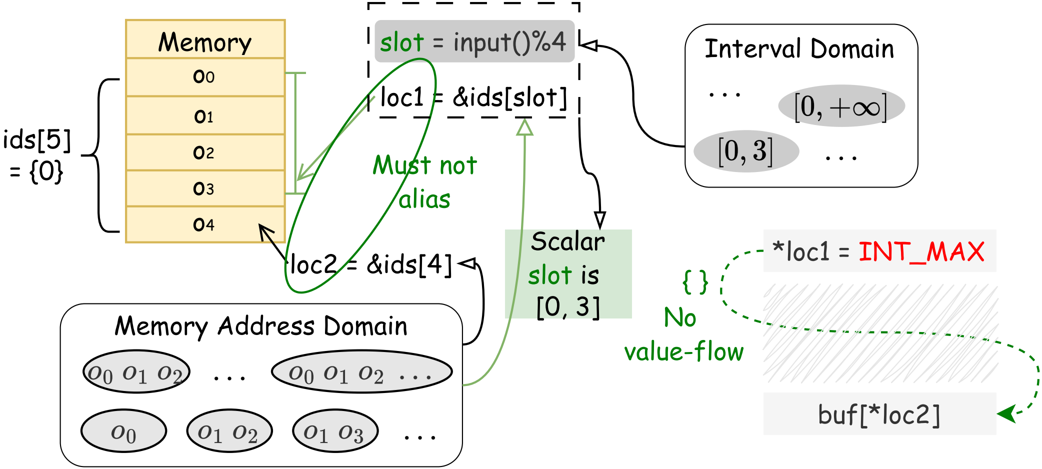
Memory Address Domain

$o_0\ o_1\ o_2$ $\dots$ $o_0\ o_1\ o_2\ \dots$

o_0 $o_1\ o_2$ $o_1\ o_3$ $\dots$

An Example

Analysis with Cross-Domain Refinement



Memory

o0

o1

o2

o3

o4

ids[5] = {0}

slot = input()%4

loc1 = &ids[slot]

loc2 = &ids[4]

Must not alias

Interval Domain

…

$[0, +\infty]$

$[0, 3]$

…

Scalar slot is [0, 3]

Memory Address Domain

$o_0\ o_1\ o_2$ … $o_0\ o_1\ o_2\ \ldots$

o_0 $o_1\ o_2$ $o_1\ o_3$ …

*loc1 = INT_MAX

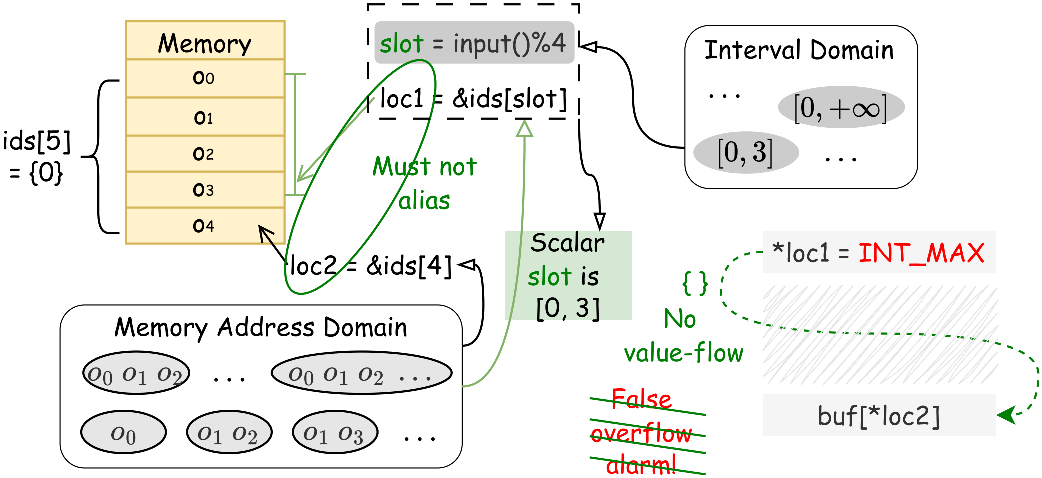
{ }

No value-flow

buf[*loc2]

An Example

Analysis with Cross-Domain Refinement



Memory

o_0

o_1

o_2

o_3

o_4

ids[5] = {0}

slot = input()%4

loc1 = &ids[slot]

Must not alias

loc2 = &ids[4]

Interval Domain

$\ldots$ $[0, +\infty]$

$[0, 3]$ $\ldots$

Scalar slot is [0, 3]

Memory Address Domain

$o_0\ o_1\ o_2$ $\ldots$ $o_0\ o_1\ o_2\ \ldots$

o_0 $o_1\ o_2$ $o_1\ o_3$ $\ldots$

*loc1 = INT_MAX

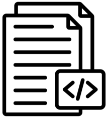
{ }

No value-flow

buf[*loc2]

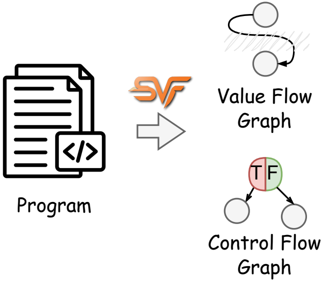
~~False overflow alarm!~~

Framework Overview

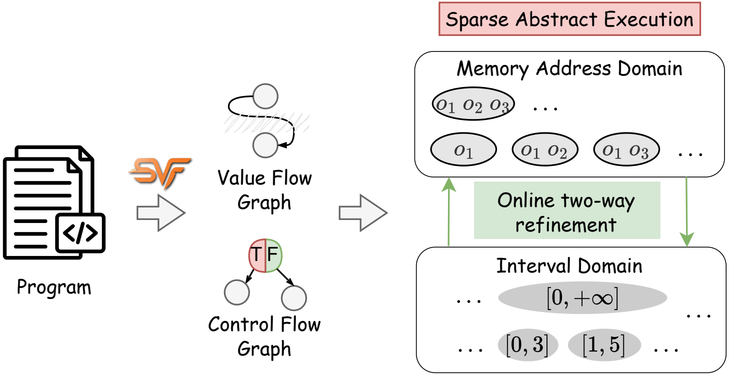


Program

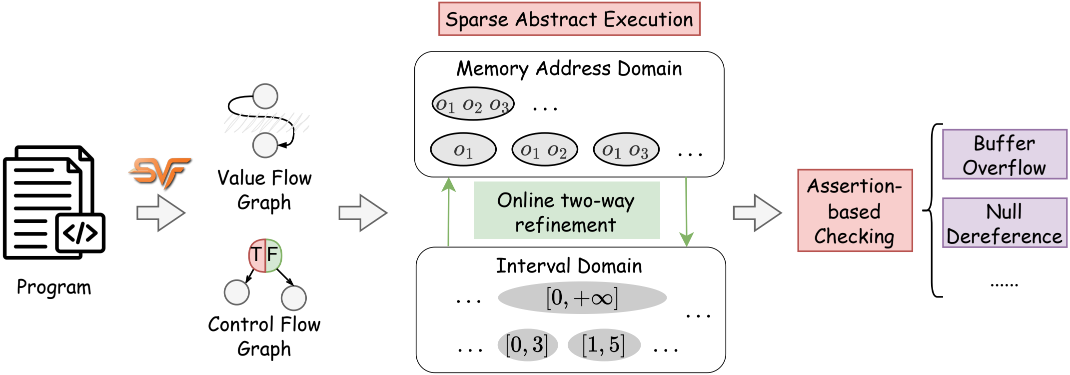
Framework Overview



Framework Overview



Framework Overview



Program

Value Flow Graph

T F

Control Flow Graph

Sparse Abstract Execution

Memory Address Domain

$o_1\ o_2\ o_3$ ...

o_1 $o_1\ o_2$ $o_1\ o_3$ ...

Online two-way refinement

Interval Domain

... $[0, +\infty]$ ...

... $[0, 3]$ $[1, 5]$ ...

Assertion-based Checking

Buffer Overflow

Null Dereference

......

LLVM-like Language

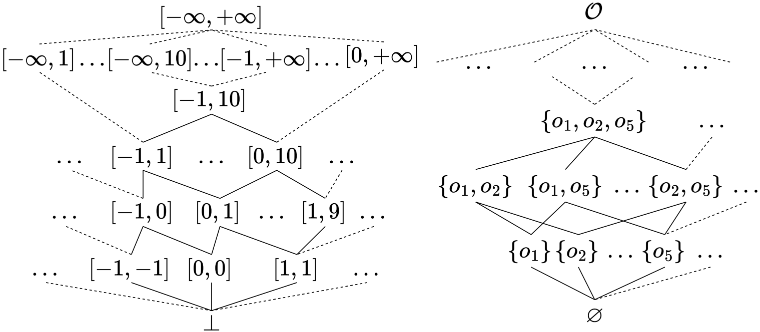
`c,fld`	$\in \mathcal{C}$	Constants
`p,q,r`	$\in \mathcal{S}$	Stack virtual registers
`g`	$\in \mathcal{G}$	Global pointer variables
`p,q,r,g`	$\in \mathcal{P} = \mathcal{S} \cup \mathcal{G}$	Top-level variables
$\mathtt{o}, \mathtt{a}, \mathtt{a}_{\mathtt{f}}, \mathtt{a.fld}, \mathtt{a[c]}$	$\in \mathcal{O}$	Abstract objects
`v`	$\in \mathcal{V} = \mathcal{P} \cup \mathcal{O}$	Program variables

ℓ ::=	Stmt
$\mathtt{p} = \mathtt{c}$	ConsStmt
$\mathtt{p} = \mathtt{alloc}_{\mathtt{o}}$	AddrStmt
$\mathtt{p} = \&(\mathtt{q} \rightarrow \mathtt{fld})$	GepStmt (Field)
$\mathtt{p} = \&\mathtt{q}[\mathtt{c}]$ (constant)	GepStmt (Array-C)
$\mathtt{p} = \&\mathtt{q}[\mathtt{v}]$ (variable)	GepStmt (Array-V)
$\mathtt{p} = *\mathtt{q}$	LoadStmt
$*\mathtt{p} = \mathtt{q}$	StoreStmt
$\mathtt{p} = \mathtt{q}$	CopyStmt
$\mathtt{p} = \mathtt{phi}(\mathtt{p}_1, \mathtt{p}_2, \ldots \mathtt{p}_n)$	PhiStmt
$\mathtt{p} = \neg \mathtt{q}$	UnaryStmt
$\mathtt{r} = \mathtt{p} \odot \mathtt{q}$	BinaryStmt

$\odot \in \{+, -, *, /, \%, <<, >>, <, >, \&, \&\&, <=, >=, \equiv, \sim, |, \wedge\}$

Interval and Memory Address Domain

- Interval abstraction (*Interval* domain) for scalar variables.
- Discrete values (*MemAddress* domain) for memory addresses.



(a) Interval domain (b) Memory address domain

Abstract Trace for Interval and Memory Address Domain

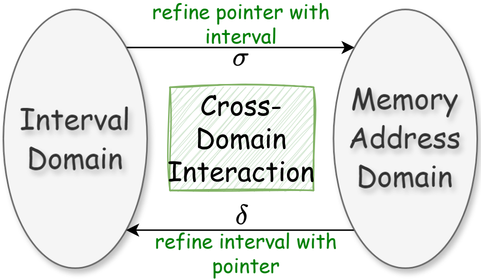
- **Symbol to value mapping**: $\sigma \in \mathcal{P} \rightarrow \mathit{Interval} \times \mathit{MemAddress}$ captures the memory addresses/interval value of top-level pointers/scalar variables.

Abstract Trace for Interval and Memory Address Domain

- **Symbol to value mapping**: $\sigma \in \mathcal{P} \rightarrow \mathit{Interval} \times \mathit{MemAddress}$ captures the memory addresses/interval value of top-level pointers/scalar variables.
- **Value to value mapping**: $\delta \in \mathbb{L} \times \mathit{MemAddress} \rightarrow \mathit{Interval} \times \mathit{MemAddress}$ captures the correlation between memory objects and memory addresses/interval values at different program locations.

Abstract Trace for Interval and Memory Address Domain

- **Symbol to value mapping**: $\sigma \in \mathcal{P} \rightarrow \mathit{Interval} \times \mathit{MemAddress}$ captures the memory addresses/interval value of top-level pointers/scalar variables.
- **Value to value mapping**: $\delta \in \mathbb{L} \times \mathit{MemAddress} \rightarrow \mathit{Interval} \times \mathit{MemAddress}$ captures the correlation between memory objects and memory addresses/interval values at different program locations.



Analysis Rules

SVFStmt	C-Like form	Abstract Execution Rule
ConsStmt	$\ell : \mathtt{p} = \mathtt{c}$	$\sigma(\mathtt{p}) := \langle [\mathtt{c}, \mathtt{c}], \top \rangle$
CopyStmt	$\ell : \mathtt{p} = \mathtt{q}$	$\sigma(\mathtt{p}) := \sigma(\mathtt{q})$
BinaryStmt	$\ell : \mathtt{r} = \mathtt{p} \otimes \mathtt{q}$	$\sigma(r) := \sigma(p) \hat{\otimes} \sigma(q)$
PhiStmt	$\ell : \mathtt{r} = \mathtt{phi}(\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_n)$	$\sigma(r) := \bigsqcup_{i=1}^{n} \sigma(p_i)$
ValueFlow	$\ell' \overset{o}{\hookrightarrow} \ell$	$\delta_{\overline{\ell}}(o) \sqsupseteq \delta_{\underline{\ell'}}(o)$
AddrStmt	$\ell : \mathtt{p} = \mathtt{alloc}_{\mathtt{o_i}}$	$\sigma(\mathtt{p}) := \langle \top, \{o_i\} \rangle$
GepStmt	$\ell : \mathtt{p} = \&(\mathtt{q} \rightarrow \mathtt{i})$ or $\mathtt{p} = \&\mathtt{q}[\mathtt{i}]$	$\sigma(\mathtt{p}) := \bigsqcup_{\mathrm{o} \in \gamma(\sigma(\mathtt{q}))} \bigsqcup_{j \in \gamma(\sigma(\mathtt{i}))} \langle \top, \{\mathtt{o.fld}_j\} \rangle$
LoadStmt	$\ell : \mathtt{p} = *\mathtt{q}$	$\sigma(\mathtt{p}) := \bigsqcup_{o \in \{o \mid (o \mapsto \_) \in \delta_{\overline{\ell}}\}} \delta_{\overline{\ell}}(o)$
StoreStmt	$\ell : *\mathtt{p} = \mathtt{q}$	$\delta_{\underline{\ell}} \sqsupseteq (\{o \mapsto \sigma(\mathtt{q}) \vert o \in \gamma(\sigma(\mathtt{p}))\} \sqcup \delta_{\overline{\ell}} \setminus \mathsf{kill}(\ell))$

$$\mathsf{kill}(\ell : *\mathrm{p} = \mathrm{q}) := \begin{cases} \{\mathrm{o} \mapsto \_ \mid \mathrm{o} \in \gamma(\sigma(\mathrm{p}))\} & \text{if } \sigma(\mathrm{p}) \equiv \langle \top, \{\mathrm{o}\} \rangle \wedge \mathrm{o} \text{ is singleton} \\ \{\mathrm{o} \mapsto \_ \mid \mathrm{o} \in \mathcal{O}\} & \text{if } \sigma(\mathrm{p}) \equiv \langle \top, \varnothing \rangle \\ \varnothing & \text{otherwise} \end{cases}$$

Evaluation

Datasets

1. A benchmark comprising 7774 programs from NIST Juliet test cases [1], which includes its null dereferences and buffer overflow vulnerabilities.
2. 10 popular open-source C/C++ projects across various application domains: `paste` (file merger), `md5sum` (file verifier), `YAJL` (JSON parsing library), `MP4v2` (MP4 file library), `RIOT` (IoT operating system), `darknet` (neural network framework), `tmux` (terminal multiplexer), `Teeworlds` (online multiplayer game), `NanoMQ` (MQTT broker for IoT edge platform) and `redis` (in-memory database).

Evaluation

Datasets

Table 1: The statistics of the open-source projects. #LOI denotes the number of lines of LLVM instructions. #Method, #Call and #Obj are the numbers of functions, method calls and memory objects, respectively. $|V|$ and $|E|$ are the numbers of ICFG nodes and ICFG edges.

Project	#LOI	#Method	#Call	#Obj	$\|V\|$	$\|E\|$
`paste`	8,416	53	758	510	9,395	9,922
`md5sum`	11,483	63	881	606	12,494	13,064
`YAJL`	20,592	151	561	208	9,253	9,922
`MP4v2`	39,178	601	610	1,991	15,595	16,733
`RIOT`	54,597	579	1,614	951	20,176	20,843
`darknet`	159,205	985	9,776	2,550	136,094	147,852
`tmux`	446,626	1,967	22,369	3,879	162,879	178,924
`Teeworlds`	529,737	2,306	28,267	5,754	251,356	246,029
`NanoMQ`	788,967	3,235	47,646	30,838	358,312	443,670
`redis`	1,363,507	6,314	68,664	13,958	589,019	704,356
Total	3,422,308	16,254	181,146	61,245	1,564,573	1,791,315

Evaluation

Research Questions

RQ1 **Is CSA effective in detecting existing bugs?** We aim to investigate whether CSA can achieve a better performance than the state-of-the-art on detecting existing bugs.

RQ2 **Can CSA find bugs with a low false positive rate in real-world projects?** We would like to examine the effectiveness and efficiency of CSA using real-world popular applications.

RQ3 **What is the influence of different components in our framework?** We aim to understand RQ3.1: the precision improvement of cross-domain refinement; and RQ3.2: efficiency improvement in terms of time and memory using equivalent correlation tracking.

Evaluation

NIST Benchmark (RQ1)

Table 2: Comparing with five tools using the NIST benchmark, with true positive rate (#TPR) and precision rate (#PCR) in percentage (%).

Tool	Buffer overflow		Null dereference		Total	
	#TPR (%)	#PCR (%)	#TPR (%)	#PCR (%)	#TPR (%)	#PCR (%)
Infer	19.23	70.57	53.17	50.19	20.20	68.48
Cppcheck	2.72	100.00	42.86	85.71	3.87	95.00
KLEE	67.78	98.81	91.27	93.12	68.45	98.58
IKOS	49.76	45.83	92.86	92.86	50.99	47.07
Sparrow	44.64	32.49	90.48	52.78	45.95	33.21
CSA	**73.84**	**84.11**	**100.00**	**100.00**	**74.58**	**84.63**
BugNum	8589		252		8841	

Evaluation

Bugs in Real-World Projects (RQ2)

Table 3: Comparing CSA with five open-source tools using ten popular applications. #TP and #FP are true positive and false positive, respectively. Time (secs), Mem (MB) are running time and memory costs. The – in the Time columns indicates a running time of more than 4h. The – in the Mem columns indicates a cost of more than 100 Gigabytes.

Project	INFER				CPPCHECK				IKOS				KLEE				SPARROW				CSA			
	Report		Time	Mem	Report		Time	Mem	Report		Time	Mem	Report		Time	Mem	Report		Time	Mem	Report		Time	Mem
	#TP	#FP	(secs)	(MB)	#TP	#FP	(secs)	(MB)	#TP	#FP	(secs)	(MB)	#TP	#FP	(secs)	(MB)	#TP	#FP	(secs)	(MB)	#TP	#FP	(secs)	(MB)
`paste`	1	15	7	61	0	17	1	9	3	21	512	1126	4	0	2911	1711	4	35	3	51	3	0	9	106
`md5sum`	2	21	8	80	0	18	1	11	2	35	986	1684	3	0	2824	1642	2	22	2	48	4	1	8	110
`YAJL`	0	17	9	110	0	14	1	12	1	1625	2895	4822	4	16	14400	17333	3	86	6	59	3	0	5	102
`MP4v2`	1	28	313	335	1	26	38	38	1	956	3684	6215	2	3	14400	21358	1	236	214	231	1	0	13	384
`RIOT`	3	29	111	155	2	19	2	22	2	1325	5216	8622	5	2	14400	23654	2	651	315	421	8	6	27	346
`darknet`	25	134	837	282	16	214	10	55	14	1265	9531	23954	25	8	14400	40015	10	842	826	984	21	10	3507	1875
`tmux`	5	142	522	909	3	156	30	39	4	1632	11325	38366	2	1	14400	70826	3	1256	1036	1894	12	10	824	5052
`Teeworlds`	10	169	684	934	4	187	2	54	2	529	13569	40368	2	1	14400	71865	10	1512	1593	2984	15	8	2886	2598
`NanoMQ`	23	154	654	305	10	147	94	38	–	–	–	–	5	2	14400	91465	6	1241	1642	3125	30	8	1143	6551
`redis`	6	137	1292	10484	8	136	516	123	–	–	–	–	3	2	14400	101475	5	1152	2654	9211	14	8	6553	3870
`Total`	76	846	4437	13655	44	934	695	401	29	7388	47718	125157	55	35	120935	441344	46	7033	8291	19008	111	51	14975	20994

Evaluation

Ablation Analysis for Cross-Domain Interaction (RQ3.1)

Table 4: Comparing with CSA-CP (a variant of CSA without cross-domain interaction) using the NIST benchmark, with true positive rate (#TPR) and precision rate (#PCR) in percentage (%).

Tool	Buffer overflow		Null dereference		Total	
	#TPR (%)	#PCR (%)	#TPR (%)	#PCR (%)	#TPR (%)	#PCR (%)
CSA-CP	73.84	42.62	100.00	42.64	74.58	42.65
CSA	**73.84**	**84.11**	**100.00**	**100.00**	**74.58**	**84.63**
BugNum	8589		252		8841	

Evaluation

Ablation Analysis for Cross-Domain Interaction (RQ3.1)

Table 5: Comparing CSA with CSA-CP using ten popular applications. #TP and #FP are true positive and false positive, respectively. Time (secs), Mem (MB) are running time and memory costs.

Project	CSA-CP				CSA			
	Report		Time	Mem	Report		Time	Mem
	#TP	#FP	(secs)	(MB)	#TP	#FP	(secs)	(MB)
`paste`	3	19	5	92	3	0	9	106
`md5sum`	4	26	15	121	4	1	8	110
`YAJL`	3	35	7	172	3	0	5	102
`MP4v2`	1	25	58	269	1	0	13	384
`RIOT`	8	38	102	366	8	6	27	346
`darknet`	21	199	3483	1982	21	10	3507	1875
`tmux`	12	360	1182	6343	12	10	824	5052
`Teeworlds`	15	244	2754	3485	15	8	2886	2598
`NanoMQ`	30	292	1801	7063	30	8	1143	6551
`redis`	14	275	8629	4421	14	8	6553	3870
`Total`	111	1513	18036	24314	111	51	14975	20994

Evaluation

Ablation Analysis for Equivalent Correlation Tracking (RQ3.2)

Table 6: Comparison between CSA and CSA-NI (a version of CSA without implication-equivalent memory addresses).

Project	CSA-NI		CSA	
	Time (secs)	Mem (MB)	Time (secs)	Mem (MB)
`tmux`	1540 (1.87×)	21016 (4.16×)	824	5052
`Teeworlds`	6176 (2.14×)	14237 (5.48×)	2886	2598
`NanoMQ`	3292 (2.88×)	48805 (7.45×)	1143	6551
`redis`	21232 (3.24×)	32314 (8.35×)	6553	3870
Geo. Mean	(2.47×)	(6.14×)		

Thank You!