PLANKTON: Reconciling Binary Code and Debug Information (Supplement Material)

August 11, 2023

1. Formalization of Transformation Rules

We follow existing formalization methods [3, 1, 2] for LLVM intermediate representations. Figure 1 shows (a fragment of) the abstract syntax for the subset of the LLVM IR formalized in PLANKTON, which is used to define peephole transformation rules. Operations in the LLVM IR compute with values *op*, which are either identifiers id naming temporaries, or constants computed from statically-known data.

Figure 2 shows more transformation rules. `subToAdd` and `ptrToPtr` are two auxiliary transformation rules. `subToAdd` converts pointer subtraction to pointer addition, which allows PLANKTON to apply `multiAdd`. `ptrToPtr` simplifies redundant type conversion produced by other rules. `ptrIcmp` and `ptrPhi` transform values that are used in phi-nodes and comparisons, where all operands should have consistent types. These two rules use type hints of one incoming value to transform all the others. `loadBitcast`, `storeBitcast`, `loadPtr`, and `storePtr` are four rules used to find correct types for memory load/store operations. `promoteLoad` and `promotePtrToInt` aggregate type hints based on all instructions that use the value and apply those hints to the current instruction.

The above promotion rules assign values with correct types and propagate type hints through the control- and data-flows because type conversions are passed on to the following instructions and uses of the values. The demotion rules are just a set of "inverse" rules for the promotion. For example, `intToPtrGep` will lower the **getelementptr** instruction into a sequence of **add** instructions.

$$
\begin{array}{lcl}
prog & ::= & \frac{pre\ nl\ stmt}{stmt} \\
stmt & ::= & stmt\ nl\ stmt \mid reg = inst \mid inst \\
cond & ::= & \textbf{eq} \mid \textbf{ne} \mid \textbf{ugt} \mid \textbf{uge} \mid \textbf{ult} \mid \textbf{ule} \mid \textbf{sgt} \mid \textbf{sge} \mid \textbf{slt} \mid \textbf{sle} \\
typ & ::= & \textbf{i}sz \mid \textbf{void} \mid typ* \mid \{\overline{typ_j}^{\,j}\} \\
binop & ::= & \textbf{add} \mid \textbf{sub} \mid \textbf{mul} \mid \textbf{udiv} \mid \textbf{sdiv} \mid \textbf{urem} \mid \textbf{srem} \\
 & & \mid \textbf{shl} \mid \textbf{lshr} \mid \textbf{ashr} \mid \textbf{and} \mid \textbf{or} \mid \textbf{xor} \\
castop & ::= & \textbf{ptrtoint} \mid \textbf{inttoptr} \mid \textbf{bitcast} \\
\phi & ::= & \textbf{phi}\ typ\ [\![[op_j, b_j]]\!]_j \\
inst & ::= & binop\ typ\ op_1, op_2 \mid castop\ typ_1\ op_1\ to\ typ_2\ op_2 \\
 & & \mid \textbf{load}\ typ*\ op \mid \textbf{store}\ tpy\ op_1\ typ*\ op_2 \\
 & & \mid \textbf{select}\ op_1\ typ\ op_2\ op_3 \mid \textbf{icmp}\ cond\ typ\ op_1, op_2 \\
 & & \mid \textbf{getelementptr}\ typ*\ op\ \textbf{i}sz\ [\![op_j]\!]_j
\end{array}
$$

Figure 1: Partial syntax for peephole transformation rules.

References

[1] Juneyoung Lee, Chung-Kil Hur, Ralf Jung, Zhengyang Liu, John Regehr, and Nuno P Lopes. Reconciling high-level optimizations and low-level code in llvm. *Proceedings of the ACM on Programming Languages*, 2(OOPSLA):1–28, 2018.

[2] Nuno P Lopes, David Menendez, Santosh Nagarakatte, and John Regehr. Provably correct peephole optimizations with alive. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 22–32, 2015.

[3] Jianzhou Zhao, Santosh Nagarakatte, Milo MK Martin, and Steve Zdancewic. Formalizing the llvm intermediate representation for verified program transformations. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 427–440, 2012.

Table 1: Bug detection results for IRs produced by WLLVM and lifted from `GCC` compiled binaries by PLANKTON. ▼ means the static analysis runs out of memory. ▲ means lifting failure. Only the client sides of `MySQL` and `MariaDB` are invoked since both SVF and PINPOINT timeout on their server sides.

Program	KLOC	WLLVM		PLANKTON					
		PINPOINT	SVF	PINPOINT			SVF		
		# Report	# Report	#FP	#FN	#TP	#FP	#FN	#TP
CWE401s01	188	321	764	83	8	313	55	117	647
CWE401s02		387	0	77	32	355	0	0	0
CWE401s03		193	450	53	6	187	36	78	372
CWE415s01	103	446	361	28	120	326	15	191	170
CWE415s02		302	0	32	84	218	0	0	0
CWE416	57	884	236	60	1	883	0	0	236
CWE476	36	280	51	10	150	130	0	0	51
bash	397	3	31	12	1	2	60	7	24
darknet	28	21	666	5	6	15	101	138	528
ffmpeg	1162	196	▼	211	90	106		▼	
git	262	68	612	47	48	20	161	79	533
libcrypto.so.3	490	114	647	21	41	73	418	50	597
libicuuc.so	833	97	257	66	34	63	34	71	186
libuv.so.1.0.0	62	0	7	1	0	0	0	0	7
mariadb	3966	39	86	2	11	28	15	13	73
mysql	4572	64	111	6	26	38	20	19	92
php	1358	189	2160	51	84	105	574	206	1783
python	749	31	719	7	15	16	326	421	420
redis-server	170	871	911	214	387	487	93	697	214
ss-local	34	1	22	4	1	0	4	6	16
tmux	60	49	292	35	10	39	43	116	176
vim	389	103	536	13	33	70	131	111	425
wget	129	42	53	19	4	38	25	12	41
wrk	596	56	516	48	33	23	217	100	416
Total		**4757**	**9488**	**1105**	**1225**	**3535**	**2328**	**2432**	**7007**
Rate				**23.81%**	**25.74%**	**74.26%**	**24.94%**	**25.77%**	**74.23%**

Promotion Rules

$$\frac{\begin{array}{c} v = \texttt{ptrtoint}\ \tau*\ v_p\ \texttt{to}\ \texttt{i}sz \\ (\mathsf{isAgg}(\tau) \wedge (v_k = v \vee v_j = v)) \Rightarrow [\![v_i = \texttt{add}\ \texttt{i}sz\ v_j,\ v_k]\!]_n \end{array}}{\begin{array}{c} v' = \texttt{getelementptr}\ \tau*\ v_p,\ \texttt{i32}\ [\![v_{idx}]\!]_m \\ v = \texttt{ptrtoint}\ \mathsf{typeOf}(v')\ v'\ \texttt{to}\ \texttt{i}sz \end{array}}\ (\texttt{multiAdd}) \qquad \frac{\begin{array}{c} v_i = \texttt{ptrtoint}\ \tau*\ v_p\ \texttt{to}\ \texttt{i}sz \\ v = \texttt{sub}\ \texttt{i}sz\ v,\ v_c \end{array}}{\begin{array}{c} v_i = \texttt{ptrtoint}\ \tau*\ v_p\ \texttt{to}\ \texttt{i}sz \\ v'_c = \texttt{sub}\ \texttt{i}sz\ 0,\ v_c; \\ v = \texttt{add}\ \texttt{i}sz\ v_i,\ v'_c \end{array}}\ (\texttt{subToAdd})$$

$$\frac{\begin{array}{c} v_i = \texttt{ptrtoint}\ \tau*\ v_j\ \texttt{to}\ \texttt{i}sz \\ v = \texttt{inttoptr}\ \texttt{i}sz\ v_i\ \texttt{to}\ \tau* \end{array}}{\mathsf{replaceUse}(v, v_j)}\ (\texttt{ptrToInt}) \qquad \frac{\begin{array}{c} \exists k, v_k = \texttt{ptrtoint}\ \tau*\ v_p\ \texttt{to}\ \texttt{i}sz; v = \texttt{phi}\ \texttt{i}sz\ [\![\texttt{[}v_i,\ b_i\texttt{]}]\!]_n \\ [\![v'_i = \mathsf{typeConv}(v_i, \tau*)]\!]_n \\ v_t = \texttt{phi}\ \tau*\ [\![\texttt{[}v'_i,\ b_i\texttt{]}]\!]_n \\ v = \texttt{ptrtoint}\ \tau*\ v_t\ \texttt{to}\ \texttt{i}sz \end{array}}{}\ (\texttt{ptrPhi})$$

$$\frac{\begin{array}{c} v'_p = \texttt{bitcast}\ ty_1*\ v_p\ \texttt{to}\ ty_2* \\ v = \texttt{load}\ ty_2*\ v'_p \\ \phi \subset \{ty_1 > ty_2\} \end{array}}{\begin{array}{c} v'_p = \begin{cases} \texttt{getelementptr}\ ty_1*\ v_p,\ 0\ \ \mathsf{isAgg}(ty_1) \\ v_p \qquad\qquad\qquad\qquad\qquad\ \ \texttt{otherwise} \end{cases} \\ v' = \texttt{load}\ \mathsf{typeOf}(v'_p)\ v'_p \\ v = \mathsf{typeConv}(v', ty_2) \end{array}}\ (\texttt{loadBitcast}) \qquad \frac{\begin{array}{c} v'_p = \texttt{bitcast}\ ty_1*\ v_p\ \texttt{to}\ ty_2* \\ \texttt{store}\ ty_2\ v,\ ty_2*\ v'_p \\ \phi \subset \{ty_1 > ty_2\} \end{array}}{\begin{array}{c} v'_p = \begin{cases} \texttt{getelementptr}\ ty_1*\ v_p,\ 0\ \ \mathsf{isAgg}(ty_1) \\ v_p \qquad\qquad\qquad\qquad\qquad\ \ \texttt{otherwise} \end{cases} \\ v' = \mathsf{typeConv}(v,\ \mathsf{typeOfPointee}(v'_p)) \\ \texttt{store}\ \mathsf{typeOf}(v')\ v',\ \mathsf{typeOf}(v'_p)\ v'_p \end{array}}\ (\texttt{storeBitcast})$$

$$\frac{\begin{array}{c} [\![v_1 \mid v_2]\!] = \texttt{ptrtoint}\ \tau*\ v_p\ \texttt{to}\ \texttt{i}sz \\ \texttt{icmp}\ \texttt{i}sz\ v_1, v_2 \end{array}}{\begin{array}{c} v'_1 = \mathsf{typeConv}(v_1,\ \tau*) \\ v'_2 = \mathsf{typeConv}(v_2,\ \tau*) \\ \texttt{icmp}\ \tau*\ v'_1, v'_2 \end{array}}\ (\texttt{ptrIcmp}) \qquad \frac{\begin{array}{c} v'_p = \texttt{load}\ \texttt{i}sz*\ v_p; \\ v = \texttt{inttoptr}\ \texttt{i}sz\ v'_p\ \texttt{to}\ \tau* \end{array}}{\begin{array}{c} v'_p = \texttt{bitcast}\ \texttt{i}sz*\ v_p\ \texttt{to}\ \tau** \\ v_l = \texttt{load}\ \tau**\ v'_p \\ v' = \texttt{ptrtoint}\ \tau*\ v_l\ \texttt{to}\ \texttt{i}sz \\ v = \texttt{inttoptr}\ \texttt{i}sz\ v'\ \texttt{to}\ \tau* \end{array}}\ (\texttt{loadPtr}) \qquad \frac{\begin{array}{c} v'_p = \texttt{ptrtoint}\ \tau*\ v_p\ \texttt{to}\ \texttt{i}sz \\ \texttt{store}\ \texttt{i}sz\ v'_p,\ \texttt{i}sz*\ v \end{array}}{\begin{array}{c} v' = \texttt{bitcast}\ \texttt{i}sz*\ v\ \texttt{to}\ \tau**; \\ \texttt{store}\ \tau*\ v_p,\ \tau**\ v' \end{array}}\ (\texttt{storePtr})$$

$$\frac{\begin{array}{c} v = \texttt{load}\ \texttt{i}sz*\ v_p \\ (n = 0 \wedge (v_k = v \vee v_j = v)) \Rightarrow [\![v_i = \texttt{add}\ \texttt{i}sz\ v_j,\ v_k]\!]_n \\ \exists m, v_m = \texttt{inttoptr}\ \texttt{i}sz\ v\ \texttt{to}\ \tau* \end{array}}{\begin{array}{c} v = \texttt{load}\ \texttt{i}sz*\ v_p \\ v' = \texttt{inttoptr}\ \texttt{i}sz\ v\ \texttt{to}\ \tau* \\ v'' = \texttt{ptrtoint}\ \tau*\ v'\ \texttt{to}\ \texttt{i}sz \\ (n = 0 \wedge (v_k = v'' \vee v_j = v'')) \Rightarrow [\![v_i = \texttt{add}\ \texttt{i}sz\ v_j,\ v_k]\!]_n \end{array}}\ (\texttt{promoteLoad}) \qquad \frac{\begin{array}{c} v = \texttt{ptrtoint}\ ty_1*\ v_p\ \texttt{to}\ \texttt{i}sz \\ \exists i, v_i = \texttt{inttoptr}\ \texttt{i}sz\ v\ \texttt{to}\ ty_2* \\ \phi \subset \{ty_2 > ty_1\} \end{array}}{\begin{array}{c} v_i = \texttt{bitcast}\ ty_1*\ v_p\ \texttt{to}\ ty_2* \\ v_j = \texttt{ptrtoint}\ ty_2*\ v_i\ \texttt{to}\ \texttt{i}sz \\ \mathsf{replaceUse}(v, v_j) \end{array}}\ (\texttt{promotePtrToInt})$$

Demotion Rules

$$\frac{\begin{array}{c} \exists k, v_k = \texttt{inttoptr}\ \texttt{i}sz\ v_p\ \texttt{to}\ \tau*; v = \texttt{phi}\ \tau*\ [\![\texttt{[}v_i,\ b_i\texttt{]}]\!]_n \\ [\![v'_i = typeConv(v_i, \texttt{i}sz)]\!]_n \\ v_t = \texttt{phi}\ \texttt{i}sz\ [\![\texttt{[}v'_i,\ b_i\texttt{]}]\!]_n \\ v = \texttt{inttoptr}\ \texttt{i}sz\ v_t\ \texttt{to}\ \tau* \end{array}}{}\ (\texttt{intPhi}) \qquad \frac{\begin{array}{c} [\![v_1 \mid v_2]\!] = \texttt{inttoptr}\ \texttt{i}sz\ v_b\ \texttt{to}\ \tau* \\ \texttt{icmp}\ \tau*\ v_1, v_2 \end{array}}{\begin{array}{c} v'_1 = \mathsf{typeConv}(v_1,\ \texttt{i}sz) \\ v'_2 = \mathsf{typeConv}(v_2,\ \texttt{i}sz) \\ \texttt{icmp}\ \texttt{i}sz\ v'_1, v'_2 \end{array}}\ (\texttt{intIcmp})$$

$$\frac{\begin{array}{c} v_p = \texttt{inttoptr}\ \texttt{i}sz\ v_b\ \texttt{to}\ \tau* \\ v = \texttt{getelementptr}\ \tau*\ v_p,\ \texttt{i32}\ [\![v_{idx}]\!]_m \\ \tau\prime = \mathsf{typeOf}(v) \end{array}}{\begin{array}{c} v_p = \texttt{inttoptr}\ \texttt{i}sz\ v_b\ \texttt{to}\ \tau* \\ v_b = \texttt{ptrtoint}\ \tau*\ v_p\ \texttt{to}\ \texttt{i}sz \\ (n = 0 \wedge (v_k = v_b \vee v_j = v_b)) \Rightarrow \\ [\![v_i = \texttt{add}\ \texttt{i}sz\ v_j,\ v_k]\!]_n \Rightarrow v_n \\ v = \texttt{inttoptr}\ \texttt{i}sz\ v_n\ \texttt{to}\ \tau\prime \end{array}}\ (\texttt{intToPtrGep}) \qquad \frac{\begin{array}{c} v'_p = \texttt{load}\ \tau**\ v_p; \\ v = \texttt{ptrtoint}\ \tau*\ v'_p\ \texttt{to}\ \texttt{i}sz \end{array}}{\begin{array}{c} v'_p = \texttt{bitcast}\ \tau**\ v_p\ \texttt{to}\ \tau* \\ v_l = \texttt{load}\ \tau*\ v'_p \\ v' = \texttt{inttoptr}\ \texttt{i}sz\ v_l\ \texttt{to}\ \tau* \\ v = \texttt{ptrtoint}\ \tau*\ v_l\ \texttt{to}\ \texttt{i}sz \end{array}}\ (\texttt{loadInt}) \qquad \frac{\begin{array}{c} v'_p = \texttt{inttoptr}\ \texttt{i}sz\ v_p\ \texttt{to}\ \tau* \\ v = \texttt{ptrtoint}\ \tau*\ v'_p\ \texttt{to}\ \texttt{i}sz \end{array}}{\mathsf{replaceUse}(v, v_p)}\ (\texttt{intToPtr})$$

Figure 2: Selected promotion and demotion transformation rules.