Wastrumentation: Portable WebAssembly Dynamic Analysis with Support for Intercession (Appendix)

Aäron Munsters
Vrije Universiteit Brussel, Brussels, Belgium

Angel Luis Scull Pupo
Vrije Universiteit Brussel, Brussels, Belgium

Elisa Gonzalez Boix
Vrije Universiteit Brussel, Brussels, Belgium

A Wastrumentation ABI

Appendix A shows the instrumentation ABI supported by Wastrumentation. This ABI maps onto the high-level API. The following traps include a generic type (l, r, o, or v): binary_l_r_to_o, unary_i_to_o, trap_local_get_v, trap_local_set_v, trap_local_tee_v, trap_global_get_v, trap_global_set_v, trap_v_store, trap_v_load. Each generic type must be further monomorphized into specific Wasm types. For example, the trap trap_local_get_v corresponds to the following specific traps: trap_local_get_i32 trap_local_get_f32 trap_local_get_i64 trap_local_get_f64.

Table 1 The dynamic analysis ABI supported by Wastrumentation. The trap name is exported by instrumented module. The typed arguments and return type are the type of the function. The associated name per value is left for documentation purposes, the last two arguments are the location of the instruction, i.e., the function index and instruction offset.

Trap name	Typed arguments	Return Type
specialized_if_then_k	cndt: I32, inputs-len: I32, results-len: I32, fidx: I64, iidx: I64,	cont: I32
trap_if_then_post	fidx: I64, iidx: I64,	void
specialized_if_then_else_k	cndt: I32, inputs-len: I32, results-len: I32, fidx: I64, iidx: I64,	cont: I32
trap_if_then_else_post	fidx: I64, iidx: I64,	void
specialized_br	lbl: I64, fidx: I64, iidx: I64,	void
specialized_br_if	cndt: I32, lbl: I32, fidx: I64, iidx: I64,	cont: I32
specialized_br_table	br_tbl_tgt_idx: I32, runtime_label: I32, dflt_idx: I32, fidx: I64, iidx: I64,	br_tbl_tgt_idx: I32
specialized_call_pre	f_tgt: I32, fidx: I64, iidx: I64,	void
specialized_call_post	f_tgt: I32, fidx: I64, iidx: I64,	void
specialized_call_indirect_pre	fn_tbl_idx: I32, fn_tbl: I32, fidx: I64, iidx: I64,	fn_tbl_idx: I32
specialized_call_indirect_post	fn_tbl: I32, fidx: I64, iidx: I64,	void
specialized_select	cndt: I32, fidx: I64, iidx: I64,	cont: I32
return_trap	fidx: I64, iidx: I64,	void
drop_trap	fidx: I64, iidx: I64,	void
trap_const_i32	const: I32, fidx: I64, iidx: I64,	res: I32
trap_const_f32	const: F32, fidx: I64, iidx: I64,	res: F32
trap_const_i64	const: I64, fidx: I64, iidx: I64,	res: I64
trap_const_f64	const: F64, fidx: I64, iidx: I64,	res: F64
binary_l_r_to_o	lopnd: l, ropnd: _r, oprtr: I32, fidx: I64, iidx: I64,	res: o
unary_i_to_o	opnd: i, oprtr: I32, fidx: I64, iidx: I64,	res: o
trap_local_get_v	value: v, idx: I64, fidx: I64, iidx: I64,	value: v
trap_local_set_v	value: v, idx: I64, fidx: I64, iidx: I64,	value: v
trap_local_tee_v	value: v, idx: I64, fidx: I64, iidx: I64,	value: v
trap_global_get_v	value: v, idx: I64, fidx: I64, iidx: I64,	value: v
trap_global_set_v	value: v, idx: I64, fidx: I64, iidx: I64,	value: v
trap_v_store	write_idx: I32, val: v, offs: I64, op: I32, fidx: I64, iidx: I64,	void
trap_v_load	load_idx: I32, offs: I64, op: I32, fidx: I64, iidx: I64,	res: v
trap_memory_size	size: I32, idx: I64, fidx: I64, iidx: I64,	size: I32
trap_memory_grow	amount: I32, idx: I64, fidx: I64, iidx: I64,	delta-or-neg-1: I32
trap_block_pre	input_c: I32, arity: I32, fidx: I64, iidx: I64,	void
trap_block_post	fidx: I64, iidx: I64,	void
trap_loop_pre	input_c: I32, arity: I32, fidx: I64, iidx: I64,	void
trap_loop_post	fidx: I64, iidx: I64,	void

B Binary Code Increase Experiments

Figure 1 depicts the results of binary code increase experiments for Wastrumentation and Wasabi.

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

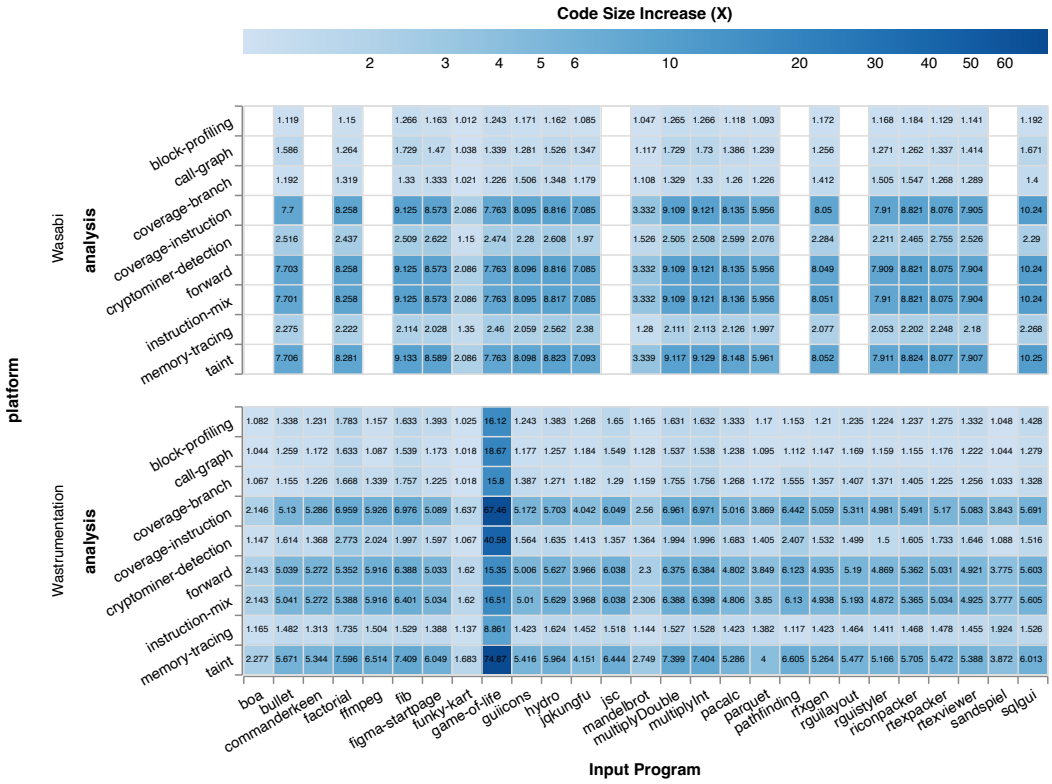

Figure 1 Code size increase after instrumentation for Wasabi and Wastrumentation using the "forward" analysis.

C Performance Overhead Experiments

Figure 3 depicts the performance overhead results for Wastrumentation and Wasabi.

D Forward Analysis across Execution Engines

Figure 3 depicts the performance overhead for the "Forward" analysis for Wastrumentation and Wasabi executing atop of NodeJS and Wasmtime. This figure confirms that while different execution engines will affect the performance, the overhead for Wastrumentation overall lies within the same order of magnitude across different execution engines.

References

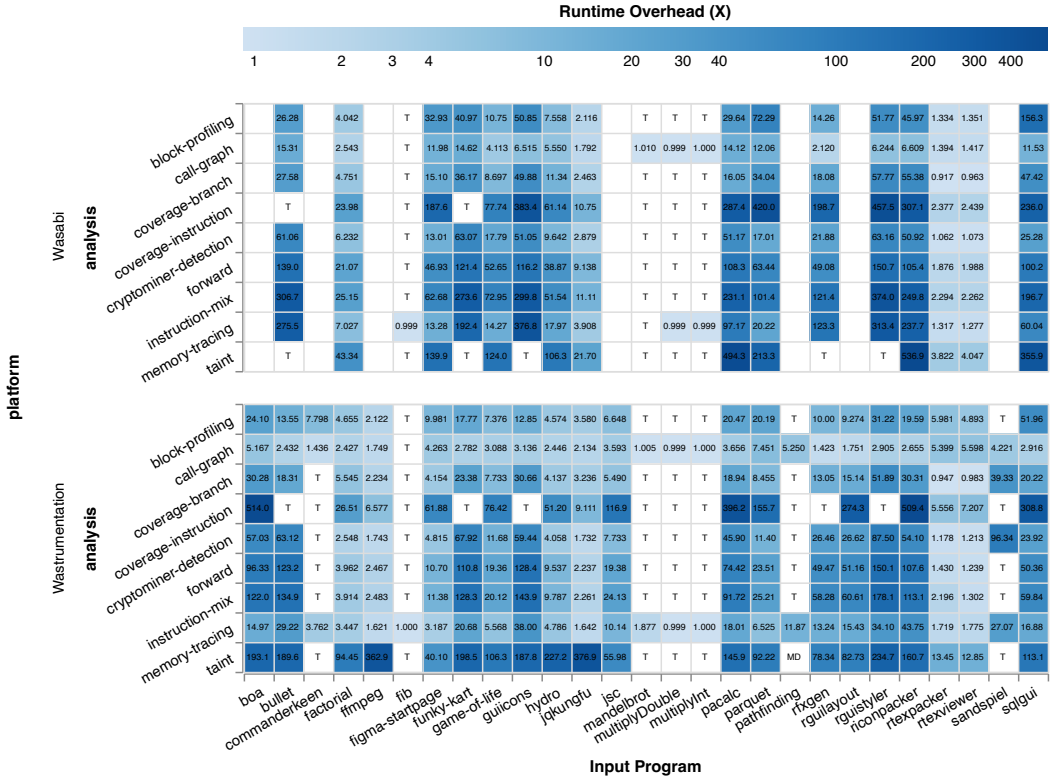

Figure 2 Overhead of Wasabi and Wastrumentation for the WasmR3 benchmark suite.

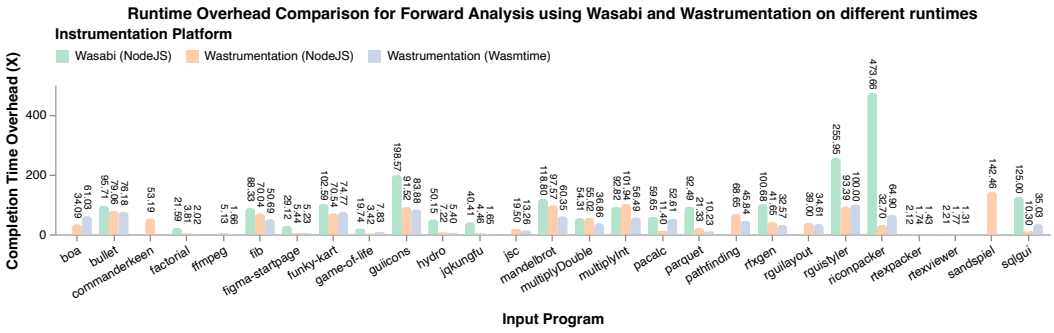

Figure 3 Overhead for the Forward analysis for different execution engines for Wasabi and Wastrumentation.