Indexed Types for Faster WebAssembly (appendix)

Anonymous Author(s)

Contents

A Complete Wasm-precheck Typing Judgment Definition

$$\boxed{C \vdash e^* : ti_1^*; l_1; \Gamma_1; \phi_1 \to ti_2^*; l_2; \Gamma_2; \phi_2}$$

$$\textsc{Unreachable}\quad \frac{}{C \vdash \mathsf{unreachable} : ti_1^*; l_1; \Gamma_1; \phi_1 \to ti_2^*; l_2; \Gamma_2; \phi_2}$$

$$\textsc{Nop}\quad \frac{}{C \vdash \mathsf{nop} : \epsilon; l; \Gamma; \phi \to \epsilon; l; \Gamma; \phi}$$

$$\textsc{Drop}\quad \frac{}{C \vdash \mathsf{drop} : (t\ \alpha); l; \Gamma; \phi \to \epsilon; l; \Gamma; \phi}$$

$$\frac{\alpha \notin \Gamma}{C \vdash t.\mathsf{const}\ c : \epsilon; l; \Gamma, \phi \to (t\ \alpha); l; \Gamma, (t\ \alpha); \phi, (= \alpha\ (t\ c))}\ \textsc{Const}$$

$$\frac{\alpha_3 \notin \Gamma}{C \vdash t.binop : (t\ \alpha_1)\ (t\ \alpha_2); l; \Gamma; \phi \to (t\ \alpha_3); l; \Gamma, (t\ \alpha_3); \phi, (= \alpha_3\ (\|binop\|\ \alpha_1\ \alpha_2))}\ \textsc{Binop}$$

$$\textsc{Div-Prechk}\quad \frac{\Gamma \vdash \phi \rightsquigarrow \neg(=\ \alpha_2\ 0) \qquad \alpha_3 \notin \Gamma}{C \vdash t.\mathsf{div}\checkmark : (t\ \alpha_1)\ (t\ \alpha_2); l; \Gamma; \phi \to (t\ \alpha_3); l; \Gamma, (t\ \alpha_3); \phi, (= \alpha_3\ (\mathsf{i32.div}\ \alpha_1\ \alpha_2))}$$

$$\frac{\alpha_3 \notin \Gamma}{C \vdash t.relop : (t\ \alpha_1)\ (t\ \alpha_2); l; \Gamma; \phi \to (t\ \alpha_3); l; \Gamma, (t\ \alpha_3); \phi, (= \alpha_3\ (\|relop\|\ \alpha_1\ \alpha_2))}\ \textsc{Relop}$$

$$\frac{\alpha_2 \notin \Gamma}{C \vdash t.testop : (t\ \alpha_1); l; \Gamma; \phi \to (t\ \alpha_2); l; \Gamma, (t\ \alpha_2); \phi, (= \alpha_2\ (\|testop\|\ \alpha_1))}\ \textsc{Testop}$$

$$\frac{\alpha_2 \notin \Gamma}{C \vdash t.unop : (t\ \alpha_1); l; \Gamma; \phi \to (t\ \alpha_2); l; \Gamma, (t\ \alpha_2); \phi, (= \alpha_2\ (\|unop\|\ \alpha_1))}\ \textsc{Unop}$$

$$\frac{\alpha \notin \Gamma}{C \vdash \mathsf{select} : (t\ \alpha_1)\ (t\ \alpha_2)\ (\mathsf{i32}\ \alpha_3); l; \Gamma, \phi \to (t\ \alpha); l; \Gamma, (t\ \alpha); \phi, (\mathrm{if}\ (= \alpha_3\ (\mathsf{i32}\ 0))\ (= \alpha\ \alpha_2)\ (= \alpha\ \alpha_1))}\ \textsc{Select}$$

$$\textsc{Block}\quad \frac{C, \mathrm{label}\ ((t_2\ \alpha_2)^*; (t_l\ \alpha_{l2})^*; \phi_2) \vdash e^* : (t_1\ \alpha_1)^*; (t_l\ \alpha_{l1})^*; \Gamma_1; \phi_1 \to (t_2\ \alpha_2)^*; (t_l\ \alpha_{l2})^*; \Gamma_2; \phi_3 \qquad \Gamma_2 \vdash \phi_3 \rightsquigarrow \phi_2}{C \vdash \mathsf{block}\ (t_1^* \to t_2^*)\ e^*\ \mathsf{end} : (t_1\ \alpha_1)^*; (t_l\ \alpha_{l1}); \Gamma_1; \phi_1 \to (t_2\ \alpha_2)^*; (t_l\ \alpha_{l2})^*; \Gamma_2; \phi_2}$$

$$\textsc{Loop}\quad \frac{\begin{array}{c} C, \mathrm{label}\ ((t_1\ \alpha_1)^*; (t_l\ \alpha_{l1})^*; \phi_3) \vdash e_1^* : (t_1\ \alpha_1)^*; (t_l\ \alpha_{l1})^*; \Gamma_1\ (t_1\ \alpha_1)^*\ (t_l\ \alpha_{l1})^*; \phi_3 \to (t_2\ \alpha_2)^*; (t_l\ \alpha_{l2})^*; \Gamma_2; \phi_4 \\ \Gamma_1 \vdash \phi_1 \rightsquigarrow \phi_3 \qquad \Gamma_2 \vdash \phi_4 \rightsquigarrow \phi_2 \end{array}}{C \vdash \mathsf{loop}\ t_1^* \to t_2^*\ e^*\ \mathsf{end} : (t_1\ \alpha_1)^*; (t_l\ \alpha_{l1}); \Gamma_1; \phi_1 \to (t_2\ \alpha_2)^*; (t_l\ \alpha_{l2})^*; \Gamma_2; \phi_2}$$

$$\frac{\begin{array}{c} C, \mathrm{label}\ ((t_2\ \alpha_2)^*; (t_l\ \alpha_{l2})^*; \phi_2) \vdash e_1^* : (t_1\ \alpha_1)^*; (t_l\ \alpha_{l1})^*; \Gamma_1; \phi_1, \neg(= \alpha\ (\mathsf{i32}\ 0)) \to (t_2\ \alpha_2)^*; (t_l\ \alpha_{l2})^*; \Gamma_2; \phi_3 \\ C, \mathrm{label}\ ((t_2\ \alpha_2)^*; (t_l\ \alpha_{l2})^*; \phi_2) \vdash e_2^* : (t_1\ \alpha_1)^*; (t_l\ \alpha_{l1})^*; \Gamma_1; \phi_1, (= \alpha\ (\mathsf{i32}\ 0)) \to (t_2\ \alpha_2)^*; (t_l\ \alpha_{l2})^*; \Gamma_2; \phi_4 \\ \Gamma_2 \vdash \phi_3 \rightsquigarrow \phi_2 \qquad \Gamma_2 \vdash \phi_4 \rightsquigarrow \phi_2 \end{array}}{C \vdash \mathsf{if}\ t_1^* \to t_2^*\ e_1^*\ \mathsf{else}\ e_2^*\ \mathsf{end} : (\mathsf{i32}\ \alpha)\ (t_1\ \alpha_1)^*; (t_l\ \alpha_{l1}); \Gamma_1; \phi_1 \to (t_2\ \alpha_2)^*; (t_l\ \alpha_{l2})^*; \Gamma_2; \phi_2}\ \textsc{If}$$

$$\frac{C_{\mathrm{return}} = (t_3\ \alpha_4)^*; \phi_3 \qquad \Gamma_1 \vdash \phi_1 \rightsquigarrow \phi_3[\alpha_4 \mapsto \alpha_3]^*}{C \vdash \mathsf{return} : ti_1^*\ ((t_3\ \alpha_3)^*)\ l_1; \Gamma_1; \phi_1 \to ti_2^*; l_2; \Gamma_2; \phi_2}\ \textsc{Return}$$

$$\frac{C_{\text{label}}(i) = (t_3\ \alpha_4)^*; (t_l\ \alpha_{l4})^*; \phi_3 \qquad \Gamma_1 \vdash \phi_1 \rightsquigarrow \phi_3[\alpha_4 \mapsto \alpha_3]^*[\alpha_{l4} \mapsto \alpha_{l3}]^*}{C \vdash \mathsf{br}\ i : ti_1^*\ (t_3\ \alpha_3)^*; (t_l\ \alpha_{l3})^*; \Gamma_1; \phi_1 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2}\ \textsc{Br}$$

$$\frac{C_{\text{label}}(i) = (t_1\ \alpha_3)^*; (t_l\ \alpha_{l3})^*; \phi_3 \qquad \Gamma_1 \vdash \phi_1, \neg(=\alpha\ (\mathsf{i32}\ 0)) \rightsquigarrow \phi_3[\alpha_3 \mapsto \alpha_1]^*[\alpha_{l3} \mapsto \alpha_{l1}]^*}{C \vdash \mathsf{br\_if}\ i : (t_1\ \alpha_1)^*; (t_l\ \alpha_{l1})^*; \Gamma_1; \phi_1 \longrightarrow (t_1\ \alpha_1)^*; (t_l\ \alpha_{l1})^*; \Gamma_1; \phi_1, (=\alpha\ (\mathsf{i32}\ 0))}\ \textsc{Br-If}$$

$$\frac{(C_{\text{label}}(i) = (t_1\ \alpha_i)^*; (t_l\ \alpha_{li})^*; \phi_i)^+ \qquad (\Gamma_1 \vdash \phi_1 \rightsquigarrow \phi_i[\alpha_i \mapsto \alpha_1]^*[\alpha_{li} \mapsto \alpha_{l1}]^*)^*}{C \vdash \mathsf{br\_table}\ i^+ : (t_1\ \alpha_1)^*\ (\mathsf{i32}\ \alpha); (t_l\ \alpha_{l1})^*; \Gamma_1; \phi_1 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2}\ \textsc{Br-Table}$$

$$\frac{C_{\text{func}}(i) = (t_1\ \alpha_3)^*; \phi_3 \longrightarrow (t_2\ \alpha_4)^*; \phi_4 \qquad \Gamma_1 \vdash \phi_1 \rightsquigarrow \phi_3[\alpha_3 \mapsto \alpha_1]^* \qquad (t_2\ \alpha_4)^* \notin \Gamma_1}{C \vdash \mathsf{call}\ i : (t_1\ \alpha_1)^*; l; \Gamma_1; \phi_1 \longrightarrow (t_2\ \alpha_4)^*; l; \Gamma_1, (t_2\ \alpha_4)^*; \phi_1 \cup \phi_4[\alpha_3 \mapsto \alpha_1]^*}\ \textsc{Call}$$

$$\frac{C_{\text{table}} = (j, tfi^*) \qquad \Gamma_1 \vdash \phi_1 \rightsquigarrow \phi_3[\alpha_3 \mapsto \alpha_1]^* \qquad (t_2\ \alpha_4)^* \notin \Gamma_1}{\begin{array}{l} C \vdash\ \mathsf{call\_indirect}\ ((t_1\ \alpha_3)^*; \phi_3 \longrightarrow (t_2\ \alpha_4)^*; \phi_4) \\ \quad :\ (t_1\ \alpha_1)^*\ (\mathsf{i32}\ \alpha); l; \Gamma_1; \phi_1 \\ \qquad \longrightarrow (t_2\ \alpha_4)^*; l; (\Gamma_1\ (t_2\ \alpha_4))^*; \phi_1 \cup \phi_4[\alpha_3 \mapsto \alpha_1]^* \end{array}}\ \textsc{Call-Indirect}$$

$$\textsc{Call-Indirect-Prechk}\quad \frac{\begin{array}{c} C_{table}(i) = (n, tfi^*) \qquad C_{\text{table}} = (j, tfi^*) \qquad \Gamma_1 \vdash \phi_1 \rightsquigarrow \phi_3[\alpha_3 \mapsto \alpha_1]^* \\ (t_2\ \alpha_4)^* \notin \Gamma_1 \forall i \leq n.\ (\Gamma_1 \vdash \phi_1 \rightsquigarrow \neg(=(\mathsf{i32}\ i)\ a)) \vee\ (tfi^*)(i) = ((t_1\ \alpha_3)^*; \phi_3 \longrightarrow (t_2\ \alpha_4)^*; \phi_4) \end{array}}{\begin{array}{l} C \vdash\ \mathsf{call\_indirect}\checkmark\ ((t_1\ \alpha_3)^*; \phi_3 \longrightarrow (t_2\ \alpha_4)^*; \phi_4) \\ \quad :\ (t_1\ \alpha_1)^*\ (\mathsf{i32}\ \alpha); l; \Gamma_1; \phi_1 \\ \qquad \longrightarrow (t_2\ \alpha_4)^*; l; \Gamma_1, (t_2\ \alpha_4)^*; \phi_1 \cup \phi_4[\alpha_3 \mapsto \alpha_1]^* \end{array}}$$

$$\frac{C_{\text{local}}(i) = t \qquad l(i) = (t\ \alpha) \qquad \alpha_2 \notin \Gamma}{C \vdash \mathsf{get\_local}\ i : \epsilon; l; \Gamma; \phi \longrightarrow (t\ \alpha_2); l; \Gamma, (t\ \alpha_2); \phi, (=\alpha\ \alpha_2)}\ \textsc{Get-Local}$$

$$\frac{C_{\text{local}}(i) = t \qquad l_2 = l_1[i := (t\ \alpha)]}{C \vdash \mathsf{set\_local}\ i : (t\ \alpha); l_1; \Gamma; \phi \longrightarrow \epsilon; l_2; \Gamma; \phi}\ \textsc{Set-Local}$$

$$\frac{C_{\text{local}}(i) = t \qquad l_2 = l_1[i := (t\ \alpha)] \qquad \alpha_2 \notin \Gamma}{C \vdash \mathsf{tee\_local}\ i : (t\ \alpha); l_1; \Gamma; \phi \longrightarrow (t\ \alpha_2); l_2; \Gamma, (t\ \alpha_2); \phi, (=\alpha\ \alpha_2)}\ \textsc{Tee-Local}$$

$$\frac{C_{\text{global}}(i) = \mathrm{mut}^?\ t \qquad \alpha \notin \Gamma}{C \vdash \mathsf{get\_global}\ i : \epsilon; l; \Gamma; \phi \longrightarrow (t\ \alpha); l; \Gamma, (t\ \alpha); \phi}\ \textsc{Get-Global}$$

$$\frac{C_{\text{global}}(i) = \mathrm{mut}\ t}{C \vdash \mathsf{set\_global}\ i : (t\ \alpha); l; \Gamma; \phi \longrightarrow \epsilon; l; \Gamma; \phi}\ \textsc{Set-Global}$$

$$\frac{C_{\text{memory}} = n \qquad 2^a \leq (|tp| <)^? |t| \qquad \alpha_2 \notin \Gamma}{C \vdash t.\mathsf{load}\ (tp\_sx)^?\ a\ o : (\mathsf{i32}\ \alpha_1); l; \Gamma; \phi \longrightarrow (t\ \alpha_2); l; \Gamma, (t\ \alpha_2); \phi}\ \textsc{Mem-Load}$$

$$\frac{C_{\text{memory}} = n \qquad 2^a \leq (|tp| <)^? |t| \qquad \alpha_3 \notin \Gamma \qquad \Gamma \vdash \phi \rightsquigarrow (\text{le } (\text{add } \alpha_1 \ (\text{i32 } o + width)) \ (\text{i32 } n * 64\text{Ki}))}{C \vdash t.\text{load}\checkmark \ (tp\_sx)^? \ a \ o : (\text{i32 } \alpha_1); l; \Gamma; \phi \longrightarrow (t \ \alpha_2); l; \Gamma, (t \ \alpha_2); \phi} \textsc{Load-Prechk}$$

$$\frac{C_{\text{memory}} = n \qquad 2^a \leq (|tp| <)^? |t|}{C \vdash t.\text{store } tp^? \ a \ o : (\text{i32 } \alpha_1) \ (t \ \alpha_2); l; \Gamma; \phi \longrightarrow \epsilon; l; \Gamma; \phi} \textsc{Mem-Store}$$

$$\frac{C_{\text{memory}} = n \qquad 2^a \leq (|tp| <)^? |t| \qquad \Gamma \vdash \phi \rightsquigarrow (\text{le } (\text{add } \alpha_1 \ (\text{i32 } o + width)) \ (\text{i32 } n * 64\text{Ki}))}{C \vdash t.\text{store}\checkmark \ tp^? \ a \ o : (\text{i32 } \alpha_1) \ (t \ \alpha_2); l; \Gamma; \phi \longrightarrow \epsilon; l; \Gamma; \phi} \textsc{Store-Prechk}$$

$$\frac{C_{\text{memory}} = n \qquad \alpha \notin \Gamma}{C \vdash \text{current\_memory} : \epsilon; l; \Gamma; \phi \longrightarrow (\text{i32 } \alpha); l; \Gamma, (\text{i32 } \alpha); \phi} \textsc{Current-Memory}$$

$$\frac{C_{\text{memory}} = n \qquad \alpha_2 \notin \Gamma}{C \vdash \text{grow\_memory} : (\text{i32 } \alpha_1); l; \Gamma; \phi \longrightarrow (\text{i32 } \alpha_2); l; \Gamma, (\text{i32 } \alpha_2); \phi} \textsc{Grow-Memory}$$

$$\frac{C \vdash e^* : ti_1^*; l_1; \Gamma_1; \phi_1 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2}{C \vdash e^* : ti^* \ ti_1^*; l_1; \Gamma_1; \phi_1 \longrightarrow ti^* \ ti_2^*; l_2; \Gamma_2; \phi_2} \textsc{Stack-Poly}$$

$$\frac{}{C \vdash \epsilon : \epsilon; l; \Gamma; \phi \longrightarrow \epsilon; l; \Gamma; \phi} \textsc{Empty}$$

$$\frac{C \vdash e_1^* : ti_1^*; l_1; \Gamma_1; \phi_1 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2 \qquad C \vdash e_2 : ti_2^*; l_2; \Gamma_2; \phi_2 \longrightarrow ti_3^*; l_3; \Gamma_3; \phi_3}{C \vdash e_1^* \ e_2 : ti_1^*; l_1; \Gamma_1; \phi_1 \longrightarrow ti_3^*; l_3; \Gamma_3; \phi_3} \textsc{Composition}$$

A.1 Module Types

The complete module typing rules are in Figure 1 (note that *im* is an import and *ex* is an export). Functions f, typecheck their body e^* under the module type context C with the expected postcondition $ti_2^*; l_2; \Gamma_2; \phi_2$ in the label stack and return position, and with the local index store $(t_1 \ a_1)^* \ (t \ a_2)^*$ constructed from the function's arguments $(t_1 \ a_1)^*$ and declared locals $(t \ a_2)^*$. Global variables *glob* must ensure that their initialization instructions e^* produce a value of the proper type t. Exported global variables cannot be mutable, if there are any exports defined, the global cannot have the mutable tag *mut*: $ex^* = \epsilon \lor tg = t$. Tables *tab* ensure that the indices i^n refer to well-typed functions and there are exactly as many indices as the expected size n. Memory *mem* simply has its declared initial size n from which it can only grow bigger. All imported functions, globals, tables, and memories are expected to have their declared type. They are typechecked during linking.

Typechecking a module involves typechecking every component of the module. Functions, f, are typechecked under the module type context, C, containing the entirety of the module. This means that functions can refer to themselves, other functions, all globals, the table, and memory. This may seem to be a circular definition, but the type of the module is declared statically (as the combined declared types of all the module components), so it is just checking against the expected module index type context. Globals, *glob*, are typechecked under the module index context containing only the global variable declarations preceding the current declaration.

$$\frac{\begin{array}{c} C_2 = C, \text{local } t_1^*\ t^*, \text{label } (ti_2^*; l_2; \phi_2), \text{return } (ti_2^*; \phi_2) \\ C_2 \vdash e^* : \epsilon; (t_1\ \alpha_0)^*\ (t\ \alpha)^*; \emptyset, (t_1\ \alpha_0)^*, (t\ \alpha)^*; (\phi_1, (= \alpha\ (t\ 0)^*))[\alpha_1 \mapsto \alpha_0] \rightarrow (t_2\ \alpha_3)^*; l_2; \Gamma_3; \phi_3 \\ \vdash (t_1\ \alpha_1)^*; \phi_1 \rightarrow (t_2\ \alpha_2)^*; \phi_2 \qquad \Gamma_3 \vdash \phi_3 \rightsquigarrow \phi_2[\alpha_2 \mapsto \alpha_3] \end{array}}{C \vdash ex^*\ \text{func}\ (t_1\ \alpha_1)^*; \phi_1 \rightarrow (t_2\ \alpha_2)^*; \phi_2\ \text{local}\ t^*\ e^* : ex^*\ (t_1\ \alpha_1)^*; \phi_1 \rightarrow (t_2\ \alpha_2)^*; \phi_2}\ \textsc{Func}$$

$$\frac{}{C \vdash ex^*\ \text{func}\ tfi\ im : ex^*\ tfi}\ \textsc{Function-Import}$$

$$\frac{tg = mut^?\ t \qquad ex^* = \epsilon \vee tg = t \qquad C \vdash e^* : \epsilon; \epsilon; \emptyset; \emptyset \rightarrow (t\ a); \epsilon; \Gamma_2; \phi_2}{C \vdash ex^*\ \text{global}\ tg\ e^* : ex^*\ tg}\ \textsc{Global}$$

$$\frac{tg = t}{C \vdash ex^*\ \text{global}\ tg\ im : ex^*\ tg}\ \textsc{Global-Import} \qquad \frac{(C_{\text{func}}(i) = tfi)^n}{C \vdash ex^*\ \text{table}\ n\ i^n : ex^*\ (n, tfi^n)}\ \textsc{Table}$$

$$\frac{}{C \vdash ex^*\ \text{table}\ (n, tfi^n)\ im : ex^*\ (n, tfi^n)}\ \textsc{Table-Import} \qquad \frac{}{C \vdash ex^*\ \text{memory}\ n : ex^*\ n}\ \textsc{Memory}$$

$$\frac{}{C \vdash ex^*\ \text{memory}\ n\ im : ex^*\ n}\ \textsc{Memory-Import}$$

$$\frac{\begin{array}{c} (C \vdash f : ex_f^*\ tfi)^* \qquad (C_i \vdash glob_i : ex_g^*\ tg_i)^* \qquad (C \vdash tab : ex_t^*\ (n, tfi^n))^? \qquad (C \vdash mem : ex_m^*\ n)^? \\ (C_i = \{\text{global } tg^{i-1}\})_i^* \qquad ex_f^{**}\ ex_g^{**}\ ex_t^{*?}\ ex_m^{*?}\ \text{distinct} \qquad C = \{\text{func } tfi^*, \text{global } tg^*, \text{table } (n, tfi^n)^?, \text{memory } n^?\} \end{array}}{\vdash \text{module}\ f^*\ glob^*\ tab^?\ mem^?}\ \textsc{Module}$$

Figure 1. Indexed Module Typing Rules

A.2 Administrative Typing Rules

While we have shown the Wasm-precheck typing rules for instructions within a static context, we still need typing rules for administrative instructions and the store used in reduction. *Administrative instructions* are introduced for reduction to keep track of information during reduction. For example, local is the result of reducing a closure call; it is used to reduce a function body within the closed environment of the closure. They are not part of the surface syntax of a language (*e.g.,* you cannot put a local block in a Wasm-precheck program), and can only appear as an intermediate term during reduction. Figure 3 shows the Wasm-precheck typing rules for module instances *inst*, the run time store *s*, and various data structures contained within *s*. There are many different judgments being introduced, so we explicitly state the form of the judgment before stating the rule for that judgment.

During reduction, we use Rule Program (Figure 2) to ensure that a Wasm-precheck program state (consisting of the store *s*, local variables v^*, and instruction sequence e^*) is well typed (notice that it has the same form as the reduction relation). It uses Rule Code and relies on the store being well-typed (Rule Store in Figure 3), to ensure that a reducible Wasm-precheck program is well typed. Rule Code checks that a sequence of instructions is well typed with an empty stack, the indexed types and constraints for the given local variables in the precondition, and an optional return postcondition (not used by Rule Program). Since local variables are values, we know that each one of them is equal to some constant, so Rule Code is really just checking that the sequence of instructions has some postcondition reachable from the given local variables. There is an optional return postcondition for Rule Code because the typing rule for local blocks (as seen in Rule Local in Figure 4) has as a premise a judgment of the exactly same form, except with a return postcondition.

In addition to getting the type of the instructions being reduced, we also need to know the type of the store *s* since it is part of the reduction relation. Rule Store checks that a run-time store, *s* is well typed by the store context *S*. The store context *S* is to *s* as *C* is to *inst*. That is, it contains the type information for everything in *s*. Rule Store ensures that every module instance *inst* in *s* has the type of the index module context *C* in *S* using Rule Instance. Further, Rule Store ensures that all of the closures in all of the tables in *s* are well typed, and the the sizes of all the tables and memory chunks in *S* do not exceed the actual size of their implementations.

To get the type of the store, we in turn have to know the types of each of the various run-time data structures. Rule Instance checks that a module instance is well-typed by the index module context under the store context *S*. It checks all of the closures

$$S ::= \{\text{inst } C^*,\ \text{tab } n^*,\ \text{mem } m^*\}$$

$\boxed{\vdash s; v^*; e^*}$

$$\frac{\vdash s : S \qquad S; \epsilon \vdash_i v^*; e^* : ti^*; l; \Gamma; \phi}{\vdash_i s; v^*; e^* : ti^*; l; \Gamma; \phi}\ \text{Program}$$

$$\boxed{S; (ti^*; \phi)^? \vdash_i v^*; e^* : ti^*; l; \Gamma; \phi}$$

$$\frac{\begin{array}{c}(\vdash v : (t\ \alpha); \phi_v)^* \qquad (\alpha_2 \notin (t\ \alpha)^*)^* C = S_{\text{inst}}(i), \text{local } t^*, \text{return } (ti^n; \phi)^? \\ S; C \vdash e^* : \epsilon; (t\ \alpha)^*; \emptyset, (t\ \alpha)^*, (t_2\ \alpha_2)^*; \phi_v^*, (= \alpha_2\ (t_2\ c_2))^* \to ti^n; l; \Gamma; \phi \qquad \Gamma \vdash \phi \rightsquigarrow \phi_2\end{array}}{S; (ti^n; \phi)^? \vdash_i v^*; e^* : ti^n; l; \Gamma; \phi_2}\ \text{Code}$$

Figure 2. Wasm-precheck Program Typing Rules

$$\boxed{\vdash s : S}$$

$$\frac{S = \{\text{inst } C^*, \text{tab } n^*, \text{mem } m^*\} \qquad (S \vdash inst : C)^* \qquad ((S \vdash cl : tfi)^*)^* \qquad (n \le |cl^*|)^* \qquad (m \le |b^*|)^*}{\vdash \{\text{inst } inst^*, \text{tab } (cl^*)^*, \text{mem } (b^*)^*\} : S}\ \text{Store}$$

$$\boxed{S \vdash inst : C}$$

$$\frac{(S \vdash cl : tfi)^* \qquad (\vdash v : (t\ a), \phi_v)^* \qquad (S_{\text{tab}}(i) = n)^? \qquad (S_{\text{mem}}(j) = m)^?}{\begin{array}{c}S \vdash \{\text{func } cl^*, \text{glob } v^*, \text{tab } i^?, \text{mem } j^?\} \\ : \{\text{func } tfi^*, \text{global } (\text{mut}^?\ t)^*, \text{table } n^?, \text{memory } m^?\}\end{array}}\ \text{Instance}$$

$$\boxed{\vdash v : ti; \phi}$$

$$\frac{}{\vdash t.\text{const } c : (t\ a); \emptyset, (\text{eq } a\ (t\ c))}\ \text{Admin-Const}$$

$$\boxed{S \vdash cl : ti_1^*; \phi_1 \to ti_2^*; \phi_2}$$

$$\frac{S_{\text{inst}}(i) \vdash f : ti_1^*; \phi_1 \to ti_2^*; \phi_2}{S \vdash \{\text{inst } i, \text{code } f\} : ti_1^*; \phi_1 \to ti_2^*; \phi_2}\ \text{Closure}$$

Figure 3. Wasm-precheck Store Typing Rules

cl^* against their expected types tfi^* in C, and similarly for all of the globals (v^* and $(\text{mut}^?\ t)^*$). The table and memory indices (i and j, respectively) are used to look up the the relevant types ((n, tfi^*) and m, respectively) in the store context S. Closures are typechecked by Rule Closure, which falls back on the module typing rules from Figure 1 to typecheck the function definition inside of the closure. Rule Admin-Const gets the postcondition indexed types and constraints on values; it is used to typecheck local and global variables.

Now we will introduce the typing rules for administrative instructions, and the administrative typing judgment in Figure 4. The administrative typing judgment $S; C \vdash e^* : tfi$ extends the Wasm-precheck typing rules for instructions to include administrative instructions and the store context S. Every rule of the judgment $C \vdash e^* : tfi$ is implicitly added to the administrative judgment by accepting any S.

Most of the rules for administrative instructions check against extra information provided by the administrative typing judgment. Rule Local typechecks a local block using Rule Code to ensure that the body e^* is well typed with the indexed types and constraints for local variables provided by the local block as the precondition and any postcondition. Since local blocks are inline expansions of function calls, we use the optional return postcondition functionality of Rule Code to ensure that returning from inside the local block will be well typed. Rule Call-Cl typechecks calling a closure by ensuring that the closure cl being called has the same type as the call instruction call cl in S. Rule Trap is always well typed under any precondition

$$\boxed{S; C \vdash e^* : tfi}$$

$$\frac{S; (ti_2^n; \phi_2) \vdash_i v_l^*; e^* : ti_2^n; l_2; \Gamma_2; \phi_2 \qquad ti_2^n \notin \Gamma_1}{S; C \vdash \mathsf{local}_n\{i; v_l^*\}\ e^*\ \mathsf{end} : \epsilon; l; \Gamma_1; \phi_1 \rightarrow ti_2^n; l; \Gamma_1, ti_2^n; \phi_1 \cup \phi_2}\ \text{Local}$$

$$\frac{S \vdash cl : (t_2\ \alpha_2)^*; \phi_2 \rightarrow (t_3\ \alpha_3)^*; \phi_3 \qquad \Gamma_1 \vdash \phi_1 \rightsquigarrow \phi_2[\alpha_2 \mapsto \alpha_1] \qquad (t_3\ \alpha_4)^* \notin \Gamma_1}{S; C \vdash \mathsf{call}\ cl : (t_2\ \alpha_1)^*; l; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^*; l; \Gamma_1, (t_3\ \alpha_4)^*; \phi_1 \cup \phi_3[\alpha_2 \mapsto \alpha_1][\alpha_3 \mapsto \alpha_4]}\ \text{Call-Cl} \qquad \frac{}{S; C \vdash \mathsf{trap} : tfi}\ \text{Trap}$$

$$\frac{\begin{array}{c} S; C \vdash e_0^* : ti_3^*; l_3; \Gamma_3; \phi_3 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_4 \\ \Gamma_3 \supseteq \Gamma_1, ti_3^*, l_3 \qquad \Gamma_4 \vdash \phi_4 \rightsquigarrow \phi_2 \qquad S; C, \text{label}\ (ti_3^*; l_3; \phi_3) \vdash e^* : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_5 \qquad \Gamma_2 \vdash \phi_5 \rightsquigarrow \phi_2 \end{array}}{S; C \vdash \mathsf{label}\{e_0^*\}\ e^*\ \mathsf{end} : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2}\ \text{Label}$$

Figure 4. Wasm-precheck Administrative Instruction Rules

and postcondition. Rule Label typechecks the body of the label block with the precondition of the saved instructions pushed onto the label stack. If the label was generated by a loop, then the precondition of the saved values is the precondition of the loop, and we know the loop is well typed. Otherwise, the saved instructions will be an empty sequence and will be well typed from the precondition.

B Erasure and Embedding Definitions and Proofs

B.1 Embedding Wasm in Wasm-precheck

We present a way to embed Wasm programs in Wasm-precheck, and prove that it will generate well-typed Wasm-precheck programs when embedding well-typped Wasm programs.

Embedding works purely over the surface syntax of the languages. The embedding function takes a Wasm program and replaces all of the type annotations with indexed function types that have no constraints on the variables. Intuitively, the type annotations are the only part of the surface syntax of Wasm that isn't in Wasm-precheck, so we must figure out a way to bring it over. While this embedding requires no additional developer effort, it provides no information to the indexed type system beyond what can be inferred from the instructions in the program.

First, we define embedding over modules: the pinnacle syntactic objects of both the Wasm and Wasm-precheck surface syntax hierarchies. Embedding a module *module* means embedding all of the functions f^* in the module. We explain how to embed functions f in Definition 4 below. We do not have to embed globals $glob^*$, the table $tab^?$, or the memory $mem^?$ as Wasm and Wasm-precheck use the same syntax to define them (although Wasm-precheck represents the types of tables differently).

We currently cannot typecheck Wasm imported tables under the Wasm-precheck type system since we do not have access to what the function types are for an imported table, which is necessary for typechecking it in Wasm-precheck. Unfortunately, this is currently a limitation that would require the developer to copy over the annotations.

We typeset Wasm-precheck instructions in a blue sans serif font and Wasm instructions in a **bold red font** to set them apart.

Definition 1. $\boxed{embed_m(m) = m}$

$$embed_m(\textbf{module}\ f^*\ glob^*\ tab^?\ mem^?) \quad = \quad \mathsf{module}\ embed_f(f)^*\ embed_g(glob)^*\ tab^?\ mem^?$$

We use lemmas that show that that well-typed Wasm global variable and function definitions embed into well-typed global variables and functions, respectively, in Wasm-precheck.

Theorem 1. Well Typed Wasm Programs Embedded in Wasm-precheck are Well Typed
If $\vdash \textbf{module}\ f^*\ glob^*\ tab^?\ mem^?$,
then $\vdash embed_m(\textbf{module}\ f^*\ glob^*\ tab^?\ mem^?)$

Proof. We must show that every premise of Rule Module holds on the embedding module.

- $(embed_C(C) \vdash embed_f(f) : ex_f^*\ tfi)^*$
  Since $(C \vdash f : ex_f^*\ tfi)^*$ is a premise of $\vdash \textbf{module}\ f^*\ glob^*\ tab^?\ mem^?$, then we have $(embed_C(C) \vdash embed_f(f) : ex_f^*\ tfi)^*$ by Lemma Sound-Embedding-of-Functions.

- $(embed_C(C_i) \vdash embed_g(glob) : ex_g^*\ tg)^*$
  Since $(C \vdash glob : ex_g^*\ tg)^*$ is a premise of $\vdash \textbf{module}\ f^*\ glob^*\ tab^?\ mem^?$, then we have $(embed_C(C) \vdash embed_g(glob) : ex_g^*\ tg)^*$ by Lemma Sound-Embedding-of-Globals.
- $(embed_C(C) \vdash tab : ex_t^*\ (n, tfi^n))^?$
  We proceed by case analysis of $tab^?$:
  - Case ϵ
    This case is vaucuous, so we have nothing to prove.
  - Case $(ex_t^*)\ \textbf{table}\ n\ i^n$
    We want to show that $embed_C(C) \vdash ex_t^*\ \mathsf{table}\ n\ i^n : ex_t^*\ (n, tfi^m)$
    To do so, we must show that $(embed_C(C)_{\text{func}}(i) = tfi)^n$.
    We have $(C_{\text{func}}(i) = tf)^n$, as it is a premise of $\vdash \textbf{module}\ f^*\ glob^*\ tab^?\ mem^?$.
    Thus, we have $(embed_C(C)_{\text{func}}(i) = tfi)^n$ by definition of $embed_C$.
  - Case $(ex_t^*)\ \textbf{table}\ n\ im$
    Unfortunately, we do not currently support embedding imported tables.
- $(embed_C(C) \vdash mem : ex_t^*\ n^?$
  We proceed by case analysis of $mem^?$:
  - Case ϵ
    This case is vaucuous, so we have nothing to prove.
  - Case $(ex_m^*)\ \textbf{memory}\ n$
    Trivially, $embed_C(C) \vdash (ex_m^*)\ \mathsf{memory}\ n : ex_m^*\ n$ by Rule Memory
  - Case $(ex_m^*)\ \textbf{memory}\ n\ im$
    Trivially, $embed_C(C) \vdash (ex_m^*)\ \mathsf{memory}\ n\ im : ex_m^*\ n$ by Rule Memory-Import
- $embed_C(C) = \{\text{func } tfi^*,\ \text{global } tg^*,\ \text{table } (n, tfi^n)^?,\ \text{memory } n^?\}$
  We have $C = \{\text{func } tf^*,\ \text{global } tg^*,\ \text{table } n^?,\ \text{memory } n^?\}$, as it is a premise of $\vdash \textbf{module}\ f^*\ glob^*\ tab^?\ mem^?$. Then this holds by definition of $embed_C$
- $embed_C(C_i) = \{\text{ global } tg^{i-1}\}$
  We have $C_i = \{\text{ global } tg^{i-1}\}$, as it is a premise of $\vdash \textbf{module}\ f^*\ glob^*\ tab^?\ mem^?$. Then this holds by definition of $embed_C$.
- $ex_f^*, ex_g^*, ex_t^*, ex_m^*$ distinct
  We know this to be true since it is a premise of $\vdash \textbf{module}\ f^*\ glob^*\ tab^?\ mem^?$.

□

The proof relies on the definition of the embedding of module type contexts, which we provide below.

Definition 2. $\boxed{embed_C(C) = C}$

$$
\begin{array}{lcl}
embed_C(\{\text{func } (t_4^* \rightarrow t_5^*)^*, & = & \{\text{func } ((t_4\ \alpha_4)^*;\ \emptyset \rightarrow (t_5\ \alpha_5)^*;\ \emptyset)^*, \\
\quad \text{global } tg^*,\ \text{table } n^?,\ \text{memory } n^?, & & \quad \text{global } tg^*,\ \text{table } (n, tfi^*)^?,\ \text{memory } n^?, \\
\quad \text{local } t_1^*,\ \text{label } (t_2^*)^*, & & \quad \text{local } t_1^*,\ \text{label } ((t_2\ \alpha_2)^*;\ (t_1\ \alpha_1)^*;\ \emptyset)^*, \\
\quad \text{return } (t_3^*)^?\}) & & \quad \text{return } ((t_3\ \alpha_3)^*;\ \emptyset)^?\}
\end{array}
$$

The first lemma shows that embedding well-typed Wasm global variable definitions and import declarations results in well-typed Wasm-precheck global variables. Before we show the lemma, we first show the definition of embedding a global variable.

Definition 3. $\boxed{embed_g(glob) = glob}$

$$
\begin{array}{rcl}
embed_g(ex^*\ \textbf{global}\ tg\ e^*) & = & ex^*\ \mathsf{global}\ tg\ embed_{e^*}(e^*)^{\epsilon} \\
embed_g(ex^*\ \textbf{global}\ tg\ im) & = & ex^*\ \mathsf{global}\ tg\ im
\end{array}
$$

This proof relies on a lemma about embedded instructions being well typed, Lemma Sound-Embedding-of-Instructions, to ensure that the instructions that initiate the global produce a value of the correct type. This lemma is introduced and defined below, along with the definition of the embedding function for instructions $embed_{e^*}$.

Lemma 1. Sound-Embedding-of-Globals
If $C \vdash glob : ex^*\ tg$, then $embed_C(C) \vdash embed_g(glob) : ex^*\ tg$.

Proof. We proceed by case anaylsis on $C \vdash glob : ex^* \ tg$.

- $C \vdash \textbf{global} \ tg \ im : ex^* \ tg$
  We want to show that $embed_C(C) \vdash \mathsf{global} \ tg \ im : ex^* \ tg$. To do so, we must show that $tg = t$, which is a premise of $C \vdash \textbf{global} \ tg \ im : ex^* \ tg$, so we have $embed_C(C) \vdash \mathsf{global} \ tg \ im : ex^* \ tg$.
- $C \vdash \textbf{global} \ tg \ e^* : ex^* \ tg$
  We have to show that $embed_C(C) \vdash \mathsf{global} \ tg \ embed_{e^*}(e^*)^{\epsilon} : ex^* \ tg$. To do so, we must show that $tg = \mathrm{mut}^? \ t$, $ex^* = \epsilon \vee tg = t$, and $embed_C(C) \vdash embed_{e^*}(e^*)^{\epsilon} : \epsilon; \epsilon; \emptyset; \emptyset \rightarrow (t \ \alpha); \epsilon; \Gamma; \phi$.
  We have $g = \mathrm{mut}^? \ t$ and $ex^* = \epsilon \vee tg = t$ because they are premises of $C \vdash \textbf{global} \ tg \ e^* : ex^* \ tg$.
  Further, we know $C \vdash e^* : \epsilon \rightarrow t$ since it is also a premise of $C \vdash \textbf{global} \ tg \ e^* : ex^* \ tg$.
  Then, we know that $embed_C(C) \vdash embed_{e^*}(e^*)^{\epsilon} : \epsilon; \epsilon; \emptyset; \emptyset \rightarrow (t \ \alpha); \epsilon; \Gamma; \phi$, for some Γ and ϕ, by Lemma Sound-Embedding-of-Instructions.

□

The embedding of functions, Definition 4, both must construct a pre- and post-condition for itself and embed its body. Function bodies have their local variables defined by the function that they are enclosed in. Thus, when the function body is embedded we pass the local types $(t_1^* \ t^*)$ so the body knows how to constrain local variables.

We construct an indexed function type that has the precondition of the expected values on the stack turned into indexed types using fresh index variables and the types t_1^* from the Wasm type, and do the same with the postcondition and t_2^*. In the precondition, the index variable context contains fresh index variables α_1^*, with associated Wasm types t_1^*, to represent the values passed as arguments. In the postcondition, the index variable context contains the index variables generated to represent the arguments passed to the function, as well as fresh index variables α_2^*, with associated Wasm types t_2^*, to represent the values returned from the function. The index constraint context is empty in both the pre- and post-condition.

Definition 4. $\boxed{embed_f(f) = \mathsf{f}}$

$$
\begin{aligned}
embed_f(\textbf{func} \ (t_1^* \rightarrow t_2^*) \ \textbf{local} \ t^* \ e^*) \quad &= \quad \mathsf{func} \ \ ((t_1 \ \alpha_1)^*; \emptyset \rightarrow (t_2 \ \alpha_2)^*; \emptyset) \\
&\qquad\qquad \mathsf{local} \ t^* \ (embed_e(e)^{(t_1^* \ t^*)})^* \\
&\qquad \mathsf{end} \\
embed_f(\textbf{func} \ (t_1^* \rightarrow t_2^*) \ im) \quad &= \quad \mathsf{func} \ \ ((t_1 \ \alpha_1)^*; \emptyset \rightarrow (t_2 \ \alpha_2)^*; \emptyset) \\
&\qquad\qquad im \\
&\qquad \mathsf{end}
\end{aligned}
$$

Now we prove that embedding a well-typed Wasm function produces a well-typed Wasm-precheck function. Similar to the proof for global variables, this proof relies on a lemma about embedded instructions being well typed, Lemma Sound-Embedding-of-Instructions, to ensure that the embedded function body is well typed. We also require an extra requirement on the implementation of implication, that any constraint set ϕ implies the empty constraint set .

Lemma 2. Sound-Embedding-of-Functions
If $C \vdash f : ex^* \ t_1^* \rightarrow t_2^*$,
then $embed_C(C) \vdash embed_f(f) : ex^* \ ((t_1 \ \alpha_1)^*; \emptyset \rightarrow (t_2 \ \alpha_2)^*; \emptyset)$.

Proof. We proceed by case anaylsis on $C \vdash f : ex^* \ tg$.

- $C \vdash \textbf{func} \ (t_1^* \rightarrow t_2^*) \ im : ex^* \ t_1^* \rightarrow t_2^*$
  Trivially, $embed_C(C) \vdash \mathsf{func} \ ((t_1 \ \alpha_1)^*; \emptyset \rightarrow (t_2 \ \alpha_2)^*; \emptyset) im : ex^* \ (t_1 \ \alpha_1)^*; \emptyset \rightarrow (t_2 \ \alpha_2)^*; \emptyset$
- $C \vdash \textbf{func} \ (t_1^* \rightarrow t_2^*) \ \textbf{local} \ t^* \ e^* : ex^* \ t_1^* \rightarrow t_2^*$
  We want to show that $embed_C(C) \vdash \mathsf{func} \ ((t_1 \ \alpha_1)^*; \emptyset \rightarrow (t_2 \ \alpha_2)^*; \emptyset)$ $\mathsf{local} \ t^* \ embed_{e^*}(e^*)^{t_1^* \ t^*} : ex^* \ (t_1 \ \alpha_1)^*; \emptyset \rightarrow (t_2 \ \alpha_2)^*; \emptyset$ To do so, we must show that

  $$C_2 \vdash embed_{e^*}(e^*)^{t_1^* \ t^*} : \epsilon; (t_1 \ \alpha_1)^* \ (t \ \alpha)^*; (\emptyset, (t_1 \ \alpha_1)^*, (t \ \alpha)^*); \emptyset, (= \alpha \ (t \ 0))^* \rightarrow (t_2 \ \alpha_4)^*; l_2; \Gamma_2; \phi_2$$

  where $\Gamma_2 \vdash \phi_2 \rightsquigarrow \emptyset[\alpha_2 \mapsto \alpha_4][\alpha_1 \mapsto \alpha_3]$ and

  $$C_2 = embed_C(C), \mathrm{local}(t_1^* \ t^*), \mathrm{return}((t_2 \ \alpha_2)^*; \emptyset), \mathrm{label}(((t_2 \ \alpha_4)^*; l_2; \emptyset)$$

  We know $C, \mathrm{local}(t_1^* \ t^*), \mathrm{return}(t_2^*), \mathrm{label}(t_2^*) \vdash e^* : \epsilon \rightarrow t_2^*$ since it is a premise of $C \vdash \textbf{func} \ (t_1^* \rightarrow t_2^*) \ \textbf{local} \ t^* \ e^* : ex^* \ t_1^* \rightarrow t_2^*$.
  By definition of $embed_C$, $C_2 = embed_C(C, \mathrm{local}(t_1^* \ t^*), \mathrm{return}(t_2^*), \mathrm{label}(t_2^*) \vdash e^* : \epsilon \rightarrow t_2^*)$.

Then, we know that

$$C_2 \vdash embed_{e^*}(e^*)^{t_1^* \, t^*} : \epsilon;\ (t_1\ \alpha_1)^*\ (t\ \alpha)^*;\ (\emptyset, (t_1\ \alpha_1)^*, (t\ \alpha)^*);\ \emptyset \to (t_2\ \alpha_4)^*;\ l_2;\ \Gamma_2;\ \phi_2$$

for some l_2, Γ_2, and ϕ_2, by Lemma Sound-Embedding-of-Instructions.

The last case is somewhat tricky, as we allow the implementation of implication to be an under-approximation, and therefore cannot immediately claim that $\Gamma_2 \vdash \phi_2 \rightsquigarrow \emptyset[\alpha_2 \mapsto \alpha_4][\alpha_1 \mapsto \alpha_3]$. However, it is a reasonable requirement that any constraint set should imply the empty set, so we accept this as another requirement on the implementation of implication.

□

Embedding instructions replaces all type annotations used within the Wasm syntax with Wasm-precheck indexed type annotations, and adds the function types for all of the functions in a table to the table's type declaration. This occurs within blocks and indirect function calls, as shown in Definition 5. The indexed types simply have fresh index variables that are different in the precondition and postcondition, and the primitive types for the stack are known from the Wasm type $t_1^* \to t_2^*$. To know what the local variables are, we parameterize the embedding over the types of local variables (t^*).

Definition 5. $\boxed{embed_e(e)^{t^*} = \mathsf{e}}$

$$\begin{aligned}
embed_{e^*}(\mathbf{block}\ (t_1^* \to t_2^*)\ e^*\ \mathbf{end})^{t^*} &= \mathsf{block}(t_1^* \to t_2^*)\ embed_{e^*}(e^*)^{t^*}\ \mathsf{end} \\
embed_{e^*}(\mathbf{loop}\ (t_1^* \to t_2^*)\ e^*\ \mathbf{end})^{t^*} &= \mathsf{loop}(t_1^* \to t_2^*)\ embed_{e^*}(e^*)^{t^*}\ \mathsf{end} \\
embed_{e^*}(\mathbf{if}\ (t_1^* \to t_2^*)\ e_1^*\ e_2^*\ \mathbf{end})^{t^*} &= \mathsf{if}(t_1^* \to t_2^*)\ embed_e(e_1^*)^{t^*}\ embed_e(e_2^*)^{t^*}\ \mathsf{end} \\
embed_{e^*}(\mathbf{call_indirect}\ (t_1^* \to t_2^*))^{t^*} &= \mathsf{call_indirect}\ ((t_1\ \alpha_1)^*;\ \emptyset \to (t_2\ \alpha_2)^*;\ \emptyset) \\
embed_{e^*}(e)^{t^*} &= e, \text{otherwise} \\
embed_{e^*}(e^*)^{t^*} &= (embed_{e^*}(e)^{t^*})^*
\end{aligned}$$

Note that $(t_2\ \alpha_2)^*, (\hat{t}\ \alpha_3)^* \in \Gamma_2$ is generally implicitly proven by the structure of Γ_2.

Lemma 3. Sound-Embedding-of-Instructions

If $C \vdash e^* : t_1^* \to t_2^*$,
and $embed_C(C)_{\text{local}} = \hat{t}^*$,
and $(t_1\ \alpha_1)^*, (\hat{t}\ \alpha)^* \in \Gamma_1$,
then $\forall \Gamma_1, \phi_1.\exists, \Gamma_2, \phi_2. embed_C(C) \vdash embed_{e^*}(e^*)^{\hat{t}^*} : (t_1\ \alpha_1)^*;\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1 \to (t_2\ \alpha_2)^*;\ (\hat{t}\ \alpha_3)^*;\ \Gamma_2;\ \phi_2$
and $(t_2\ \alpha_2)^*, (\hat{t}\ \alpha_3)^* \in \Gamma_2$.

Proof. We proceed by induction on the typing derivation.

Note: we use $\hat{t}^*$ to represent the types of local variables passed to $embed_{e^*}(e^*)$ to set them apart from other ts.

- Case: $C \vdash t.\mathbf{const}\ c : \epsilon \to t$
  Trivially, $embed_C(C) \vdash t.\mathsf{const}\ c : \epsilon;\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1 \to (t\ \alpha_2);\ (\hat{t}\ \alpha)^*;\ \Gamma, (t\ \alpha_2);\ \phi_1, (=\ \alpha_2\ (t\ c))$, by Rule Const.
- Case: $C \vdash t.binop : t\ t \to t$
  Trivially, $embed_C(C) \vdash t.binop : (t\ \alpha_1)\ (t\ \alpha_2);\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1 \to (t\ \alpha_3);\ (\hat{t}\ \alpha)^*;\ \Gamma_1, (t\ \alpha_3);\ \phi_1, (=\ \alpha_3\ (\|binop\|\ \alpha_1\ \alpha_2)$ by Rule Binop.
- Case: $C \vdash t.testop : t \to \mathbf{i32}$
  Trivially, $embed_C(C) \vdash t.testop : (t\ \alpha_1);\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1 \to (\mathsf{i32}\ \alpha_2)(\hat{t}\ \alpha)^*;\ \Gamma_1, (\mathsf{i32}\ \alpha_2);\ \phi_1, (=\ \alpha_2\ (\|testop\|\ \alpha_1)$ by Rule Testop.
- Case: $C \vdash t.relop : t\ t \to \mathbf{i32}$
  Trivially, $embed_C(C) \vdash t.binop : (t\ \alpha_1)\ (t\ \alpha_2);\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1 \to (\mathsf{i32}\ \alpha_3);\ (\hat{t}\ \alpha)^*;\ \Gamma_1, (\mathsf{i32}\ \alpha_3);\ \phi_1, (=\ \alpha_3\ (\|relop\|\ \alpha_1\ \alpha_2)$ by Rule Relop.
- Case: $C \vdash \mathbf{unreachable} : t_1^* \to t_2^*$
  Trivially, $embed_C(C) \vdash \mathsf{unreachable} : (t_1\ \alpha_1)^*;\ (t\ \alpha)^*;\ \Gamma_1;\ \phi_1 \to (t_2\ \alpha_2)^*;\ (\hat{t}\ \alpha)^*;\ \Gamma_2;\ \phi_2$ by Rule Unreachable.
- Case: $C \vdash \mathbf{nop} : \epsilon \to \epsilon$
  Trivially, $embed_C(C) \vdash \mathsf{nop} : \epsilon;\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1 \to \epsilon;\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1$ by Rule Nop.
- Case: $C \vdash \mathbf{drop} : t \to \epsilon$
  Trivially, $embed_C(C) \vdash \mathsf{nop} : (t\ \alpha_1);\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1 \to \epsilon;\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1$ by Rule Drop.
- Case: $C \vdash \mathbf{select} : t\ t\ \mathbf{i32} \to t$
  Trivially, $embed_C(C) \vdash t.binop : (t\ \alpha_1)\ (t\ \alpha_2);\ (\mathsf{i32}\ \alpha_3);\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1$
  $$\to (t\ \alpha_4);\ (\hat{t}\ \alpha)^*;\ \Gamma_1, (t\ \alpha_4);\ \phi_1, (\text{if}\ (=\ \alpha_3\ (\mathsf{i32}\ 0))(=\ \alpha_4\ \alpha_2)(=\ \alpha_4\ \alpha_1))$$
  by Rule Select.

- Case: $C \vdash \textbf{block}\ \ (t_1^* \to t_2^*)\ e^*\ \textbf{end} : t_1^* \to t_2^*$
  We want to show that $embed_C(C) \vdash \text{block}\ (t_1^* \to t_2^*)\ embed_{e^*}(e^*)^{\hat{t}^*}\ \text{end} : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \to (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2$.
  To do so, we must show that $C' \vdash embed_{e^*}(e^*)^{\hat{t}_*} : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \to (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2$, where $C' = embed_C(C), \text{label}((t_2\ \alpha_2)^*; l_2; \phi_2)$.
  We have $C, \text{label}(t_2^*) \vdash e^* : t_1^* \to t_2^*$, as it is a premise of $C \vdash \textbf{block}\ \ (t_1^* \to t_2^*)\ \textbf{end} : t_1^* \to t_2^*$.
  Further, we know $\hat{t}^* = embed_C(C)_{\text{local}}$ as it is a premise of the lemma we are trying to prove, and therefore $C'_{\text{local}} = \hat{t}^*$
  Now we can invoke the inductive hypothesis on e^*, since $C' = embed_C(C, \text{label}(t_2^*))$, to get
  $$C' \vdash embed_{e^*}(e^*)^{\hat{t}_*} : (t_1\ \alpha_1)^*; (\hat{t}\ \alpha_{l1})^*; (\emptyset, (t_1\ \alpha_1)^*, (\hat{t}\ \alpha_{l1})^*); \emptyset \to (t_2\ \alpha_2)^*; (\hat{t}\ \alpha_2)^*; \Gamma_2; \phi_3$$
  The other premise, $\Gamma_2 \vdash \phi_3 \rightsquigarrow \phi_2$ is tricky. It is tricky because we allow the implementation of $\implies$ to be an under-approximation, so we cannot necessarily claim this immediately. However, it is a reasonable requirement of the implementation that any constraint set should imply the empty constraint set. Thus, we simply pick ϕ_2 to be the empty constraint set $\emptyset$, and therefore any ϕ_3 necessarily implies ϕ_2.
- Case: $C \vdash \textbf{loop}\ \ (t_1^* \to t_2^*)\ e^*\ \textbf{end} : t_1^* \to t_2^*$
  We want to show that $embed_C(C) \vdash \text{loop}\ (t_1^* \to t_2^*)\ embed_{e^*}(e^*)^{t^*}\ \text{end} : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \to (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2$.
  To do so, we must show that $C' \vdash embed_{e^*}(e^*)^{\hat{t}_*} : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_3 \to (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2$, where $C' = embed_C(C), \text{label}((t_1\ \alpha_1)^*; l_1; \phi_3)$, and $\Gamma_1 \vdash \phi_1 \rightsquigarrow \phi_3$.
  We have $C, \text{label}(t_1^*) \vdash e^* : t_1^* \to t_2^*$, as it is a premise of $C \vdash \textbf{loop}\ \ (t_1^* \to t_2^*)\ \textbf{end} : t_1^* \to t_2^*$.
  Further, we know $\hat{t}^* = embed_C(C)_{\text{local}}$ as it is a premise of the lemma we are trying to prove, and therefore $C'_{\text{local}} = \hat{t}^*$
  Now we can invoke the inductive hypothesis on e^*, since $C' = embed_C(C, \text{label}(t_1^*))$, to get
  $$C' \vdash embed_{e^*}(e^*)^{\hat{t}_*} : (t_1\ \alpha_1)^*; (\hat{t}\ \alpha_{l1})^*; (\emptyset, (t_1\ \alpha_1)^*, (\hat{t}\ \alpha_{l1})^*); \emptyset \to (t_2\ \alpha_2)^*; (\hat{t}\ \alpha_2)^*; \Gamma_2; \phi_4$$
  The other premise, $\Gamma_2 \vdash \phi_4 \rightsquigarrow \phi_2$, follows as described for the **block** case.
- Case: $C \vdash \textbf{if}\ \ (t_1^* \to t_2^*)\ e_1^*\ e_2^*\ \textbf{end} : t_1^* \to t_2^*$
  We want to show that $embed_C(C) \vdash \text{if}\ (t_1^* \to t_2^*)\ embed_{e^*}(e_1^*)^{t^*}\ embed_{e^*}(e_2^*)^{t^*}\ \text{end} : (t_1\ \alpha_1)^*\ (\text{i32}\ \alpha); l_1; \Gamma_1; \phi_1 \to (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2$.
  To do so, we must show that
  $$C' \vdash embed_{e^*}(e_1^*)^{\hat{t}_*} : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1, \neg(=\ \alpha(\text{i32}\ 0)) \to (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_3$$
  and
  $$C' \vdash embed_{e^*}(e_2^*)^{\hat{t}_*} : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1, (=\ \alpha(\text{i32}\ 0)) \to (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_4$$
  where $\Gamma_2 \vdash \phi_3 \rightsquigarrow \phi_2$, $\Gamma_2 \vdash \phi_4 \rightsquigarrow \phi_2$, and $C' = embed_C(C), \text{label}((t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2)$.
  We have $C, \text{label}(t_2^*) \vdash e_1^* : t_1^* \to t_2^*$ and $C, \text{label}(t_2^*) \vdash e_2^* : t_1^* \to t_2^*$ as they are premises of $C \vdash \textbf{if}\ \ (t_1^* \to t_2^*) e_1^*\ e_2^*\ \textbf{end} : t_1^* \to t_2^*$.
  Further, we know $\hat{t}^* = embed_C(C)_{\text{local}}$ as it is a premise of the lemma we are trying to prove, and therefore $C'_{\text{local}} = \hat{t}^*$
  Now we can invoke the inductive hypothesis on e_1^* and e_2^*, since $C' = embed_C(C, \text{label}(t_1^*))$, to get
  $$C' \vdash embed_{e^*}(e_1^*)^{\hat{t}_*} : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \to (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_3$$
  and
  $$C' \vdash embed_{e_2^*}(e_2^*)^{\hat{t}_*} : (t_1\ \alpha_1)^*; (\hat{t}\ \alpha_{l1})^*; \Gamma_1; \phi_1 \to (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_4$$
  The other two premises, $\Gamma_2 \vdash \phi_3 \rightsquigarrow \phi_2$, $\Gamma_2 \vdash \phi_4 \rightsquigarrow \phi_2$, follow as described for the **block** case.
- $C \vdash \textbf{br}\ i : t^*\ t_1^* \to t_2^*$
  We want to show that $embed_C(C) \vdash \text{br}\ i : (t\ \alpha)^*\ (t_1\ \alpha_1)^*; (\hat{t}\ \alpha_{l1})^*; \Gamma_1; \phi_1 \to ti_2; (\hat{t}\ \alpha_4)^*; \emptyset; \emptyset$
  To do so, we must show that $embed_C(C)_{\text{label}}(i) = (t_1\ \alpha_2)^*; (\hat{t}\ \alpha_{l2}); \phi_3$, where $\Gamma_1 \vdash \phi_1 \rightsquigarrow \phi_3[\alpha_2 \mapsto \alpha_1][\alpha_{l2} \mapsto \alpha_{l1}]$, for some α_2^* and α_{l2}^*.
  We know $C_{\text{label}}(i) = t_1^*$ because it is a premise of $C \vdash \textbf{br}\ i : t^*\ t_1^* \to t_2^*$.
  Then, $embed_C(C)_{\text{label}}(i) = (t_1\ \alpha_2)^*; (\hat{t}\ \alpha_{l2}); \emptyset$, by the definition of $embed_C$.
  The other premise, $\Gamma_1 \vdash \phi_1 \rightsquigarrow \emptyset[\alpha_2 \mapsto \alpha_1][\alpha_{l2} \mapsto \alpha_{l1}]$, follows as described for the **block** case.
- $C \vdash \textbf{br\_if}\ i : t_1^*\ \text{i32} \to t_2^*$
  We want to show that $embed_C(C) \vdash \text{br\_if}\ i : (t_1\ \alpha_1)^*\ (\text{i32}\ \alpha); (\hat{t}\ \alpha_3)^*; \Gamma_1; \phi_1 \to (t_1\ \alpha_1)^*; (\hat{t}\ \alpha_3); \Gamma_1; \phi_1, (=\ \alpha\ (\text{i32}\ 0))$
  To do so, we must show that $embed_C(C)_{\text{label}}(i) = (t_1\ \alpha_2)^*; (\hat{t}\ \alpha_{l2}); \phi_3$, where $\Gamma_1 \vdash \phi_1, \neg(=\ \alpha\ (\text{i32}\ 0)) \rightsquigarrow \phi_3[\alpha_2 \mapsto \alpha_1][\alpha_{l2} \mapsto \alpha_{l1}]$, for some α_2^* and α_{l2}^*.

We know $C_{\text{label}}(i) = t^*$ because it is a premise of $C \vdash$ **br_if** $i : t_1^*$ i32 $\rightarrow t_2^*$.
Then, $embed_C(C)_{\text{label}}(i) = (t_1\ \alpha_2)^*; (\hat{t}\ \alpha_{l2}); \emptyset$, by the definition of $embed_C$.
The other premise, $\Gamma_1 \vdash \phi_1, \neg(=\ \alpha\ (\text{i32}\ 0)) \rightsquigarrow \emptyset[\alpha_2 \mapsto \alpha_1][\alpha_{l2} \mapsto \alpha_{l1}]$, follows as described for the **block** case.

- $C \vdash$ **br_table** $i^+ : t_0^*\ t_1^*$ **i32** $\rightarrow t_2^*$
We want to show that $embed_C(C) \vdash$ br_table $i^+ : (t_0\ \alpha_0)^*\ (t_1\ \alpha_1)^*\ (\text{i32}\ \alpha); (\hat{t}\ \alpha_{l1})^*; \Gamma_1; \phi_1 \rightarrow \epsilon; \epsilon; \emptyset; \emptyset$
To do so, we must show that $embed_C(C)_{\text{label}}(i) = (t_1\ \alpha_2)^*; (\hat{t}\ \alpha_{l2}); \phi_3$, where $\Gamma_1 \vdash \phi_1 \rightsquigarrow \phi_3[\alpha_2 \mapsto \alpha_1][\alpha_{l2} \mapsto \alpha_{l1}]^+$, for some α_2^* and α_{l2}^*.
We know that $(C_{\text{label}}(i) = t_1^*)^+$ because it is a premise of $C \vdash$ **br_table** $i^+ : t_0^*\ t_1^*$ **i32** $\rightarrow t_2^*$.
Then, $embed_C(C)_{\text{label}}(i) = ((t_1\ \alpha_2)^*; (\hat{t}\ \alpha_{l2}); \emptyset)^+$, by the definition of $embed_C$.
The other premise, $(\Gamma_1 \vdash \phi_1 \rightsquigarrow \emptyset[\alpha_2 \mapsto \alpha_1][\alpha_{l2} \mapsto \alpha_{l1}])^+$, follows as described for the **block** case.
- $C \vdash$ **return** $: t^*\ t_1^* \rightarrow t_2^*$
We want to show that $embed_C(C) \vdash$ return$i : (t\ \alpha)^*\ (t_1\ \alpha_1)^*; (\hat{t}\ \alpha_{l1})^*; \Gamma_1; \phi_1 \rightarrow ti_2; (\hat{t}\ \alpha_4)^*; \emptyset; \emptyset$
To do so, we must show that $embed_C(C)_{\text{return}} = (t_1\ \alpha_2)^*; \phi_3$, where $\Gamma_1 \vdash \phi_1 \rightsquigarrow \phi_3[\alpha_2 \mapsto \alpha_1]$, for some α_2^*.
We know $C_{\text{return}} = t_1^*$ because it is a premise of $C \vdash$ **return**$i : t^*\ t_1^* \rightarrow t_2^*$.
Then, $embed_C(C)_{\text{return}} = (t_1\ \alpha_2)^*; ; \emptyset$, by the definition of $embed_C$.
The other premise, $\Gamma_1 \vdash \phi_1 \rightsquigarrow \emptyset[\alpha_2 \mapsto \alpha_1]$, follows as described for the **block** case.
- $C \vdash$ **call_indirect** $(t_1^* \rightarrow t_2^*) : t_1^* \rightarrow t_2^*$
We want to show that

$$\begin{aligned} embed_C(C) \vdash\ & \text{call\_indirect}\ ((t_1\ \alpha_3)^*; \emptyset \rightarrow (t_2\ \alpha_4)^*; \emptyset) \\ & : (t_1\ \alpha_1)^*; (\hat{t}\ \alpha_3)^*; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^*; (\hat{t}\ \alpha_3)^*; \Gamma_1, (t_2\ \alpha_4)^*; \phi_1 \end{aligned}$$

To do so, we must show that $embed_C(C)_{\text{table}} = (n, tfi^n)$, and that $\Gamma_1 \vdash \phi_1 \rightsquigarrow \emptyset[\alpha_1 \mapsto \alpha_2]$.
We know $C_{\text{table}} = n$, as it is a premise of $C \vdash$ **call_indirect** $(t_1^* \rightarrow t_2^*) : t_1^* \rightarrow t_2^*$.
Then, $embed_C(C)_{\text{table}} = (n, tfi^n)$, by definition of $embed_C$.
The other premise, $\Gamma_1 \vdash \phi_1 \rightsquigarrow \emptyset[\alpha_1 \mapsto \alpha_2$, follows as described for the **block** case.
- $C \vdash$ **get_local** $i : \epsilon \rightarrow t$
We want to show that $embed_C(C) \vdash$ get_local $i : \epsilon; (\hat{t}\ \alpha_1)^*; \Gamma_1; \phi_1 \rightarrow (t\ \alpha); (\hat{t}\ \alpha_1)^*; \Gamma_1; \phi_1$
To do so, we must show that $embed_C(C)_{\text{local}}(i) = t$, and that $(t\ \alpha) = ((\hat{t}\ \alpha_1)^*)(i)$.
$(t\ \alpha) = ((\hat{t}\ \alpha_1)^*)(i)$ is trivially correct by construction (we simply choose a t and α such that $(t\ \alpha) = ((\hat{t}\ \alpha_1)^*)(i)$.
Then, since $embed_C(C)_{\text{local}} = \hat{t}^*$ is a premise of this lemma, we have that $embed_C(C)_{\text{local}}(i) = t$.
- $C \vdash$ **set_local** $i : t \rightarrow \epsilon$
We want to show that $embed_C(C) \vdash$ set_local $i : (t\ \alpha); (\hat{t}\ \alpha_1)^*; \Gamma_1; \phi_1 \rightarrow \epsilon; (\hat{t}\ \alpha_2)^*; \Gamma_1; \phi_1$
To do so, we must show that $embed_C(C)_{\text{local}}(i) = t$, and that $(\hat{t}\ \alpha_2)^* = ((\hat{t}\ \alpha_1)^*)[i := (t\ \alpha)]$.
$(\hat{t}\ \alpha_2)^* = ((\hat{t}\ \alpha_1)^*)[i := (t\ \alpha)]$ is trivially correct by construction (we simply choose a $(\hat{t}\ \alpha_2)^*$ such that $(\hat{t}\ \alpha_2)^* = ((\hat{t}\ \alpha_1)^*)[i := (t\ \alpha)]$.
Then, since $embed_C(C)_{\text{local}} = \hat{t}^*$ is a premise of this lemma, we have that $embed_C(C)_{\text{local}}(i) = t$.
- $C \vdash$ **tee_local** $i : t \rightarrow t$
We want to show that $embed_C(C) \vdash$ set_local $i : (t\ \alpha); (\hat{t}\ \alpha_1)^*; \Gamma_1; \phi_1 \rightarrow (t\ \alpha); (\hat{t}\ \alpha_2)^*; \Gamma_1; \phi_1$
To do so, we must show that $embed_C(C)_{\text{local}}(i) = t$, and that $(\hat{t}\ \alpha_2)^* = ((\hat{t}\ \alpha_1)^*)[i := (t\ \alpha)]$.
$(\hat{t}\ \alpha_2)^* = ((\hat{t}\ \alpha_1)^*)[i := (t\ \alpha)]$ is trivially correct by construction (we simply choose a $(\hat{t}\ \alpha_2)^*$ such that $(\hat{t}\ \alpha_2)^* = ((\hat{t}\ \alpha_1)^*)[i := (t\ \alpha)]$.
Then, since $embed_C(C)_{\text{local}} = \hat{t}^*$ is a premise of this lemma, we have that $embed_C(C)_{\text{local}}(i) = t$.
- Case: $C \vdash$ **get_global** $i : \epsilon \rightarrow t$
We want to show that $embed_C(C) \vdash$ **get_global** $i : \epsilon; (\hat{t}\ \alpha)^*; \Gamma_1; \phi_1 \rightarrow (t\ \alpha_2); (\hat{t}\ \alpha)^*; \Gamma, (t\ \alpha_2); \phi_1$.
To do so, we must show that $embed_C(C)_{\text{global}}(i) = \text{mut}^?\ t$.
We know $C_{\text{global}}(i) = \text{mut}^?\ t$, as it is a premise of $C \vdash$ **get_global** $i : \epsilon \rightarrow t$.
Then, $embed_C(C)_{\text{global}}(i) = \text{mut}^?\ t$, by definition of $embed_C$.
- Case: $C \vdash$ **set_global** $i : t \rightarrow \epsilon$
We want to show that $embed_C(C) \vdash$ **set_global** $i : (t\ \alpha_1); (\hat{t}\ \alpha)^*; \Gamma_1; \phi_1 \rightarrow \epsilon; (\hat{t}\ \alpha)^*; \Gamma_1; \phi_1$.
To do so, we must show that $embed_C(C)_{\text{global}}(i) = \text{mut}^?\ t$.
We know $C_{\text{global}}(i) = \text{mut}^?\ t$, as it is a premise of $C \vdash$ **get_global** $i : \epsilon \rightarrow t$.
Then, $embed_C(C)_{\text{global}}(i) = \text{mut}^?\ t$, by definition of $embed_C$.
- Case: $C \vdash t$.**load** $(tp\_sx)^?\ align\ o$: **i32** $\rightarrow t$

We want to show that $embed_C(C) \vdash \textbf{get\_global}\ i : (\textbf{i32}\ ()\ \alpha);\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1 \rightarrow (t\ \alpha_2);\ (\hat{t}\ \alpha)^*;\ \Gamma, (t\ \alpha_2);\ \phi_1$.
To do so, we must show that $embed_C(C)_{\text{memory}} = n$ and $2^{align} \leq (|tp| <)^? t$.
We know $C_{\text{memory}} = n$ and $2^{align} \leq (|tp| <)^? t$ as they are premises of $C \vdash t.\textbf{load}\ (tp\_sx)^?\ align\ o : \textbf{i32} \rightarrow t$.
Then, $embed_C(C)_{\text{memory}} = n$, by definition of $embed_C$.

- Case: $C \vdash t.\textbf{store}\ tp^?\ align\ o : \textbf{i32}\ t \rightarrow \epsilon$
  We want to show that $embed_C(C) \vdash \textbf{set\_global}\ i : (\textbf{i32}\ \alpha_1)\ (t\ \alpha_2);\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1 \rightarrow \epsilon;\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1$.
  To do so, we must show that $embed_C(C)_{\text{memory}} = n$ and $2^{align} \leq (|tp| <)^? t$.
  We know $C_{\text{memory}} = n$ and $2^{align} \leq (|tp| <)^? t$ as they are premises of $C \vdash t.\textbf{store}\ tp^?\ align\ o : \textbf{i32}\ t \rightarrow \epsilon$.
  Then, $embed_C(C)_{\text{memory}} = n$, by definition of $embed_C$.
- Case: $C \vdash \textbf{current\_memory} : \epsilon \rightarrow \textbf{i32}$
  We want to show that $embed_C(C) \vdash \textbf{get\_global}\ i : \epsilon;\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1 \rightarrow (\textbf{i32}\ ()\ \alpha);\ (\hat{t}\ \alpha)^*;\ \Gamma, (\textbf{i32}\ ()\ \alpha);\ \phi_1$.
  To do so, we must show that $embed_C(C)_{\text{memory}} = n$.
  We know $C_{\text{memory}} = n$ as it is a premise of $C \vdash \textbf{current\_memory} : \epsilon \rightarrow \textbf{i32}$.
  Then, $embed_C(C)_{\text{memory}} = n$, by definition of $embed_C$.
- Case: $C \vdash \textbf{grow\_memory} : \textbf{i32} \rightarrow \textbf{i32}$
  We want to show that $embed_C(C) \vdash \textbf{get\_global}\ i : (\textbf{i32}\ \alpha_1);\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1 \rightarrow (\textbf{i32}\ ()\ \alpha_2);\ (\hat{t}\ \alpha)^*;\ \Gamma, (\textbf{i32}\ ()\ \alpha_2);\ \phi_1$.
  To do so, we must show that $embed_C(C)_{\text{memory}} = n$.
  We know $C_{\text{memory}} = n$ as it is a premise of $C \vdash \textbf{current\_memory} : \epsilon \rightarrow \textbf{i32}$.
  Then, $embed_C(C)_{\text{memory}} = n$, by definition of $embed_C$.
- Case: $C \vdash \epsilon : \epsilon \rightarrow \epsilon$
  Trivially, $embed_C(C) \vdash \epsilon : \epsilon;\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1 \rightarrow \epsilon;\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1$ by Rule Empty.
- Case: $C \vdash e_1^*\ e_2 : t_1^* \rightarrow t_3^*$
  We want to show that $embed_C(C) \vdash e_1^*\ e_2 : (t_1\ \alpha_1)^*;\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1 \rightarrow (t_3\ \alpha_3)^*;\ (\hat{t}\ \alpha_4)^*;\ \Gamma_3;\ \phi_3$.
  To do so, we must show that
  $$embed_C(C) \vdash embed_{e^*}(e_1^*)^{\hat{t}^*} : (t_1\ \alpha_1)^*;\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1 \rightarrow (t_2\ \alpha_2)^*;\ (\hat{t}\ \alpha_5)^*;\ \Gamma_2;\ \phi_2$$
  and
  $$embed_C(C) \vdash embed_{e^*}(e_2)^{\hat{t}^*} : (t_2\ \alpha_2)^*;\ (\hat{t}\ \alpha_5)^*;\ \Gamma_2;\ \phi_2 \rightarrow (t_3\ \alpha_3)^*;\ (\hat{t}\ \alpha_4)^*;\ \Gamma_3;\ \phi_3$$
  We have $C \vdash e_1^* : t_1^* \rightarrow t_2^*$, and $C \vdash e_2 : t_2^* \rightarrow t_3^*$, as they are premises of $C \vdash e_1^*\ e_2 : t_1^* \rightarrow t_3^*$.
  Then, by the inductive hypothesis, we have
  $$embed_C(C) \vdash embed_{e^*}(e_1^*)^{\hat{t}^*} : (t_1\ \alpha_1)^*;\ (\hat{t}\ \alpha)^*;\ \Gamma_1;\ \phi_1 \rightarrow (t_2\ \alpha_2)^*;\ (\hat{t}\ \alpha_5)^*;\ \Gamma_2;\ \phi_2$$
  and
  $$embed_C(C) \vdash embed_{e^*}(e_2)^{\hat{t}^*} : (t_2\ \alpha_2)^*;\ (\hat{t}\ \alpha_5)^*;\ \Gamma_2;\ \phi_2 \rightarrow (t_3\ \alpha_3)^*;\ (\hat{t}\ \alpha_4)^*;\ \Gamma_3;\ \phi_3$$
- Case: $C \vdash e^* : t^*\ t_1^* \rightarrow t^*\ t_2^*$
  We want to show that
  $$embed_C(C) \vdash e^* : (t\ \alpha)^*\ (t_1\ \alpha_1)^*;\ (\hat{t}\ \alpha_3)^*;\ \Gamma_1;\ \phi_1 \rightarrow (t\ \alpha)^*\ (t_2\ \alpha_2)^*;\ (\hat{t}\ \alpha_5)^*;\ \Gamma_2;\ \phi_2$$
  .
  To do so, we must show that
  $$embed_C(C) \vdash embed_{e^*}(e_1^*)^{\hat{t}^*} : (t_1\ \alpha_1)^*;\ (\hat{t}\ \alpha_3)^*;\ \Gamma_1;\ \phi_1 \rightarrow (t_2\ \alpha_2)^*;\ (\hat{t}\ \alpha_5)^*;\ \Gamma_2;\ \phi_2$$
  We have $C \vdash e^* : t_1^* \rightarrow t_2^*$, as it is a premis of $C \vdash e^* : t^*\ t_1^* \rightarrow t^*\ t_2^*$.
  Then, by the inductive hypothesis, we have
  $$embed_C(C) \vdash embed_{e^*}(e_1^*)^{\hat{t}^*} : (t_1\ \alpha_1)^*;\ (\hat{t}\ \alpha_3)^*;\ \Gamma_1;\ \phi_1 \rightarrow (t_2\ \alpha_2)^*;\ (\hat{t}\ \alpha_5)^*;\ \Gamma_2;\ \phi_2$$

□

These are not the only differences in the surface syntax between Wasm and Wasm-precheck: we also introduced four new instructions (the ✓-tagged instructions). The definition of embedding we have introduced has been entirely syntactic, but that will not work for replacing non-✓-tagged instructions with ✓-tagged versions during embedding since we must be able to ensure that stronger guarantees are met. Thus, we do not have an explicit embedding that provides ✓-tagged instructions, though we do posit the existence of a trivial embedding that would provide ✓-tagged instructions. One could, for example, check at every div, call_indirect , load , and store whether the ✓-tagged version of the instruction is well typed, and only if it is well typed replace the instruction with the ✓-tagged version. However, a more sophisticated static analysis could provide more precise type annotations and therefore potentially allow even more check eliminations.

B.2 Erasing Wasm-precheck to Wasm

We provide an erasure function for Wasm-precheck that transforms Wasm-precheck programs into Wasm programs by discarding the extra information from the Wasm-precheck type system and replacing ✓-tagged instructions with their non-tagged counterparts. Erasure is used in the type safety proof. Therefore, we define erasure not just for the surface syntax, like we did for embedding, but also for typing constructs (such as the module type context), administrative instructions, and runtime data structures (such as the store). We show that erasing a well-typed Wasm-precheck (run time) program produces a well-typed Wasm (run time) program.

As with the presentation of the embedding, we typeset Wasm-precheck instructions in a blue sans serif font and Wasm instruction in a **bold red font**.

Erasing Surface Syntax. As with embedding, we start by defining erasure with the pinnacle syntactic object: the module. Defining and erasure for modules relies on the erasure of tables and functions, and therefore instructions and indexed function types. Keep in mind that the proofs of sound erasure work over the typing rules for these constructs, so we also define erasure of module type contexts since they are used in the typing rules for modules.

Erasing a module erases all of the functions f^* and the table $tab^?$. The globals $glob^*$ and optional memory $mem^?$ both have the same syntax in Wasm-precheck as in Wasm.

Definition 6. $\boxed{erase_{module}(module) = \mathbf{module}}$

$$\begin{array}{rcl} erase_{module}(\mathsf{module}\ f^*\ glob^*\ tab^?\ mem^?) & = & \mathbf{module}\ erase_f(f)^* \\ & & erase_g(glob)^* \\ & & erase_t(tab)^? \\ & & mem^? \end{array}$$

Erasing a table definition $\mathsf{table}\ n\ i^n$ does nothing, since a table definition has the same syntax in Wasm-precheck and in Wasm. However, erasing an imported table declaration $\mathsf{table}\ (n, tfi^n)\ im$ must get rid of the indexed function types tfi^n. We do not use or care about the exports, since they are unchanged and only used for linking, so we omit them.

Definition 7. $\boxed{erase_t(tab) = \mathbf{tab}}$

$$\begin{array}{rcl} (ex^*)\ \mathsf{table}\ n\ i^n & = & (ex^*)\ \mathbf{table}\ n\ i^n \\ (ex^*)\ \mathsf{table}\ (n, tfi^n)\ im & = & (ex^*)\ \mathbf{table}\ n\ im \end{array}$$

Erasing a global definition erases the instruction sequence used to initialize the global variable. However, erasing an imported global declaration does nothing. We do not use or care about the exports, since they are unchanged and only used for linking, so we omit them.

Definition 8. $\boxed{erase_g(glob) = \mathbf{glob}}$

$$\begin{array}{rcl} (ex^*)\ \mathsf{global}\ tg\ e^* & = & (ex^*)\ \mathbf{global}\ tg\ erase_{e^*}(e^*) \\ (ex^*)\ \mathsf{global}\ tg\ im & = & (ex^*)\ \mathbf{global}\ tg\ im \end{array}$$

To erase a function definition f, we erase both the type declaration $ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ and the body e^*. We can also erase an imported function by erasing the declared type tfi.

Definition 9. $\boxed{erase_f(f) = \mathbf{f}}$

$$\begin{array}{rcl} erase(\mathsf{func}\ ((t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^*; l_2; \Gamma_1; \phi_2)\ \mathsf{local}\ t^*\ e^*) & = & \mathbf{func}\ (t_1^* \rightarrow t_2^*)\ \mathbf{local}\ t^*\ erase_{e^*}(e^*) \\ erase_f(\mathsf{func}\ ((t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^*; l_2; \Gamma_1; \phi_2)\ im) & = & \mathbf{func}\ (t_1^* \rightarrow t_2^*)\ im \end{array}$$

We show that erasing a Wasm-precheck function f, that is well typed under a module type context C, produces a Wasm function $erase_f(f)$ that is well typed under the erased module type context $erase_C(C)$. This is useful not just for erasing the surface syntax, but also because functions are a part of closures which are used at run time (as part of module instances and tables). The proof relies on Lemma Sound-Static-Typing-Erasure to prove that the body is still well typed. The case of imported functions is trivial because an imported function is well typed under absolutely any context and with any function type, so it is omitted.

Lemma 4. Sound-Function-Typing-Erasure
If $C \vdash \text{func}\ \ (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2); l_2; \Gamma_2; \phi_2)\ \text{local}\ t^*\ e^* : ex^*\ (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2); l_2; \Gamma_2; \phi_2$
then $erase_C(C)\ \vdash \textbf{func}\ (t_1^* \rightarrow t_2^*)\ \text{local}\ t^*\ erase_{e^*}(e^*)) : ex^*\ t_1^* \rightarrow t_2^*$

Proof. We must show that $erase_C(C), \text{local}(t_1^*\ t^*), \text{label}(t_2^*), \text{return}(t_2^*) \vdash erase_{e^*}(e^*) : t_1^* \rightarrow t_2^*$ since it is the only premise of typechecking a function definition in Wasm.

We know the following because it is a premise of Rule Func which we have assumed to hold.

$$C, \text{local}(t_1^*\ t^*), \text{label}((t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2), \text{return}((t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2) \vdash e^* : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2); l_2; \Gamma_2; \phi_2$$

Then, by Lemma Sound-Static-Typing-Erasure, we have that

$$erase_C(C), \text{local}(t_1^*\ t^*), \text{label}(t_2^*), \text{return}(t_2^*) \vdash erase_{e^*}(e^*) : t_1^* \rightarrow t_2^*$$

□

Erasing an indexed type function keeps only the primitive Wasm types (t_1^* and t_2^*) from the indexed types representing the stack ($(t_1\ \alpha_1)^*$ and $(t_2\ \alpha_2)^*$), and discards everything else.

Definition 10. $\boxed{erase_{tfi}(tfi) = \textbf{tf}}$
$erase_{tfi}((t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^*; l_2; \Gamma_2\phi_2) = t_1^* \rightarrow t_2^*$

Erasing instructions involves erasing the indexed function types for indirect function calls, which includes it as part of their syntax. We must also remove the ✓tag from ✓-tagged instructions to turn them into instructions that exist in Wasm.

Definition 11. $\boxed{erase_{e^*}(e) = \textbf{e}}$

$$\begin{array}{rcl}
erase_{e^*}(\text{block}\ (t_1^n \rightarrow t_2^n)\ e^*\ \text{end}) & = & \textbf{block}\ \ (t_1^n \rightarrow t_2^n)\ erase[e^*]e^*\ \ \textbf{end} \\
erase_{e^*}(\text{loop}\ (t_1^n \rightarrow t_2^n)\ e^*\ \text{end}) & = & \textbf{loop}\ \ (t_1^n \rightarrow t_2^n)\ erase[e^*]e^*\ \ \textbf{end} \\
erase_{e^*}(\text{if}\ (t_1^n \rightarrow t_2^n)\ e_1^*\ e_2^*\ \text{end}) & = & \textbf{if}\ \ (t_1^n \rightarrow t_2^n)\ erase[e^*]e_1^*\ erase[e^*]e_2^*\ \ \textbf{end} \\
erase_{e^*}(\text{label}_n\ \{e_0^*\}\ e^*\ \text{end}) & = & \textbf{label}_n\ \{erase_{e^*}(e_0^*)\} \\
 & & \qquad erase_{e^*}(e^*) \\
 & & \ \textbf{end} \\
erase_{e^*}(\text{local}_n\ \{i; v^*\}\ e^*\ \text{end}) & = & \textbf{local}_n\ \{i; v^*\} \\
 & & \qquad erase_{e^*}(e^*) \\
 & & \ \textbf{end} \\
erase_{e^*}(\text{call_indirect}\ tfi) & = & \textbf{call_indirect}\ erase_{tfi}(tfi) \\
erase_{e^*}(\text{call_indirect}\checkmark\ tfi) & = & t.\textbf{call_indirect}\ erase_{tfi}(tfi) \\
erase_{e^*}(t.\text{div}\checkmark\ ) & = & t.\textbf{div} \\
erase_{e^*}(t.\text{store}\checkmark\ tp^?\ align\ o) & = & t.\textbf{store}\ tp^?\ align\ o \\
erase_{e^*}(t.\text{load}\checkmark\ (tp_sx)^?\ align\ o) & = & t.\textbf{load}\ (tp_sx)^?\ align\ o \\
erase_{e^*}(e) & = & e, \text{otherwise} \\
erase_{e^*}(e^*) & = & erase_{e^*}(e)^*
\end{array}$$

Erasing Typing Constructs. Here, we prove that erasing a Wasm-precheck static typing derivation is sound with respect to Wasm's type system. This means that erasure on the Wasm-precheck static typing judgment is sound with respect to Wasm's type system. Specifically, a Wasm-precheck instruction sequence e^*, that is well typed under a module type context C, produces a Wasm instruction sequence $e'^* = erase_{e^*}(e^*)$ that is well typed under the erased module type context $C' = erase_C(C)$.

Lemma 5. Sound-Static-Typing-Erasure
If $C \vdash e^* : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2$,
then $erase_C(C) \vdash erase_{e^*}(e^*) : t_1^* \rightarrow t_2^*$

Proof. We proceed by induction over typing derivations. Most proof cases are omitted as they are simple, but we provide a few to give an idea of what the proofs look like. Intuitively, we want to show that erasing the typing derivation produces a valid Wasm typing derivation.

For most of the cases, the sequence of instructions e^* contains only a single instruction e_2, so we elide the step of turning $erase_{e^*}(e^*)$ into $erase_{e^*}(e_2)$.

- Case: $C \vdash t.binop : (t\ \alpha_1)\ (t\ \alpha_2); l_1; \Gamma_1; \phi_1 \rightarrow (t\ \alpha_3); l_1; \Gamma_1, (t\ \alpha_3); \phi_1, (= a_3\ (t.binop\ a_1\ a_2))$
  We want to show that $erase_C(C)\ \vdash erase_{e^*}(t.binop) : t\ t \rightarrow t$
  Then, by the definition of $erase_e$, we want to show that $erase_C(C) \vdash t.binop : t\ t \rightarrow t$ is valid in Wasm.
  Trivially, we have $erase_C(C) \vdash t.binop : t\ t \rightarrow t$.
- Case: $C \vdash \mathsf{block}\ (t_1^n \rightarrow t_2^n)e^*\ \mathsf{end} : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2$
  We want to show that $erase_C(C) \vdash \textbf{block}\ \ (t_1^* \rightarrow t_2^*)\ erase_{e^*}(e^*)\ \textbf{end} : (t_1^* \rightarrow t_2^*)$.
  This proof is slightly more involved, since the derivation for this rule includes a premise that

  $$C, \mathrm{label}((t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2) \vdash e^* : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2$$

  By the inductive hypothesis for the well-typedness of e^*, we have that $erase_C(C, \mathrm{label}((t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2)) \vdash erase_{e^*}(e^*) : t_1^* \rightarrow t_2^*$
  Then we have $erase_C(C), \mathrm{label}(t_2^*) \vdash erase_{e^*}(e^*) : t_1^* \rightarrow t_2^*$ by definition of $erase_C$.
- Case: $C \vdash \mathsf{br}\ \ i : (t_1\ \alpha_1)^*\ (t\ a)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2$
  We want to show that $erase_C(C) \vdash \textbf{br}\ i : t_1^*\ t^* \rightarrow t_2^*$
  We have to reason about $erase_C(C)$ because the typing judgment relies on the label stack from C.
  From $C\ \vdash\ \mathsf{br}\ \ i\ :\ (t_1\ \ \alpha_1)^*\ \ (t\ \ a)^*; l_1; \Gamma_1; \phi_1\ \rightarrow\ (t_2\ \ \alpha_2)^*; l_2; \Gamma_2; \phi_2$, we have that $C_{\mathrm{label}}(i)\ =\ (t\ \ a)^*; l_1; \Gamma_3; \phi_3$, where $\Gamma_1 \vdash \phi_1 \Rightarrow \phi_3$, since it is a premise.
  Then $erase_C(C)_{\mathrm{label}}(i) = t^*$, by the definition of $erase_C$.
  Thus, $erase_C(C) \vdash \textbf{br}\ i : t_1^* \rightarrow t_2^*$.
- Case: $C \vdash \mathsf{call}\ \ i : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2$
  We want to show that $erase_C(C) \vdash \textbf{call}\ i : t_1^* \rightarrow t_2^*$
  We again have to reason about $erase_C(C)$ because the typing judgment relies on the function type from C.
  From $C \vdash \mathsf{call}\ \ i : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2$, we have that $C_{\mathrm{func}}(i) = (t_1\ \alpha_1)^*; l_1; \Gamma_3; \phi_3 \rightarrow (t_2\ \alpha_2)^*; l_4; \Gamma_4; \phi_4$.
  Then, $erase_C(C)_{\mathrm{func}}(i) = t_1^* \rightarrow t_2^*$, by the definition of $erase_C$.
  Thus, $erase_C(C) \vdash \textbf{call}\ i : t_1^* \rightarrow t_2^*$.
- Case: $C \vdash \mathsf{set\_local}\ \ i : (t\ a); l_1; \Gamma_1; \phi_1 \rightarrow \epsilon; l_1[i := a]; \Gamma_1; \phi_1$
  We want to show that $erase_C(C) \vdash \textbf{set\_local}\ i : t \rightarrow \epsilon$
  We again have to reason about $erase_C(C)$ because the typing judgment relies on the local variable types from C.
  From $C \vdash \mathsf{set\_local}\ \ i : (t\ a); l_1; \Gamma_1; \phi_1 \rightarrow \epsilon; l_1[i := a]; \Gamma_1; \phi_1$, we have that $C_{\mathrm{local}}(i) = t$, since it is a premise.
  Then, we have that $erase_C(C)_{\mathrm{local}}(i) = t$, by the definition of $erase_C$.
  Thus, $erase_C(C) \vdash \textbf{set\_local}\ i : t \rightarrow \epsilon$.
- Case: $C \vdash e_1^*\ e_2 : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2); l_2; \Gamma_2; \phi_2$
  We want to show that $erase_C(C) \vdash erase_{()}[e^*](e_1^*\ e_2) : t_1^* \rightarrow t_2^*$
  For this typing rule, we must invoke the inductive hypothesis twice: one on the sequence e_1^* and once on the instruction e_2. Then we must show that we can compose the erased subsequences together to get a well-typed sequence.
  We know that $C \vdash e_1^* : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_3\ \alpha_3); l_3; \Gamma_3; \phi_3$ and that $C \vdash e_2 : (t_3\ \alpha_3)^*; l_3; \Gamma_3; \phi_3 \rightarrow (t_2\ \alpha_2); l_2; \Gamma_2; \phi_2$ because they are premises of Rule Composition which we have assumed to hold.
  $erase_C(C) \vdash erase_{e^*}(e_1^*) : t_1^* \rightarrow t_3^*$ by the inductive hypothesis on e_1^*, and $erase_C(C) \vdash erase_{e^*}(e_2) : t_3^* \rightarrow t_2^*$, by the inductive hypothesis on e_2.
  Thus, $erase_C(C)\ \vdash\ erase_{e^*}(e_1^*)\ erase_{e^*}(e_2) : t_1^* \rightarrow t_2^*$.

□

To erase a module type context, we must erase all of the function types tfi^*, the table type (n, tfi_2^*) if one is present, and the postconditions in the label stack $((t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1)^*$ and the return stack $((t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2)^?$. We erase postconditions the same way we erase the postconditions of indexed function types: by keeping only the primitive Wasm types (t_1^* in the case of a label postcondition). Recall that erasing a table type means discarding the type information about the functions in the table.

Definition 12. $\boxed{erase_C(C) = \mathrm{C}}$

$$\begin{aligned}
erase_C(\{&\mathrm{func}\ tfi^*,\ \mathrm{global}\ tg^*, &&= \{\mathrm{func}\ erase_{tfi}(tfi)^*,\\
&\mathrm{table}\ (n, tfi_2^*)^?, && \quad \mathrm{global}\ tg^*,\ \mathrm{table}\ n^?,\\
&\mathrm{memory}\ m^?,\ \mathrm{local}\ t^*, && \quad \mathrm{memory}\ \ m^?, \mathrm{local}\ t^*,\\
&\mathrm{label}\ ((t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1)^*, && \quad \mathrm{label}\ (t_1^*)^*,\ \mathrm{return}\ (t_2^*)^?\}\\
&\mathrm{return}\ ((t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2)^?\})
\end{aligned}$$

Erasing Programs. Defining and erasure for programs relies on the erasure of the store and its various structures, as well as the erasure of instructions which we have already defined and proven. Remember, the proofs of sound erasure work over the typing rules for these constructs, so we have to show sound erasure for all of the various typing rules that Rule Program relies on.

Now we will show that erasing a well-typed Wasm-precheck program in reduction form (s; v^*; e^*) is sound with respect to Wasm's type system. Intuitively, we accomplish this by showing that erasing typing derivations of the Rule Program judgment produce valid Wasm typing derivations, like in Lemma Sound-Static-Typing-Erasure. To do so, we must show sound erasure for Rule Code, as it is a premise of Rule Program; this is done by Lemma Sound-Code-Typing-Erasure. Erasing programs involves erasing many run-time data structures, including the store s and store context S, as well as modules instances $inst^*$ in s, and closures cl in modules instances and the optional table. Erasing the store is shown to be safe by Lemma Sound-Store-Erasure.

Theorem 2. Sound-Program-Typing-Erasure
If $\vdash_i s; v^*\ e^* : (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2$,
then $\vdash_i erase_s(s); v^*; erase_{e^*}(e^*) : t_2^*$

Proof. We must show that $\vdash erase_s(s) : S$ for some Wasm store context S, and that $erase_S(S);\ \vdash_i erase_{e^*}(e^*) : \epsilon \rightarrow t_2^*$.

We have $\vdash s : S'$, where S' is a Wasm-precheck store context, because it is a premise of Rule Program which we have assumed to hold.

Then, $\vdash erase_s(s) : erase_S(S')$ by Lemma Sound-Store-Erasure.

We also have $S;\ \vdash_i v^*; e^* : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2$ as a premise of Rule Program.

In which case we have $erase_S(S);\ \vdash_i v^*; erase_{e^*}(e^*) : t_1^* \rightarrow t_2^*$ by Lemma Sound-Code-Typing-Erasure. □

The sound erasure of Rule Code is used in the sound erasure of programs. Thus, we only prove the case when the optional return stack $((t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2)^?$ is empty because we are only proving this to use later in Rule Program, which never uses the return stack. Lemma Sound-Code-Typing-Erasure relies on Lemma Sound-Admin-Typing-Erasure, which shows a similar property, but for the administrative typing judgment $S; C \vdash e^* : tfi$.

Lemma 6. Sound-Code-Typing-Erasure
If $S;\ \vdash_i v^*; e^* : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2$,
then $erase_S(S);\ \vdash_i v^*; erase_{e^*}(e^*) : epsilon \rightarrow t_2^*$

Proof. We must show that $(\vdash v : t_v)^*$ and $erase_S(S); erase_C(S_{\text{inst}}(i)), \text{local } t_v^* \vdash erase_{e^*}(e^*) : epsilon \rightarrow t_2^*$

We have $(\vdash v : t_v)^*$ trivially since it is a premise of Rule Code which we have assumed to hold.

We also have $S; S_{\text{inst}}(i), \text{local } t_v^* \vdash e^* : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, for some Γ_1,ϕ_1, and ϕ_2, by inversion on the Code judgment.

Then, by Lemma Sound-Admin-Typing-Erasure, we have that $erase_S(S); erase_C(()S_{\text{inst}}(i), \text{local } t_v^*) \vdash erase_{e^*}(e^*) : \epsilon \rightarrow t_2^*$ □

Lemma Sound-Admin-Typing-Erasure builds on Lemma Sound-Static-Typing-Erasure by adding the store context S and typing rules for administrative instructions. It is necessary to add these rules and extra information because they are used for typechecking programs. Note that while we add S to the judgment used in Lemma Sound-Static-Typing-Erasure to get $S; C \vdash e^*; tfi$, none of the rules previously proven reference S in any way, they simply match any store context.

Lemma 7. Sound-Admin-Typing-Erasure
If $S; C \vdash e^* : (t_1\ \alpha_1)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^*; l_2; \Gamma_2; \phi_2$,
then $erase_S(S); erase_C(C) \vdash erase_{e^*}(e^*) : t_1^* \rightarrow t_2^*$

Proof. We proceed by induction over typing rules. In addition to the prior cases from Lemma Sound-Static-Typing-Erasure, which trivially still hold since the value of S does not matter to those rules, we add proves for a few administrative typing rules, which may refer to S. Again, several proof cases are omitted as they are simple.

- $S; C \vdash \text{label}_n\{e_0^*\}\ e^*\ \text{end} : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^n; l_2; \Gamma_2; \phi_2$
  We must show that $erase_S(S); erase_C(C) \vdash erase_{e^*}(e_0^*) : t_3^* \rightarrow t_2^n$ and $erase_S(S); erase_C(C, \text{label}((t_3\ \alpha_3)^*; l_3; \Gamma_3; \phi_3)) \vdash erase_{e^*}(e^*) : \epsilon \rightarrow t_3^*$ as they are the premises of typechecking a label block in Wasm.
  We have that $S; C \vdash e_0^* : (t_3\ \alpha_3)^*; l_3; \Gamma_3; \phi_3 \rightarrow (t_2\ \alpha_2)^n; l_2; \Gamma_2; \phi_2$ since it is a premise of Rule Label which we have assumed to hold.
  Then, by the inductive hypothesis for the stored instructions e_0^* being well typed, we have that $erase_S(S); erase_C(C) \vdash erase_{e^*}(e_0^*) : t_3^* \rightarrow t_2^n$

$S; C, \text{label}((t_3\ \alpha_3)^*; l_3; \Gamma_3; \phi_3)) \vdash e^* : \epsilon; l_1; \Gamma_1; \phi_1 \longrightarrow (t_3\ \alpha_3)^*; l_3; \Gamma_3; \phi_3$, because it is a premise of Rule Label which we have assumed to hold.
By the inductive hypothesis for the body e^* being well typed, we have that

$$erase_S(S); erase_C(C, \text{label}((t_3\ \alpha_3)^*; l_3; \Gamma_3; \phi_3)) \vdash erase_{e^*}(e^*) : \epsilon \longrightarrow t_3^*$$

- $S; C \vdash \text{local}_n\{i; v^*\}\ e^*\ \text{end} : \epsilon; l_1; \Gamma_1; \phi_1 \longrightarrow (t_2\ \alpha_2)^n; l_2; \Gamma_2; \phi_2$
  The premise of this rule relies on Rule Code with a non-empty return postcondition, which we have not yet proved sound erasure for, so instead we must derive Rule Code for the erased program.
  Thus, we must show that $(\vdash v : t_v)^*$ and $erase_S(S); erase_C(S_{\text{inst}}(i), \text{local}\ t_v^*,\ \text{return}((t_2\ \alpha_2)^n; l_2; \Gamma_2; \phi_2))) \vdash erase_{e^*}(e^*) : \epsilon \longrightarrow t_2^n$
  We have Rule Code as a premise of Rule Local, which we have assumed to hold.
  Therefore, $(\vdash v : t_v)^*$ trivially, since neither values nor primitive types are affected by erasure.
  We also know that $S; S_{\text{inst}}(i), \text{local}\ t_v^*,\ \text{return}((t_2\ \alpha_2)^n; l_2; \Gamma_2; \phi_2)) \vdash e^*\ \text{end} : \epsilon; l_1; \Gamma_1; \phi_1 \longrightarrow (t_2\ \alpha_2)^n; l_2; \Gamma_2; \phi_2$
  Therefore, we have that $erase_S(S); erase_C(S_{\text{inst}}(i), \text{local}\ t_v^*,\ \text{return}((t_2\ \alpha_2)^n; l_2; \Gamma_2; \phi_2)) \vdash erase_{e^*}(e^*) : \epsilon \longrightarrow t_2^n$, by the inductive hypothesis of the well-typedness of e^*

□

We must prove safe erasure about the store s for use in Theorem 2. First though, we must define erasure for s. Erasing the store erases all of the modules instances and closures in the tables inside the store. Note that in the definition we have expanded the definition of a table instance to $(\{\text{inst}\ i,\ \text{func}\ f\}^*)^*$ for extra clarity.

Definition 13. $\boxed{erase_s(s) = \mathbf{s}}$

$$\begin{array}{lcl} erase_s(\{\text{inst}\ inst^*, & = & \{\text{inst}\ erase_{inst}(inst)^*, \\ \quad \text{tab}\ (\{\text{inst}\ i,\ \text{func}\ f\}^*)^*, & & \quad \text{tab}\ (\{\text{inst}\ i,\ \text{func}\ erase_f(f)\}^*)^*, \\ \quad \text{mem}\ meminst^*\}) & & \quad \text{mem}\ meminst^*\} \end{array}$$

Lemma Sound-Store-Erasure proves that erasing a well-typed Wasm-precheck store results in a well-typed Wasm store.

Lemma 8. Sound-Store-Erasure
If $\vdash s : S$, then $\vdash erase_s(s) : erase_S(S)$

Proof. Note that
$s \quad = \quad \{\text{inst}\ inst^*, \text{tab}\ (\{\text{inst}\ i,\ \text{func}\ f\}^*)^*,\ \text{mem}\ meminst^*\}$ and
$S \quad = \quad \{\text{inst}\ C^*,\ \text{tab}\ (n, tfi^*)^*,\ \text{mem}\ m^*\}$
Then,

$$erase_S(S) = \{\text{func}\ erase_{tfi}(tfi)^*,\ \text{global}\ tg^*,\ \text{table}\ n,\ \text{memory}\ n^?, ...\}$$

by the definition of $erase_C$.
Then, we must prove the following properties, as they are the premises of $\vdash erase_s(s) : erase_S(S)$:

1. $(erase_S(S) \vdash erase_{inst}(inst) : erase_C(C))^*$
   We have that $(S \vdash inst : C)^*$, because it is a premise of Rule Store that we have assumed to hold.
   Then, we have $erase_S(S) \vdash erase_{inst}(inst) : erase_C(C))^*$ by Lemma Sound-Instance-Typing-Erasure.
2. $((erase_S(S) \vdash \{\text{inst}\ i,\ \text{func}\ erase_f(f)\} : erase_{tfi}(tfi))^*)^*$
   We have $((S \vdash cl : tfi)^*)^*)$, because it is a premise of Rule Store that we have assumed to hold.
   Then, $((erase_S(S) \vdash \{\text{inst}\ i,\ \text{func}\ erase_f(f)\} : erase_{tfi}(tfi))^*)^*$ by Lemma Sound-Closure-Typing-Erasure
3. $(n \leq |\{\text{inst}\ i,\ \text{func}\ erase_f(f)\}|)^*$
   We have that $(n \leq |\{\text{inst}\ i,\ \text{func}\ f\}|)^*$, because it is a premise of Rule Store that we have assumed to hold.
   Because the number of closures is not affected by erasure, we can then say that $(n \leq |\{\text{inst}\ i,\ \text{func}\ erase_f(f)\}|)^*$
4. $(m \leq |b^*|)^*$
   Trivially, we have that $(m \leq |b^*|)^*$, because it is a premise of Rule Store that we have assumed to hold.

□

Erasing a module instance erases all of the functions f in the closures (which we have expanded inline to $\{\text{inst}\ i,\ \text{func}\ f\}$) within the module instance.

Definition 14. $\boxed{erase_{inst}(inst) = \textbf{inst}}$

$$\begin{array}{lll} erase_{inst}(\{\text{func } \{\text{inst } i,\ \text{func } f\}^*, & = & \{\text{func } \{\text{inst } i,\ \text{func } erase_f(f)\}^*, \\ \quad \text{global } v^*,\ \text{table } i^?, & & \text{global } v^*,\ \text{table } i^?, \\ \quad \text{memory } j^?\}) & & \text{memory } j^?\} \end{array}$$

We now prove that if a Wasm-precheck module instance $inst$ has type C under the store context S, then the erased Wasm instance $erase_{inst}(inst)$ will have the erased type $erase_c(C)$ under the erased store context $erase_S(S)$. To do this, we rely on the above lemmas to safely erase index information from function declarations and table declarations (globals and memory have the same type information in both Wasm-precheck and Wasm). This will be useful for proving that a well-typed Wasm-precheck store s erases to a well-typed Wasm store $erase_s(s)$ since stores contain many instances. To do this, we rely on the above lemmas to safely erase index type information about closures and tables (globals and memory have the same type information in both Wasm-precheck and Wasm).

Lemma 9. Sound-Instance-Typing-Erasure If $S \vdash inst : C$, then $erase_S(S) \vdash erase_{inst}(inst) : erase_C(C)$

Proof. Note that

$$\begin{array}{rcl} S & = & \{\text{inst } C^*,\ \text{tab } (n, tfi^*)^*,\ \text{mem } m^*\} \\ inst & = & \{\text{func } \{\text{inst } i,\ \text{func } f\}^*, \text{global } v^*,\ \text{table } i^?, \text{memory } j^?\} \\ C & = & \{\text{func } tfi^*,\ \text{global } tg^*,\ \text{table } (n, tfi_2)^?,\ \text{memory } n^?, \dots\} \end{array}$$

Then,

$$erase_C(C) = \{\text{func } erase_{tfi}(tfi)^*,\ \text{global } tg^*,\ \text{table } n,\ \text{memory } n^?, ...\}$$

by the definition of $erase_C$.

Then, we must prove the following properties, as they are the premises of $erase_S(S) \vdash erase_{inst}(inst) : erase_C(C)$:

1. $erase_S(S) \vdash \{\text{inst } i,\ \text{func } f\} : erase_{tfi}(tfi))^*$
   We have that $S \vdash \{\text{inst } i,\ \text{func } f\} : tfi$, because it is a premise of Rule Instance that we have assumed to hold.
   Then, we have $erase_S(S) \vdash \{\text{inst } i,\ \text{func } f\} : erase_{tfi}(tfi))^*$ by Lemma Sound-Closure-Typing-Erasure.
2. $(\vdash v : tg)^*$
   Trivially, this is a premise of $S \vdash inst : C$ and is not affected by erasure, so therefore it holds.
3. $erase_S(S)_{tab}(i) = n$
   $erase_S(S)_{tab}(i) = n$ by definition of $erase_S$.
   Therefore, $erase_S(S)_{tab}(i) = n$.
4. $erase_S(S)_{mem}(i) = n^?$
   Trivially, this is a premise of $S \vdash inst : C$ and is not affected by erasure, so therefore it holds.

□

We erase store contexts by erasing all of the module type instances C^* and table types $(n, tfi^*)^*$ within.

Definition 15. $\boxed{erase_S(S) = \text{S}}$

$$\begin{array}{lll} erase_S(\{\text{inst } C^*, & = & \{\text{inst } erase_c(C)^*, \\ \quad \text{tab } (n, tfi^*)^*,\ \text{mem } m^*\}) & & \text{tab } n^*, \text{mem } m^*\} \end{array}$$

Finally, we prove that if a Wasm-precheck closure is well typed than the erased closure is well typed.

Lemma 10. Sound-Closure-Typing-Erasure
If $S \vdash \{\text{inst } i,\ \text{func } f\}^* : tfi$,
then $erase_S(S) \vdash \{\text{inst } i,\ \text{func } erase_f(f)\}^* : erase_{tfi}(tfi)$

Proof. We must show that $erase_S(S)_{\text{inst}}(i) \vdash erase_f(f) : erase_{tfi}(tfi)$.
We have $S_{\text{inst}}(i) \vdash f : tfi$ since it is a premise of Rule Closure which we have assumed to hold.
Then, $erase_S(S)_{\text{inst}}(i) \vdash erase_f(f) : erase_{tfi}(tfi)$ by Lemma Sound-Function-Typing-Erasure. □

C Type Safety Proof

Before we jump into the type safety proof, we will first introduce several of the reasoning principles.

Lemma Well-Formedness reasons about the index variables in an instruction type. It says that every index variable used to represent a value on the stack or the value of a local variable must be present in the index environment Γ, and that the constraint set ϕ cannot refer to variables not in Γ.

Lemma 11. Well-Formedness
If $C \vdash e^* : ti_1^*;\, ti_3^*;\, \Gamma_1;\, \phi_1 \rightarrow ti_2^*;\, ti_4^*;\, \Gamma_2;\, \phi_2$,
(or $S;\, C \vdash e^* : ti_1^*;\, ti_3^*;\, \Gamma_1;\, \phi_1 \rightarrow ti_2^*;\, ti_4^*;\, \Gamma_2;\, \phi_2$)
then $(ti_1 \in \Gamma_1)^*$, $(ti_3 \in \Gamma_1)^*$, and $domain(\phi_1) \subset \Gamma_1$,
and $(ti_2 \in \Gamma_2)^*$, $(ti_4 \in \Gamma_4)^*$, and $domain(\phi_2) \subset \Gamma_2$,
and $\Gamma_2 \subseteq \Gamma_1$

Proof. Informally, no typing rule can produce a malformed instruction type, so all instruction types with a valid derivation must be well formed. Every rule that introduces fresh index variables in the stack, local index store, or constraint set ϕ also declares that index variable into Γ, unless the variable is already on the stack/in the local environment, in which case we can assume it is already in Γ. Similarly, functions must declare their local variables in Γ when typechecking their body.

Thus, this follows from induction over the typing judgment. □

Lemma 12. Strengthening
If $C \vdash e^* : ti_1^*;\, l_1;\, \Gamma_1;\, \phi_1 \rightarrow ti_2^*;\, l_2;\, \Gamma_2;\, \phi_2$,
and $\Gamma_1 \vdash [\Rightarrow \phi_3][\phi_1]$
then, $C \vdash e^* : ti_1^*;\, l_1;\, \Gamma_1;\, \phi_3 \rightarrow ti_2^*;\, l_2;\, \Gamma_2;\, \phi_4$, for some ϕ_4
where $\Gamma_2 \vdash [\Rightarrow \phi_4][\phi_2]$

Proof. Proof Sketch: Intuitively, this proof follows from the fact that the only time ϕ_1 appears in a premise of a typing rule is on the left hand side of an implication assertion. Thus, since ϕ_3 implies ϕ_1, any ϕ implied by ϕ_1 is also implied by ϕ_3. Similarly, ϕ_2 is generally constructed syntactically by adding onto ϕ_1, so the where clause holds trivially. For the block rules, where ϕ_2 is a joinpoint of multiple paths, and not syntactically generated, each path will have stronger constraints, still implying the old path constraints, and therefore still implying ϕ_2.

The proof is by straightforward induction over typing derivations. □

Lemma Threading-Constraints is crucial for the inductive part of the proof; it allows us to add more index variables, and constraints on those variables, to the pre and post condition of an instruction type. The use case of this lemma is that when a program is evaluated inside a hole, there is no additional type information, but then we plug the reduced version of the program back into the hole and have to compose it with other instructions which may have more type information.

Lemma 13. Threading-Constraints
If $S;\, C \vdash e^* : ti_1^*;\, ti_3^*;\, \Gamma_1;\, \phi_1 \rightarrow ti_2^*;\, ti_4^*;\, \Gamma_2;\, \phi_2$,
and $domain(\phi) \subset \Gamma$
then $S;\, C \vdash e^* : ti_1^*;\, ti_3^*;\, \Gamma_1 \cup \Gamma;\, \phi_1 \cup \phi \rightarrow ti_2^*;\, ti_4^*;\, \Gamma_2 \cup \Gamma;\, \phi_2 \cup \phi$,

Proof. The idea is that if we have a constraint set and add more constraints to it, then the new constraint set must be stronger than the old one and imply the old one. In a little more detail, if $\Gamma \vdash \phi_1 \Rightarrow \phi_2$, then $\forall P.\Gamma \vdash \phi_1, P \Rightarrow \phi_2$, so if $\Gamma \vdash \phi_1 \Rightarrow \phi_2$, then $\forall \phi_3.\Gamma \vdash \phi_1, \phi_3 \Rightarrow \phi_2$. In effect, we are lifting common logic laws into the run-time type system.

Additionally, we can add in fresh index variable declarations to the index environments, because by Lemma Well-formedness those variable won't be referred to by the derivation for e^*.

The proof works by induction over typing derviations. □

Lemma Inversion-On-Instruction-Typing tells us what typing rules can apply to a given Wasm-precheck instruction sequence, and therefore lets us reason about what the type of that sequence looks like. For example, if we have a typing derivation, D for $S;\, C \vdash t.\mathsf{const}\ c : ti_1^*;\, l_1;\, \phi_1 \rightarrow ti_2^*;\, l_2;\, \phi_2$, then we know that D must have at its base Rule Const, because that is the only way we have of typing constant instructions. D can also include any number of applications of Rule Stack-Poly, because they can be applied to any well-typed sequence of instructions.

We do not know the exact types of instructions just from them being well typed, since the typing rules are non-deterministic. However, we can reason about the general shape of the types given the base type on top of which Rule Stack-Poly get applied. Additionally, Rule Composition can be used with the empty sequence and any well-typed single instruction. The addition of Rule Composition with the empty sequence is trivial because the postcondition of an empty instruction sequence must be immediately reachable from the precondition. Therefore the stack and local index store must be the same in both the precondition and postcondition of the empty sequence in the above case, and the postcondition index type context must be reachable from the precondition index type context.

Lemma 14. Inversion-On-Instruction-Typing

1. If $S; C \vdash t.\mathsf{const}\ c : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_2^* = ti_1^*\ (t\ \alpha)$, $l_1 = l_2$, $\Gamma_2 = \Gamma_1, (t\ \alpha)$, and $\phi_2 = \phi_1, (= \alpha\ (t\ c))$, for some $\alpha \notin \Gamma_1$.
2. If $S; C \vdash t.binop : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti^*\ (t\ \alpha_1)\ (t\ \alpha_2)$, $ti_2^* = ti^*\ (t\ \alpha_3)$, $l_1 = l_2$, $\Gamma_2 = \Gamma_1, (t\ \alpha_3)$, and $\phi_2 = \phi_1, (= \alpha_3\ (binop\ \alpha_1\ \alpha_2))$, for some $\alpha_3 \notin \Gamma_1$.
3. If $S; C \vdash t.testop : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti^*\ (t\ \alpha_1)$, $ti_2^* = ti^*\ (\mathsf{i32}\ \alpha_2)$, $l_1 = l_2$, $\Gamma_2 = \Gamma_1, (\mathsf{i32}\ \alpha_2)$, and $\phi_2 = \phi_1, (= \alpha_2\ (testop\ \alpha_1))$, for some $\alpha_2 \notin \Gamma_1$.
4. If $S; C \vdash t.relop : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti^*\ (t\ \alpha_1)\ (t\ \alpha_2)$, $ti_2^* = ti^*\ (\mathsf{i32}\ \alpha_3)$, $l_1 = l_2$, $\Gamma_2 = \Gamma_1, (\mathsf{i32}\ \alpha_3)$, and $\phi_2 = \phi_1, (\mathsf{i32}\ \alpha_3), (= \alpha_3\ (relop\ \alpha_1\ \alpha_2))$, for some $\alpha_3 \notin \Gamma_1$.
5. If $S; C \vdash \mathsf{nop} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti_2^*$, $l_1 = l_2$, $\Gamma_1 = \Gamma_2$, and $\phi_1 = \phi_2$.
6. If $S; C \vdash \mathsf{drop} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti_0^*\ (t\ \alpha)$, $ti_2^* = ti_0^*$, $l_1 = l_2$, $\Gamma_1 = \Gamma_2$, and $\phi_1 = \phi_2$.
7. If $S; C \vdash \mathsf{select} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$,
   then $ti_1^* = ti_0^*\ (t\ \alpha_1)\ (t\ \alpha_2)\ (\mathsf{i32}\ \alpha_3)$, $ti_2^* = ti_0^*; (t\ \alpha)$, $l_1 = l_2$, $\Gamma_2 = \Gamma_1, (t\ \alpha)$, and $\phi_2 = \phi_1, (t\ \alpha), (\text{if}(= \alpha_3\ (\mathsf{i32}\ 0))\ (= \alpha_1\ \alpha)\ (= \alpha_2\ \alpha))$, for some $\alpha \notin \Gamma_1$.
8. If $S; C \vdash \mathsf{block}\ ((t_1^n \rightarrow t_2^m))e^*\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then
   - $ti_1^* = ti_0^*\ (t_1\ \alpha_1)^n$
   - $ti_2^* = ti_0^*\ (t_2\ \alpha_2)^m$
   - $S; C, \text{label}\ ((t_2\ \alpha_2)^m; l_2; \phi_2) \vdash e^* : (t_1\ \alpha_1)^n; l_1; \Gamma_1; \phi_1 \rightarrow (t_2\ \alpha_2)^m; l_2; \Gamma_2; \phi_3$,
   - $\Gamma_2 \vdash \phi_3 \Rightarrow \phi_2$
9. If $S; C \vdash \mathsf{loop}\ ((t_1\ \alpha_3)^*; l_3; \phi_3 \rightarrow (t_2\ \alpha_4)^*; l_4; \phi_4)\ e^*\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti_0^*\ (t_1\ \alpha_1)^*$, $ti_2^* = ti_0^*\ (t_2\ \alpha_2)^*$, $l_1 = (t_l\ \alpha_{l1})^*$, $l_3 = (t_l\ \alpha_{l3})^*$, $l_2 = (t_l\ \alpha_{l2})^*$, $l_4 = (t_l\ \alpha_{l4})^*$, $\Gamma_2 = \Gamma_1, (t_2\ \alpha_2)^*, (t_l\ \alpha_{l2})^*$, $\phi_2 = \phi_1 \cup \phi_4'$, $\phi_3' = \phi_3[\alpha_3 \mapsto \alpha_1]^*[\alpha_{l3} \mapsto \alpha_{l1}]^*$, $\phi_4' = \phi_4[\alpha_4 \mapsto \alpha_2]^*[\alpha_{l4} \mapsto \alpha_{l2}]^*$, $S; C, \text{label}\ ((t_1\ \alpha_3)^*; l_3; \phi_3) \vdash e^* : (t_1\ \alpha_1)^*; l_1; \emptyset, (t_1\ \alpha_1)^*, l_1; \phi_3' \rightarrow (t_2\ \alpha_2)^*; l_2; \Gamma_5; \phi_5$, $\Gamma_1 \vdash \phi_1 \Rightarrow \phi_3'$, and $\Gamma_5 \vdash \phi_5 \Rightarrow \phi_4'$.
10. If $S; C \vdash \mathsf{if}\ ((t_1\ \alpha_3)^*; l_3; \phi_3 \rightarrow (t_2\ \alpha_4)^*; l_4; \phi_4)\ e_1^*\ \mathsf{else}\ e_2^*\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti_0^*\ (t_1\ \alpha_1)^*\ (\mathsf{i32}\ \alpha)$, $ti_2^* = ti_0^*\ (t_2\ \alpha_4)^*$, $l_1 = (t_l\ \alpha_{l1})^*$, $l_3 = (t_l\ \alpha_{l3})^*$, $l_2 = (t_l\ \alpha_{l2})^*$, $l_4 = (t_l\ \alpha_{l4})^*$, $\Gamma_2 = \Gamma_1, (t_2\ \alpha_2)^* \cup (t_l\ \alpha_{l2})^*$, $\phi_2 = \phi_1 \cup \phi_4'$, $\phi_3' = \phi_3[\alpha_3 \mapsto \alpha_1]^*[\alpha_{l3} \mapsto \alpha_{l1}]^*$, $\phi_4' = \phi_4[\alpha_4 \mapsto \alpha_2]^*[\alpha_{l4} \mapsto \alpha_{l2}]^*$, $S; C, \text{label}\ ((t_2\ \alpha_4)^*; l_4; \phi_4) \vdash e_1^* : (t_1\ \alpha_1)^*; l_1; \emptyset, (t_1\ \alpha_1)^*, l_1; \phi_3' \rightarrow (t_2\ \alpha_2)^*; l_2; \Gamma_5; \phi_5$, $S; C, \text{label}\ ((t_2\ \alpha_4)^*; l_4; \phi_4) \vdash e_2^* : (t_1\ \alpha_1)^*; l_1; \emptyset, (t_1\ \alpha_1)^*, l_1; \phi_3' \rightarrow (t_2\ \alpha_2)^*; l_2; \Gamma_6; \phi_6$, $\Gamma_1 \vdash \phi_1 \Rightarrow \phi_3'$, $\Gamma_6 \vdash \phi_6 \Rightarrow \phi_4'$, and $\Gamma_5 \vdash \phi_5 \Rightarrow \phi_4'$.
11. If $S; C \vdash \mathsf{label}_n\{e_0^*\}e^*\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_2^* = ti_1^*\ ti^*$, $S; C \vdash e_0^* : ti_3^*; l_3; \Gamma_3; \phi_3 \rightarrow ti^*; l_2; \Gamma_4; \phi_4$, $S; C, \text{label}\ (ti_3^*; l_3; \phi_3) \vdash e^* : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti^*; l_2; \Gamma_5; \phi_5$, $ti_3^*, l_3 \notin \Gamma_1$, $\Gamma_4 \vdash \phi_4 \Rightarrow \phi_2$, and $\Gamma_5 \vdash \phi_5 \Rightarrow \phi_2$.
12. If $S; C \vdash \mathsf{br}\ i : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti_0^*\ (t_1\ \alpha_1)^*$ and $l_1 = (t_{l1}\ \alpha_{l1})^*$, where $C_{\text{label}}(i) = (t_1\ \alpha_2); (t_{l1}\ \alpha_{l2})^*; \phi_3$ and $\Gamma_1 \vdash \phi_1 \Rightarrow \phi_3[\alpha_2 \mapsto \alpha_1][\alpha_{l2} \mapsto \alpha_{l1}]$.
13. If $S; C \vdash \mathsf{br\_if}\ i : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti_2^*\ (\mathsf{i32}\ \alpha)$, $l_2 = l_1$, $\Gamma_1 = \Gamma_2$, $\phi_2 = \phi_1, (= \alpha\ (\mathsf{i32}\ 0))$, $ti_1^* = (t_1\ \alpha_1)^*\ (\mathsf{i32}\ \alpha)$ and $l_1 = (t_{l1}\ \alpha_{l1})^*$, where $C_{\text{label}}(i) = (t_1\ \alpha_2); (t_{l1}\ \alpha_{l2})^*; \phi_3$ and $\Gamma_1 \vdash \phi_1, \neg(= \alpha\ (\mathsf{i32}\ 0)) \Rightarrow \phi_3[\alpha_2 \mapsto \alpha_1][\alpha_{l2} \mapsto \alpha_{l1}]$.
14. If $S; C \vdash \mathsf{br\_table}\ i^+ : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti_0^*\ (t_1\ \alpha_1)^*\ (\mathsf{i32}\ \alpha)$ and $l_1 = (t_{l1}\ \alpha_{l1})^*$, where $(C_{\text{label}}(i) = (t_1\ \alpha_i); (t_{l1}\ \alpha_{li})^*; \phi_i)^+$, and $(\Gamma_1 \vdash \phi_1 \Rightarrow \phi_i[\alpha_i \mapsto \alpha_1][\alpha_{li} \mapsto \alpha_{l1}])^+$.
15. If $S; C \vdash \mathsf{call}\ i : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = (t_1\ \alpha_1)^*$, $ti_2^* = (t_2\ \alpha_5)$, $l_1 = l_2$, $\Gamma_2 = \Gamma_1 \cup \Gamma_4$, $\phi_2 = \phi_1 \cup \phi_4[\alpha_4 \mapsto \alpha_5]^*$, where $\alpha_5^* \notin \Gamma_1$, $C_{\text{func}}(i) = (t_1\ \alpha_3)^*; \phi_3 \rightarrow (t_2\ \alpha_4)^*; \phi_4$, and $\Gamma_1 \vdash \phi_1 \Rightarrow \phi_3[\alpha_3 \mapsto \alpha_1]$.
16. If $S; C \vdash \mathsf{call\_indirect}\ ((t_1\ \alpha_3)^*; \phi_3 \rightarrow (t_2\ \alpha_4)^*; \phi_4) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = (t_1\ \alpha_1)^*$, $ti_2^* = (t_2\ \alpha_5)$, $l_1 = l_2$, $\Gamma_2 = \Gamma_1 \cup \Gamma_4$, $\phi_2 = \phi_1 \cup \phi_4[\alpha_4 \mapsto \alpha_5]^*$, where $\alpha_5^* \notin \Gamma_1$, $C_{\text{table}} = (j, (ti_6^*; \phi_6 \rightarrow ti_7^*; \phi_7)^*)$, and $\Gamma_1 \vdash \phi_1 \Rightarrow \phi_3[\alpha_3 \mapsto \alpha_1]$.
17. If $S; C \vdash \mathsf{call}\ cl : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = (t_1\ \alpha_1)^*$, $ti_2^* = (t_2\ \alpha_5)$, $l_1 = l_2$, $\Gamma_2 = \Gamma_1 \cup \Gamma_4$, $\phi_2 = \phi_1 \cup \phi_4[\alpha_4 \mapsto \alpha_5]^*$, where $\alpha_5^* \notin \Gamma_1$, $S \vdash cl : (t_1\ \alpha_3)^*; \phi_3 \rightarrow (t_2\ \alpha_4)^*; \phi_4$, and $\Gamma_1 \vdash \phi_1 \Rightarrow \phi_3[\alpha_3 \mapsto \alpha_1]$.
18. If $S; C \vdash \mathsf{local}_n\{i; v_l^*\}\ e^*\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_2^* = ti_1^*\ ti^n$, $l_1 = l_2$, $\Gamma_2 = \Gamma_1 \cup \Gamma_3$, $\phi_2 = \phi_1 \cup \phi_3$, and $S; (ti^n; l_3; \Gamma_3; \phi_3) \vdash_i v_l^*; e^* : ti^n; l_3; \Gamma_3; \phi_3$.
19. If $S; C \vdash \mathsf{return} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti_0^*\ (t_1\ \alpha_1)^*$, where $C_{\text{return}} = (t_1\ \alpha_2); \phi_3$ and $\Gamma_1 \vdash \phi_1 \Rightarrow \phi_3[\alpha_2 \mapsto \alpha_1]$.
20. If $S; C \vdash \mathsf{get\_local}\ i : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_2^* = ti_1^*\ (t\ \alpha)$, $l_1 = l_2$, $\Gamma_1 = \Gamma_2$, $\phi_1 = \phi_2$, $C_{\text{local}}(i) = t$, and $l_1(i) = (t\ \alpha)$.
21. If $S; C \vdash \mathsf{set\_local}\ i : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti_2^*\ (t\ \alpha)$, $\Gamma_1 = \Gamma_2$, $\phi_1 = \phi_2$, $C_{\text{local}}(i) = t$, and $l_2 = l_1[i := (t\ \alpha)]$.
22. If $S; C \vdash \mathsf{tee\_local}\ i : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti_2^* = ti^*\ (t\ \alpha)$, $\Gamma_1 = \Gamma_2$, $\phi_1 = \phi_2$, $C_{\text{local}}(i) = t$, and $l_2 = l_1[i := (t\ \alpha)]$.
23. If $S; C \vdash \mathsf{get\_global}\ i : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_2^* = ti_1^*\ (t\ \alpha)$, $l_1 = l_2$, $\Gamma_2 = \Gamma_1, (t\ \alpha)$, $\phi_1 = \phi_2$, $C_{\text{global}}(i) = \text{mut}^?\ t$, and $\alpha \notin \Gamma_1$.

24. If $S; C \vdash \text{set\_global } i : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti_2^*\ (t\ \alpha), l_1 = l_2, \Gamma_1 = \Gamma_2, \phi_1 = \phi_2$, and $C_{\text{global}}(i) = \text{mut } t$.
25. If $S; C \vdash t.\text{load } (tp\_sx)^?\ align\ o : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti^*\ (\text{i32}\ \alpha_1), ti_2^* = ti^*\ (t\ \alpha_2), l_1 = l_2$, $\Gamma_2 = \Gamma_1, (t\ \alpha_2), \phi_1 = \phi_2, C_{\text{memory}} = n, 2^{align} \leq (|tp| <)^?|t|$, and $\alpha_2 \notin \Gamma_1$.
26. If $S; C \vdash t.\text{store } tp^?\ align\ o : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti_2^*\ (\text{i32}\ \alpha_1)\ (t\ \alpha_2), l_1 = l_2, \Gamma_1 = \Gamma_2, \phi_1 = \phi_2$, $C_{\text{memory}} = n$, and $2^{align} \leq (|tp| <)^?|t|$.
27. If $S; C \vdash \text{current\_memory} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$,
then $ti_2^* = ti_1^*\ (\text{i32}\ \alpha), l_1 = l_2, \Gamma_2 = \Gamma_1, (\text{i32}\ \alpha), \phi_1 = \phi_2, C_{\text{memory}} = n$, and $\alpha \notin \Gamma_1$.
28. If $S; C \vdash \text{grow\_memory} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, then $ti_1^* = ti^*\ (\text{i32}\ \alpha_1), ti_2^* = ti^*\ (\text{i32}\ \alpha_2), l_1 = l_2, \Gamma_2 = \Gamma_1, (\text{i32}\ \alpha_2)$, $\phi_1 = \phi_2, C_{\text{memory}} = n$, and $\alpha_2 \notin \Gamma_1$.
29. If $S; C \vdash e_1^*\ e_2 : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$, then $S; C \vdash e_1^* : ti_0^*\ ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_1'^*; l_1'; \Gamma_1'; \phi_1'$, and $S; C \vdash e_2 : ti_1'^*; l_1'; \Gamma_1'; \phi_1' \rightarrow ti_0^*\ ti_2^*; l_2; \Gamma_2; \phi_2$.

Proof. Follows from straightforward induction over typing derivations. □

Corollary 3. Type-of-Values
If $S; C \vdash (t.\text{const}\ c)^n : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, *then* $ti_2^* = ti_1^*(t\ \alpha)^*, l_2 = l_1, \Gamma_2^* = \Gamma_1, (t\ \alpha)^*, \phi_2 = \phi_1, (= \alpha(t\ c))^*$.

Proof. Follows from straightforward induction over the typing derivations and Lemma Inversion-On-Instruction-Typing. □

Corollary 4. Values-Any-Context
If $S; C \vdash (t.\text{const}\ c)^n : ti^*; l_1; \Gamma; \phi \rightarrow ti_1^*\ (t\ \alpha)^*; l_2; \Gamma, (t\ \alpha)^*; \phi, (=\ \alpha\ (t\ c))$
then $\hat{S}; \hat{C} \vdash (t.\text{const}\ c)^n : t\hat{i}^*; l_1; \hat{\Gamma}; \phi_1 \rightarrow ti_1^*\ (t\ \alpha)^*; l_2; \hat{\Gamma}, (t\ \alpha)^*; \hat{\phi}, (=\ \alpha\ (t\ c))$ *for all* $\hat{S}, \hat{C}, \hat{\Gamma}$, *and* $\hat{\phi}$.

Proof. Follows from straightforward induction over the typing derivations, Lemma Inversion-On-Instruction-Typing, Rule Const, and Rule Composition. □

Corollary 5. Sequence-Composition
If $S; C \vdash e_1^* : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, *and* $S; C \vdash e_2^* : ti_2^*; l_2; \Gamma_2; \phi_2 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$.

Proof. Follows from straightforward induction over the typing derivations, Lemma Inversion-On-Instruction-Typing, and Rule Composition. □

Corollary 6. Sequence-Decomposition
If $S; C \vdash e_1^*\ e_2^* : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, *and* $S; C \vdash e_1^* : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$,
then $S; C \vdash e_2^* : ti_3^*; l_3; \Gamma_3; \phi_3 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$

Proof. Follows from straightforward induction over the typing derivations, Lemma Inversion-On-Instruction-Typing, and Rule Composition. □

Corollary 7. Consts-In-Br
If $S; C \vdash e_1^* : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, *and* $S; C \vdash e_2^* : ti_2^*; l_2; \Gamma_2; \phi_2 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$.

Proof. Follows from straightforward induction over the typing derivations, Lemma Inversion-On-Instruction-Typing, and Rule Composition. □

Lemma 15. Erasure-Same-Semantics
If $\vdash s; v^*; e^* : ti_2^*; l_2; \Gamma_2; \phi_2$,
then $erase_s(s); v^*; erase_{e^*}(e^*) \hookrightarrow_i erase_s(s'); v'^*; erase_{e^*}(e'^*)$ for some s' and e'^*.

Proof. The intuition for this is that the reduction semantics are syntactically the same except for the representation of types. Therefore, we can relate the semantics of a Wasm-precheck program to the Wasm semantics by erasing the Wasm-precheck program to a Wasm program, reducing under the Wasm semantics, and then adding back the Wasm-precheck types.
The proof follows then by induction over the typing derivation. □

The next lemma, Lemma Values-Br-In-Context, shows that if a sequence of constants, v^n, has a certain postcondition within a nested context, L^k, then it has the same postcondition outside of that context with the precondition of the context extended with equality constraints on fresh variables. We use this rule for branching and returning when we have some values v^n inside a reduction context L^k.

Lemma 16. Values-Br-In-Context

If $S; C \vdash L^k[(t.\mathsf{const}\ c)^n \mathsf{br}\ j] : ti_1^*; l_1; \Gamma_1; phi_1 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2$

where $C_{\text{label}}(j-k) = (ti_3^n; l_3; \phi_3)$

where ti_3^n and l_3 are entirely fresh (no variable in ti_3^n or l_3 appears in Γ_1 or Γ_2), which we can safely assume through alpha-renaming,

then $S; C' \vdash (t.\mathsf{const}\ c)^n : ti_1^*; l_1; \Gamma_1, (t_0\ \alpha_0)^*; \phi_1, (=\ \alpha_0\ (t_0\ c_0))^* \longrightarrow ti_1^*\ ti_3^n; l_1; \Gamma_1, (t_0\ \alpha_0)^*, ti_3^n; \phi_1, (=\ \alpha_0\ (t_0\ c_0))^*, (=\ \alpha_3\ (t\ c))^*$

where $\Gamma_1, (t_0\ \alpha_0)^*, ti_3^n \vdash \phi_1, (=\ \alpha_0\ (t_0\ c_0))^*, (=\ \alpha_3\ (t\ c))^* \Rightarrow \phi_3[l_3 \mapsto l_1]$

where $ti_3^n = (t\ \alpha_3)^n$

Proof. By induction on k.

- Base case: $k = 0$
  1. Expanding L^0, we have $S; C \vdash (t_0.\mathsf{const}\ c_0)^*\ (t.\mathsf{const}\ c)^n\ \
\ j\ e^* : ti_1^*; l_1; \Gamma_1; \phi_1 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ for some $(t_0.\mathsf{const}\ c_0)^*$ and e^*
  2. By Lemma Sequence-Decomposition on 1, we know that the following hold for some ti_4^*, l_4, Γ_4, and ϕ_4
     a. $S; C \vdash (t_0.\mathsf{const}\ c_0)^* : ti_1^*; l_1; \Gamma_1; \phi_1 \longrightarrow ti_4^*; l_4; \Gamma_4; \phi_4$
     b. $S; C \vdash (t.\mathsf{const}\ c)^n\ \
\ j\ e^* : ti_4^*; l_4; \Gamma_4; \phi_4 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  3. By Lemma Type-Of-Values on 2a, we know that $ti_4^* = ti_1^*\ (t_0\ \alpha_0)^*$, $l_4 = l_1$, $\Gamma_4 = \Gamma_1, (t_0\ \alpha_0)^*$, and $\phi_4 = \phi_1, (=\ \alpha_0\ (t_0\ c_0))^*$
  4. Therefore, $S; C \vdash (t.\mathsf{const}\ c)^n\ \
\ j\ e^* : ti_1^*\ (t_0\ \alpha_0)^*; l_1; \Gamma_1, (t_1\ \alpha_0)^*; \phi_1, (=\ \alpha_0\ (t_0\ c_0))^* \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  5. By Lemma Sequence-Decomposition on 4, we have $S; C \vdash (t.\mathsf{const}\ c)^n\ \
\ j : ti_1^*\ (t_1\ \alpha_1)^*; l_1; \Gamma_1, (t_0\ \alpha_0)^*; \phi_1, (=\ \alpha_0\ (t_0\ c_0))^* \longrightarrow ti_5^*; l_5; \Gamma_5; \phi_5$ for some ti_5^*, l_5, Γ_5, and ϕ_5.
  6. By Lemma Inversion-On-Instruction-Typing on $S; C \vdash (t.\mathsf{const}\ c)^n\ \
\ j : ti_1^*\ (t_0\ \alpha_0)^*; l_1; \Gamma_1, (t_0\ \alpha_0)^*; \phi_1, (=\ \alpha_0\ (t_0\ c_0))^* \longrightarrow ti_5^*; l_5; \Gamma_5; \phi_5$, we have that
     a. $S; C \vdash (t.\mathsf{const}\ c)^n : ti_1^*\ (t_0\ \alpha_0)^*; l_1; \Gamma_1, (t_0\ \alpha_0)^*; \phi_1, (=\ \alpha_0\ (t_0\ c_0))^* \longrightarrow ti_6^*; l_6; \Gamma_6; \phi_6$ for some ti_6^*, l_6, Γ_6, and ϕ_6
     b. $S; C \vdash \mathsf{br}\ j : ti_6^*; l_6; \Gamma_6; \phi_6 \longrightarrow ti_5^*; l_5; \Gamma_5; \phi_5$
  7. By Lemma Type-Of-Values on 4a, we know that $ti_6^* = ti_1^*\ (t_0\ \alpha_0)^*\ (t\ \alpha_6)^n$, $l_6 = l_1$, $\Gamma_6 = \Gamma_1, (t_0\ \alpha_0)^*(t\ \alpha_6)^n$, and $\phi_6 = \phi_1, (=\ \alpha_0\ (t_0\ c_0))^*, (=\ \alpha_6\ (t\ c))^*$
  8. By Lemma Inversion-On-Instruction-Typing on 4b, we know that $(t = t_3)^n$, where $ti_3^n = ((t_3\ \alpha_3))^n$, $l_1 = (t_{l3}\ \alpha_{l1})^*$, where $l_3 = (t_{l3}\ \alpha_{l3})^*$ and $\Gamma_0, (t_0\ \alpha_0)^* \vdash \phi_1, (=\ \alpha_0\ (t_0\ c_0))^*, (=\ \alpha_6\ (t\ c))^* \Rightarrow \phi_3[\alpha_3 \mapsto \alpha_6][\alpha_{l3} \mapsto \alpha_{l6}]$
  9. Then, by combining Lemma Values-Any-Context, 4a, 5, and 6, and since ti_3^n are fresh, we have that $S; C' \vdash (t.\mathsf{const}\ c)^n : ti_1^*\ (t_0\ \alpha_0)^*; l_3; \Gamma_1, (t_0\ \alpha_0)^*; \phi_1, (=\ \alpha_0\ (t_0\ c_0))^* \longrightarrow ti_1^*\ (t_1\ \alpha_1)^*\ ti_3^n; l_3; \Gamma_1, (t_1\ \alpha_1)^*; \phi_1, (=\ \alpha_1\ (t_1\ c_1))^*, (=\ \alpha_3\ (t\ c))^*$
  10. Finally, by 6, and because ti_3^n is fresh, $\Gamma_1, (t_1\ \alpha_1)^*, ti_3^n \vdash \phi_1, (=\ \alpha_1\ (t_1\ c_1))^*, (=\ \alpha_3\ (t\ c))^* \Rightarrow \phi_3[\alpha_{l3} \mapsto \alpha_{l6}]$ (here we are essentially moving the renaming $[\alpha_3 \mapsto \alpha_6]$ to the left-hand-side of the implication, which is safe because we control what the names are at this point)
- Inductive case: $k > 0$

  This proof case is simpler than above as we only have to reason about br in the base case, so in the inductive case the inductive hypothesis handles it for us.
  1. Expanding L^k, we have $S; C \vdash (t_0.\mathsf{const}\ c_0)^* \mathsf{label}\{e_0^*\}\ L^{k-1}[(t.\mathsf{const}\ c)^n\ \
\ j]\ \mathsf{end} e^* : ti_1^*; l_1; \Gamma_1; \phi_1 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ for some $(t_0.\mathsf{const}\ c_0)^*$, e_0^*, and e^*
  2. By Lemma Sequence-Decomposition on 1, we know that the following hold for some ti_4^*, l_4, Γ_4, and ϕ_4
     a. $S; C \vdash (t_0.\mathsf{const}\ c_0)^* : ti_1^*; l_1; \Gamma_1; \phi_1 \longrightarrow ti_4^*; l_4; \Gamma_4; \phi_4$
     b. $S; C \vdash \mathsf{label}\{e_0^*\}\ L^{k-1}[(t.\mathsf{const}\ c)^n\ \
\ j]\ \mathsf{end} e^* : ti_4^*; l_4; \Gamma_4; \phi_4 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  3. By Lemma Type-Of-Values on 2a, we know that $ti_4^* = ti_1^*\ (t_0\ \alpha_0)^*$, $l_4 = l_1$, $\Gamma_4 = \Gamma_1, (t_0\ \alpha_0)^*$, and $\phi_4 = \phi_1, (=\ \alpha_0\ (t_0\ c_0))^*$
  4. Therefore, $S; C \vdash \mathsf{label}\{e_0^*\}\ L^{k-1}[(t.\mathsf{const}\ c)^n\ \
\ j]\ \mathsf{end} e^* : ti_1^*\ (t_0\ \alpha_0)^*; l_1; \Gamma_1, (t_1\ \alpha_0)^*; \phi_1, (=\ \alpha_0\ (t_0\ c_0))^* \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  5. By Lemma Sequence-Decomposition on 4, we have $S; C \vdash \mathsf{label}\{e_0^*\}\ L^{k-1}[(t.\mathsf{const}\ c)^n\ \
\ j]\ \mathsf{end} e^* : ti_1^*\ (t_0\ \alpha_0)^*; l_1; \Gamma_1, (t_0\ \alpha_0)^*; \phi_1, (=\ \alpha_0\ (t_0\ c_0))^* \longrightarrow ti_5^*; l_5; \Gamma_5; \phi_5$ for some ti_5^*, l_5, Γ_5, and ϕ_5.
  6. By Lemma Inversion-On-Instruction-Typing on 5, we have that $S; C, \text{label}(ti^*; l; \phi) \vdash L^{k-1}[(t.\mathsf{const}\ c)^n\ \
\ j]\ \mathsf{end} e^* : ti_1^*\ (t_0\ \alpha_0)^*; l_1; \Gamma_1, (t_0\ \alpha_0)^*; \phi_1, (=\ \alpha_0\ (t_0\ c_0))^* \longrightarrow ti_5^*; l_5; \Gamma_5; \phi_5$
  7. Then, by the inductive hypothesis on 6, we have that $S; C, \text{label}(ti^*; l; \phi) \vdash (t.\mathsf{const}\ c)^n : ti_1^*\ (t_0\ \alpha_0)^*; l_3; \Gamma_1, (t_0\ \alpha_0)^*, (t'\ \alpha')^*; \phi_1, (=\ \alpha_0\ (t_0\ c_0))^*, (=\ \alpha'\ (t'\ c'))^* \longrightarrow ti_1^*\ (t_0\ \alpha_0)^*\ ti_3^n; l_3; \Gamma_1, (t_0\ \alpha_0)^*, (t'\ \alpha')^*, ti_3^n; \phi_1, (=\ \alpha_0\ (t_0\ c_0))^*, (=\ \alpha'\ (t'\ c'))^*, (=\ \alpha_3\ (t\ c))^*$, where $\Gamma_1, (t_0\ \alpha_0)^*, (t'\ \alpha')^*, ti_3^n \vdash \phi_1, (=\ \alpha_0\ (t_0\ c_0))^*, (=\ \alpha'\ (t'\ c'))^*, (=\ \alpha_3\ (t\ c))^* \Rightarrow \phi_3[l_3 \mapsto l_1]$, for some t'^*, c'^*, and α'^*.

8. Finally, by Lemma Values-Any-Context on 7, we have $S; C' \vdash (t.\mathrm{const}\ c)^n : ti_1^*\ (t_0\ \alpha_0)^*; l_1; \Gamma_1, (t_0\ \alpha_0)^*, (t\ \alpha)^*; \phi_1, (= \alpha_0\ (t_0\ c_0))^*, (= \alpha\ (t\ c))^* \rightarrow ti_1^*\ (t_0\ \alpha_0)^*\ ti_3^n; l_1; \Gamma_1, (t_0\ \alpha_0)^*, (t'\ \alpha')^*, ti_3^n; \phi_1, (= \alpha_0\ (t_0\ c_0))^*, (= \alpha'\ (t'\ c'))^*, (= \alpha_3\ (t\ c))^*$

□

Lemma 17. Values-Return-In-Context
If $S; C \vdash L^k[(t.\mathrm{const}\ c)^n \mathrm{return}] : ti_1^*; l_1; \Gamma_1; phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
where $C_{\mathrm{return}} = (ti_3^n; \phi_3)$
where ti_3^n are entirely fresh (no variable in ti_3^n appears in Γ_1 or Γ_2), which we can safely assume through alpha-renaming, then $S; C' \vdash (t.\mathrm{const}\ c)^n : ti_1^*; l_1; \Gamma_1, (t_0\ \alpha_0)^*; \phi_1, (= \alpha_0\ (t_0\ c_0))^* \rightarrow ti_1^*\ ti_3^n; l_1; \Gamma_1, (t_0\ \alpha_0)^*, ti_3^n; \phi_3$

Proof. Similar to Lemma Values-Br-In-Context. □

Note: while we parameterize Wasm-precheck over implication, since we start with an under-approximation of implication, we can assume that implication is sound.

C.1 Subject Reduction Lemmas and Proofs

Lemma 18 (Subject Reduction for Programs). If $\vdash s; v^*; e^* : ti^*; l; \Gamma; \phi$,
and $s; v^*; e^* \hookrightarrow s'; v'^*; e'^*$,
then $\vdash s'; v'^*; e'^* : ti^*; l; \Gamma; \phi$

Proof.

1. Because $\vdash_i s; v^*; e^* : ti^*; l; \Gamma; \phi$, we know that for some S
   a. $\vdash s : S$
   b. $S; \epsilon \vdash_i v^*; e^* : ti^*; l; \Gamma; \phi$
   by inversion on Rule Program.
2. by inversion on Rule Code and 1b, for some α^*, t^*, c^*, we have that
   a. $(\vdash v : (t_v\ \alpha_v); \phi_v)^*$
   b. $S; S_{\mathrm{inst}}(i), \mathrm{local}\ (t_v^*) \vdash e^* : \epsilon; (t_v\ \alpha_v)^*; \emptyset, (t_v\ \alpha_v)^*, (t\ \alpha)^*; \phi_v^*, (= \alpha\ (t\ c))^* \rightarrow ti^*; l; \Gamma; \phi_2$
   c. $\Gamma \vdash \phi_2 \Rightarrow \phi$
3. By Lemma 19 on 1a, 2a, 2b, and Lemma Well-Formedness, we have
   a. $S; S_{\mathrm{inst}}(i), \mathrm{local}\ (t_v^*) \vdash e'^* : \epsilon; (t_v\ \alpha_v)^*; \emptyset, (t_v\ \alpha_v)^*, (t\ \alpha)^*; \phi_v^*, (= \alpha\ (t\ c))^* \rightarrow ti^*; l; \Gamma; \phi_2$
   b. $\vdash \epsilon; (t_v\ \alpha_v)^*; \emptyset, (t_v\ \alpha_v)^*, (t\ \alpha)^*; \phi_v^*, (= \alpha\ (t\ c))^* \rightarrow ti^*; l; \Gamma; \phi_2$
   c. $\vdash s' : S$ for some s'
   d. $(\vdash v' : (t_v\ \alpha_v'); \phi_v')^*$, where $l_2 = (t_v\ \alpha_v')$ and $\phi_3 = \phi_1 \bigcup \phi_v'^*$, $\alpha^* \notin \Gamma_3$ for some v'^*
   e. $\alpha^* \notin \Gamma_1$
   f. $\Gamma_4 \vdash \phi_4 \Rightarrow \phi_2$
4. By Rule Code, 3a, and 3d, $S; \epsilon \vdash_i v'^*; e'^* : ti^*; l; \Gamma; \phi$.
5. by Rule Program, 4, and 3c, $\vdash_i s'; v'^*; e'^* : ti^*; l; \Gamma; \phi$

□

Lemma 19 (Subject Reduction for Instructions). If $S; C \vdash e^* : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ and $s; v^*; e^* \hookrightarrow s'; v'^*; e'^*$ (we may omit s and v^* on rules that don't refer to them), where

Assumption 1) $\vdash ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
Assumption 2) $\vdash s : S$, where $C = S_{\mathrm{inst}}(i)$
Assumption 3) $(\vdash v : (t_v\ \alpha_v); \phi_v)^*$, where $l_1 = (t_v\ \alpha_v)^*$ and $\phi_v \subset \phi_1$

then $S; C \vdash e'^* : ti_1^*; l_3; \Gamma_1, (t\ \alpha)^*; \phi_3, (= \alpha\ (t\ c))^* \rightarrow ti_2^*; l_2; \Gamma_4; \phi_4$ for some α^*, t^*, and c^* where

Subgoal 1) $\vdash ti_1^*; l_3; \Gamma_1, (t\ \alpha)^*; \phi_3, (= \alpha\ (t\ c))^* \rightarrow ti_2^*; l_2; \Gamma_4; \phi_4$,
Subgoal 2) $\vdash s' : S$ (implicitly true and omitted when $s' = s$),
Subgoal 3) $(\vdash v' : (t_v\ \alpha_v'); \phi_v')^*$, $l_3 = (t_v'\ \alpha_v')^*$,
Subgoal 4) $\phi_3 = \phi_1 \bigcup \phi_v'^*$ for some v'^* (implicitly true and omitted when $v'^* = v^*$ and $l_1 = l_3$, or when $\phi_3 = \phi_1[\alpha_v \mapsto \alpha_v']$),
Subgoal 5) $\alpha^* \notin \Gamma_1$ (ensures α^* don't appear in the typing derivation for e'^*, according to Lemma Well-formedness),
Subgoal 6) and $\Gamma_4 \vdash \phi_4 \Rightarrow \phi_2$ (implicitly true and omitted when $\phi_4 = \phi_2$ and $\Gamma_2 \subseteq \Gamma_4$)

Proof. By case analysis on the reduction rules.

- $S; C \vdash L^0[\text{trap}] : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ L^0[\text{trap}] \hookrightarrow \text{trap}$
  This case is trivial since trap accepts any precondition and postcondition. Thus, $S; C \vdash \text{trap} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ by Rule Trap, where: subgoal 1 follows from assumption 4; subgoals 2, 4, and 6 trivially hold; subgoal 3 follows from assumption 2, and subgoal 5 holds since $\alpha^* = \epsilon$.

- $S; C \vdash (t.\text{const}\ c_1)\ (t.\text{const}\ c_2)\ t.binop : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ (t.\text{const}\ c_1)\ (t.\text{const}\ c_2)\ t.binop \hookrightarrow t.\text{const}\ c$ where $c = binop(c_1, c_2)$
  We begin by reasoning about the type of the original instructions $(t.\text{const}\ c_1)\ (t.\text{const}\ c_2)\ t.binop$
  By Lemma Inversion-On-Instruction-Typing on $S; C \vdash (t.\text{const}\ c_1)\ (t.\text{const}\ c_2)\ t.binop : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, we know that $\Gamma_2 = \Gamma_1, (t\ \alpha_1), (t\ \alpha_2), (t\ \alpha_3)$ where $\alpha_1, \alpha_2, \alpha_3 \notin \Gamma_1$, $ti_2^* = ti_1^*\ (t\ \alpha_3)$, $l_2 = l_1$, and $\phi_2 = \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)), (= \alpha_3\ (\|binop\|\ \alpha_1\ \alpha_2))$
  Now we know that we want to show that

$$\begin{aligned}&S; C \vdash t.\text{const}\ c : ti_1^*; l_1; \Gamma_1, (t\ \alpha_1), (t\ \alpha_2); \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2))\\ &\rightarrow ti_1^*\ (t\ \alpha_3); l_1; \Gamma_1, (t\ \alpha_1), (t\ \alpha_2), (t\ \alpha_3); \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)), (= \alpha_3\ c)\end{aligned}$$

  where $(t\ \alpha_3); l_1 \subset \Gamma_1, (t\ \alpha_1), (t\ \alpha_2)$ and $\Gamma_1, (t\ \alpha_1), (t\ \alpha_2), (t\ \alpha_3) \vdash \phi_3 \Rightarrow (\phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)), (= \alpha_3\ (\|binop\|\ \alpha_1\ \alpha_2)))$
  Now we show that $t.\text{const}\ c$ has the appropriate type.
  By Rule Const,

$$\begin{aligned}&S; C \vdash t.\text{const}\ c : \epsilon; l_1; \Gamma_1, (t\ \alpha_1), (t\ \alpha_2); \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2))\\ &\qquad \rightarrow (t\ \alpha_3); l_1; \Gamma_1, (t\ \alpha_1), (t\ \alpha_2), (t\ \alpha_3); \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)), (= \alpha_3\ c)\end{aligned}$$

  since $\alpha_3 \notin \Gamma_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2))$.
  Then, by Rule Stack-Poly,

$$\begin{aligned}&S; C \vdash t.\text{const}\ c : ti_1^*; l_1; \Gamma_1, (t\ \alpha_1), (t\ \alpha_2); \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2))\\ &\qquad \rightarrow ti_1^*\ (t\ \alpha_3); l_1; \Gamma_1, (t\ \alpha_1), (t\ \alpha_2), (t\ \alpha_3); \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)), (= \alpha_3\ c)\end{aligned}$$

  By assumption 1, $l_1 \subset \Gamma_1$, so $(t\ \alpha_3), l_1 \subset \Gamma_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)), (t\ \alpha_3)$.
  Further, since $c = binop(c_1, c_2)$, then $\Gamma_1, (t\ \alpha_1), (t\ \alpha_2), (t\ \alpha_3) \vdash \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)), (= \alpha_3\ (t\ c)) \Rightarrow \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)), (= \alpha_3\ (\|binop\|\ \alpha_1\ \alpha_2))$.

- $S; C \vdash (t.\text{const}\ c_1)\ (t.\text{const}\ c_2)\ t.\text{div}\checkmark : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ (t.\text{const}\ c_1)\ (t.\text{const}\ c_2)\ t.\text{div}\checkmark \hookrightarrow t.\text{const}\ c$ where $c = div(c_1, c_2)$
  Same as above.

- $C \vdash (t.\text{const}\ c_1)\ (t.\text{const}\ c_2)\ t.binop : \epsilon; l_1; \Gamma_1\phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ (t.\text{const}\ c_1)\ (t.\text{const}\ c_2)\ t.binop \hookrightarrow \text{trap}$
  This case is trivial since trap accepts any precondition and postcondition. Thus, $S; C \vdash \text{trap} : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ by Rule Trap.

- $S; C \vdash (t.\text{const}\ c_1)\ t.testop : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ (\text{i32.const}\ c)\ t.testop \hookrightarrow \text{i32.const}\ c_2$ where $c_2 = testop(c)$
  We begin by reasoning about the type of the original instructions $(t.\text{const}\ c_1)\ t.testop$
  By Lemma Inversion-On-Instruction-Typing on $S; C \vdash (t.\text{const}\ c_1)\ t.testop : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, we know that $ti_2^* = ti_1^*\ (\text{i32}\ \alpha_2)$, $\Gamma_2 = \Gamma_1, (t\ \alpha_1), (\text{i32}\ \alpha_2)$ where $\alpha_1, \alpha_2 \notin \Gamma_1$, $ti_2^* = (\text{i32}\ \alpha_2)$, $l_2 = l_1$, and $\phi_2 = \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (testop\ \alpha_1))$
  Now we must show that

$$S; C \vdash \text{i32.const}\ c : ti_1^*; l_1; \Gamma_1, (t\ \alpha_1); \phi_1, (= \alpha_1\ (t\ c_1)) \rightarrow ti_1^*\ (t\ \alpha_2); l_1; \Gamma_1, (t\ \alpha_1), (\text{i32}\ \alpha_2); \phi_3$$

  where $ti^*; l \subset \Gamma_3$ and $\Gamma_1, (t\ \alpha_1), (t\ \alpha_2) \vdash \phi_3 \Rightarrow \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (testop\ \alpha_1))$
  We show that i32.const c has the appropriate type.
  By Rule Const, $S; C \vdash \text{i32.const}\ c : \epsilon; l_1; \Gamma_1, (t\ \alpha_1); \phi_1 \rightarrow (\text{i32}\ \alpha_2); l_1; \Gamma_1, (t\ \alpha_1), (\text{i32}\ \alpha_2); \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c))$.
  Then, $S; C \vdash \text{i32.const}\ c : ti_1^*; l_1; \Gamma_1, (t\ \alpha_1); \phi_1 \rightarrow ti_1^*\ (\text{i32}\ \alpha_2); l_1; \Gamma_1, (t\ \alpha_1), (\text{i32}\ \alpha_2); \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c))$ by Rule Stack-Poly.

We have $\Gamma_3 = \Gamma_1, (t\ \alpha_1), (\text{i32}\ \alpha_2)$, and by Lemma Well-formedness, $l_1 \subset \Gamma_1$, so $(\text{i32}\ \alpha_2), l_1 \subset \Gamma_1, (t\ \alpha_1), (\text{i32}\ \alpha_2)$. Further, since $c_2 = testop(c_1)$, then $\Gamma_1, (t\ \alpha_1), (t\ \alpha_2) \vdash \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)) \Rightarrow \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (\|testop\|\ (t\ c_2)))$.

- $S; C \vdash (t.\text{const}\ c_1)\ (t.\text{const}\ c_2)\ t.relop : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2 \phi_2$
  $\wedge\ (\text{i32.const}\ c_1)\ (t.\text{const}\ c_2)\ t.relop \hookrightarrow t.\text{const}\ c$ where $c = relop(c_1, c_2)$
  This case is identical to the $(t.\text{const}\ c_1)\ (t.\text{const}\ c_2)\ t.binop \hookrightarrow t.\text{const}\ c$ case, except that $binop$ is replaced with $relop$, and the result type is replaced with i32.

- $S; C \vdash \text{unreachable} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge$ unreachable $\hookrightarrow$ trap
  This case is trivial since trap accepts any precondition and postcondition. Thus, $S; C \vdash \text{trap} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ by $trap$.

- $S; C \vdash \text{nop} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge$ nop $\hookrightarrow \epsilon$
  By Lemma Inversion-On-Instruction-Typing on Rule Nop, we know that $ti_2^* = ti_1^*$, $l_2 = l_1$, $\Gamma_2 = \Gamma_1$ and $\phi_2 = \phi_1$.
  Then, we want to show that $S; C \vdash \epsilon : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_3; \phi_3$, where $ti^*; l \subset \Gamma_3$ and $\Gamma_3 \vdash \phi_3 \Rightarrow \phi_1$
  Then, $S; C \vdash \epsilon : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_1^*; l_1; \Gamma_1; \phi_1$ by Rule Empty and Rule Stack-Poly.

- $S; C \vdash (t.\text{const}\ c)\ \text{drop} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ (t.\text{const}\ c)\ \text{drop} \hookrightarrow \epsilon$
  By Lemma Inversion-On-Instruction-Typing on $S; C \vdash (t.\text{const}\ c)\ \text{drop} : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, we know that $ti_2^* = ti_1^*$, $l_2 = l_1$, and $\Gamma_2 = \Gamma_1, (t\ \alpha)$. $\phi_2 = \phi_1, (= \alpha\ (t\ c))$.
  We want to show that $S; C \vdash \epsilon : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_3; \phi_3$.
  We have $\alpha \notin \Gamma_1$, as it is a premise of Rule Const.
  By Lemma Well-formedness, $l_1 \subset \Gamma_1$.
  By Rule Empty, $S; C \vdash \epsilon : \epsilon; l_1; \Gamma_1, (t\ \alpha); \phi_1(= \alpha\ (t\ c)) \rightarrow \epsilon; l_1; \Gamma_1, (t\ \alpha); \phi_1(= \alpha\ (t\ c))$.
  Thus, $S; C \vdash \epsilon : ti_1^*; l_1; \Gamma_1, (t\ \alpha); \phi_1(= \alpha\ (t\ c)) \rightarrow ti_1^*; l_1; \Gamma_1, (t\ \alpha); \phi_1(= \alpha\ (t\ c))$ since $ti_2^* = ti_1^*$, by Rule Stack-Poly.

- Case: $S; C \vdash (t.\text{const}\ c_1)\ (t.\text{const}\ c_2)\ (\text{i32.const}\ k+1)\ \text{select} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ (t.\text{const}\ c_1)\ (t.\text{const}\ c_2)\ (\text{i32.const}\ k+1)\ \text{select} \hookrightarrow (t.\text{const}\ c_1)$
  By Lemma Inversion-On-Instruction-Typing on $S; C \vdash (t.\text{const}\ c_1)\ (t.\text{const}\ c_2)\ (\text{i32.const}\ k{+}1)\ \text{select} : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, we know that $ti_2^* = ti_1^*\ (\alpha_3\ )$, $l_2 = l_1$, $\Gamma_2 = \Gamma_1, (t\ \alpha_1), (t\ \alpha_2), (\text{i32}\ \alpha), (t\ \alpha_3)$ and $\phi_2 = \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)), (= \alpha\ (\text{i32}\ k+1)), (if\ (= \alpha\ (\text{i32}\ 0))\ (= \alpha_3\ \alpha_2)\ (= \alpha_3\ \alpha_1))$
  Then, we want to show that $S; C \vdash (t.\text{const}\ c_1) : \epsilon; l_1; \Gamma_1, (t\ \alpha_1), (t\ \alpha_2), (\text{i32}\ \alpha); \phi_1 \rightarrow (t\ \alpha_3); l_2; \Gamma_3; \phi_3$, where $(t\ \alpha_3), l_2 \subset \Gamma_3$ and

  $$\Gamma_3 \vdash \phi_3 \Rightarrow \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)), (= \alpha\ (\text{i32}\ k+1)), (if\ (= \alpha\ (\text{i32}\ 0))\ (= \alpha_3\ \alpha_2)\ (= \alpha_3\ \alpha_1))$$

  By Rule Const,

  $$\begin{aligned} &S; C \vdash (t.\text{const}\ c_1) : \epsilon; l_1; \Gamma_1, (t\ \alpha_1), (t\ \alpha_2), (\text{i32}\ \alpha); \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)), (= \alpha\ (\text{i32}\ k+1)), (= \alpha_3\ (t\ c_1)) \\ &\rightarrow (t\ \alpha_3); l_1; \Gamma_1, (\text{i32}\ \alpha_2), (t\ \alpha_3); \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)), (= \alpha\ (\text{i32}\ k+1)), (= \alpha_3\ (t\ c_1)), (= \alpha_1\ (t\ c_1)) \end{aligned}$$

  since $\alpha_3 \notin \Gamma_1, (t\ \alpha_1), (t\ \alpha_2), (\text{i32}\ \alpha)$.
  We have $\Gamma_3 = \Gamma_1, (t\ \alpha_1), (t\ \alpha_2), (\text{i32}\ \alpha), (t\ \alpha_3)$, and by Lemma Well-formedness, $l_1 \subset \Gamma_1$, so

  $$(t\ \alpha_3), l_1 \subset \Gamma_1, (t\ \alpha_1), (t\ \alpha_2), (\text{i32}\ \alpha), (t\ \alpha_3)$$

  Recall that we know $\Gamma_3 \vdash \phi_3 \Rightarrow \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)), (= \alpha\ (\text{i32}\ k+1)), (if\ (= \alpha\ (\text{i32}\ 0))\ (= \alpha_3\ \alpha_2)\ (= \alpha_3\ \alpha_1))$
  Finally,

  $$\begin{aligned} &S; C \vdash (t.\text{const}\ c_1) : ti_1^*; l_1; \Gamma_1, (t\ \alpha_1), (t\ \alpha_2), (\text{i32}\ \alpha); \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)), (= \alpha\ (\text{i32}\ k+1)), (= \alpha_3\ (t\ c_1)) \\ &\rightarrow ti_1^*\ (t\ \alpha_3); l_1; \Gamma_1, (\text{i32}\ \alpha_2), (t\ \alpha_3); \phi_1, (= \alpha_1\ (t\ c_1)), (= \alpha_2\ (t\ c_2)), (= \alpha\ (\text{i32}\ k+1)), (= \alpha_3\ (t\ c_1)), (= \alpha_1\ (t\ c_1)) \end{aligned}$$

  since $ti_2 = ti_1^*\ (t\ \alpha_3)$, by Rule Stack-Poly.

- Case: $S; C \vdash (t.\text{const}\ c_1)\ (t.\text{const}\ c_2)\ (\text{i32.const}\ 0)\ \text{select} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ (t.\text{const}\ c_1)\ (t.\text{const}\ c_2)\ (\text{i32.const}\ 0)\ \text{select} \hookrightarrow (t.\text{const}\ c_2)$

Same as above, except 0 instead of $k+1$, and $(= \alpha_3\ (t\ c_2))$ instead of $(= \alpha_3\ (t\ c_1))$.

- Case: $S; C \vdash (t.\mathsf{const}\ c)^n\ \mathsf{block}\ (t_1^n \to t_2^m)\ e^*\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \to ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ (t.\mathsf{const}\ c)^n\ \mathsf{block}\ (t_1^n \to t_2^m)\ e^*\ \mathsf{end}$
$\hookrightarrow \mathsf{label}_m\{\epsilon\}\ (t.\mathsf{const}\ c)^n\ e^*\ \mathsf{end}$
We want to show that $S; C \vdash \mathsf{label}_m\{\epsilon\}\ (t.\mathsf{const}\ c)^n\ e^*\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \to ti_2^*; l_2; \Gamma_2; \phi_2$.
  1. The following hold by Lemma Inversion-On-Instruction-Typing on $S; C \vdash (t.\mathsf{const}\ c)^n\ \mathsf{block}\ (t_1^n \to t_2^m)\ e^*\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \to ti_2^*; l_2; \Gamma_2; \phi_2$, for some $ti_1'^*$, l_1', Γ_1', ϕ_1', and ti_3^*
     a. $S; C \vdash (t.\mathsf{const}\ c)^n\ : ti_3^*\ ti_1^*; l_1; \Gamma_1; \phi_1 \to ti_1'^*; l_1'; \Gamma_1'; \phi_1'$
     b. $S; C \vdash \mathsf{block}\ (t_1^n \to t_2^m)\ e^*\ \mathsf{end} : ti_1'^*; l_1'; \Gamma_1'; \phi_1' \to ti_3^*\ ti_2^*; l_2; \Gamma_2; \phi_2$
  2. Then, by Lemma Inversion-On-Instruction-Typing on $S; C \vdash \mathsf{block}\ (t_1^n \to t_2^m)\ e^*\ \mathsf{end} : ti_1'^*; l_1'; \Gamma_1'; \phi_1' \to ti_3^*\ ti_2^*; l_2; \Gamma_2; \phi_2$, we have that
     a. $ti_1'^* = ti_4^*\ (t_1\ \alpha_1)^n$, for some ti_4^*
     b. $ti_2^* = ti_4^*\ (t_2\ \alpha_2)^m$
     c. $S; C, \mathrm{label}\ ((t_2\ \alpha_2)^m; l_2; \phi_2) \vdash e^* \vdash e^* : (t_1\ \alpha_1)^n; l_1'; \Gamma_1'; \phi_1' \to (t_2\ \alpha_2)^m; l_2; \Gamma_2; \phi_3$
     d. $\Gamma_2 \vdash \phi_3 \Rightarrow \phi_2$
  3. By Rule Stack-Poly and 2c, we have that $S; C, \mathrm{label}\ ((t_2\ \alpha_2)^m; l_2; \phi_2) \vdash e^* \vdash e^* : ti_4^*\ (t_1\ \alpha_1)^n; l_1'; \Gamma_1'; \phi_1' \to ti_4^*\ (t_2\ \alpha_2)^m; l_2; \Gamma_2; \phi_3$
  4. Then, by 2a and 2b, it follows that $S; C, \mathrm{label}\ ((t_2\ \alpha_2)^m; l_2; \phi_2) \vdash e^* \vdash e^* : ti_1'^*; l_1'; \Gamma_1'; \phi_1' \to ti_2^*; l_2; \Gamma_2; \phi_3$
  5. By Lemma Values-Any-Context and 1a, we have that $S; C, \mathrm{label}\ ((t_2\ \alpha_2)^m; l_2; \phi_2) \vdash (t.\mathsf{const}\ c)^n\ : ti_3^*\ ti_1^*; l_1; \Gamma_1; \phi_1 \to ti_1'^*; l_1'; \Gamma_1'; \phi_1'$
  6. Then, by Lemma Sequence-Composition, 4, and 5, we have that $S; C, \mathrm{label}\ ((t_2\ \alpha_2)^m; l_2; \phi_2) \vdash (t.\mathsf{const}\ c)^n\ e^* : ti_3^*\ ti_1^*; l_1; \Gamma_1; \phi_1 \to ti_2^*; l_2; \Gamma_2; \phi_2$
  7. By Rule Empty, $S; C \vdash \epsilon : ti_2^*; l_2; \Gamma_2; \phi_2 \to ti_2^*; l_2; \Gamma_2; \phi_2$.
  8. We have $\Gamma_2 \supseteq \Gamma_1, ti_2^*; l_2$ as a result of Lemma Well-Formedness
  9. Finally, by Rule Label, 6, 7, 8, and 2d, we have that $S; C \vdash \mathsf{label}_m\{\epsilon\}\ (t.\mathsf{const}\ c)^n\ e^*\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \to ti_2^*; l_2; \Gamma_2; \phi_2$
- Case: $S; C \vdash (t.\mathsf{const}\ c)^n\ \mathsf{loop}\ (t_1^n \to t_2^m)\ e^*\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \to ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ (t.\mathsf{const}\ c)^n\ \mathsf{loop}\ ti_3^n; l_3; \phi_3 \to ti_4^m; l_4; \phi_4\ e^*\ \mathsf{end} \hookrightarrow \mathsf{label}_n\{\mathsf{loop}\ ti_3^n; l_3; \phi_3 \to ti_4^m; l_4; \phi_4\ e^*\ \mathsf{end}\}\ (t.\mathsf{const}\ c)^n\ e^*\ \mathsf{end}$
This rule is similar to the above one, except that we must reason a little more about the stored instructions since we are storing the loop.
We want to show that $S; C \vdash \mathsf{label}_m\{\mathsf{loop}\ (t_1^n \to t_2^m)\ e^*\ \mathsf{end}\}\ (t.\mathsf{const}\ c)^n\ e^*\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \to ti_2^*; l_2; \Gamma_2; \phi_2$.
  1. The following hold by Lemma Inversion-On-Instruction-Typing on $S; C \vdash (t.\mathsf{const}\ c)^n\ \mathsf{loop}\ (t_1^n \to t_2^m)\ e^*\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \to ti_2^*; l_2; \Gamma_2; \phi_2$, for some $ti_1'^*$, l_1', Γ_1', ϕ_1', and ti_3^*
     a. $S; C \vdash (t.\mathsf{const}\ c)^n\ : ti_3^*\ ti_1^*; l_1; \Gamma_1; \phi_1 \to ti_1'^*; l_1'; \Gamma_1'; \phi_1'$
     b. $S; C \vdash \mathsf{loop}\ (t_1^n \to t_2^m)\ e^*\ \mathsf{end} : ti_1'^*; l_1'; \Gamma_1'; \phi_1' \to ti_3^*\ ti_2^*; l_2; \Gamma_2; \phi_2$
  2. Then, by Lemma Inversion-On-Instruction-Typing on $S; C \vdash \mathsf{loop}\ (t_1^n \to t_2^m)\ e^*\ \mathsf{end} : ti_1'^*; l_1'; \Gamma_1'; \phi_1' \to ti_3^*\ ti_2^*; l_2; \Gamma_2; \phi_2$, we have that
     a. $ti_1'^* = ti_4^*\ (t_1\ \alpha_1)^n$, for some ti_4^*
     b. $ti_2^* = ti_4^*\ (t_2\ \alpha_2)^m$

c. $S; C, \text{label}\ ((t_1\ \alpha_1)^m; l_1; \phi_1) \vdash e^* : (t_1\ \alpha_1)^n; l_1'; \Gamma_1'; \phi_1' \rightarrow (t_2\ \alpha_2)^m; l_2; \Gamma_2; \phi_3$

d. $\Gamma_2 \vdash \phi_3 \Rightarrow \phi_2$

3. By Rule Stack-Poly and 2c, we have that $S; C, \text{label}\ ((t_1\ \alpha_1)^m; l_1; \phi_1) \vdash e^* : ti_4^*\ (t_1\ \alpha_1)^n; l_1'; \Gamma_1'; \phi_1' \rightarrow ti_4^*\ (t_2\ \alpha_2)^m; l_2; \Gamma_2; \phi_3$
4. Then, by 2a and 2b, it follows that $S; C, \text{label}\ ((t_1\ \alpha_1)^m; l_1; \phi_1) \vdash e^* : ti_1^{'*}; l_1'; \Gamma_1'; \phi_1' \rightarrow ti_2^*; l_2; \Gamma_2; \phi_3$
5. By Lemma Values-Any-Context and 1a, we have that $S; C, \text{label}\ ((t_1\ \alpha_1)^m; l_1; \phi_1) \vdash (t.\text{const}\ c)^n\ : ti_3^*\ ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_1^{'*}; l_1'; \Gamma_1'; \phi_1'$
6. Then, by Lemma Sequence-Composition, 4, and 5, we have that $S; C, \text{label}\ ((t_1\ \alpha_1)^m; l_1 \phi_1) \vdash (t.\text{const}\ c)^n\ e^* : ti_3^*\ ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
7. We have $\Gamma_1 \supseteq \Gamma_1, ti_1^*; l_1$ as a result of Lemma Well-Formedness
8. Finally, by Rule Label, 6, 7, and 2d, we have that $S; C \vdash \text{label}_m\{\text{loop}\ (t_1^n \rightarrow t_2^m)\ e^*\ \text{end}\}\ (t.\text{const}\ c)^n\ e^*\ \text{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$

- Case: $S; C \vdash (\text{i32.const}\ 0)\ \text{if}\ (ti_3^n \rightarrow ti_4^m)\ e_1^*\ \text{else}\ e_2^*\ \text{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ (\text{i32.const}\ 0)\ \text{if}\ (ti_3^n \rightarrow ti_4^m)\ e_1^*\ \text{else}\ e_2^*\ \text{end} \hookrightarrow \text{block}\ (ti_3^n \rightarrow ti_4^m)\ e_2^*\ \text{end}$
We want to show that
$$\text{block}\ (ti_3^n \rightarrow ti_4^m)\ e_1^*\ \text{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$$
First, we reason about $ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
1. By Lemma Inversion-On-Instruction-Typing on $S; C \vdash (\text{i32.const}\ 0)\ \text{if}\ (ti_3^n \rightarrow ti_4^m)\ e_1^*\ \text{else}\ e_2^*\ \text{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, we have that
a. $S; C \vdash (t.\text{const}\ 0)^n\ : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_4^*; l_4; \Gamma_4; \phi_4$

b. $S; C \vdash \text{if}\ (ti_3^n \rightarrow ti_4^m)\ e_1^*\ \text{else}\ e_2^*\ \text{end} : ti_4^*; l_4; \Gamma_4; \phi_4 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
2. By Lemma Type-Of-Values and 1a, we have that $ti_4^* = ti_1^*\ (\text{i32}\ \alpha)$, $l_4 = l_1$, $\Gamma_4 = \Gamma_1, (\text{i32}\ \alpha)$, and $\phi_4 = \phi_1, (=\ \alpha\ (\text{i32}\ \alpha))$
3. Then, by Lemma Inversion-On-Instruction-Typing on $S; C \vdash \text{if}\ (ti_3^n \rightarrow ti_4^m)\ e_1^*\ \text{else}\ e_2^*\ \text{end} : ti_1^*\ (\text{i32}\ \alpha); l_1; \Gamma_1, (\text{i32}\ \alpha); \phi_1, (=\ \alpha\ (\text{i32}\ \alpha)) \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, we have that $S; C, \text{label}(ti_2^m; l_2; \phi_2) \vdash e_2^* : ti_1^*; l_1; \Gamma_1, (\text{i32}\ \alpha); \phi_1, (=\ \alpha\ (\text{i32}\ \alpha)) \rightarrow ti_2^*; l_2; \Gamma_2; \phi_5$, where $\Gamma_2 \vdash \phi_5 \Rightarrow \phi_2$
4. Then, by Rule Block, we have that $S, C \vdash \text{block}\ (ti_3^n \rightarrow ti_4^m)\ e_2^*\ \text{end} : ti_1^*; l_1; \Gamma_1, (\text{i32}\ \alpha); \phi_1, (=\ \alpha\ (\text{i32}\ \alpha)) \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$

- Case: $S; C \vdash (\text{i32.const}\ k+1)\ \text{if}\ (ti_3^n; l_3; \phi_3 \rightarrow ti_4^m; l_4; \phi_4)\ e_1^*$
else e_2^*
end
$: ti_1^*; l_1; \phi_1 \rightarrow ti_2^*; l_2; \phi_2$
$\wedge\ (\text{i32.const}\ k+1)\ \text{if}\ ti_3^n; l_3; \phi_3 \rightarrow ti_4^m; l_4; \phi_4\ e_1^*\ \text{else}\ e_2^*\ \text{end}$
$\hookrightarrow \text{block}\ ti_3^n; l_3; \phi_3 \rightarrow ti_4^m; l_4; \phi_4\ e_1^*\ \text{end}$
This case is the same as above, except with e_2 instead of e_1 and $k+1$ instead of 0.

- Case: $S; C \vdash \text{label}_n\{e^*\}\ (t.\text{const}\ c)^n\ \text{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ \text{label}_n\{e^*\}\ (t.\text{const}\ c)^*\ \text{end} \hookrightarrow (t.\text{const}\ c)^*$
By Lemma Inversion-On-Instruction-Typing on $S; C \vdash \text{label}_n\{e^*\}\ (t.\text{const}\ c)^n\ \text{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, we have that $ti_2^* = ti_1^*\ ti^n$.
We have $S; C \vdash (t.\text{const}\ c)^n : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (t\ \alpha_3)^n; (t_{l2}\ \alpha_{l3})^*; \Gamma_3; \phi_3$, where $(t\ \alpha)^n = ti^*$, $(t_{l2}\ \alpha_{l2})^* = l_2; \Gamma_3 \vdash \phi_3 \Rightarrow \phi_2[\alpha \mapsto \alpha_3][\alpha_{l2} \mapsto \alpha_{l3}]$; and $ti^*\ l_2 \notin \Gamma_1$, as they are premises of Rule Label, which we have assumed to hold.
By Lemma Inversion-On-Instruction-Typing on $S; C \vdash (t.\text{const}\ c)^n : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (t\ \alpha_3)^n; l_1; \Gamma_3; \phi_3$, we know $(t_{l2}\ \alpha_{l2})^* = l_2 = l_1$, $\Gamma_3 = \Gamma_1, (t\ \alpha_3)^n$, and $\phi_3 = \phi_1, (=\alpha_3\ (t\ c))^n$.

Since $ti^*\ l_2 \notin \Gamma_1$, we can get $S; C \vdash (t.\mathsf{const}\ c)^* : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (t\ \alpha)^*; l_1; \Gamma_1, (t\ \alpha)^*; \phi_1, (= \alpha\ (t\ c))^*$ by Rule Const. Then, by Rule Const,

$$S; C \vdash (t.\mathsf{const}\ c)^* : \epsilon; (t_{l2}\ \alpha_{l2})^*; \Gamma_1[\alpha_{l1} \mapsto \alpha_{l2}]; \phi_1[\alpha_{l1} \mapsto \alpha_{l2}] \\ \rightarrow (t\ \alpha)^*; (t_{l2}\ \alpha_{l2})^*; \Gamma_1[\alpha_{l1} \mapsto \alpha_{l2}], (t\ \alpha)^*; \phi_1[\alpha_{l1} \mapsto \alpha_{l2}], (= \alpha\ (t\ c))^*$$

Therefore,

$$S; C \vdash (t.\mathsf{const}\ c)^* : ti_1^*; (t_{l2}\ \alpha_{l2})^*; \Gamma_1[\alpha_{l1} \mapsto \alpha_{l2}]; \phi_1[\alpha_{l1} \mapsto \alpha_{l2}] \\ \rightarrow ti_2^*; (t_{l2}\ \alpha_{l2})^*; \Gamma_1[\alpha_{l1} \mapsto \alpha_{l2}], (t\ \alpha)^*; \phi_1[\alpha_{l1} \mapsto \alpha_{l2}], (= \alpha\ (t\ c))^*$$

since $\Gamma_1[\alpha_{l1} \mapsto \alpha_{l2} \vdash, \Rightarrow (t\ \alpha)^*]\phi_1[\alpha_{l1} \mapsto \alpha_{l2}], (= \alpha\ (t\ c))^*\phi_2$ and $ti_2^* = ti_1^*\ (t\ \alpha)^*$.

- Case: $S; C \vdash \mathsf{label}_n\{e^*\}\ \mathsf{trap}\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ \mathsf{label}_n\{e^*\}\ \mathsf{trap}\ \mathsf{end} \hookrightarrow \mathsf{trap}$
  Trivially, $C \vdash \mathsf{trap} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ by Rule Trap since trap accepts any precondition and postcondition.

- Case: $S; C \vdash \mathsf{label}_n\{e^*\}\ L^j[(t.\mathsf{const}\ c)^n\ (\mathsf{br}\ j)]\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ \mathsf{label}_n\{e^*\}\ L^j[(t.\mathsf{const}\ c)^n\ (\mathsf{br}\ j)] \hookrightarrow (t.\mathsf{const}\ c)^n\ e^*$
  Intuitively, this proof works because the premise of Rule Br assumes that $C_{\text{label}}(i)$ is the precondition ($ti_3^n; l_3; \phi_3$, as we will soon see) of the stored instructions e^* in the $i+1$th label, and the postcondition of the label block is immediately reachable from the postcondition of e^*. Meanwhile, that assumption is ensured by Rule Label, which ensures that e^* has the same precondition as the $i+1$th branch postcondition on the label stack and the same postcondition as the label block instruction.
  First, we derive the type of $(t.\mathsf{const}\ c)^n$ from the precondition of Rule Br.
  1. By Lemma Inversion-On-Instruction-Typing and $S; C \vdash \mathsf{label}_n\{e^*\}\ L^j[(t.\mathsf{const}\ c)^n\ (\mathsf{br}\ j)]\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, we have
     a. $S; C \vdash e^* : ti_3^n; l_3; \Gamma_3; \phi_3 \rightarrow ti_5^*; l_2; \Gamma_2; \phi_4$
     b. $\Gamma_2 \vdash \phi_4 \Rightarrow \phi_2$
     c. $\Gamma_3 \supset \Gamma_1, ti_3^n, l_3$
     d. $S; C, \mathrm{label}(ti_3^n; l_3; \phi_3) \vdash L^j[(t.\mathsf{const}\ c)^n\ (\mathsf{br}\ j)] : ti_0^*\ ti_4^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_0^*\ ti_5^*; l_2; \Gamma_2; \phi_5$, where $ti_1^* = ti_0^*\ ti_4^*$ and $ti_2^* = ti_0^*\ ti_5^*$ for some ti_0^*
  2. Then, by Lemma Values-Br-In-Context and 1d, we have that
     a. $S; C' \vdash (t.\mathsf{const}\ c)^n : ti_1^*; l_1; \Gamma_1, (t_0\ \alpha_0)^*; \phi_1, (= \alpha_0\ (t_0\ c_0))^* \rightarrow ti_1^*\ ti_3^n; l_1; \Gamma_1, (t_0\ \alpha_0)^*, ti_3^n; \phi_1, (= \alpha_0\ (t_0\ c_0))^*, (= \alpha_3\ (t\ c))^n$, for some α_0^*, t_0^*, and c_0^*, where $ti_3^n = (t\ \alpha_3)^n$
     b. $\Gamma_1, (t_0\ \alpha_0)^*, ti_3^n \vdash \phi_1, (= \alpha_0\ (t_0\ c_0))^*, (= \alpha_3\ (t\ c))^n \Rightarrow \phi_3[l_3 \mapsto l_1]$
     c. l_3 is fresh with respect to Γ_1 and Γ_2
  3. By assumption 3, we know $(\vdash v : (t_v\ \alpha_v); \phi_v)^*$, where $l_1 = (t_v\ \alpha_v)^*$ and $\phi_v \subset \phi_1$
  4. Trivially then, we have $(\vdash v : (t_v\ \alpha_{v3}); \phi_{v3})^*$, where $l_3 = (t_v\ \alpha_{v3})^*$
  5. Replacing ϕ_v with ϕ_{v3} in ϕ_1 to obtain $\hat{\phi}_1$, we have that $\Gamma_1, (t_0\ \alpha_0)^*, ti_3^n \vdash p\hat{h}i_1, (= \alpha_0\ (t_0\ c_0))^*, (= \alpha_3\ (t\ c))^n \Rightarrow \phi_3$
  6. We choose $(t\ \alpha)^*$ to be the set difference between Γ_3 and $\Gamma_1, (t_0\ \alpha_0)^*, l_3, ti_3^n$
  7. Then, by Lemma Values-Any-Context, and 2a, we have that $S; C \vdash (t.\mathsf{const}\ c)^n : ti_0^*; l_3; \Gamma_1, (t\ \alpha)^*, l_3, (t_0\ \alpha_0)^*; p\hat{h}i_1, (= \alpha_0\ (t_0\ c_0))^*, (= \alpha_3\ (t\ c))^n \rightarrow ti_0^*\ ti_3^n; l_3; \Gamma_3; p\hat{h}i_1, (= \alpha_0\ (t_0\ c_0))^*, (= \alpha_3\ (t\ c))^n$
  8. Then, by Lemma Strengthening, 5, and 1a, we have that $S; C \vdash e^* : ti_0^*\ ti_3^n; l_3; \Gamma_3; p\hat{h}i_1, (= \alpha_0\ (t_0\ c_0))^*, (= \alpha_3\ (t\ c))^n \rightarrow ti_2^*; l_2; \Gamma_2; \phi_6$, where $\Gamma_2 \vdash \phi_6 \Rightarrow \phi_2$

9. Finally, by Lemma Sequence-Composition, we have that $S; C \vdash (t.\mathsf{const}\ c)^n\ e^* : ti_0^*; l_3; \Gamma_1, (t\ \alpha)^*, l_3, (t_0\ \alpha_0)^*; \hat{phi}_1, (= \alpha_0\ (t_0\ c_0))^*, (= \alpha_3\ (t\ c))^n \rightarrow ti_2^*; l_2; \Gamma_2; \phi_6$

- Case: $S; C \vdash (\mathsf{i32.const}\ 0)\ (\mathsf{br\_if}\ j) : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ (\mathsf{i32.const}\ 0)\ (\mathsf{br\_if}\ j) \hookrightarrow \epsilon$
In the case that br_if does not branch, it acts exactly like drop (consumes (i32.const 0) and reduces to the empty sequence). Thus, this case is the same as the drop case.

- Case: $S; C \vdash (\mathsf{i32.const}\ k+1)\ (\mathsf{br\_if}\ j) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ (\mathsf{i32.const}\ k+1)\ (\mathsf{br\_if}\ j) \hookrightarrow \mathsf{br}\ j$
We want to show that $S; C \vdash \mathsf{br}\ j : ti_1^*; l_1; \Gamma_1, (\mathsf{i32}\ \alpha); \phi_1, (= \alpha\ (\mathsf{i32}\ k+1)) \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
We know $S; C \vdash \mathsf{br\_if}\ j : ti_1^*\ (\mathsf{i32}\ \alpha); l_1; \Gamma_1, (\mathsf{i32}\ \alpha); \phi_1, (= \alpha\ (\mathsf{i32}\ k+1)) \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, where $\alpha \notin \Gamma_1$ by Lemma Inversion-On-Instruction-Typing on $S; C \vdash (\mathsf{i32.const}\ k+1)\ (\mathsf{br\_if}\ j) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
Then we know $C_{\text{label}}(j) = (ti_3^*; l_3; \phi_3)$, where $ti_1^* = ti_0^*\ (t_1\ \alpha_1)^*$, $ti_3^* = (t_1\ \alpha_3)^*$, $l_1 = (t_l\ \alpha_{l1})^*$, $l_3 = (t_l\ \alpha_{l3})^*$, and $\Gamma_1, (\mathsf{i32}\ \alpha) \vdash \phi_1, \neg(= \alpha\ (\mathsf{i32}\ 0)) \Rightarrow \phi_3[\alpha_3 \mapsto \alpha_1]^*[\alpha_{l3} \mapsto \alpha_{l1}]^*$ by Lemma Inversion-On-Instruction-Typing on $S; C \vdash \mathsf{br\_if}\ j : ti_1^*\ (\mathsf{i32}\ \alpha); l_1; \Gamma_1; \phi_1, (= \alpha\ (\mathsf{i32}\ k+1)) \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
Then we have $S; C \vdash \mathsf{br}\ j : ti_0^*\ (t_1\ \alpha_1)^*; l_1; \Gamma_1, (\mathsf{i32}\ \alpha); \phi_1, (= \alpha\ (\mathsf{i32}\ k+1)) \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ by Rule Br.

- Case: $S; C \vdash (\mathsf{i32.const}\ k)\ (\mathsf{br\_table}\ j_1^k\ j\ j_2^*) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ (\mathsf{i32.const}\ k)\ (\mathsf{br\_table}\ j_1^k\ j\ j_2^*) \hookrightarrow \mathsf{br}\ j$
We want to show that $S; C \vdash \mathsf{br}\ j : ti_1^*; l_1; \Gamma_1, (\mathsf{i32}\ \alpha); \phi_1, (= \alpha\ (\mathsf{i32}\ k)) \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
This case is similar in structure to the (i32.const $k+1$) (br_if j) case.
We know $S; C \vdash \mathsf{br\_table}\ j_1^k\ j\ j_2^* : ti_1^*\ (\mathsf{i32}\ \alpha); l_1; \Gamma_1, (t\ \alpha); \phi_1, (= \alpha\ (\mathsf{i32}\ k)) \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ by Lemma Inversion-On-Instruction-Typing on $S; C \vdash (\mathsf{i32.const}\ k)\ (\mathsf{br\_table}\ j_1^k\ j\ j_2^*) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
Then we know $C_{\text{label}}(j) = (ti_3^*; l_1; \phi_3)$, where $ti_1^* = ti_0^*\ (t_1\ \alpha_1)^*$, $ti_3^* = (t_1\ \alpha_3)^*$, $l_1 = (t_l\ \alpha_{l1})^*$, $l_3 = (t_l\ \alpha_{l3})^*$, and $\Gamma_1, (\mathsf{i32}\ \alpha) \vdash \phi_1, (= \alpha\ (\mathsf{i32}\ k)) \Rightarrow \phi_3[\alpha_3 \mapsto \alpha_1]^*[\alpha_{l3} \mapsto \alpha_{l1}]^*$ by Lemma Inversion-On-Instruction-Typing on $S; C \vdash \mathsf{br\_table}\ j_1^k\ j\ j_2^* : ti_1^*\ (\mathsf{i32}\ \alpha); l_1; \Gamma_1, (\mathsf{i32}\ \alpha); \phi_1, (= \alpha\ (\mathsf{i32}\ k)) \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
Therefore we have $S; C \vdash \mathsf{br}\ j : ti_0^*\ (t_1\ \alpha_1)^*; l_1; \Gamma_1, (\mathsf{i32}\ \alpha); \phi_1, (= \alpha\ (\mathsf{i32}\ k)) \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ by Rule Br.

- Case: $C \vdash (\mathsf{i32.const}\ k+n)\ (\mathsf{br\_table}\ j_1^k\ j) : ti_1^*; l_1; \phi_1 \rightarrow ti_2^*; l_2; \phi_2$
$\wedge\ (\mathsf{i32.const}\ k+n)\ (\mathsf{br\_table}\ j_1^k\ j) \hookrightarrow \mathsf{br}\ j$
Same as above.

- Case: $S; C \vdash \mathsf{call}\ j : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ s; v^*; \mathsf{call}\ j \hookrightarrow_i \mathsf{call}\ s; v^*; s_{\text{func}}(i, j)$
We want to show that $S; C \vdash \mathsf{call}\ s_{\text{func}}(i, j) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
By Lemma Inversion-On-Instruction-Typing on $S; C \vdash \mathsf{call}\ j : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, we know that $l_2 = l_1$, $ti_1^* = ti_0^*\ (t_3\ \alpha_3)^*$, $ti_2^* = ti_0^*\ (t_4\ \alpha_4)^*$, $\Gamma_1 \vdash \phi_1 \Rightarrow \phi_3[\alpha_5 \mapsto \alpha_3]$, $\phi_2 = \phi_1 \cup \phi_4[\alpha_5 \mapsto \alpha_3][\alpha_6 \mapsto \alpha_4]$, and $\Gamma_2 = \Gamma_1, (t_4\ \alpha_4)^*$ where $(t_3\ \alpha_5)^*; \phi_3 \rightarrow (t_4\ \alpha_6)^*; \phi_4 = C_{\text{func}}(j)$, and $(t_4\ \alpha_4)^* \notin \Gamma_1$.
Thus, we want to show that $S; C \vdash \mathsf{call}\ s_{\text{func}}(i, j) : ti_0^*\ (t_3\ \alpha_3)^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_0^*\ (t_4\ \alpha_4)^*; l_2; \Gamma_1, (t_3\ \alpha_3)^*; \phi_1 \cup \phi_4[\alpha_5 \mapsto \alpha_3][\alpha_6 \mapsto \alpha_4]$.
Then we know $S \vdash s_{\text{func}}(i, j) : (t_3\ \alpha_5)^*; \phi_3 \rightarrow (t_4\ \alpha_6)^*; \phi_4$ because it is a premise of $S \vdash s_{\text{inst}}(i) : C$, which is a premise of $\vdash s : S$.
Therefore, $S; C \vdash \mathsf{call}\ s_{\text{func}}(i, j) : (t_3\ \alpha_3)^*; l_1; \Gamma_1; \phi_1 \rightarrow (t_4\ \alpha_4)^*; l_1; \Gamma_1, (t_4\ \alpha_4)^*; \phi_1 \cup \phi_4[\alpha_5 \mapsto \alpha_3][\alpha_6 \mapsto \alpha_4]$ by Rule Call-Cl.
Thus, $S; C \vdash \mathsf{call}\ s_{\text{func}}(i, j) : ti_0^*\ (t_3\ \alpha_3)^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_0^*\ (t_4\ \alpha_4)^*; l_2; \Gamma_1, (t_4\ \alpha_4)^*; \phi_1 \cup \phi_4[\alpha_5 \mapsto \alpha_3][\alpha_6 \mapsto \alpha_4]$ by Rule Stack-Poly and since $l_1 = l_2$.

- Case: $S; C \vdash (\mathsf{i32.const}\ j)\ \mathsf{call\_indirect}\ ((t_3\ \alpha_5)^*; \phi_3 \rightarrow (t_4\ \alpha_6)^*; \phi_4) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ s; (\mathsf{i32.const}\ j)\ \mathsf{call\_indirect}\ ((t_3\ \alpha_5)^*; \phi_3 \rightarrow (t_4\ \alpha_6)^*; \phi_4) \hookrightarrow_i \mathsf{call}\ s_{\text{tab}}(i, j)$
where $s_{\text{tab}}(i, j)_{\text{code}} = (\mathsf{func}\ (t_3\ \alpha_5)^*; \phi_3 \rightarrow (t_4\ \alpha_6)^*; \phi_4\ \mathsf{local}\ t^*\ e^*)$
We want to show that $\mathsf{call}\ s_{\text{tab}}(i, j) : ti_1^*; l_1; \Gamma_1, (\mathsf{i32}\ \alpha); \phi_1, (= \alpha\ (\mathsf{i32}\ j)) \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$. The extra index variable is added to the precondition since the instruction that creates it is reduced away.

We know that $S; C \vdash \text{call\_indirect}\ ((t_3\ \alpha_5)^*; \phi_3 \rightarrow (t_4\ \alpha_6)^*; \phi_4) : ti_1^*\ (\text{i32}\ \alpha); l_1; \Gamma_1, (\text{i32}\ \alpha); \phi_1, (= \alpha\ (\text{i32}\ j)) \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ by Lemma Inversion-On-Instruction-Typing on $S; C \vdash (\text{i32.const}\ j)\ \text{call\_indirect}\ ((t_3\ \alpha_5)^*; \phi_3 \rightarrow (t_4\ \alpha_6)^*; \phi_4) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
By Lemma Inversion-On-Instruction-Typing on $S; C \vdash \text{call\_indirect}\ ((t_3\ \alpha_5)^*; \phi_3 \rightarrow (t_4\ \alpha_6)^*; \phi_4) : ti_1^*\ (\text{i32}\ \alpha); l_1; \Gamma_1, (\text{i32}\ \alpha); \phi_1, (= \alpha\ (\text{i32}\ j)) \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, we know that $ti_1^* = ti_0^*\ (t_3\ \alpha_3)^*$, and $ti_2^* = ti_0^*\ (t_4\ \alpha_4)^*$ for some ti_0^*, $l_1 = l_2$, $\Gamma_1 \vdash \phi_1 \Rightarrow \phi_3[\alpha_5 \mapsto \alpha_3]$, $\Gamma_2 = \Gamma_1, (\text{i32}\ \alpha), (t_4\ \alpha_4)^*$, and $\phi_2 = \phi_1, (= \alpha\ (\text{i32}\ j)) \cup \phi_4[\alpha_5 \mapsto \alpha_3][\alpha_6 \mapsto \alpha_4]$.
We know $S \vdash s_{\text{tab}}(i, j) : (t_3\ \alpha_5)^*; \phi_3 \rightarrow (t_4\ \alpha_6)^*; \phi_4$ since it is a premise of $\vdash s : S$ which we have assumed to hold.
Then, $S; C \vdash \text{call}\ s_{\text{tab}}(i, j) : (t_3\ \alpha_3)^*; l_1; \Gamma_1, (\text{i32}\ \alpha); \phi_1, (= \alpha\ (\text{i32}\ j)) \rightarrow (t_4\ \alpha_4)^*; l_1; \Gamma_1, (\text{i32}\ \alpha), (t_4\ \alpha_4)^*; \phi_1, (= \alpha\ (\text{i32}\ j)) \cup \phi_4[\alpha_5 \mapsto \alpha_3][\alpha_6 \mapsto \alpha_4]$ by Rule Call-Cl.
Therefore, $S; C \vdash \text{call}\ s_{\text{tab}}(i, j) : ti_0^*\ (t_3\ \alpha_3)^*; l_1; \Gamma_1, (\text{i32}\ \alpha); \phi_1, (= \alpha\ (\text{i32}\ j)) \rightarrow ti_0^*\ (t_4\ \alpha_4)^*; l_1; \Gamma_1, (\text{i32}\ \alpha), (t_4\ \alpha_4); \phi_1, (= \alpha\ (\text{i32}\ j)) \cup \phi_4[\alpha_5 \mapsto \alpha_3][\alpha_6 \mapsto \alpha_4]$ by Rule Stack-Poly.

- Case: $S; C \vdash (\text{i32.const}\ j)\ \text{call\_indirect}✓\ (ti_3^*; l_3; \Gamma_3; \phi_3 \rightarrow ti_4^*; l_4; \Gamma_4; \phi_4) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ s; (\text{i32.const}\ j)\ \text{call\_indirect}✓\ ti_3^*; l_3; \Gamma_3; \phi_3 \rightarrow ti_4^*; l_4; \Gamma_4; \phi_4 \hookrightarrow_i \text{call}\ s_{\text{tab}}(i, j)$
  where $s_{\text{tab}}(i, j)_{\text{code}} = (\text{func}\ ti_3^*; \epsilon; \Gamma_3; \phi_3 \rightarrow ti_4^*; \epsilon; \Gamma_4; \phi_4\ \text{local}\ t^*\ e^*)$
  Same as above.

- Case: $S; C \vdash (\text{i32.const}\ j)\ \text{call\_indirect}\ tfi : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ s; (\text{i32.const}\ j)\ \text{call\_indirect}\ tfi \hookrightarrow_i \text{trap}$.
  Trivially, $S; C \vdash \text{trap} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ by Rule Trap.

- Case: $S; C \vdash v^n\ \text{call}\ cl : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ s; v^n\ \text{call}\ cl \hookrightarrow_i \text{local}_m\ \{i; v^n\ (t.\text{const}\ 0)^k\}$
  $\qquad \text{block}\ tfi$
  $\qquad\qquad e^*$
  $\qquad \text{end}$
  $\quad \text{end}$
  where $cl_{\text{code}} = \text{func}\ (ti_3^n; \phi_3 \rightarrow ti_4^m; \phi_4)\ \text{local}\ t^k\ e^*$ and $cl_{\text{inst}} = i$
  Let $tfi = \epsilon; ti_3^n\ (t\ \alpha)^k; \phi_3 \rightarrow (t_4\ \alpha_5)^m; l_4; \phi_4[\alpha_4 \mapsto \alpha_5]$
  We want to show that

$$S; C \vdash \text{local}_m\ \{i; v^n\ (t.\text{const}\ 0)^k\}\ (\text{block}\ (\epsilon; ti_3^n\ (t_2\ \alpha_2)^n; \phi_3 \rightarrow ti_4^m; l_4; \phi_4)\ e^*\ \text{end})\ \text{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$$

By Lemma Inversion-On-Instruction-Typing on Rule Composition, Rule Const, and Rule Call-Cl, we know $l_2 = l_1$, $ti_2^* = ti_1^*\ (t_4\ \alpha_5)^m$, $\Gamma_1 \vdash \phi_1, (t_2\ \alpha_2), (\text{eq}\ \alpha_2\ (t_2\ c)) \Rightarrow \phi_3[\alpha_3 \mapsto \alpha_2]$, $\phi_2 = \phi_1, (= \alpha_2\ (t_2\ c))^n \cup \phi_4[\alpha_4 \mapsto \alpha_5][\alpha_3 \mapsto \alpha_2]$, $\Gamma_2 = \Gamma_1, (t_2\ \alpha_2)^n, (t_4\ \alpha_5)^m$, $(t_4\ \alpha_5)^m \notin \Gamma_1$ and $S \vdash cl : ti_3^n; \phi_3 \rightarrow ti_4^m; \phi_4$, where $(t_4\ \alpha_4)^m = ti_4^m$.
We also know that

$$S; C \vdash (t_2.\text{const}\ c)^n : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_1^*\ (t_2\ \alpha_2)^n; l_1; \Gamma_1, (t_2\ \alpha_2)^n; \phi_1, (= \alpha_2\ (t_2\ c))^n$$

where $v^n = (t_2.\text{const}\ c)^n$, and

$$\begin{aligned} S; C \vdash \text{call}\ cl : {} & ti_1^*\ (t_2\ \alpha_2)^n; l_1; \Gamma_1, (t_2\ \alpha_2)^n; \phi_1, (= \alpha_2\ (t_2\ c))^n \\ & \rightarrow (t_4\ \alpha_5)^m\ ti_2^m; l_1; \Gamma_1, (t_2\ \alpha_2)^n, (t_4\ \alpha_5)^m; \phi_1, (= \alpha_2\ (t_2\ c))^n \cup \phi_4[\alpha_4 \mapsto \alpha_5][\alpha_3 \mapsto \alpha_2] \end{aligned}$$

by Rule Inversion-On-Instruction-Typing on Rule Composition.
We have $C \vdash \text{func}\ ti_3^n; \phi_3 \rightarrow ti_4^m; \phi_4\ \text{local}\ t^k\ e^* : ti_3^n; \phi_3 \rightarrow ti_4^m; \phi_4$ because it is a premise of $S \vdash cl : ti_3^n; \phi_3 \rightarrow ti_4^m; \phi_4$.
Then,

$$\begin{aligned} & S; C, \text{local}\ t_2^n\ t^k, && \vdash e^* : \epsilon; (t_2\ \alpha_2)^n\ (t\ \alpha)^k; \emptyset, (t_2\ \alpha_2)^n\ (t\ \alpha)^k; (\phi_3, (= \alpha\ (t\ 0))^k)[\alpha_3 \mapsto \alpha_2] \\ & \quad \text{label}\ (ti_4^m; l_4; \phi_4), && \quad \rightarrow (t_4\ \alpha_6)^m; l_4; \Gamma_6; \phi_6 \\ & \quad \text{return}\ ((t_4\ \alpha_5)^m; \phi_4[\alpha_4 \mapsto \alpha_5]) && \end{aligned}$$

where $\Gamma_6 \vdash \phi_6 \Rightarrow \phi_4[\alpha_4 \mapsto \alpha_6]$ because it is a premise of the above derivation.
We can now reconstruct the type after reduction.

$$\begin{aligned} & S; C, \text{local}\ t_2^n\ t^k, && \vdash \text{block}\ tfi\ e^*\ \text{end} : (\epsilon; ti_3^n\ (t_2\ \alpha_2)^n; \Gamma_1; \phi_3 \\ & \quad \text{return}\ ((t_4\ \alpha_5)^m; \phi_4[\alpha_4 \mapsto \alpha_5]) && \quad \rightarrow (t_4\ \alpha_5)^m; l_4; \Gamma_1, (t_4\ \alpha_5)^m, l_4; \phi_1 \cup \phi_4[\alpha_4 \mapsto \alpha_5][\alpha_3 \mapsto \alpha_2]) \end{aligned}$$

by Rule Block, since $(t_4\ \alpha_5)^m \notin \Gamma_1$.
$\vdash v : (t_2\ \alpha_2); \emptyset, (t_2\ \alpha_2), (\mathsf{eq}\ \alpha_2\ (t_2\ c)))^n$ by Rule Admin-Const, and $(\vdash (t\mathsf{const}\ 0) : (t\ \alpha); \emptyset, (t\ \alpha), (\mathsf{eq}\ a\ (t\ 0)))^k$ by Rule Admin-Const.
Then, $S; (ti_4^m; \phi_4)\ \vdash v^n\ (t\mathsf{const}\ 0)^k; \mathsf{block}\ tfi\ e^*\ \mathsf{end} : (t_4\ \alpha_5)^m; l_4; \Gamma_1, (t_4\ \alpha_5)^m, l_4; \phi_1 \cup \phi_4[\alpha_4 \mapsto \alpha_5][\alpha_3 \mapsto \alpha_2])$ by Rule Code.
$S; C \vdash \mathsf{local}_m\{j; v^n\ (t.\mathsf{const}\ 0)^k\}\ \mathsf{block}\ tfi\ e^*\ \mathsf{end}\ \mathsf{end} : ti_1^*; l_1; \Gamma_1, (t_2\ \alpha_2)^n; \phi_1, (= \alpha_2\ (t_2\ c))^n \epsilon; l_1; \Gamma_1, (t_2\ \alpha_2)^n; \phi_1, (= \alpha_2\ (t_2\ c))^n \rightarrow (t_4\ \alpha_5)^m; l_4; \Gamma_1, (t_4\ \alpha_5)^m; \phi_1 \cup \phi_4[\alpha_4 \mapsto \alpha_5][\alpha_3 \mapsto \alpha_2])$ by Rule Local.
Finally,

$$\begin{aligned} S; C \vdash\ & \mathsf{local}_m\{j; v^n\ (t.\mathsf{const}\ 0)^k\}\ \mathsf{block}\ tfi\ e^*\ \mathsf{end}\ \mathsf{end} \\ :\ & ti_1^*; l_1; \Gamma_1, (t_2\ \alpha_2)^n; \phi_1, (= \alpha_2\ (t_2\ c))^n \epsilon; l_1; \Gamma_1, (t_2\ \alpha_2)^n; \phi_1, (= \alpha_2\ (t_2\ c))^n \\ & \rightarrow ti_1^*\ (t_4\ \alpha_5)^m; l_4; \Gamma_1, (t_4\ \alpha_5)^m; \phi_1 \cup \phi_4[\alpha_4 \mapsto \alpha_5][\alpha_3 \mapsto \alpha_2]) \end{aligned}$$

by Rule Stack-Poly.

- Case: $S; C \vdash \mathsf{local}_n\{i; v_l^*\}\ v^n\ \mathsf{end} : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ \mathsf{local}_n\{i; v_l^*\}\ v^n\ \mathsf{end} \hookrightarrow_j\ v^n$
  We want to show that $v^n : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma'; \phi'$, where $\Gamma' \vdash \phi' \Rightarrow \phi_2$.
  First we derive the type of v^n from the precondition of the local.
  We have $S; (ti^n; \phi_3) \vdash_i v_l^*; v^n : (t\ \alpha)^n; l_3; \Gamma_3; \phi_3$, as it is a premise of Rule Local that we have assumed to hold.
  By Lemma Inversion-On-Instruction-Typing on Rule Local, $ti_2^* = ti_1^*\ (t\ \alpha)^n$, $l_1 = l_2$, $\Gamma_3 = \Gamma_1, ti^n$, and $\phi_2 = \phi_1 \cup \phi_3$.
  $(\vdash v_l : ti_l; \phi_l)^*$ and $S; C_l \vdash v^n : \epsilon : ti_l^*; \emptyset, ti_l^*; \phi_l^* \rightarrow (t\ \alpha)^n; l_3; \Gamma_3; \phi_3$ because they are premises of Rule Code which we have assumed to hold.
  $ti_l = (t_l\ \alpha_l)^*$, and $\phi_l^* = \emptyset, (= \alpha_l\ (t_l\ c_l))^*$ because they are premises of Rule Admin-Const which we have assumed to hold.
  By Lemma Inversion-On-Instruction-Typing on Rule Const, $\Gamma_3 = ti_l^*, (t\ \alpha)^n$, $\phi_3 = \phi_l^*, (= \alpha\ (t\ c))^n$, where $v^n = t.\mathsf{const}\ c$.
  $S; C \vdash v^n : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (t\ \alpha)^n; l_1; \Gamma_1, (t\ \alpha)^n; \phi_1, (= \alpha\ (t\ c))^n$ by Rule Const and Rule Composition.
  By Lemma Well-formedness, $l_1 \subset \Gamma_1$, so $(t\ \alpha)^n, l_1 \subset \Gamma_1, (t\ \alpha)^n$.
  Trivially, $\Gamma_1, \Gamma_v, (t_l\ \alpha_l)^* \vdash \phi_1, (= \alpha\ (t\ c))^n, (= \alpha_l\ (t_l\ c_l))^* \Rightarrow \phi_2$ since $\phi_2 = \phi_1 \cup \phi_3$ and $\phi_3 = \emptyset, (= \alpha_l\ (t_l\ c_l))^*, (= \alpha\ (t\ c))^n$.
  Finally, $S; C \vdash v^n : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_1^*\ (t\ \alpha)^n; l_1; \Gamma_1, (t\ \alpha)^n; \phi_1, (= \alpha\ (t\ c))^n$, by Rule Stack-Poly.

- Case: $S; C \vdash \mathsf{local}_n\{i; v_l^*\}\ \mathsf{trap}\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ \mathsf{local}_n\{i; v_l^*\}\ \mathsf{trap}\ \mathsf{end} \hookrightarrow\ \mathsf{trap}$
  Trivially, $S; C \vdash \mathsf{trap} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ by Rule Trap.

- Case: $S; C \vdash \mathsf{local}_n\{i; v_l^*\}\ L^k[(t.\mathsf{const}\ c)^n\ \mathsf{return}]\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ \mathsf{local}_n\{i; v_l^*\}\ L^k[(t.\mathsf{const}\ c)^n\ \mathsf{return}]\ \mathsf{end} \hookrightarrow_j\ (t.\mathsf{const}\ c)^n$
  This proof is similar to the br case above, but with a few extra steps.
  First, we derive the type of $(t.\mathsf{const}\ c)^n$ from the precondition of return.
  $ti_2^* = ti_1^*\ (t\ \alpha)^n$, $l_1 = l_2$, $\Gamma_2 = \Gamma_1 \cup \Gamma_3$, $\phi_2 = \phi_1 \cup \phi_3$, $S; ((t\ \alpha)^n; \phi_3) \vdash_i v_l^*; L^k[(t.\mathsf{const}\ c)^n\ \mathsf{return}] : ti_3^n; l_3; \Gamma_3; \phi_3$ by Lemma Inversion-On-Instruction-Typing on $S; C \vdash \mathsf{local}_n\{i; v_l^*\}\ L^k[(t.\mathsf{const}\ c)^n\ \mathsf{return}]\ \mathsf{end} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
  $(\vdash v_l : ti_l; \phi_l)^*$ and $S; C_l \vdash L^k[(t.\mathsf{const}\ c)^n\ \mathsf{return}] : \epsilon; ti_l^*; \emptyset, ti_l^*; \phi_l^* \rightarrow (t\ \alpha)^n; l_3; \Gamma_3; \phi_3$,
  where $C_l = C$, local t^*, return $((t\ \alpha)^n; \phi_3)$, by inversion on the Code judgment.
  $ti_l^* = (t_l\ \alpha_l)^*$ because it is a premise of Rule Admin-Const which we have assumed to hold.
  By Lemma Inversion-On-Instruction-Typing on Rule Composition and Rule Return, $S; C_l \vdash (t.\mathsf{const}\ c)^n : ti_4^*; l_4; \Gamma_4; \phi_4 \rightarrow ti_4^*\ (t\ \alpha_5)^n; l_3; \Gamma_5; \phi_5$, $S; C_l \vdash \mathsf{return} : ti_4^*\ (t_3\ \alpha_5)^n; l_3; \Gamma_5; \phi_5 \rightarrow ti_\emptyset^*; l_\emptyset; \Gamma_\emptyset; \phi_\emptyset$, $ti_3^n \notin \Gamma_1$ and $\Gamma_5 \vdash \phi_5 \Rightarrow \phi_3[\alpha_3 \rightarrow \alpha_5]$, where $(t\ \alpha_3) = ti_3^n$.
  By Lemma Inversion-On-Instruction-Typing on $S; C_l \vdash (t.\mathsf{const}\ c)^n : ti_4^*; l_4; \Gamma_4; \phi_4 \rightarrow ti_4^*\ (t_3\ \alpha_5)^n; l_3; \Gamma_5; \phi_5$, we have $l_4 = l_3$, $\Gamma_5 = \Gamma_4, (t\ \alpha)^n$, $\phi_5 = \phi_4, (= \alpha_5\ (t\ c))^n$, and $S; C_l \vdash (t.\mathsf{const}\ c)^n : \epsilon; l_4; \Gamma_4; \phi_4 \rightarrow (t_3\ \alpha_5)^n; l_3; \Gamma_5; \phi_5$.
  Then, $S; C \vdash (t.\mathsf{const}\ c)^n\ : \epsilon; l_1; \Gamma_1, (t_1\ \alpha_1)^*; \phi_1, (= \alpha_\emptyset\ (t_\emptyset\ c_\emptyset))^* \rightarrow (t_3\ \alpha_5)^n; l_3; \Gamma_5; \phi_5$ by Lemma Lift-Consts.
  We have $S; C \vdash (t.\mathsf{const}\ c)^n\ : ti_1^*; l_1; \Gamma_1, (t_1\ \alpha_1)^*; \phi_1, (= \alpha_\emptyset\ (t_\emptyset\ c_\emptyset))^* \rightarrow ti_1^*\ (t_3\ \alpha_5)^n; l_2; \Gamma_5; \phi_5$ by Rule Stack-Poly.
  Further, $S; C \vdash (t.\mathsf{const}\ c)^n\ : ti_1^*; l_1; \Gamma_1, (t_1\ \alpha_1)^*; \phi_1, (= \alpha_\emptyset\ (t_\emptyset\ c_\emptyset))^* \rightarrow (t_3\ \alpha_5)^n; l_3; \Gamma_5 \cup \Gamma_1; \phi_5 \cup \phi_1$ by Lemma Threading-Constraints, since $\Gamma_1, (t_1\ \alpha_1)^* \cup \Gamma_1 = \Gamma_1, (t_1\ \alpha_1)^*$, and $\phi_1, (= \alpha_\emptyset\ (t_\emptyset\ c_\emptyset))^* \cup \phi_1 = \phi_1, (= \alpha_\emptyset\ (t_\emptyset\ c_\emptyset))^*$.
  We have $\Gamma_5 \vdash \phi_5 \Rightarrow \phi_3[\alpha_3 \mapsto \alpha_5]$, so $\Gamma_5 \cup \Gamma_1 \vdash \phi_5 \cup \phi_1 \Rightarrow (\phi_3[\alpha_3 \mapsto \alpha_5]) \cup \phi_1$.

Then, since $(t_3\ \alpha_3)^n \notin \Gamma_1$ are fresh, we can perform renaming to get $S; C \vdash (t.\text{const}\ c)^n\ : ti_1^*; l_1; \Gamma_1, (t_1\ \alpha_1)^*; \phi_1, (= \alpha_\emptyset\ (t_\emptyset\ c_\emptyset))^* \rightarrow ti_1^*\ (t_3\ \alpha_3)^n; l_3; \Gamma_5 \cup \Gamma_1; \phi_5 \cup \phi_1$

- Case: $S; C \vdash v\ (\text{tee\_local}\ j) : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ v\ (\text{tee\_local}\ j) \hookrightarrow v\ v\ (\text{set\_local}\ j)$
  Note that the reduction of tee_local does not actually need to reason about locals since it gets reduced to a set_local , so we only have to do the reasoning in the set_local case.
  As usual, we start by figuring out what $\epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ looks like.
  By Lemma Inversion-On-Instruction-Typing on Rule Composition, we know that $S; C \vdash v : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$, and $S; C \vdash \text{tee\_local}\ j : ti_3^*; l_3; \Gamma_3; \phi_3 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
  By Lemma Inversion-On-Instruction-Typing on Rule Tee-Local, we also know that $ti_3^* = ti^*\ (t\ \alpha)$, $ti_2^* = ti^*\ (t\ \alpha_2)$, $l_2 = l_3[j := (t\ \alpha)]$, $\Gamma_2 = \Gamma_3, (t\ \alpha_2)$, and $\phi_2 = \phi_3, (= \alpha\ \alpha_2)$.
  Then, by Lemma Inversion-On-Instruction-Typing on Rule Const, $t.\text{const}\ c = v$, $ti^* = \epsilon$, $l_1 = l_3$, $\Gamma_3 = \Gamma_1, (t\ \alpha)$, and $\phi_3 = \phi_1, (= \alpha\ (t\ c))$.
  Now, we can show that $v\ v\ (\text{set\_local}\ j) : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_1, (= \alpha\ (t\ c)), (= \alpha\ \alpha_2)$ and $\Gamma_2 \vdash \phi_1, (= \alpha\ (t\ c)), (= \alpha\ \alpha_2) \implies \phi_2$.
  By Rule Const and Rule Composition, $S; C \vdash v\ v : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (t\ \alpha_2)\ (t\ \alpha); l_1; \Gamma_1, (t\ \alpha_2), (t\ \alpha); \phi_1, (= \alpha_2\ (t\ c)), (= \alpha\ (t\ c))$.
  By Rule Set-Local, $S; C \vdash \text{set\_local}\ j : (t\ \alpha); l_1; \Gamma_1; \phi_2 \rightarrow \epsilon; l_1[j := (t\ \alpha)]; \Gamma_1, (t\ \alpha_2), (t\ \alpha); \phi_1, (= \alpha_2\ (t\ c)), (= \alpha\ (t\ c))$.
  By Rule Composition, $S; C \vdash v\ v\ (\text{set\_local}\ j) : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_1[j := (t\ \alpha)]; \Gamma_1, (t\ \alpha_2), (t\ \alpha); \phi_1, (= \alpha_2\ (t\ c)), (= \alpha\ (t\ c))$.
  By Lemma Well-formedness, $l_1 \subset \Gamma_1$, so $(t\ \alpha_2), l_1[j := \alpha] \subset \Gamma_1, (t\ \alpha), (t\ \alpha_2)$.
  Finally, $\Gamma_1, (t\ \alpha), (t\ \alpha_2) \vdash \phi_1, (= \alpha_2\ (t\ c)), (= \alpha\ (t\ c)) \Rightarrow \phi_1, (= \alpha\ (t\ c)), (= \alpha\ \alpha_2)$ trivially.

- Case: $S; C \vdash \text{get\_local}\ j : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ s; v_1^j\ (t.\text{const}\ c)\ v_2^k; \text{get\_local}\ j \hookrightarrow s; v_1^j\ (t.\text{const}\ c)\ v_2^k; (t.\text{const}\ c)$
  We know $l_1 = (t_1\ \alpha_1)^j\ (t\ \alpha)\ (t_2\ \alpha_2)^j$, where $\vdash t.\text{const}\ c : (t\ \alpha); \emptyset, (= \alpha\ (t\ c))$ and $(= \alpha\ (t\ c)) \in \phi_1$ as it is one of our assumptions.
  Then, $S; C \vdash \text{get\_local}\ j : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (t\ \alpha_2); l_1; \Gamma_1, (t\ \alpha_2); \phi_1, (= \alpha_2\ \alpha)$, $ti_2^* = ti_1^*\ (t\ \alpha_2)$, $l_1 = l_2$, $\Gamma_2 = \Gamma_1, (t\ \alpha_2)$, and $\phi_2 = \phi_1, (= \alpha_2\ \alpha)$ by Lemma Inversion-On-Instruction-Typing on $S; C \vdash \text{get\_local}\ j : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
  We have $S; C \vdash (t.\text{const}\ c) : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (t\ \alpha_2); l_1; \Gamma_1, (t\ \alpha_2); \phi_1, (= \alpha_2\ (t\ c))$ by Rule Const.
  Then, $S; C \vdash (t.\text{const}\ c) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_1^*\ (t\ \alpha_2); l_1; \Gamma_1, (t\ \alpha_2); \phi_1, (= \alpha_2\ (t\ c))$ by Rule Stack-Poly.
  Finally, since $(= \alpha\ (t\ c)) \in \phi_1, \Gamma_1, (t\ \alpha_2) \vdash \phi_1, (= \alpha_2\ (t\ c)) \Rightarrow \phi_1, (= \alpha_2\ \alpha)$.

- Case: $S; C \vdash (t'.\text{const}\ c')\ \text{set\_local}\ j : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  $\wedge\ s; v_1^j\ (t.\text{const}\ c)\ v_2^k; (t.\text{const}\ c')\ \text{set\_local}\ j \hookrightarrow s; v_1^j\ (t.\text{const}\ c')\ v_2^k; \epsilon$
  We know $l_1 = (t_1\ \alpha_1)^j\ (t\ \alpha)\ (t_2\ \alpha_2)^k$, where $\vdash v_1^j\ (t.\text{const}\ c)\ v_2^k : (t\ \alpha); (= \alpha\ (t\ c))$ and $(= \alpha\ (t\ c)) \in \phi_1$ as it is one of our assumptions.
  Then, $S; C \vdash (t'.\text{const}\ c') : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$, $S; C \vdash \text{set\_local}\ j : ti_3^*; l_3; \Gamma_3; \phi_3 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ by Lemma Inversion-On-Instruction-Typing on $S; C \vdash (t'.\text{const}\ c')\ \text{set\_local}\ j : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
  Then, $C_{\text{local}}(j) = t$, $ti_3^* = ti_2^*\ (t\ a')$, $\Gamma_3 = \Gamma_2$, $\phi_3 = \phi_1$, and $l_2 = l_3[j := (t\ \alpha')]$ by Lemma Inversion-On-Instruction-Typing on
  $S; C \vdash \text{set\_local}\ j : ti_3^*; l_3; \Gamma_3; \phi_3 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
  Further, $ti_3^* = ti_1^*\ (t'\ \alpha')$, $l_1 = l_3$, $\Gamma_3 = \Gamma_1, (t'\ \alpha')$, $\phi_3 = \phi_1, (= \alpha'\ (t'\ c'))$, by Lemma Inversion-On-Instruction-Typing on
  $S; C \vdash (t'.\text{const}\ c') : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$.
  Since $ti_3^* = ti_1^*\ (t'\ \alpha')$ and $ti_3^* = ti_1^*(t\ \alpha')$, $t' = t$.
  Then, $S; C \vdash \epsilon : \epsilon; l_2; \Gamma_2; \phi_2 \rightarrow \epsilon; l_2; \Gamma_2; \phi_2$ by Rule Empty.
  Further, $S; C \vdash \epsilon : ti_1^*; l_2; \Gamma_2; \phi_2 \rightarrow ti_1^*; l_2; \Gamma_2; \phi_2$ by Rule Stack-Poly.
  Note that since ϕ_1 contains the equality constraints for all the locals except for index j, the only novel case is for the modified local.
  Then $\vdash (t.\text{const}\ c') : (t\ \alpha'); \emptyset, (= \alpha'\ (t\ c'))$ by Rule Admin-Const.
  Finally, since $\phi_2 = \phi_1, (= \alpha'\ (t\ c'))$, $\phi_2 = \phi_1 \bigcup(\emptyset, (= \alpha'\ (t\ c')))$.

- Case: $S; C \vdash \mathsf{get\_global}~j : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge~s; v^*; \mathsf{get\_global}~j \hookrightarrow_i s; v^*; s_{\text{glob}}(i, j)$
We know $ti_2^* = ti_1^*~(t~\alpha)$, $l_1 = l_2$, $C_{\text{global}}(j) = \text{mut}^? t$, $\Gamma_2 = \Gamma_1, (t~\alpha)$, and $\phi_1 = \phi_2$ by Lemma Inversion-On-Instruction-Typing on
$S; C \vdash \mathsf{get\_global}~j : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
Recall that we assume $\vdash s : S$, then we know $S \vdash s_{\text{inst}}(i) : C$ because it is a premise of Rule Store.
Recall that $C_{\text{global}}(j) = \text{mut}^? t$, then $\vdash s_{\text{glob}}(i, j) : (t~\alpha_\emptyset); \phi_\emptyset$ because it is a premise of Rule Instance that we have assumed to hold.
Now, we can show that $s_{\text{glob}}(i, j)$ has the appropriate type.
Then $S; C \vdash (t.\mathsf{const}~c) : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (t~\alpha); l_1; \Gamma_1, (t~\alpha); \phi_1, (= \alpha~(t~c))$, where $t.\mathsf{const}~c = s_{\text{glob}}(i, j)$, by Rule Const.
Further $S; C \vdash (t.\mathsf{const}~c) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_1^*~(t~\alpha); l_1; \Gamma_1, (t~\alpha); \phi_1, (= \alpha~(t~c))$ by Rule Stack-Poly.
Finally, $\Gamma_1, (t~\alpha) \vdash \phi_1, (= \alpha~(t~c)) \Rightarrow \phi_1$ trivially.

- Case: $S; C \vdash (t.\mathsf{const}~c)~(\mathsf{set\_global}~j) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge~s; v^*; (t.\mathsf{const}~c)~(\mathsf{set\_global}~j) \hookrightarrow_i s'; v^*; \epsilon$, where $s' = s$ with $\text{glob}(i, j) = (t.\mathsf{const}~c)$
We know $S; C \vdash (t.\mathsf{const}~c) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$, $S; C \vdash \mathsf{set\_global}~j : ti_3^*; l_3; \Gamma_3; \phi_3 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, by Lemma Inversion-On-Instruction-Typing on $S; C \vdash (t.\mathsf{const}~c)~(\mathsf{set\_global}~j) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
Then, $ti_3^* = ti_1^*~(t~\alpha)$, $l_1 = l_3$, $\Gamma_3 = \Gamma_1, (t~\alpha)$, $\phi_3 = \phi_1, (= \alpha~(t~c))$, and $\alpha \notin \Gamma_1$, by Lemma Inversion-On-Instruction-Typing on $S; C \vdash (t.\mathsf{const}~c) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$.
Then, $ti_3^* = ti_2^*~(t~\alpha)$, $l_3 = l_2$, $\Gamma_2 = \Gamma_3$, $\phi_2 = \phi_3$, $C_{\text{global}} = \text{mut}~t$, by Lemma Inversion-On-Instruction-Typing on $S; C \vdash \mathsf{set\_global}~j : ti_3^*; l_3; \Gamma_3; \phi_3 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
Now we construct a type for ϵ.
We have $S; C \vdash \epsilon : \epsilon; l_1; \Gamma_1, (t~\alpha); \phi_1, (= \alpha~(t~c)) \rightarrow \epsilon; l_1; \Gamma_1, (t~\alpha); \phi_1, (= \alpha~(t~c))$ by Rule Empty.
Then, $S; C \vdash \epsilon : ti_1^*; l_1; \Gamma_1, (t~\alpha); \phi_1, (= \alpha~(t~c)) \rightarrow ti_2^*; l_1; \Gamma_1, (t~\alpha); \phi_1, (= \alpha~(t~c))$ by Rule Stack-Poly.
Now we must ensure that the new store s' is well typed: $\vdash s' : S$.
Recall that we assume $\vdash s : S$, then we know $S \vdash s_{\text{inst}}(i) : C$ because it is a premise of Rule Store.
Recall that $C_{\text{global}}(j) = \text{mut}~t$ and $s_{\text{glob}}(i, j) = (t.\mathsf{const}~c')$, where $\vdash (t.\mathsf{const}~c') : (t~\alpha_\emptyset); \emptyset, (= \alpha_\emptyset~(t~c'))$ because it is a premise of Rule Instance that we have assumed to hold.
We know $\vdash: (t.\mathsf{const}~c) : (t~\alpha); \emptyset, (= \alpha~(t~c))$ by Rule Admin-Const.
Therefore $\vdash s' : S$ by Rule Store.

- Case: $S; C \vdash (\mathsf{i32.const}~k)~(t.\mathsf{load}~a~o) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge~s; v^*; (\mathsf{i32.const}~k)~(t.\mathsf{load}~a~o) \hookrightarrow_i s; v^*; t.\mathsf{const}~\text{const}_t(b^*)$, where $s_{\text{mem}}(i, k + o, |t|) = b^*$
We know $ti_2^* = ti_1^*~(t~\alpha)$, $l_1 = l_2$, $\Gamma_2 = \Gamma_1, (t~\alpha_k), (t~\alpha)$, $\phi_2 = \phi_1, (= \alpha_k~(\mathsf{i32}~k))$, and $\alpha_k \notin \Gamma_1$, by Lemma Inversion-On-Instruction-Typing on $S; C \vdash (\mathsf{i32.const}~k)~(t.\mathsf{load}~a~o) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
Let $c = \text{const}_t(b^*)$. Although note that the actual value of c is irrelevant for the rest of the proof case.
Now we can restruct a type for $t.\mathsf{const}~c$.
We have $S; C \vdash t.\mathsf{const}~c : \epsilon; l_1; \Gamma_1, (\mathsf{i32}~\alpha_k); \phi_1, (= \alpha_k~(\mathsf{i32}~k)) \rightarrow (t~\alpha); l_1; \Gamma_1, (\mathsf{i32}~\alpha_k), (t~\alpha); \phi_1, (= \alpha_k~(\mathsf{i32}~k)), (= \alpha~(t~c))$ by Rule Const.
Then, $S; C \vdash t.\mathsf{const}~c : ti_1^*; l_1; \Gamma_1, (\mathsf{i32}~\alpha_k); \phi_1, (= \alpha_k~(\mathsf{i32}~k)) \rightarrow ti_1^*~(t~\alpha); l_1; \Gamma_1, (\mathsf{i32}~\alpha_k), (t~\alpha); \phi_1, (= \alpha_k~(\mathsf{i32}~k)), (= \alpha~(t~c))$ by Rule Stack-Poly.
Finally, $\Gamma_1, (t~\alpha_k), (t~\alpha) \vdash \phi_1, (= \alpha_k~(\mathsf{i32}~k)), (= \alpha~(t~c)) \Rightarrow \phi_1, (= \alpha_k~(\mathsf{i32}~k))$ trivially.

- Case: $S; C \vdash (\mathsf{i32.const}~k)~(t.\mathsf{load}\checkmark~a~o) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge~s; v^*; (\mathsf{i32.const}~k)~(t.\mathsf{load}\checkmark~tp\_sx^?~a~o) \hookrightarrow_i s; t.\mathsf{const}~\text{const}_t(b^*)$, where $s_{\text{mem}}(i, k + o, |t|) = b^*$
Proceeds the same as the above case, the reduction of $\mathsf{load}\checkmark$ is equivalent to a successful load.

- Case: $S; C \vdash (\mathsf{i32.const}~k)~(t.\mathsf{load}~tp\_sx~a~o) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge~s; v^*; (\mathsf{i32.const}~k)~(t.\mathsf{load}~tp\_sx~a~o) \hookrightarrow_i s; t.\mathsf{const}~\text{const}_t^{sx}(b^*)$, where $s_{\text{mem}}(i, k + o, |tp|) = b^*$
This case is the same as the above two cases except that tp_sx is present in the load instruction.
Similar to above case, except with $|tp|$ replacing $|t|$ and $\text{const}_t^{sx}(b^*)$ instead of $\text{const}_t(b^*)$.

- Case: $S; C \vdash (\mathsf{i32.const}~k)~(t.\mathsf{load}\checkmark~tp\_sx~a~o) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge~s; v^*; (\mathsf{i32.const}~k)~(t.\mathsf{load}\checkmark~tp\_sx^?~a~o) \hookrightarrow_i s; t.\mathsf{const}~\text{const}_t(b^*)$, where $s_{\text{mem}}(i, k + o, |tp|) = b^*$

Proceeds the same as the above case.

- Case: $S; C \vdash (\text{i32.const } k)\ (t.\text{load } tp\_sx^?\ a\ o) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ s; v^*; (\text{i32.const } k)\ (t.\text{load } tp\_sx^?\ a\ o) \hookrightarrow_i s; v^*; \text{trap}$
Trivially, we have $S; C \vdash \text{trap} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ by Rule Trap.

- Case: $S; C \vdash (\text{i32.const } k)\ (t.\text{const } c)\ (t.\text{store } a\ o) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ s; (\text{i32.const } k)\ (t.\text{const } c)\ (t.\text{store } a\ o) \hookrightarrow_i s'; \epsilon$, where $s' = s$ with $\text{mem}(i, k + o, |t|) = \text{bits}_t^{|t|}(c)$
We know $ti_1^* = ti_2^*, l_1 = l_2, \Gamma_2 = \Gamma_1, (\text{i32 } \alpha_k), (t\ \alpha_c), \phi_2 = \phi_1, (= \alpha_k\ (\text{i32 } k)), (= \alpha_c\ (t\ c)), C_{\text{memory}} = n$, and $\alpha_k, \alpha_c \notin \Gamma_1$ by Lemma Inversion-On-Instruction-Typing on $S; C \vdash (\text{i32.const } k)\ (t.\text{const } c)\ (t.\text{store } a\ o) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
Now we construct a type for ϵ.
We have $S; C \vdash \epsilon : \epsilon; l_1; \Gamma_1, (\text{i32 } \alpha_k), (t\ \alpha_c); \phi_1, (= \alpha_k\ (\text{i32 } k)), (= \alpha_c\ (t\ c)) \rightarrow \epsilon; l_1; \Gamma_1, (\text{i32 } \alpha_k), (t\ \alpha_c); \phi_1, (= \alpha_k\ (\text{i32 } k)), (= \alpha_c\ (t\ c))$ by Rule Empty.
Now we must ensure that the new store s' is well typed: $\vdash s' : S$.
Recall $\vdash s : S$ and $C_{\text{memory}} = n$, then $S_{\text{mem}}(i) = n$ and $s_{\text{mem}}(i) = b^*$ where $n \leq |b^*|$ because it's a premise of Rule Store.
Since $s' = s$ with $\text{mem}(i, k + o, |t|) = \text{bits}_t^{|t|}(c)$, then $|s'_{\text{mem}}(i)| = |s_{\text{mem}}(i)|$, and therefore $n \leq |s'_{\text{mem}}(i)|$, so $s' : S$ by Rule Store.

- Case: $S; C \vdash (\text{i32.const } k)\ (t.\text{const } c)\ (t.\text{store}\checkmark\ a\ o) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ s; (\text{i32.const } k)\ (t.\text{const } c)\ (t.\text{store}\checkmark\ a\ o) \hookrightarrow_i s'; \epsilon$, where $s' = s$ with $\text{mem}(i, k + o, |t|) = \text{bits}_t^{|t|}(c)$
Proceeds the same as the above case, the reduction of store✓ is equivalent to a successful store.

- Case: $S; C \vdash (\text{i32.const } k)\ (t.\text{const } c)\ (t.\text{store } tp\ a\ o) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ s; v^*; (\text{i32.const } k)\ (t.\text{const } c)\ (t.\text{store } tp\ a\ o) \hookrightarrow_i s'; v^*; \epsilon$, where $s' = s$ with $\text{mem}(i, k + o, |tp|) = \text{bits}_t^{|tp|}(c)$
This case is the same as the above two cases except that tp is present in the store instruction.
Similar to above case, except with $|tp|$ replacing $|t|$ and $\text{const}_t^{sx}(b^*)$ instead of $\text{const}_t(b^*)$.

- Case: $S; C \vdash (\text{i32.const } k)\ (t.\text{const } c)\ (t.\text{store}\checkmark\ tp\ a\ o) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ s; v^*; (\text{i32.const } k)\ (t.\text{const } c)\ (t.\text{store}\checkmark\ tp\ a\ o) \hookrightarrow_i s'; v^*; \epsilon$, where $s' = s$ with $\text{mem}(i, k + o, |tp|) = \text{bits}_t^{|tp|}(c)$
Proceeds the same as above.

- Case: $S; C \vdash (\text{i32.const } k)\ (t.\text{const } c)\ (t.\text{store } tp^?\ a\ o) : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ s; v^*; (\text{i32.const } k)\ (t.\text{const } c)\ (t.\text{store } tp^?\ a\ o) \hookrightarrow_i s; v^*; \text{trap}$
Trivially, we have $S; C \vdash \text{trap} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ by Rule Trap.

- Case: $S; C \vdash \text{current\_memory} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ s; v^*; \text{current\_memory} \hookrightarrow_i s; v^*; \text{i32.const } |s_{\text{mem}}(i, *)|/64\text{Ki}$
We know $ti_2^* = ti_1^*\ (\text{i32 } \alpha), l_1 = l_2, \Gamma_2 = \Gamma_1, (\text{i32 } \alpha), \phi_2 = \phi_1$, and $\alpha \notin \Gamma_1$ by Lemma Inversion-On-Instruction-Typing on $S; C \vdash \text{current\_memory} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
Now we can construct a type for the reduced program.
Let $c = |s_{\text{mem}}(i, *)|/64\text{Ki}$. Although note that the actual value of c is irrelevant to the rest of the proof case.
We have $S; C \vdash \text{i32.const } c : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (\text{i32 } \alpha); l_1; \Gamma_1, (\text{i32 } \alpha); \phi_1, (= \alpha\ (\text{i32 } c))$ by Rule Const.
Then, $S; C \vdash \text{i32.const } c : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_1^*\ (\text{i32 } \alpha); l_1; \Gamma_1, (\text{i32 } \alpha); \phi_1, (= \alpha\ (\text{i32 } c))$ by Rule Stack-Poly.
Finally, $\Gamma_1, (\text{i32 } \alpha) \vdash \phi_1, (= \alpha\ (\text{i32 } c)) \Rightarrow \phi_1$ trivially.

- Case: $S; C \vdash (\text{i32.const } k)\ \text{grow\_memory} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ s; v^*; (\text{i32.const } k)\ \text{grow\_memory} \hookrightarrow_i s'; v^*; \text{i32.const } |s_{\text{mem}}(i, *)|/64\text{Ki}$,
where $s' = s$ with $\text{mem}(i, *) = s_{\text{mem}}(i, *)(0)^{k \cdot 64\text{Ki}}$
We know $ti_2^* = ti_1^*\ (\text{i32 } \alpha), l_1 = l_2, \Gamma_2 = \Gamma_1, (\text{i32 } \alpha_k), (\text{i32 } \alpha), \phi_2 = \phi_1, (= \alpha_k\ (\text{i32 } k)), C_{\text{memory}} = n$, and $\alpha_k, a \notin \Gamma_1$ by Lemma Inversion-On-Instruction-Typing on $S; C \vdash (\text{i32.const } k)\ \text{grow\_memory} : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
Further, $S_{\text{mem}}(i) \leq |s_{\text{mem}}(i, *)|$ because it is a premise of Rule Store on $\vdash s : S$, which we have assumed.
Now we will construct a type for the reduced instruction sequence.
Let $c = \text{i32.const } |s_{\text{mem}}(i, *)|/64\text{Ki}$. Although note that the actual value of c is irrelevant to the rest of the proof case.

We have $S; C \vdash \text{i32.const } c : \epsilon; l_1; \Gamma_1, (\text{i32 } \alpha_k); \phi_1, (= \alpha_k\ (\text{i32 } k)) \longrightarrow (\text{i32 } \alpha); l_1; \Gamma_1, (\text{i32 } \alpha_k), (\text{i32 } \alpha); \phi_1, (= \alpha_k\ (\text{i32 } k)), (= \alpha\ (\text{i32 } c))$ by Rule Const.
Then, $\Gamma_1, (\text{i32 } \alpha_k), (\text{i32 } \alpha) \vdash \phi_1, (= \alpha_k\ (\text{i32 } k)), (= \alpha\ (\text{i32 } c)) \Rightarrow \phi_1, (= \alpha_k\ (\text{i32 } k))$ trivially.
Now we must ensure that the new store s' is well typed: $\vdash s' : S$.
Recall that we assumed $\vdash s : S$ and that we have $C_{\text{memory}} = n$, then $S_{\text{mem}}(i) = n$ and $s_{\text{mem}}(i) = b^*$ where $n \leq |b^*|$ because it's a premise of Rule Store.
Since $s' = s$ with $\text{mem}(i, *) = s_{\text{mem}}(i, *)(0)^{k \cdot 64\text{Ki}}$, then $|s'_{\text{mem}}(i)| > |s_{\text{mem}}(i)|$, and therefore $n \leq |s'_{\text{mem}}(i)|$, so $s' : S$ by Rule Store.

- Case: $S; C \vdash (\text{i32.const } k)\ \text{grow\_memory} : ti_1^*; l_1; \Gamma_1; \phi_1 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ s; (\text{i32.const } k)\ \text{grow\_memory} \hookrightarrow_i \text{i32.const } (-1)$
Same as above case since the value of c is irrelevant (and can therefore be -1).

- Case: $S; C \vdash \text{local}_n\{i; v^*\}\ e^* : ti_1^*; l_1; \Gamma_1; \phi_1 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ s; v_0^*; \text{local}_n\{i; v^*\}\ e^* \hookrightarrow_j s'; v_0^*; \text{local}_n\{i; v'^*\}\ e'^*$
where $s; v^*; e^* \hookrightarrow_i s'; v'^*; e'^*$
First, we will derive the type of the body of the local block.
We know $(\vdash v : (t_v\ \alpha_v); \phi_v)^*$, and $S; (ti^n; \phi) \vdash_i v^*; e^* : ti^n; l_3; \Gamma_3; \phi_3$, where $ti_2^* = ti_1^*\ ti^n$ and $\phi_2 = \phi_1 \cup \phi_3$ by Lemma Inversion-On-Instruction-Typing on $S; C \vdash \text{local}_n\{i; v^*\}\ e^* : ti_1^*; l_1; \Gamma_1; \phi_1 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
Then $S; S_{\text{inst}}(i), \text{local } t_v^*, \text{return } (ti^n; \phi) \vdash e^* : \epsilon; (t_v\ \alpha_v)^*; \emptyset, (t_v\ \alpha_v)^*; \phi_v^* \longrightarrow ti^n; l_3; \Gamma_3; \phi_4$, where $\Gamma_3 \vdash \phi_4 \Rightarrow \phi_3$ because it is a premise of Rule Code.
Now, we invoke the inductive hypothesis and use it to rebuild the original type.
Since $S; S_{\text{inst}}(i), \text{return}(ti^n; \phi) \vdash e^* : \epsilon; (t_v\ \alpha_v)^*; \emptyset, (t_v\ \alpha_v)^*; \phi_v^* \longrightarrow ti^n; l_3; \Gamma_3; \phi, s \vdash S, (\vdash v : (t_v\ \alpha_v); \phi_v)^*$, and $s; v^*; e^* \hookrightarrow_i s'; v'^*; e'^*$, then by the inductive hypothesis we know that $\vdash s' : S$ and $S; S_{\text{inst}}(i), \text{return}(ti^n; \phi) \vdash e'^* : \epsilon; (t_v\ \alpha'_v); \Gamma_4; \phi_4 \longrightarrow ti^n; l_3; \Gamma_3; \phi_3$, where $(\vdash v' : (t_v\ \alpha'_v); \phi'_v)^*, (t_v\ \alpha'_v) \subset \Gamma_3$, and $(\phi'_v \subset \phi_4)^*$.
We know, $S; (ti^n); \phi) \vdash_i v'^*; e'^* : ti^n; l_3; \Gamma_3; \phi_3$ by Rule Code.
Then, $S; C \vdash \text{local}_n\{i; v'^*\}\ e'^* : \epsilon; l_1; \Gamma_1; \phi_1 \longrightarrow ti^n; l_2; \Gamma_2; \phi_2$ by Rule Local.
Thus, $S; C \vdash \text{local}_n\{i; v'^*\}\ e'^* : ti_1^*; l_1; \Gamma_1; \phi_1 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ by Rule Stack-Poly, since $ti_2^* = ti_1^*\ ti^n$

- Case: $S; C \vdash L^k[e^*] : ti_1^*; l_1; \Gamma_1; \phi_1 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
$\wedge\ s; v^*; L^k[e^*] \hookrightarrow_i s'; v'^*; L^k[e'^*]$
where $s; v^*; e^* \hookrightarrow_i s'; v'^*; e'^*$
The intuition for the proof is that there is no requirement on what the label stack is of the module type context C under which $L^k[e^*]$ is typed. Thus, we can reduce e^* outside of L^k, but with the module type context C as if it were inside of L^k.
The proof continues via induction on k.
  - Case: $k = 0$ Expanding $L^0[e^*]$, we get $v_0^*\ e^*\ e_0^*$.
    1. By Lemma Sequence-Decomposition on $S; C \vdash v_0^*\ e^*\ e_0^* : ti_1^*; l_1; \Gamma_1; \phi_1 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ we have
       a. $S; C \vdash v_0^* : ti_1^*; l_1; \Gamma_1; \phi_1 \longrightarrow ti_3^*; l_3; \Gamma_3; \phi_3$
       b. $S; C \vdash e^*\ e_0^* : ti_3^*; l_3; \Gamma_3; \phi_3 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, and therefore, by Lemma Sequence-Decomposition
       c. $S; C \vdash e^* : ti_3^*; l_3; \Gamma_3; \phi_3 \longrightarrow ti_4^*; l_4; \Gamma_4; \phi_4$
       d. $S; C \vdash e_0^* : ti_4^*; l_4; \Gamma_4; \phi_4 \longrightarrow ti_2^*; l_2; \Gamma_2; \phi_2$

       Now, we invoke the outer inductive hypothesis (for Lemma 19) and rebuild the type using the reduced expression.
    2. By 1c and the inductive hypothesis (for Lemma 19) we know that
       a. $s; v^*; e^* \hookrightarrow_i s'; v'^*; e'^*$,
       b. $\vdash s' : S$,
       c. $S; C \vdash e'^* : ti_3^*; l_5; \Gamma_3, (t\ \alpha)^*; \phi_3, (= \alpha\ (t\ c))^* \bigcup \phi_v \longrightarrow ti_4^*; l_4; \Gamma_6; \phi_6$,
       d. $ti_1\ (t\ \alpha)^*\ l_5 \subset \Gamma_3$,

e. $\Gamma_6 \vdash \phi_6 \Rightarrow \phi_4$,

f. $l_3 = (t_v\ \alpha_v)$, where $(\vdash v' : (t_v\ \alpha_v); \phi_v)^*$

g. $(\alpha \notin \Gamma_3)^*$

3. By Lemma Values-Any-Context, 1a, and 2e we have that $S; C \vdash v_0^* : ti_1^*; l_5; \Gamma_1, (t\ \alpha)^*; \phi_1, (= \alpha\ (t\ c))^* \bigcup \phi_v \rightarrow ti_3^*; l_5; \Gamma_3, (t\ \alpha)^*; \phi_3, (= \alpha\ (t\ c))^* \bigcup \phi_v$
4. Then by Lemma Sequence-Composition, 2c, and 3, we have that $S; C \vdash v_0^*\ e'^* : ti_1^*; l_5; \Gamma_1, (t\ \alpha)^*; \phi_1, (= \alpha\ (t\ c))^* \bigcup \phi_v \rightarrow ti_2^*; l_2; \Gamma_6; \phi_6$
5. By Lemma Strengthening and 1d, we have $S; C \vdash e_0^* : ti_4^*; l_4; \Gamma_6; \phi_6 \rightarrow ti_2^*; l_2; \Gamma_7; \phi_7$, where $\Gamma_7 \vdash \phi_7 \Rightarrow \phi_2$
6. Finally, by Lemma Sequence-Composition, we have $S; C \vdash v_0^*\ e'^*\ e_0^* : ti_1^*; l_5; \Gamma_1, (t\ \alpha)^*; \phi_1, (= \alpha\ (t\ c))^* \bigcup \phi_v \rightarrow ti_2^*; l_2; \Gamma_7; \phi_7$

– Case: $k > 0$
$L^k[e^*] = v_k^*\ \text{label}_n\{e_0^*\}\ L^{k-1}[e^*]\ \text{end}\ e_k^*$.
By Lemma Inversion-On-Instruction-Typing on $S; C \vdash v_k^*\ \text{label}_n\{e_0^*\}\ L^{k-1}[e^*]\ \text{end}\ e_k^* : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, we have that $S; C \vdash v^k : ti_1^*; l_1; \Gamma_1; \phi_1 \rightarrow ti_3^*; l_1; \Gamma_3; \phi_3$, $S; C \vdash \text{label}_n\{e_0^*\}\ L^{k-1}[e^*]\ \text{end} : ti_3^*; l_1; \Gamma_3; \phi_3 \rightarrow ti_4^*; l_4; \Gamma_4; \phi_4$, and $S; C \vdash e_k^* : ti_4^*; l_4; \Gamma_4; \phi_4 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
By Lemma Inversion-On-Instruction-Typing on $S; C \vdash \text{label}_n\{e_0^*\}\ L^{k-1}[e^*]\ \text{end} : ti_3^*; l_1; \Gamma_3; \phi_3 \rightarrow ti_4^*; l_4; \Gamma_4; \phi_4$, we have that $S; C \vdash e_0^* : ti_5^*; l_5; \emptyset, ti_5^*, l_5; \phi_5 \rightarrow ti_4^*; l_4; \Gamma_4; \phi_4$, and $S; C, \text{label}\ (ti_5^*; l_5; \phi_5) \vdash L^{k-1}[e^*] : ti_3^*; l_1; \Gamma_3; \phi_3 \rightarrow ti_4^*; l_4; \Gamma_4; \phi_4$.
Now, we invoke the inner inductive hypothesis (for this inductive proof) on $L^{k-1}[e^*]$ and rebuild the type using the reduced expression.
Since $S; C, \text{label}\ (ti_5^*; l_5; \phi_5) \vdash L^{k-1}[e^*] : ti_3^*; l_1; \Gamma_3; \phi_3 \rightarrow ti_4^*; l_4; \Gamma_4; \phi_4$, and $s \vdash S$, then by the inductive hypothesis on $L^{k-1}[e^*]$ we know that $s; v^*; e^* \hookrightarrow_i s'; v'^*; e'^*$, $\vdash s' : S$, $S; C, \text{label}\ (ti_5^*; l_5; \phi_5) \vdash e'^* : ti_3^*; l_6; \Gamma_3, (t_2\ \alpha_2)^*; \phi_3, (= \alpha_2\ (t_2\ c_2))^* \bigcup \phi_v^* \rightarrow ti_4^*; l_4; \Gamma_6; \phi_6$, where $ti_3^*\ (t_v\ \alpha_v)^* \subset \Gamma_6$, $\Gamma_6 \vdash \phi_6 \Rightarrow \phi_4$, $l_6 = (t_v\ \alpha_v)$, where $(\vdash v' : (t_v\ \alpha_v); \phi_v)^*$, and $(\alpha_2 \notin \Gamma_3)^*$.
$S; C \vdash v^k : ti_1^*; l_6; \Gamma_1, (t_v\ \alpha_v), (t_2\ \alpha_2)^*; \phi_1, (= \alpha_2\ (t_2\ c_2))^* \bigcup \phi_v^* \rightarrow ti_3^*; l_6; \Gamma_3, (t_v\ \alpha_v), (t_2\ \alpha_2)^*; \phi_3, (= \alpha_2\ (t_2\ c_2))^* \bigcup \phi_v^*$ by Rule Const and Lemma Threading-Constraints.
$S; C \vdash \text{label}_n\{e_0^*\}\ L^{k-1}[e'^*]\ \text{end} : ti_3^*; l_6; \Gamma_3, (t_v\ \alpha_v), (t_2\ \alpha_2)^*; \phi_3, (= \alpha_2\ (t_2\ c_2))^* \bigcup \phi_v^* \rightarrow ti_4^*; l_4; \Gamma_4; \phi_4$ by Rule Label.
Thus, $S; C \vdash v_k^*\ \text{label}_n\{e_0^*\}\ L^{k-1}[e'^*]\ \text{end}\ e_k^* : \epsilon; ti_{v'}^*; \phi_{v'}^* \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$ by Lemma Sequence-Composition.

□

C.2 Progress Lemmas and Proofs

Lemma 20. Progress If $\vdash_i s; v^*; e^* : ti^*; l; \Gamma; \phi$ then either $e^* = v'^*$, $e^* = \text{trap}$, or $s; v^*; e^* \hookrightarrow_i s'; v'^*; e'^*$.

Proof. Because $\vdash_i s; v^*; e^* : ti^*; l; \Gamma; \phi$, we know that $\vdash s : S$ for some S, and that $S; \epsilon \vdash_i v^*; e^* : ti^*; l; \Gamma; \phi$ by inversion on Rule Program.

Then we know that $(\vdash v : (t_v\ \alpha_v); \phi_v)^*$ and $S; S_{\text{inst}}(i), \text{local}\ (t_v^*) \vdash e^* : \epsilon; (t_v\ \alpha_v)^*; \emptyset, (t_v\ \alpha_v)^*, (t\ \alpha)^*; \phi_v^*, (= \alpha\ (t\ c))^* \rightarrow ti^*; l; \Gamma; \phi_2$, for some a^*, t^*, c^*, where $\Gamma \vdash \phi_2 \Rightarrow \phi$ by inversion on Rule Code.

We have $s_{\text{inst}}(i)_{\text{mem}} = b^n$, where $S_{\text{inst}}(i)_{\text{memory}} = n$, and $s_{\text{inst}}(i)_{\text{tab}} = \{\text{inst}\ i, \text{func}\ \text{func}\ tfi\ ...\}^n$, where $S_{\text{inst}}(i)_{\text{table}} = (n, tfi^n)$, as they are subderivations of $\vdash s : S$.

We decompose $e^* = (t.\text{const}\ c)^*\ e_2^*$.

Then, by Lemma Progress-for-Instructions on e_2^*, we have that either
$\exists s'; v'^*; e'^*. s; v^*; e^* \hookrightarrow_i s'; v'^*; e'^*$,
or $e^* = v_2^*$,
or $e^* = \text{trap}$. □

Lemma 21 (Progress-for-Instructions). If $S; S_{\text{inst}}(i) \vdash (t.\text{const}\ c)^* : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$,
and $S; S_{\text{inst}}(i) \vdash e^* : ti_2^*; l_2; \Gamma_2; \phi_2 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$, where $e^* \neq (t.\text{const}\ c_2)^*$,
and if $(t.\text{const}\ c)^*\ e^* = L^k[\text{br}\ i]$, then $i \leq k$,

and $s_{\text{inst}}(i)_{\text{mem}} = b^n$, where $C_{\text{memory}} = n$,
and $s_{\text{inst}}(i)_{\text{tab}} = \{\text{inst } i, \text{func func } tfi \ldots\}^n$, where $C_{\text{table}} = (n, tfi^n)$,
and $(\vdash v : (t_v\ \alpha_v);\ \phi_v)^*$, where $C_{\text{local}} = t_v^*$,
then, either $\exists s'; v'^*; e'^*.s; v^*; (t.\text{const } c)^*\ e^* \hookrightarrow_i s'; v'^*; e'^*$,
or $e^* = v_2^*$,
or $e^* = \text{trap}$ and $(t.\text{const } c)^* = \epsilon$

Proof. We proceed by induction on $S; C \vdash e^* : ti_2^*; l_2; \Gamma_2; \phi_2 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$.

We only give the cases for prechecked instructions and cases that use the inductive hypothesis. While block, loop, and if all have inductive typing rules, but their proofs do not require the inductive hypothesis to be used, so they are omitted. For all the other cases, by Theorem 2, if they are well-typed in Wasm-precheck, then they must be well-typed in Wasm, so they must take a step or be irreducible. Then, since the reduction rules in Wasm and Wasm-precheck are equivalent for non-prechecked instructions, they must also take a step or be irreducible in Wasm-precheck.

- Case: $S; C \vdash \text{div}\checkmark\ : ti_2^*; l_2; \Gamma_2; \phi_2 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$
  We will show that $s; v^*; (t.\text{const } c)^*\ t.\text{div}\checkmark\ \hookrightarrow s; v^*; (t.\text{const } c_3)$ for some $(t.\text{const } c_3)$.
  We know $S; C \vdash (t.\text{const } c)^* : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (t\ \alpha_1)\ (t\ \alpha_2); l_1; \Gamma_1, (t\ \alpha_1), (t\ \alpha_2); \phi_1, (=\alpha_1\ (t\ c_1)), (=\alpha_2\ (t\ c_2))$ where $\Gamma_1, (t\ \alpha_1), (t\ \alpha_2) \vdash \phi_1, (=\alpha_1\ (t\ c_1)), (=\alpha_2\ (t\ c_2)) \Rightarrow \neg(=\ a_2\ 0)$ by Lemma Inversion-On-Instruction-Typing on $S; C \vdash \text{div}\checkmark\ : ti_2^*; l_2; \Gamma_2; \phi_2 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$.
  Then, since $S; C \vdash (t.\text{const } c)^* : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (t\ \alpha_1)\ (t\ \alpha_2); l_1; \Gamma_1, (t\ \alpha_1), (t\ \alpha_2); \phi_1, (=\alpha_1\ (t\ c_1)), (=\alpha_2\ (t\ c_2))$, we have $(t.\text{const } c)^* = (t.\text{const } c_1)\ (t.\text{const } c_2)$.
  Since we have $(=\alpha_2\ (t\ c_2))$ on the left hand side (and since a_2 is fresh that can be on the only constraint on a_2, and that implies that $a_2 = 0$, then it must be the case that $c_2 \neq 0$.
  Therefore, there must exist some c_3 such that $c_3 = div(c_1, c_2)$,since $div(c_1, c_2)$ is well-defined when c_2 is non-zero.
  Then, $s; v^*; (t.\text{const } c_1)\ (t.\text{const } c_2)\ t.\text{div}\checkmark\ \hookrightarrow_i s; v^*; (t.\text{const } c_3)$.
- Case: $S; C \vdash (t.\text{load}\checkmark\ (tp\_sx)\ align\ o) : ti_2^*; l_2; \Gamma_2; \phi_2 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$
  We will show that $s; (t.\text{const } c)^*\ (t.\text{load}\checkmark\ (tp\_sx)\ align\ o) \hookrightarrow t.\text{const } c$ for some c.
  We know $S; C \vdash (t.\text{const } c)^* : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (\text{i32 } \alpha); l_2; \Gamma_1, (\text{i32 } \alpha); \phi_1, (=\alpha\ (\text{i32 } k))$ where

$$\begin{aligned}\phi_1, (=\alpha\ (\text{i32 } k)) \implies\ & (\text{ge } (\text{add} a\ (\text{i32 } o))(\text{i32 } 0)),\\ & (\text{le } (\text{add} a\ (\text{add}(\text{i32 } o + width)))\\ & \quad (\text{i32 } n_2 * 64\text{Ki}))\end{aligned}$$

  where $width = |t|$ if $tp^? = \epsilon$, and $|tp|$ otherwise, and $n_2 {*} 64\text{Ki} = S_{\text{mem}}(i, j)$ by Lemma Inversion-On-Instruction-Typing on $S; C \vdash (t.\text{load}\checkmark\ (tp\_sx)\ align\ o) : ti_2^*; l_2; \Gamma_2; \phi_2 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$.
  Then, since $S; C \vdash (t.\text{const } c)^* : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (\text{i32 } \alpha); l_2; \Gamma_1, (\text{i32 } \alpha); \phi_1, (=\alpha\ (\text{i32 } k))$, we know that $(t.\text{const } c)^* = \text{i32.const } k$.
  Then, it must be the case that $k + o \geq 0$ and $k + o + width \leq n_2 * 64\text{Ki}$.
  Since $n_2 * 64\text{Ki} = C_{\text{memory}}$, we have $s_{\text{inst}}(i)_{\text{mem}}(i, j) = b_2^*$ where $C_memory = n_2 * 64\text{Ki} = |b_2^*|$.
  Therefore, it must be the case that $k + o \geq 0$ and $k + o + width < |b_2^*|$, and therefore $s_{\text{mem}}(i, k + o, width) = b_3^*$ for some b_3^* that is a subsequence of b_2^*. Then, $s; v^*; (\text{i32.const } k)\ (t.\text{load}\checkmark\ (tp\_sx)\ align\ o) \hookrightarrow_i s; v^*; t.\text{const const}_t^{sx}(b_3^*)$.
- Case: $S; C \vdash (t.\text{store}\checkmark\ tp\ align\ o) : ti_2^*; l_2; \Gamma_2; \phi_2 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$
  We will show that $s; v^*; (t.\text{const } c)^*\ (t.\text{store}\checkmark\ (tp\_sx)\ align\ o) \hookrightarrow s'; v^*; t.\text{const } c$ for some s' and $t.\text{const } c$.
  We know $S; C \vdash (t.\text{const } c)^* : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (\text{i32 } \alpha); l_2; \Gamma_1, (\text{i32 } \alpha); \phi_1, (=\alpha\ (\text{i32 } k))$ where

$$\begin{aligned}\phi_1, (=\alpha\ (\text{i32 } k)) \implies\ & (\text{ge } (\text{add} a\ (\text{i32 } o))(\text{i32 } 0)),\\ & (\text{le } (\text{add} a\ (\text{add}(\text{i32 } o + width)))\\ & \quad (\text{i32 } n_2 * 64\text{Ki}))\end{aligned}$$

  where $width = |t|$ if $tp^? = \epsilon$, and $|tp|$ otherwise, and $n_2 {*} 64\text{Ki} = S_{\text{mem}}(i, j)$ by Lemma Inversion-On-Instruction-Typing on $S; C \vdash (t.\text{store}\checkmark\ (tp\_sx)\ align\ o) : ti_2^*; l_2; \Gamma_2; \phi_2 \rightarrow ti_3^*; l_3; \Gamma_3; \phi_3$.
  Then, since $S; C \vdash (t.\text{const } c)^* : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow (\text{i32 } \alpha); l_2; \Gamma_1, (\text{i32 } \alpha); \phi_1, (=\alpha\ (\text{i32 } k))$, we know that $(t.\text{const } c)^* = \text{i32.const } k$.
  Then, it must be the case that $k + o \geq 0$ and $k + o + width \leq n_2 * 64\text{Ki}$.
  Since $n_2 * 64\text{Ki} = C_{\text{memory}}$, we have $s_{\text{inst}}(i)_{\text{mem}}(i, j) = b_2^*$ where $C_memory = n_2 * 64\text{Ki} = |b_2^*|$.
  Therefore, it must be the case that $k + o \geq 0$ and $k + o + width < |b_2^*|$, and therefore $s_{\text{mem}}(i, k + o, width) = b_3^*$ for some b_3^* that is a subsequence of b_2^*.

Then, we can construct $s' = s$ with $s'_{\text{mem}}(i, k + o, width) = bits_t^{width}(c)$ because $|bits_t^{width}(c) \models |b_3^*|$.
Then,

$$s; v^*; (\text{i32.const } k)\ (\text{i32.const } c)\ (t.\text{store}\checkmark\ tp\ align\ o) \hookrightarrow_i s'; v^*; \epsilon$$

- Case: $S; C \vdash \text{call\_indirect } (ti_4^*; \phi_4 \to ti_5^*; \phi_5) : ti_2^*; l_2; \Gamma_2; \phi_2 \to ti_3^*; l_3; \Gamma_3; \phi_3$
  We will show that $s; v^*; (t.\text{const } c)^*\ (\text{call\_indirect } (ti_4^*; \phi_4 \to ti_5^*; \phi_5)) \hookrightarrow s; v^*; \text{call } cl$ for some cl.
  We know $S; C \vdash (t.\text{const } c)^* : \epsilon; l_1; \Gamma_1; \phi_1 \to (\text{i32 } \alpha); l_2; \Gamma_1, (\text{i32 } \alpha); \phi_1, (= \alpha\ (\text{i32 } k))$, where $\Gamma_1; (\text{i32 } \alpha) \vdash \phi_1, (= \alpha\ (\text{i32 } k)) \Rightarrow (\text{le } \ a\ n) \wedge (\text{gt}(\text{i32 } 0)\ a)$, $C_{\text{table}} = (n, tfi^n)$, and $\forall 0 < j \leq n.\ (\Gamma_1; (\text{i32 } \alpha) \vdash \phi_1, (= \alpha\ (\text{i32 } k)) \Rightarrow \neg(= (\text{i32 } j)\ a)) \vee\ (tfi^*)(i) = (ti_4^*; \phi_4 \to ti_5^*; \phi_5)$ as they are premises of Rule CallIndirect, which we have assumed to hold.
  Then, since $S; C \vdash (t.\text{const } c)^* : \epsilon; l_1; \Gamma_1; \phi_1 \to (\text{i32 } \alpha); l_2; \Gamma_1, (\text{i32 } \alpha); \phi_1, (= \alpha\ (\text{i32 } k))$, we know that $(t.\text{const } c)^* = \text{i32.const } k$.
  Then it has to be the case that $0 \leq k < n$, and that $C_{\text{table}}(k) = ti_4^*; \phi_4 \to ti_5^*; \phi_5$.
  Therefore $s_{\text{inst}}(i)_{\text{tab}}(k) = \{\text{inst } j, \text{ func func } (ti_4^*; \phi_4 \to ti_5^*; \phi_5) \text{ local } t^*\ e^*\}$, because it is an assumption of the proof.
  Thus, $s; (\text{i32.const } c)\ \text{call\_indirect } \epsilon; l_1; \Gamma_1; \phi_1 \to ti_3^*; l_3; \Gamma_3; \phi_3$.
  $\hookrightarrow s; v^*; \text{call } \{\text{inst } j, \text{ func func } (ti_4^*; \epsilon; \Gamma_4; \phi_4 \to ti_5^*; \epsilon; \Gamma_5; \phi_5) \text{ local } t^*\ e^*\}$
- Case: $S; C \vdash e_1^*\ e_2 : ti_2^*; l_2; \Gamma_2; \phi_2 \to ti_3^*; l_3; \Gamma_3; \phi_3$
  We have $S; C \vdash e_1^* : ti_2^*; l_2; \Gamma_2; \phi_2 \to ti_4^*; l_4; \Gamma_4; \phi_4$ and $S; C \vdash e_2 : ti_4^*; l_4; \Gamma_4; \phi_4 \to ti_3^*; l_3; \Gamma_3; \phi_3$ as they are sub-derivations of $S; C \vdash e_1^*\ e_2 : ti_2^*; l_2; \Gamma_2; \phi_2 \to ti_3^*; l_3; \Gamma_3; \phi_3$.
  We proceed on cases by the inductive hypothesis on e^*:
  - Case: $s; v^*; (t.\text{const } c)^*\ e_1^* \hookrightarrow_i s'; v'^*; e'^*$.
    Then we have $s; v^*; L^0[(t.\text{const } c)^*\ e_1^*] \hookrightarrow_i s'; v'^*; L^0[e'^*]$, where $L^0 = \epsilon\ []\ e_2$.
  - Case: $e_1^* = \text{trap}$ and $(t.\text{const } c)^* = \epsilon$
    Then, $e_1^*\ e_2 = L^0[\text{trap}]$, and therefore $s; v^*; e_1^*\ e_2 \hookrightarrow_i s; v^*; \text{trap}$.
  - Case: $e_1 = (t_2.\text{const } c_2)^*$
    We proceed by cases on the inductive hypothesis on e_2:
    * Case: $s; v^*; (t.\text{const } c)^*\ (t_2.\text{const } c_2)^*\ e_2 \hookrightarrow_i s'; v'^*; e'^*$.
    * Case:$e_2 = \text{trap}$ and $(t.\text{const } c)^*\ (t_2.\text{const } c_2)^* = \epsilon$
      Then we have $e_1^*\ e_2 = \text{trap}$
    * Case: $e_2 = (t_3.\text{const } c_3)$
      We have $e_1^*\ e_2 = (t_2.\text{const } c_2)^*\ t_3.\text{const } c_3)$.
- Case: $S; C \vdash e^* : ti^*\ ti_2^*; l_2; \Gamma_2; \phi_2 \to ti^*\ ti_3^*; l_3; \Gamma_3; \phi_3$
  We know $S; C \vdash (t.\text{const } c)^* : \epsilon; l_1; \Gamma_1; \phi_1 \to ti^*\ ti_2^*; l_2; \Gamma_1, ti^*\ ti_2^*; \phi_2$, where $\Gamma_1 = \Gamma_1, ti^*\ ti_2^*$, by Lemma Inversion-On-Instruction-Typing on $S; C \vdash e^* : ti^*\ ti_2^*; l_2; \Gamma_2; \phi_2 \to ti^*\ ti_3^*; l_3; \Gamma_3; \phi_3$.
  Then, by Lemma Inversion-On-Instruction-Typing on $S; C \vdash (t.\text{const } c)^* : \epsilon; l_1; \Gamma_1; \phi_1 \to ti^*\ ti_2^*; l_2; \Gamma_1, ti^*\ ti_2^*; \phi_2$, we know that $(t.\text{const } c)^* = (t_1.\text{const } c_1)^*\ (t.\text{const } c_2)^*$, where $S; C \vdash (t_1.\text{const } c_1)^* : \epsilon; l_1; \Gamma_1; \phi_1 \to ti^*; l_2; \Gamma_1, ti^*; \phi_4$, and $S; C \vdash (t_2.\text{const } c_2)^* : ti^*; l_2; \Gamma_1, ti^*; \phi_4 \to ti^*\ ti_2^*; l_2; \Gamma_1, ti^*\ ti_2^*; \phi_2$.
  Then, $S; C \vdash (t_2.\text{const } c_2)^* : \epsilon; l_2; \Gamma_1, ti^*; \phi_4 \to ti_2^*; l_2; \Gamma_1, ti_2^*; \phi_2$
  We have $S; C \vdash e^* : ti_2^*; l_2; \Gamma_2; \phi_2 \to ti_3^*; l_3; \Gamma_3; \phi_3$ as a subderivation of $S; C \vdash e^* : ti^*\ ti_2^*; l_2; \Gamma_2; \phi_2 \to ti^*\ ti_3^*; l_3; \Gamma_3; \phi_3$.
  Then, we proceed by cases on the inductive hypothesis on e^*:
  - Case: $s; v^*; (t_2.\text{const } c_2)^*\ e^* \hookrightarrow_i s'; v'^*; e'^*$
    Then, we have $s; v^*; L^0[(t_2.\text{const } c_2)^*\ e^*] \hookrightarrow_i s'; v'^*; L^0[e'^*]$, where $L^0 = (t_1.\text{const } c_1)*\ []\epsilon$.
    Thus $s; v^*; (t_1.\text{const } c_1)^*\ (t_2.\text{const } c_2)^*\ e^* \hookrightarrow_i s'; v'^*; (t_1.\text{const } c_1)^*\ e'^*$
  - $e^* = (t_3.\text{const } c_3)^*$
  - $e^* = \text{trap}$ and $(t_2.\text{const } c_2)^* = \epsilon$
    Then, $(t_1.\text{const } c_1)^*\ e^* = L^0[\text{trap}]$, and therefore $s; v^*; (t_1.\text{const } c_1)^*\ e^* \hookrightarrow_i s; v^*; \text{trap}$.
- Case: $S; C \vdash \text{label}_n\{e_0^*\}\ e^*\ \text{end} : \epsilon; l_1; \Gamma_1; \phi_1 \to ti_2^*; l_2; \Gamma_2; \phi_2$
  We know $S; C, \text{label}(ti_3^*; l_3; \Gamma_3; \phi_3) \vdash e^* : \epsilon; l_1; \Gamma_1; \phi_1 \to ti_2^*; l_2; \Gamma_2; \phi_2$, and $S; C \vdash e_0^* : ti_3^*; l_3; \Gamma_3; \phi_3 \to ti_2^n; l_2; \Gamma_2; \phi_2$, as they are subderivations of $S; C \vdash \text{label}_n\{e_0^*\}\ e^*\ \text{end} : \epsilon; l_1; \Gamma_1; \phi_1 \to ti_2^*; l_2; \Gamma_2; \phi_2$.
  If $e^* = L^k[\text{br } k]$, then we reuse the Wasm proof [Watt 2018] using Theorem 2 and Lemma Erasure-Same-Semantics.
  Otherwise, we decompose $e^* = (t_2.\text{const } c_2)^*\ e_2^*$ and then we can invoke the inductive hypothesis on e_2^*. We proceed by cases:
  - Case: $s; v^*; (t_2.\text{const } c_2)^*\ e_2^* \hookrightarrow_i s'; v'^*; e'^*$
    Let $(t.\text{const } c)^*\ \text{label}_n\{e_0^*\}\ (t_2.\text{const } c_2)^*\ e_2^*\ \text{end} = L^{k+1}[(t_2.\text{const } c_2)^*\ e_2^*]$, where $L^k = (t_2.\text{const } c_2)^*\ e_2^*$.
    Then, $s; v^*; L^{k+1}[e^*] \hookrightarrow_i s'; v'^*; L^{k+1}[e'^*]$, since $s; v^*; (t_2.\text{const } c_2)^*\ e_2^* \hookrightarrow_i s'; v'^*; e'^*$.

Thus, $s; v^*; (t.\mathsf{const}\ c)^*\ \mathsf{label}_n\{e_0^*\}\ e^*\ \mathsf{end} \hookrightarrow_i s'; v'^*; (t.\mathsf{const}\ c)^*\ \mathsf{label}_n\{e_0^*\}\ e'^*\ \mathsf{end}$

- Case: $e_2^* = (t_3.\mathsf{const}\ c_3)^*$
  Then, $\mathsf{label}_n\{e_0^*\}\ (t_2.\mathsf{const}\ c_2)^*\ e_2^*\ \mathsf{end} = \mathsf{label}_n\{e_0^*\}\ (t_2.\mathsf{const}\ c_2)^*\ (t_3.\mathsf{const}\ c_3)^*\ \mathsf{end}$.
  Then, $s; v^*; \mathsf{label}_n\{e_0^*\}\ e^*\ \mathsf{end} \hookrightarrow_i s; v^*; (t_2.\mathsf{const}\ c_2)^*\ (t_3.\mathsf{const}\ c_3)^*$.
  We have $s; v^*; L^0[\mathsf{label}_n\{e_0^*\}\ e^*\ \mathsf{end}] \hookrightarrow_i s'; v'^*; L^0[(t_2.\mathsf{const}\ c_2)^*\ (t_3.\mathsf{const}\ c_3)^*]$, where $L^0 = (t.\mathsf{const}\ c)*\ []\epsilon$.
  Thus, $s; v^*; (t.\mathsf{const}\ c)*\ \mathsf{label}_n\{e_0^*\}\ e^*\ \mathsf{end} \hookrightarrow_i s'; v'^*; (t.\mathsf{const}\ c)*\ (t_2.\mathsf{const}\ c_2)^*\ (t_3.\mathsf{const}\ c_3)^*$
- Case: $e_2^* = \mathsf{trap}$ and $(t_2.\mathsf{const}\ c_2)^* = \epsilon$
  Then, $\mathsf{label}_n\{e_0^*\}\ (t_2.\mathsf{const}\ c_2)^*\ e_2^*\ \mathsf{end} = \mathsf{label}_n\{e_0^*\}\ \mathsf{trap}\ \mathsf{end}$
  Then, $s; v^*; \mathsf{label}_n\{e_0^*\}\ \mathsf{trap}\ \mathsf{end} \hookrightarrow_i s; v^*; \mathsf{trap}$.
  We have $s; v^*; L^0[\mathsf{label}_n\{e_0^*\}\ \mathsf{trap}\ \mathsf{end}] \hookrightarrow_i s'; v'^*; L^0[\mathsf{trap}]$, where $L^0 = (t.\mathsf{const}\ c)*\ []\epsilon$.
  Thus, $s; v^*; (t.\mathsf{const}\ c)*\ \mathsf{label}_n\{e_0^*\}\ \mathsf{trap}\ \mathsf{end} \hookrightarrow_i s'; v'^*; (t.\mathsf{const}\ c)*\ \mathsf{trap}$

- Case: $S; C \vdash \mathsf{local}_n\{j; v_l^*\}\ e^*\ \mathsf{end} : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$
  We know $S; C, \mathrm{local}(t_l^*), \mathrm{return}(ti_3^*; l_3; \Gamma_3; \phi_3) \vdash e^* : \epsilon; (t_l\ \alpha_l)^*; (t_l\ \alpha_l)^*; \bigcup \phi_l^* \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$, where $v_l^* = (t_l.\mathsf{const}\ c_l)^*$, and $(\vdash v_l : (t_l\ \alpha_l); \phi_l)^*$, as they are subderivations of $S; C \vdash \mathsf{local}_n\{j; v_l^*\}\ e^*\ \mathsf{end} : \epsilon; l_1; \Gamma_1; \phi_1 \rightarrow ti_2^*; l_2; \Gamma_2; \phi_2$.
  If $e^* = L^k[\mathsf{return}k]$, then we reuse the Wasm proof [Watt 2018] using Theorem 2 and Lemma Erasure-Same-Semantics. Otherwise, we decompose $e^* = (t_2.\mathsf{const}\ c_2)^*\ e_2^*$ and then we can invoke the inductive hypothesis on e_2^*. We proceed by cases:
  - Case: $s; v_l^*; (t_2.\mathsf{const}\ c_2)^*\ e_2^* \hookrightarrow_j s'; v_l'^*; e'^*$
    Then, $s; v^*; \mathsf{local}_n\{j; v_l\}\ e^*\ \mathsf{end} \hookrightarrow_i s'; v^*; \mathsf{local}_n\{j; v_l'^*\}\ e^*\ \mathsf{end}$, since $s; v^*; (t_2.\mathsf{const}\ c_2)^*\ e_2^* \hookrightarrow_j s'; v'^*; e'^*$.
    We have $s; v^*; L^0[\mathsf{local}_n\{j; v_l^*\}\ e^*\ \mathsf{end}] \hookrightarrow_i s'; v^*; L^0[\mathsf{local}_n\{j; v_l'^*\}\ e^*\ \mathsf{end}]$, where $L^0 = (t.\mathsf{const}\ c)*\ []\epsilon$.
    Thus, $s; v^*; (t.\mathsf{const}\ c)^*\ \mathsf{local}_n\{j; v_l^*\}\ e^*\ \mathsf{end} \hookrightarrow_i s'; v^*; (t.\mathsf{const}\ c)^*\ \mathsf{local}_n\{j; v_l'^*\}\ e'^*\ \mathsf{end}$
  - Case: $e_2^* = (t_3.\mathsf{const}\ c_3)^*$
    Then, $\mathsf{local}_n\{j; v_l^*\}\ (t_2.\mathsf{const}\ c_2)^*\ e_2^*\ \mathsf{end} = \mathsf{local}_n\{j; v_l^*\}\ (t_2.\mathsf{const}\ c_2)^*\ (t_3.\mathsf{const}\ c_3)^*\ \mathsf{end}$.
    Then, $s; v^*; \mathsf{local}_n\{j; v_l^*\}\ e^*\ \mathsf{end} \hookrightarrow_i s; v^*; (t_2.\mathsf{const}\ c_2)^*\ (t_3.\mathsf{const}\ c_3)^*$.
    We have $s; v^*; L^0[\mathsf{local}_n\{j; v_l^*\}\ e^*\ \mathsf{end}] \hookrightarrow_i s'; v'^*; L^0[(t_2.\mathsf{const}\ c_2)^*\ (t_3.\mathsf{const}\ c_3)^*]$, where $L^0 = (t.\mathsf{const}\ c)*\ []\epsilon$.
    Thus, $s; v^*; (t.\mathsf{const}\ c)*\ \mathsf{local}_n\{j; v_l^*\}\ e^*\ \mathsf{end} \hookrightarrow_i s'; v'^*; (t.\mathsf{const}\ c)*\ (t_2.\mathsf{const}\ c_2)^*\ (t_3.\mathsf{const}\ c_3)^*$
  - Case: $e_2^* = \mathsf{trap}$ and $(t_2.\mathsf{const}\ c_2)^* = \epsilon$
    Then, $\mathsf{local}_n\{j; v_l^*\}\ (t_2.\mathsf{const}\ c_2)^*\ e_2^*\ \mathsf{end} = \mathsf{local}_n\{j; v_l^*\}\ \mathsf{trap}\ \mathsf{end}$
    Then, $s; v^*; \mathsf{local}_n\{j; v_l^*\}\ \mathsf{trap}\ \mathsf{end} \hookrightarrow_i s; v^*; \mathsf{trap}$.
    We have $s; v^*; L^0[\mathsf{local}_n\{j; v_l^*\}\ \mathsf{trap}\ \mathsf{end}] \hookrightarrow_i s'; v'^*; L^0[\mathsf{trap}]$.
    Thus, $s; v^*; (t.\mathsf{const}\ c)*\ \mathsf{local}_n\{j; v_l^*\ \mathsf{trap}\ \mathsf{end} \hookrightarrow_i s'; v'^*; (t.\mathsf{const}\ c)*\ \mathsf{trap}$
- Otherwise, we reuse the Wasm proof [Watt 2018] using Theorem 2 and Lemma Erasure-Same-Semantics.

□

D Extra Experiment Details

D.1 Full Data for Size of Binaries

We present the full data for the comparison between the binary sizes of hand-annotated Wasm-precheck programs and their unmodified Wasm counterparts in Table 1. Unlike in the main body of the paper, the sizes here are broken down into code and type sections.

D.2 Full System Details

System and Hardware details. The benchmarks are performed on a cloud compute instance, running on OpenStack Ussuri. The instance runs Ubuntu-22.04.2-Jammy-x64-2023-02.

The instance is allocated 16 vCPUs, from a pool of Intel® Xeon® Processor E5-2680 v4 physical CPUs. The instance has 120GiB of ECC RAM, of unknown speed.

Environment Variables. We include the full environment variables of the machine on which the benchmarks were run in Figure 5, as these have been shown to impact cache behaviour and thus performance measurements [Curtsinger and Berger 2013].

References

Charlie Curtsinger and Emery D. Berger. 2013. STABILIZER: statistically sound performance evaluation. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/2451116.2451141

Table 1. Comparison of the binary sizes of hand-annotated Wasm-precheck programs vs Wasm versions. All numbers are in bytes.

	Wasm		Wasm-precheck	
Benchmark	Code	Type	Code	Type
`correlation`	20376	204	20525	1852
`covariance`	20178	203	20297	1259
`2mm`	20394	218	20633	2707
`3mm`	20513	206	20825	3761
`atax`	19995	193	20154	1126
`bicg`	20107	193	20330	1195
`doitgen`	20165	205	20293	1848
`mvt`	20204	193	20459	1562
`gemm`	20170	204	20344	1737
`gemver`	20400	227	20847	3496
`gesummv`	20071	206	20300	1091
`symm`	20251	204	20419	1519
`syr2k`	20171	204	20336	1502
`syrk`	20080	203	20194	1100
`trmm`	20040	202	20161	923
`cholesky`	20327	193	20440	1563
`durbin`	19954	193	20063	729
`gramschmidt`	20367	193	20541	1438
`lu`	20309	193	20419	1480
`ludcmp`	20630	193	20901	2632
`trisolv`	19920	193	20092	858
`deriche`	21106	237	21262	1653
`floyd-warshall`	16582	176	16645	420
`nussinov`	16749	176	16843	759
`adi`	20532	193	20693	1807
`fdtd-2d`	20609	193	20827	2145
`heat-3d`	20446	193	20570	1008
`jacobi-1d`	19890	193	20002	566
`jacobi-2d`	20139	193	20259	704
`seidel-2d`	20012	193	20079	529

Conrad Watt. 2018. Mechanising and verifying the WebAssembly specification. In *Proceedings of the ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP)*. https://doi.org/10.1145/3167082

Figure 5. Fully reproduced environment variables of the machine we used to run the experiments.

```
SHELL=/bin/bash
PWD=/home/ubuntu
LOGNAME=ubuntu
XDG_SESSION_TYPE=tty
MOTD_SHOWN=pam
HOME=/home/ubuntu
LANG=C.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or
    ↪ =40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar
    ↪ =01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh
    ↪ =01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z
    ↪ =01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.
    ↪ tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm
    ↪ =01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace
    ↪ =01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm
    ↪ =01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif
    ↪ =01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm
    ↪ =01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx
    ↪ =01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.webp
    ↪ =01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv
    ↪ =01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli
    ↪ =01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm
    ↪ =01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a
    ↪ =00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra
    ↪ =00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
SSH_CONNECTION=154.16.162.143 62340 192.168.0.6 22
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm-256color
LESSOPEN=| /usr/bin/lesspipe %s
USER=ubuntu
SHLVL=1
XDG_SESSION_ID=2169
XDG_RUNTIME_DIR=/run/user/1000
SSH_CLIENT=154.16.162.143 62340 22
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
PATH=/home/ubuntu/.local/bin:/home/ubuntu/.cargo/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/
    ↪ usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
SSH_TTY=/dev/pts/0
_=/usr/bin/printenv
```