A Container-Usage-Pattern-based Context Debloating Approach for Object-Sensitive Pointer Analysis (Supplementary Materials)

DONGJIE HE, YUJIANG GUI, WEI LI, YONGGANG TAO, CHANGWEI ZOU, YULEI SUI, and JINGLING XUE, University of New South Wales, Australia

1 Regularization of L_{FT} into $\mathcal{A}_{comm}$

The purpose of $\mathcal{A}_{comm}$ is to over-approximate both the incoming and outgoing value-flows of object fields, including the related aliases, in XPAG. This is done intentionally to determine whether an object is a container or not according to Definition 1. It is important to note that $\mathcal{A}_{comm}$ is not intended for computing the points-to information in a program.

To derive $\mathcal{A}_{comm}$, depicted in Figure 7a, we apply regularization to L_{FT}, a CFL-reachability formulation for context-insensitive Java pointer analysis first presented in [Sridharan et al. 2005], utilizing the same fundamental principle as described in [He et al. 2021b].

We use the assign edges to represent parameter passing for calls to static methods, constructors, and private methods, following [X-Special] in Figure 5 of the main paper. At this point, we assume no virtual calls in the program, and therefore, [X-Virtual] in Figure 5 is not applied. We utilize L_{FT} to model how objects flow on XPAG in a context-insensitive but field-sensitive manner. To obtain $\mathcal{A}_{comm}$, we start with $L_0 = L_{FT}$, which we extract from Figure 3 of [Sridharan et al. 2005] with some notational modifications shown below:

$$\begin{array}{rcl} \mathsf{flowsto} & \longrightarrow & \mathsf{new}\ \mathsf{flows}^* \\ \mathsf{flows} & \longrightarrow & \mathsf{assign} \mid \mathsf{store}[\mathsf{f}]\ \overline{\mathsf{flowsto}}\ \mathsf{flowsto}\ \mathsf{load}[\mathsf{f}] \\ \overline{\mathsf{flowsto}} & \longrightarrow & \overline{\mathsf{flows}}^*\ \overline{\mathsf{new}} \\ \overline{\mathsf{flows}} & \longrightarrow & \overline{\mathsf{assign}} \mid \overline{\mathsf{load}[\mathsf{f}]}\ \overline{\mathsf{flowsto}}\ \mathsf{flowsto}\ \overline{\mathsf{store}[\mathsf{f}]} \end{array} \tag{1}$$

To obtain L_1, we over-approximate L_0 by disregarding its field-sensitivity requirement.

$$\begin{array}{rcl} \mathsf{flowsto} & \longrightarrow & \mathsf{new}\ \mathsf{flows}^* \\ \mathsf{flows} & \longrightarrow & \mathsf{assign} \mid \mathsf{store}\ \overline{\mathsf{flowsto}}\ \mathsf{flowsto}\ \mathsf{load} \\ \overline{\mathsf{flowsto}} & \longrightarrow & \overline{\mathsf{flows}}^*\ \overline{\mathsf{new}} \\ \overline{\mathsf{flows}} & \longrightarrow & \overline{\mathsf{assign}} \mid \overline{\mathsf{load}}\ \overline{\mathsf{flowsto}}\ \mathsf{flowsto}\ \overline{\mathsf{store}} \end{array} \tag{2}$$

We aim to approximate object value flow rather than perform pointer analysis, which led us to remove the non-terminals flowsto and $\overline{\text{flowsto}}$ from L_1. We also reformatted the productions by removing $*$ to adopt a more formal form. This transformation results in L_2, shown below:

$$\begin{array}{rcl} \mathsf{flows} & \longrightarrow & \mathsf{assign}\ \mathsf{flows} \mid \mathsf{store}\ \overline{\mathsf{flows}}\ \overline{\mathsf{new}}\ \mathsf{new}\ \mathsf{flows}\ \mathsf{load}\ \mathsf{flows} \mid \epsilon \\ \overline{\mathsf{flows}} & \longrightarrow & \overline{\mathsf{assign}}\ \overline{\mathsf{flows}} \mid \overline{\mathsf{load}}\ \overline{\mathsf{flows}}\ \overline{\mathsf{new}}\ \mathsf{new}\ \mathsf{flows}\ \ \overline{\mathsf{store}}\ \overline{\mathsf{flows}}\ \mid \epsilon \end{array} \tag{3}$$

Furthermore, we use a standard regularization approximation algorithm for context-free grammar [Mohri and Nederhof 2001] to approximate L_2. To convert the grammar into a right linear form, we introduce a new non-terminal O and obtain L_3, which is presented below:

$$\begin{array}{rcl} \mathsf{flows} & \longrightarrow & \mathsf{assign}\ \mathsf{flows} \mid \mathsf{load}\ \mathsf{flows} \mid \mathsf{store}\ \overline{\mathsf{flows}} \mid \overline{\mathsf{store}}\ \overline{\mathsf{flows}} \mid \epsilon \\ \overline{\mathsf{flows}} & \longrightarrow & \overline{\mathsf{assign}}\ \overline{\mathsf{flows}} \mid \overline{\mathsf{load}}\ \overline{\mathsf{flows}} \mid\ \overline{\mathsf{new}}\ \mathsf{O} \mid \epsilon \\ \mathsf{O} & \longrightarrow & \mathsf{new}\ \mathsf{flows} \end{array} \tag{4}$$

Lemma 1.1. $L_i \subseteq L_{i+1}$.

Proof. Follows from that each transformation over-approximates the previous language. □

We now turn our attention to virtual calls in the program. As the DFA $\mathcal{A}_{in}^{f}$ ($\mathcal{A}_{out}^{f}$) shown in Figure 7b (7c) runs, it checks whether the current value flow may originate from any parameter (be returned by the return variable or any parameter) of some method. Since a virtual call can have multiple call targets, particularly when discovered using an imprecise call graph analysis (such as CHA and Spark), we need to over-approximate interprocedural value flows across virtual calls to ensure that DebloaterX's analysis times remain within acceptable limits. Similar to [He et al. 2021b], we avoid analyzing the methods invoked at a call site while still tracking all interprocedural value flows over-approximately. According to [X-Virtual] in Figure 5 of the main paper, we assume that an argument (or a return value) at a virtual/interface call is stored into (or loaded from) any field of its receiver variable and distinguish its XPAG edge by cstore (cload). To transform L_3 into L_4, we add such cstore and cload terminals and their inverses, $\overline{\text{cstore}}$ and $\overline{\text{cload}}$, to L_3:

$$\begin{array}{rcl} \text{flows} & \longrightarrow & \text{assign flows} \mid \text{load flows} \mid \text{cload flows} \mid \epsilon \\ & \longrightarrow & \text{store } \overline{\text{flows}} \mid \text{cstore } \overline{\text{flows}} \mid \overline{\text{store}}\ \overline{\text{flows}} \mid \overline{\text{cstore}}\ \overline{\text{flows}} \\ \overline{\text{flows}} & \longrightarrow & \overline{\text{assign}}\ \overline{\text{flows}} \mid \overline{\text{load}}\ \overline{\text{flows}} \mid \overline{\text{cload}}\ \overline{\text{flows}} \mid \overline{\text{new}}\ \text{O} \mid \epsilon \\ \text{O} & \longrightarrow & \text{new flows} \end{array} \tag{5}$$

Finally, we arrive at the final DFA, $\mathcal{A}_{comm}$, which is equivalent to L_4. It is shown in Figure 1 below and can also be found in Figure 7a of the main paper.

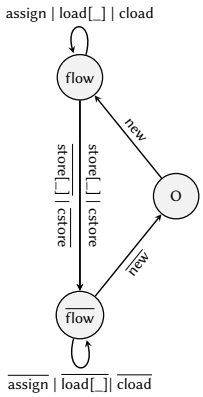

Fig. 1. $\mathcal{A}_{comm}$

$\mathcal{A}_{comm}$ over-approximates the side-effect of a target method at a virtual call regarding the reachability of a value in its arguments to another argument or its receiver variable, as shown below. Without analyzing the target, we conservatively assume that a_i, a_j, and x are aliases.

Lemma 1.2. *Given a virtual call $x = a_0.m'(a_1, ..., a_n)$, we have $\forall\ i \in [0, n]$: a_i flows x and $\forall\ i, j \in [0, n]$: a_i flows a_j.*

Proof. Due to [X-Virtual], $a_i \xrightarrow{\text{cstore}} a_0$ and $a_0 \xrightarrow{\text{cload}} x$ appear in XPAG. Thus, we have:

$$\text{flows} \Longrightarrow^{+} \ldots\ a_i \xrightarrow{\text{cstore}} a_0\ \overline{\text{flows}}\ \ \overline{\text{new}}\ \text{O new}\ \ \text{flows}\ a_0 \xrightarrow{\text{cload}} x\ \ldots \tag{6}$$

As a result, we have established $\forall\ i \in [0, n]$: a_i flows x. Similarly, we have:

$$\text{flows} \Longrightarrow^{+} \ldots\ a_i \xrightarrow{\text{cstore}} a_0\ \overline{\text{flows}}\ \ \overline{\text{new}}\ \text{O new}\ \ \text{flows}\ a_0 \xrightarrow{\overline{\text{cstore}}} a_j\ \ldots \tag{7}$$

since cstore can be matched by $\overline{\text{cstore}}$, leading us to conclude that $\forall\, i, j \in [0, n] : a_i$ flows a_j. □

2 Experimental Results on DaCapo 2006 and Third-Party Applications

Table 1. Main results for the ten DaCapo 2006 benchmarks and three third-party applications.

Program	Metrics	Spark	2obj	E-2obj	Z-2obj	C-2obj	X-2obj	3obj	E-3obj	Z-3obj	C-3obj	X-3obj
antlr	Time (s)	6.9	51.0	35.3	25.5	12.8	10.5	2201.3	1553.4	504.2	292.6	14.7
	#fail-casts	1127	511	511	529	511	511	451	451	474	451	451
	#call-edges	57472	51319	51319	51446	51319	51319	51292	51292	51419	51292	51295
	#reachables	8194	7806	7806	7836	7806	7806	7805	7805	7835	7805	7805
	#poly-calls	1987	1643	1643	1662	1643	1643	1636	1636	1655	1636	1639
bloat	Time (s)	8.3	2245.4	1277.5	1743.2	1395.4	26.0	OoT	29640.8	36961.7	14547.8	34.4
	#fail-casts	2088	1316	1316	1337	1316	1316	-	1223	1247	1223	1223
	#call-edges	67856	56837	56837	57044	56837	56839	-	56602	56809	56602	56607
	#reachables	9464	9021	9021	9063	9021	9021	-	9005	9047	9005	9005
	#poly-calls	2344	1714	1714	1745	1714	1714	-	1694	1725	1694	1697
chart	Time (s)	13.6	232.8	164.2	38.4	94.1	35.5	OoM	37176.3	1050.3	5657.2	433.4
	#fail-casts	2563	1348	1348	1388	1348	1348	-	1241	1290	1241	1241
	#call-edges	86806	72805	72805	73144	72805	72813	-	72371	72698	72371	72382
	#reachables	15933	15158	15158	15243	15158	15158	-	15128	15213	15128	15128
	#poly-calls	2732	2068	2068	2086	2068	2068	-	2044	2062	2044	2046
eclipse	Time (s)	31.7	8268.5	5573.5	2674.7	4509.2	1497.4	OoM	OoM	OoM	OoM	OoM
	#fail-casts	5114	3648	3648	3688	3648	3648	-	-	-	-	-
	#call-edges	183288	162934	162934	163089	162934	162934	-	-	-	-	-
	#reachables	23387	22628	22628	22644	22628	22628	-	-	-	-	-
	#poly-calls	10738	9773	9773	9800	9773	9773	-	-	-	-	-
fop	Time (s)	5.8	19.7	12.8	8.6	6.4	4.9	1643.0	1282.9	262.2	258.4	8.3
	#fail-casts	914	396	396	416	396	396	337	337	370	337	337
	#call-edges	40558	34424	34424	34556	34424	34424	34404	34404	34536	34404	34406
	#reachables	8001	7591	7591	7621	7591	7591	7591	7591	7621	7591	7591
	#poly-calls	1223	842	842	864	842	842	836	836	858	836	838
hsqldb	Time (s)	5.9	23.1	15.2	9.2	7.4	5.2	2834.0	2750.1	347.2	438.6	9.5
	#fail-casts	922	408	408	428	408	408	356	356	381	356	356
	#call-edges	41841	34936	34936	35075	34936	34936	34909	34909	35048	34909	34912
	#reachables	7389	6981	6981	7015	6981	6981	6980	6980	7014	6980	6980
	#poly-calls	1213	859	859	880	859	859	852	852	873	852	855
luindex	Time (s)	5.6	20.2	13.3	10.5	6.8	4.8	1287.5	1587.0	489.2	298.6	8.7
	#fail-casts	923	396	396	418	396	396	342	342	365	342	342
	#call-edges	39809	33643	33643	33774	33644	33644	33616	33616	33747	33617	33620
	#reachables	7413	7019	7019	7048	7019	7019	7018	7018	7047	7018	7018
	#poly-calls	1294	935	935	956	935	935	928	928	949	928	931
lusearch	Time (s)	6.2	32.0	23.1	18.1	7.8	5.3	3581.0	2828.2	1208.7	408.2	9.0
	#fail-casts	1035	411	411	436	411	411	359	359	385	359	359
	#call-edges	43153	36525	36525	36659	36525	36525	36498	36498	36632	36498	36501
	#reachables	8098	7671	7671	7700	7671	7671	7670	7670	7699	7670	7670
	#poly-calls	1505	1133	1133	1157	1133	1133	1126	1126	1150	1126	1129
pmd	Time (s)	9.5	61.6	44.4	30.7	35.7	12.1	3302.0	3393.8	564.2	448.5	17.6
	#fail-casts	2273	1416	1416	1445	1416	1416	1351	1351	1392	1351	1351
	#call-edges	69713	60030	60030	60165	60030	60030	59971	59971	60106	59971	59973
	#reachables	12369	11852	11852	11883	11852	11852	11852	11852	11883	11852	11852
	#poly-calls	2989	2390	2390	2411	2390	2390	2384	2384	2405	2384	2386
xalan	Time (s)	7.1	1116.5	724.7	387.9	713.3	34.0	OoM	13645.2	3543.5	1504.2	41.7
	#fail-casts	1305	601	601	618	601	601	-	547	569	547	547
	#call-edges	54147	46856	46856	46990	46856	46856	-	46824	46958	46824	46827
	#reachables	10096	9670	9670	9701	9670	9670	-	9668	9699	9668	9668
	#poly-calls	2101	1657	1657	1679	1657	1657	-	1650	1672	1650	1653
checkstyle	Time (s)	11.0	9542.1	5384.0	2759.3	6396.0	33.0	OoM	OoT	OoT	OoT	47.0
	#fail-casts	1941	1117	1117	1139	1117	1117	-	-	-	-	1005
	#call-edges	80291	67285	67285	67474	67285	67285	-	-	-	-	66543
	#reachables	12773	12315	12315	12350	12315	12315	-	-	-	-	12268
	#poly-calls	2778	2241	2241	2270	2241	2241	-	-	-	-	2197
JPC	Time (s)	13.7	137.3	112.2	42.5	68.5	61.8	4537.5	2646.1	368.3	419.3	78.8
	#fail-casts	2254	1357	1357	1379	1357	1357	1207	1207	1239	1207	1207
	#call-edges	95055	81465	81465	81653	81478	81488	79797	79797	79985	79810	79818
	#reachables	16144	15556	15556	15585	15556	15557	15209	15209	15236	15209	15210
	#poly-calls	4960	4282	4282	4318	4283	4288	4146	4146	4183	4147	4150
findbugs	Time (s)	13.6	1983.4	952.0	319.1	106.8	26.4	OoM	OoT	8377.0	1697.3	34.0
	#fail-casts	3457	2058	2058	2090	2058	2058	-	-	1899	1682	1702
	#call-edges	106065	88107	88107	88153	88107	88107	-	-	87300	87185	87192
	#reachables	16777	16265	16265	16277	16265	16265	-	-	16242	16218	16218
	#poly-calls	4534	3679	3679	3683	3679	3679	-	-	3652	3645	3646

3 Experimental Results on DaCapo-9.12

Table 2. Main results for the nine DaCapo-9.12 benchmarks.

Program	Metrics	Spark	2obj	E-2obj	Z-2obj	C-2obj	X-2obj	3obj	E-3obj	Z-3obj	C-3obj	X-3obj
avrora	Time (s)	10.3	336.3	292.3	213.2	18.5	11.5	**OoM**	**OoM**	**OoM**	2292.9	21.5
	#fail-casts	1215	659	659	712	663	663	-	-	-	584	584
	#call-edges	60549	53799	53799	53871	53799	53799	-	-	-	53677	53679
	#reachables	12181	11828	11828	11836	11828	11828	-	-	-	11817	11817
	#poly-calls	1573	1247	1247	1275	1247	1247	-	-	-	1219	1221
batik	Time (s)	30.8	3641.8	4283.9	1134.3	1003.3	993.4	**OoM**	**OoM**	**OoM**	**OoM**	**OoM**
	#fail-casts	3684	2422	2422	2488	2424	2425	-	-	-	-	-
	#call-edges	140623	123302	123302	123503	123302	123304	-	-	-	-	-
	#reachables	23445	22791	22791	22821	22791	22791	-	-	-	-	-
	#poly-calls	6276	5682	5682	5723	5682	5684	-	-	-	-	-
h2	Time (s)	15.2	5099.2	3944.2	1682.1	716.6	483.7	**OoM**	**OoM**	**OoM**	**OoM**	**OoM**
	#fail-casts	2030	1436	1436	1488	1448	1448	-	-	-	-	-
	#call-edges	108316	97404	97404	97447	97404	97404	-	-	-	-	-
	#reachables	15693	15250	15250	15258	15250	15250	-	-	-	-	-
	#poly-calls	4388	4035	4035	4053	4035	4035	-	-	-	-	-
luindex	Time (s)	9.0	326.0	288.4	206.4	16.2	10.9	**OoM**	**OoM**	**OoM**	1935.5	19.6
	#fail-casts	1095	553	553	608	557	557	-	-	-	459	459
	#call-edges	52444	45737	45737	45806	45740	45740	-	-	-	45550	45552
	#reachables	9600	9261	9261	9268	9261	9261	-	-	-	9242	9242
	#poly-calls	1613	1305	1305	1330	1305	1305	-	-	-	1268	1270
lusearch	Time (s)	8.6	334.0	301.6	202.2	15.4	10.4	**OoM**	**OoM**	**OoM**	2041.6	19.4
	#fail-casts	1092	499	499	555	505	505	-	-	-	425	425
	#call-edges	51192	44462	44462	44528	44462	44462	-	-	-	44372	44374
	#reachables	9425	9063	9063	9070	9063	9063	-	-	-	9057	9057
	#poly-calls	1729	1428	1428	1451	1428	1428	-	-	-	1416	1418
pmd	Time (s)	11.1	466.9	428.5	279.0	169.7	14.2	**OoM**	**OoM**	**OoM**	3246.6	25.0
	#fail-casts	1732	1054	1054	1099	1054	1054	-	-	-	961	963
	#call-edges	60980	53167	53167	53215	53167	53167	-	-	-	53049	53051
	#reachables	11427	11058	11058	11064	11058	11058	-	-	-	11053	11053
	#poly-calls	1976	1537	1537	1561	1537	1537	-	-	-	1519	1521
sunflow	Time (s)	16.5	922.0	842.0	574.2	28.4	21.7	**OoM**	**OoM**	**OoM**	2233.2	44.6
	#fail-casts	2269	1384	1384	1452	1388	1388	-	-	-	1291	1291
	#call-edges	80511	70122	70122	70179	70123	70127	-	-	-	69787	69793
	#reachables	15733	15289	15289	15295	15289	15290	-	-	-	15272	15273
	#poly-calls	2832	2360	2360	2390	2360	2360	-	-	-	2335	2337
tradebeans	Time (s)	10.5	1066.3	999.7	634.1	22.8	14.9	**OoM**	**OoM**	**OoM**	6105.6	32.7
	#fail-casts	1262	648	648	713	652	653	-	-	-	564	565
	#call-edges	57400	49476	49476	49645	49476	49476	-	-	-	49070	49072
	#reachables	10438	10009	10009	10016	10009	10009	-	-	-	9979	9979
	#poly-calls	1742	1416	1416	1460	1416	1416	-	-	-	1400	1402
xalan	Time (s)	14.7	1142.9	944.0	675.0	713.0	33.7	**OoM**	**OoM**	**OoM**	4579.2	50.8
	#fail-casts	2033	1142	1142	1206	1146	1146	-	-	-	1065	1065
	#call-edges	80874	71864	71864	71947	71864	71864	-	-	-	71717	71719
	#reachables	14219	13808	13808	13815	13808	13808	-	-	-	13800	13800
	#poly-calls	3863	3349	3349	3392	3349	3349	-	-	-	3330	3332