

Association rules obtained from the dataset
described in *Improving type information inferred
by decompilers with supervised machine learning*

Javier Escalada
escaladajavier@uniovi.es

Francisco Ortin*
ortin@uniovi.es

January 21, 2021

Contents

1	Introduction	2
2	Rules	2
2.1	Class <code>bool</code>	3
2.2	Class <code>char</code>	4
2.3	Class <code>short</code>	4
2.4	Class <code>int</code>	5
2.5	Class <code>pointer</code>	6
2.6	Class <code>struct</code>	7
2.7	Class <code>long long</code>	8
2.8	Class <code>float</code>	8
2.9	Class <code>double</code>	9
2.10	Class <code>void</code>	9
3	Generalizations	9

*Corresponding author

1 Introduction

This technical report is a support document for the article *Improving type information inferred by decompilers with supervised machine learning*, written by Javier Escalada, Ted Scully and Francisco Ortin [1].

This document shows a complete set of valid association rules that correlate RET and POST CALL patterns (Section 4.3.1 of [1]) with the high-level return type of functions (Section 6.2 of [1]). The rules analyzed in Section 7 of [1]) are a condensed version of some of these rules.

To create and select the association rules we have used the following procedure: We start extracting and generalizing all the possible assembly patterns from a C code corpus [2]. Then, we select the most important features with the 5 feature-selection algorithms described in Section 5.4 of [1]. Later, we choose the intersection of all the feature sets. After that, we create a set of 26239 association rules using the Apriori algorithm [3]. Finally, we select those rules with a minimum confidence of 95%, a minimum support of 0.0002 and 3 antecedents at most. The 6 possible formats are the following ones:

- $\{ret_pattern\} \Rightarrow \{returned_type\}$
- $\{post_call_pattern\} \Rightarrow \{returned_type\}$
- $\{ret_pattern, post_call_pattern\} \Rightarrow \{returned_type\}$
- $\{post_call_pattern, post_call_pattern\} \Rightarrow \{returned_type\}$
- $\{ret_pattern, post_call_pattern, post_call_pattern\} \Rightarrow \{returned_type\}$
- $\{post_call_pattern, post_call_pattern, post_call_pattern\} \Rightarrow \{returned_type\}$

Section 3 lists all the generalizations (Section 4.3.2 of [1]) used in the association rules. This list contains the ones in Table 8 of [1]) and new ones.

2 Rules

In the following tables, *reg* variables represent registers, *literal* integer literals, *address* absolute addresses, **address* absolute addresses dereferences and *offset* relative addresses.

2.1 Class bool

Antecedents	Consequent	Confidence	Support
(RET) <i>return_bool_cast</i> al, [<i>reg_bp</i>] <i>callee_epilogue</i>			
(POST_CALL) <i>caller_epilogue</i> movzx ecx, al	bool	1	0.00113
(POST_CALL) <i>caller_epilogue</i> movzx edx, al			
(RET) <i>return_bool_literal</i> al <i>callee_epilogue</i>	bool	1	0.00059
(POST_CALL) <i>caller_epilogue</i> movzx edx, al			
(RET) <i>return_bool_cast</i> al, [<i>reg_bp</i>] <i>callee_epilogue</i>			
(POST_CALL) <i>caller_epilogue</i> mov address, al	bool	1	0.00059
(POST_CALL) <i>caller_epilogue</i> movzx eax, al			
(POST_CALL) <i>caller_epilogue</i> mov [<i>reg_bp</i>], al			
(POST_CALL) <i>caller_epilogue</i> movzx eax, al	bool	1	0.000492
(POST_CALL) <i>caller_epilogue</i> movzx edx, al			
(POST_CALL) <i>caller_epilogue</i> mov address, al			
(POST_CALL) <i>caller_epilogue</i> movzx eax, al	bool	1	0.000394
(POST_CALL) <i>caller_epilogue</i> movzx ecx, al			
(POST_CALL) <i>caller_epilogue</i> mov address, al			
(POST_CALL) <i>caller_epilogue</i> movzx eax, al	bool	1	0.000394
(POST_CALL) <i>caller_epilogue</i> movzx edx, al			
(RET) xor al, al <i>callee_epilogue</i>	bool	1	0.000295
(POST_CALL) <i>caller_epilogue</i> mov [<i>reg_bp</i>], al			
continues			

Antecedents	Consequent	Confidence	Support
(POST_CALL) <i>caller_epilogue</i> mov <i>address</i> , al	bool	0.964	0.00133
(POST_CALL) <i>caller_epilogue</i> movzx <i>eax</i> , al			
(RET) return_bool_cast al, [<i>reg_bp</i>] <i>callee_epilogue</i>	bool	0.95	0.000935
(POST_CALL) <i>caller_epilogue</i> movzx <i>eax</i> , al			
(POST_CALL) <i>caller_epilogue</i> movzx <i>edx</i> , al			

2.2 Class char

Antecedents	Consequent	Confidence	Support
(RET) assign_char_literal <i>callee_epilogue</i>	char	1	0.00536
(RET) return_int_math_op al, ecx <i>callee_epilogue</i>	char	1	0.000738

2.3 Class short

Antecedents	Consequent	Confidence	Support
(RET) return_bool_cast ax, [<i>reg_bp</i>] <i>callee_epilogue</i>	short	1	0.0186
(POST_CALL) <i>caller_epilogue</i> cwde	short	1	0.0119
(POST_CALL) <i>caller_epilogue</i> movsx <i>ecx</i> , ax	short	1	0.00644
(POST_CALL) <i>caller_epilogue</i> movzx <i>eax</i> , ax	short	1	0.00418
(POST_CALL) <i>caller_epilogue</i> movzx <i>ecx</i> , ax	short	1	0.00408
(POST_CALL) <i>caller_epilogue</i> movzx <i>edx</i> , ax	short	1	0.00384
(POST_CALL) <i>caller_epilogue</i> mov [<i>reg_bp</i>], ax	short	1	0.0033
(POST_CALL) <i>caller_epilogue</i> mov <i>address</i> , ax	short	1	0.00271
continues			

Antecedents	Consequent	Confidence	Support
(RET) movzx ax, [reg _{bp}] callee_epilogue	short	1	0.00207
(RET) mov ax, [reg _{bp}] callee_epilogue	short	1	0.00182
(RET) mov ax, [reg _{ax}] callee_epilogue	short	1	0.00157
(RET) mov ax, [reg _{dx}] callee_epilogue	short	1	0.00138
(RET) return_int_math_op ax, ecx callee_epilogue	short	1	0.00123
(RET) mov ax, [reg _{cx}] callee_epilogue	short	1	0.00108
(RET) mov ax, address callee_epilogue	short	1	0.000787

2.4 Class int

Antecedents	Consequent	Confidence	Support
(RET) return_bool_cast eax, [reg_bp] callee_epilogue	int	1	0.0159
(RET) mov eax, literal callee_epilogue	int	1	0.00212
(POST_CALL) caller_epilogue push eax			
(RET) mov eax, literal callee_epilogue	int	1	0.00157
(POST_CALL) caller_epilogue mov [reg_bp], eax			
(RET) return_int_math_op eax, ecx callee_epilogue	int	1	0.00103
(RET) mov eax, literal callee_epilogue	int	1	0.000639
(POST_CALL) caller_epilogue mov address, eax			
(RET) xor eax, eax callee_epilogue	int	1	0.000492
(POST_CALL) caller_epilogue mov address, eax			
continues			

Antecedents	Consequent	Confidence	Support
(RET) movsx eax, [reg _{bp}] callee_epilogue	int	1	0.000295
(POST_CALL) caller_epilogue push eax			
(RET) or eax, literal callee_epilogue	int	1	0.000295
(POST_CALL) caller_epilogue mov [reg _{bp}], eax			
(RET) mov eax, literal callee_epilogue			
(POST_CALL) caller_epilogue mov [reg _{bp}], eax	int	1	0.000295
(POST_CALL) caller_epilogue push eax			
(RET) idiv ecx callee_epilogue	int	1	0.000246
(POST_CALL) caller_epilogue push eax			
(RET) movsx eax, [reg _{bp}] callee_epilogue	int	0.96	0.00118

2.5 Class pointer

Antecedents	Consequent	Confidence	Support
(RET) mov eax, offset callee_epilogue	pointer	1	0.0059
(RET) add eax, offset callee_epilogue	pointer	1	0.00359
(RET) gen_mov_chain eax, [reg _{bp}], offset callee_epilogue	pointer	1	0.00231
(RET) lea eax, [reg _{bp} +reg _{ax}] callee_epilogue	pointer	1	0.00207
(RET) lea eax, [reg _{bp} +reg _{cx}] callee_epilogue	pointer	1	0.00172
(RET) return_add_assign_int_to_ptr eax, [reg _{bp}], ecx, [reg _{bp}] callee_epilogue	pointer	1	0.00157
(RET) lea eax, [reg _{bp} +reg _{dx}] callee_epilogue	pointer	1	0.00152
continues			

Antecedents	Consequent	Confidence	Support
(RET) lea eax, [reg _{bp}] callee_epilogue	pointer	1	0.00148
(POST_CALL) caller_epilogue push eax			
(RET) return_add_assign_int_to_ptr eax, [reg _{bp}], eax, [reg _{bp}] callee_epilogue	pointer	1	0.000885
(RET) add_int_to_ptr eax, offset callee_epilogue	pointer	1	0.000639
(RET) mov eax, [reg _{cx}] callee_epilogue	pointer	1	0.000295
(POST_CALL) caller_epilogue mov address, eax			
(RET) mov eax, [reg _{ax}] callee_epilogue	pointer	1	0.000295
(POST_CALL) caller_epilogue mov address, eax			
(RET) mov eax, [reg _{bp}] callee_epilogue			
(POST_CALL) caller_epilogue mov address, eax	pointer	1	0.000246
(POST_CALL) caller_epilogue push eax			
(RET) lea eax, [reg _{bp}] callee_epilogue	pointer	0.992	0.00585
(RET) add eax, literal callee_epilogue			
(POST_CALL) caller_epilogue push eax	pointer	0.962	0.00246
(RET) mov eax, address callee_epilogue			
(POST_CALL) caller_epilogue push eax	pointer	0.958	0.00113

2.6 Class struct

Antecedents	Consequent	Confidence	Support
(RET) mov eax, [ebp+8] callee_epilogue	struct	1	0.0632

2.7 Class long long

Antecedents	Consequent	Confidence	Support
(RET) cdq callee_epilogue	long long	1	0.0444
(RET) mov edx, [reg _{dx}] callee_epilogue	long long	1	0.00128
(POST_CALL) caller_epilogue mov address, eax (POST_CALL) caller_epilogue push edx	long long	1	0.000738
(RET) mov edx, [reg _{bp} +reg _{cx}] callee_epilogue	long long	1	0.00059
(RET) mov edx, address callee_epilogue	long long	1	0.00059
(POST_CALL) caller_epilogue push edx	long long	0.99	0.00984
(RET) mov edx, [reg _{cx}] callee_epilogue	long long	0.98	0.00236

2.8 Class float

Antecedents	Consequent	Confidence	Support
(RET) assign_int_cast_to_float [reg _{bp}], [reg _{bp}], [reg _{bp}] callee_epilogue	float	1	0.0381
(POST_CALL) caller_epilogue fstp_dword [reg _{sp}]	float	1	0.00836
(RET) fld_dword [reg _{bp} +reg _{dx}] callee_epilogue	float	1	0.000885
(RET) fld_dword address callee_epilogue	float	0.99	0.00954
(RET) fld_dword [reg _{ax}] callee_epilogue	float	0.983	0.0028
(RET) fld_dword [reg _{dx}] callee_epilogue	float	0.952	0.00197

2.9 Class double

Antecedents	Consequent	Confidence	Support
(RET) <i>assign_int_cast_to_double</i> [reg _{bp}], [reg _{bp}], [reg _{bp}] <i>callee_epilogue</i>	double	1	0.0357
(RET) <i>fld_qword</i> <i>address</i> <i>callee_epilogue</i>	double	1	0.0103
(RET) <i>return_assign_double</i> [reg _{bp}], xmm0 <i>callee_epilogue</i>	double	1	0.0093
(RET) <i>fld_qword</i> [reg _{bp}] <i>callee_epilogue</i>	double	1	0.00266
(RET) <i>fld_qword</i> [reg _{dx}] <i>callee_epilogue</i>	double	1	0.00207
(RET) <i>fld_qword</i> [reg _{ax}] <i>callee_epilogue</i>	double	1	0.00192
(RET) <i>fld_qword</i> [reg _{cx}] <i>callee_epilogue</i>	double	1	0.00177
(RET) <i>fld_qword</i> [reg _{bp} +reg _{ax}] <i>callee_epilogue</i>	double	1	0.000738
(RET) <i>fld_qword</i> [reg _{bp} +reg _{cx}] <i>callee_epilogue</i>	double	1	0.000689

2.10 Class void

Antecedents	Consequent	Confidence	Support
(RET) <i>mov</i> [reg _{bp}], <i>literal</i> <i>callee_epilogue</i>	void	1	0.00349
(RET) <i>mov</i> <i>address</i> , <i>literal</i> <i>callee_epilogue</i>	void	1	0.00285
(RET) <i>mov</i> [reg _{ax}], <i>ecx</i> <i>callee_epilogue</i>	void	1	0.00182

3 Generalizations

In the following table, *reg* variables represent registers, *literal* integer literals, *address* absolute addresses, **address* absolute addresses dereferences and *offset* relative addresses. Variables between normal-font brackets ([]) represent optional arguments, while typewriter-font brackets ([[]]) are the assembly brackets denoting register-based dereferences.

Assembly pattern	Feature (Generalization)
<code>cond_jump offset₁</code> <code>mov [reg₁], literal₁</code> <code>jmp offset₂</code> <code>mov [reg₁], literal₂</code>	
where: $cond_jump \in \{jo, jno, js, jns, je, jz, jne, jnz, jb, jnae, jc, \hookrightarrow jnb, jae, jnc, jbe, jna, ja, jnbe, jl, jnge, jge, jnl, jle, \hookrightarrow jng, jg, jnle, jp, jpe, jnp, jpo, jcz, jecz\}$ $literal_1, literal_2 \in \{0, 1\}$ $literal_1 \neq literal_2$ $offset_1 \neq offset_2$	$bool_cast(reg_1)$
<code>mov arg₂, arg₁</code> <code>mov arg₃, arg₂</code> <code>...</code> <code>mov arg_n, arg_{n-1}</code>	$gen_mov_chain(arg_n, \dots, arg_1)$
where: $mov \in \{mov, movzx, movsx\}$ $arg_1 \in \{reg, [reg], *address, literal\}$ $arg_2, \dots, arg_n \in \{reg, [reg], *address\}$	
<code>mov arg₂, arg₁</code> <code>mov arg₃, arg₂</code> <code>...</code> <code>mov arg_n, arg_{n-1}</code>	$mov_chain(arg_n, \dots, arg_1)$
where: $arg_1 \in \{reg, [reg], *address, literal\}$ $arg_2, \dots, arg_n \in \{reg, [reg], *address\}$	
<code>bool_cast(reg₁)</code> <code>mov_chain(reg_{ax}, reg_n, ..., reg₁)</code> <code>callee_epilogue</code>	$return_bool_cast(reg_{ax}, reg_n, \dots, reg_1)$
where: $reg_{ax} \in \{eax, ax, al, ah\}$	
<code>mov_chain(reg_{ax}, reg_n, ..., reg₁, literal)</code> <code>callee_epilogue</code>	$return_bool_literal(reg_{ax}, reg_n, \dots, reg_1)$
where: $reg_{ax} \in \{eax, ax, al, ah\}$ $literal \in \{0, 1\}$	
<code>gen_mov_chain(reg₂, reg₁)</code> <code>add reg₂, 1</code> <code>mov_chain(reg₁, reg₂)</code>	$inc_int(reg_1)$
<code>gen_mov_chain(reg₂, reg₁)</code> <code>sub reg₂, 1</code> <code>mov_chain(reg₁, reg₂)</code>	$dec_int(reg_1)$
<code>gen_mov_chain(reg₂, reg₁)</code> <code>add reg₂, literal</code> <code>mov_chain(reg₁, reg₂)</code>	$inc_ptr(reg_1)$
where: $literal > 1$	
<code>gen_mov_chain(reg₂, reg₁)</code> <code>sub reg₂, literal</code> <code>mov_chain(reg₁, reg₂)</code>	$dec_ptr(reg_1)$
where: $literal > 1$	
<code>shl reg₂, literal</code> <code>mov_chain(reg₁, *address)</code> <code>sub reg₁, reg₂</code>	$sub_int_to_ptr(reg_1, *address, reg_2)$
<code>call __allmul</code> <code>mov_chain(reg, *address)</code> <code>sub reg, eax</code>	$sub_int_to_ptr(reg, *address)$
<code>call __allmul</code> <code>add eax, *address</code>	$add_int_to_ptr(*address)$
continues	

Assembly pattern	Feature (Generalization)
$sub_int_to_ptr(reg_1, *address, [reg_2])$ $mov_chain(reg_{ax}, reg_1)$ $callee_epilogue$ where: $reg_{ax} \in \{eax, ax, al, ah\}$	$return_sub_int_to_ptr(reg_{ax}, reg_1, *address,$ $\hookrightarrow [reg_2])$
$sub_int_to_ptr(reg_1, *address, [reg_2])$ $gen_mov_chain(reg_{ax}, arg_1, reg_1)$ $callee_epilogue$ where: $reg_{ax} \in \{eax, ax, al, ah\}$ $arg_1 \in \{reg, [reg], *address\}$	$return_sub_assign_int_to_ptr(reg_{ax}, arg_1,$ $\hookrightarrow reg_1, *address, [reg_2])$
$add_int_to_ptr(reg_1, *address)$ $gen_mov_chain(reg_{ax}, arg_1, reg_1)$ $callee_epilogue$ where: $reg_{ax} \in \{eax, ax, al, ah\}$ $arg_1 \in \{reg, [reg], *address\}$	$return_add_assign_int_to_ptr(reg_{ax}, arg_1,$ $\hookrightarrow reg_1, *address)$
$lea reg_1, *address$ $gen_mov_chain(reg_{ax}, arg_1, reg_1)$ $callee_epilogue$ where: $reg_{ax} \in \{eax, ax, al, ah\}$ $arg_1 \in \{reg, [reg], *address\}$	$return_add_assign_int_to_ptr(reg_{ax}, arg_1,$ $\hookrightarrow reg_1, *address)$
$math_op arg_1$ $callee_epilogue$ where: $math_op \in \{div, idiv\}$ $arg_1 \in \{reg, [reg], *address\}$	$return_int_math_op(arg_1)$
$math_op arg_1$ $mov_chain(reg_{ax}, reg_{rem})$ $callee_epilogue$ where: $math_op \in \{div, idiv\}$ $arg_1 \in \{reg, [reg], *address\}$ $reg_{ax} \in \{eax, ax, al\}$ $reg_{rem} \in \{edx, dx, ah\}$	$return_int_math_op(reg_{ax}, arg_1)$
$math_op arg_1 arg_2$ $mov_chain(arg_{ax}, reg_1)$ $callee_epilogue$ where: $math_op \in \{imul, sub, add, sar, sal, shr, shl, xor, or, and\}$ $arg_1 \in \{reg, [reg], *address\}$ $arg_2 \in \{reg, [reg], *address, literal\}$ $reg_{ax} \in \{eax, ax, al, ah\}$	$return_int_math_op(reg_{ax}, arg_1, arg_2)$
$math_op arg_2$ $mov_chain(reg_{ax}, arg_1)$ $callee_epilogue$ where: $math_op \in \{div, idiv\}$ $arg_1 \in \{reg, [reg], *address\}$ $arg_2 \in \{reg, [reg], *address\}$ $reg_{ax} \in \{eax, ax, al, ah\}$	$return_int_math_op_assign(reg_{ax}, arg_1, arg_2)$
$math_op arg_2$ $mov_chain(reg_{ax}, arg_1, reg_{rem})$ $callee_epilogue$ where: $math_op \in \{div, idiv\}$ $arg_1 \in \{reg, [reg], *address\}$ $arg_2 \in \{reg, [reg], *address\}$ $reg_{rem} \in \{edx, dx, ah\}$	$return_int_math_op_assign(reg_{ax}, arg_1, arg_2)$
continues	

Assembly pattern	Feature (Generalization)
$math_op \ arg_2 \ arg_3$ $mov_chain(reg_{ax}, arg_1, arg_2)$ $callee_epilogue$ where: $math_op \in \{imul, sub, add, sar, sal, shr, shl, xor, or, and\}$ $arg_1 \in \{reg, [reg], *address\}$ $arg_2 \in \{reg, [reg], *address\}$ $arg_3 \in \{reg, [reg], *address, literal\}$ $reg_{ax} \in \{eax, ax, al, ah\}$	$return_int_math_op_assign(reg_{ax}, arg_1, arg_2, \hookrightarrow arg_3)$
$fstp \ arg$ where: $opcode(fstp)[0] = 0xD9$ $arg \in \{[reg], *address\}$	$fstp_dword(arg)$
$fstp \ arg$ where: $opcode(fstp)[0] = 0xDD$ $arg \in \{[reg], *address\}$	$fstp_qword(arg)$
$fld \ arg$ where: $opcode(fld)[0] = 0xD9$ $arg \in \{[reg], *address\}$	$fld_dword(arg)$
$fld \ arg$ where: $opcode(fld)[0] = 0xDD$ $arg \in \{[reg], *address\}$	$fld_qword(arg)$
$fild \ arg_1$ $fstp_dword(arg_2)$ $fld_dword(arg_3)$ where: $arg_1 \in \{[reg], *address\}$ $arg_2 \in \{[reg], *address\}$ $arg_3 \in \{[reg], *address\}$	$assign_int_cast_to_float(arg_3, arg_2, arg_1)$
$fild \ arg_1$ $fstp_qword(arg_2)$ $fld_qword(arg_3)$ where: $arg_1 \in \{[reg], *address\}$ $arg_2 \in \{[reg], *address\}$ $arg_3 \in \{[reg], *address\}$	$assign_int_cast_to_double(arg_3, arg_2, arg_1)$
$movss \ arg_2, arg_1$ $fld_dword(arg_2)$ where: $arg_1 \in \{reg, [reg], *address\}$ $arg_2 \in \{[reg], *address\}$	$return_assign_float(arg_2, arg_1)$
$movsd \ arg_2, arg_1$ $fld_qword(arg_2)$ where: $arg_1 \in \{reg, [reg], *address\}$ $arg_2 \in \{[reg], *address\}$	$return_assign_double(arg_2, arg_1)$
$mov \ al, literal$ where: $literal > 1$	$assign_char_literal()$

References

- [1] J. Escalada, T. Scully, F. Ortin, Improving type information inferred by decompilers with supervised machine learning (2021). arXiv:2101.08116.
 URL <https://arxiv.org/abs/2101.08116>

- [2] J. Escalada, F. Ortin, T. Scully, An Efficient Platform for the Automatic Extraction of Patterns in Native Code, *Scientific Programming* 2017 (2017) 1–16.
- [3] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994, p. 487–499.