MSWasm: Soundly Enforcing Memory-Safe Execution of Unsafe Code*
(Technical Report)

Contents

1 Memory Safety Monitor **3**

2 Source Language **5**
2.1 Syntax . . . 5
2.2 Typing . . . 6
2.3 Source Semantics . . . 8
2.4 Source Examples . . . 12
2.5 Source Memory Safety . . . 13

3 Target Language: MSWasm **15**
3.1 Wasm Syntax . . . 15
3.2 Wasm Static Semantics . . . 17
3.3 Wasm Dynamic Semantics . . . 19
3.4 MSWasm Extensions . . . 22
3.5 MSWasm Memory Safety . . . 25
3.6 MSWasm Language Properties . . . 26
3.6.1 Type Preservation . . . 27
3.6.2 Memory Safety Properties . . . 29

4 Compiler **36**
4.1 Compilation of Programs . . . 36
4.2 Evaluation Context Compilation . . . 38
4.3 Trace Relation . . . 41
4.4 Equivalence Relation for State . . . 41

*To better explain and clarify notions, this paper uses colours. For a better experience, please print or view this paper in colours.
Specifically, in this paper we use a blue, sans-serif font for source elements, an **orange**, **bold** font for **target** elements, and a *black*, *italic* font for elements common to either languages (to avoid repeating similar definitions twice). Thus, P is a source program, **P** is a target program, and C is generic notation for either program.

4.5 Runtime Compilation 42
4.6 Configuration Compilation 47
4.7 Type Preserving Compilation 47
4.8 Functional Correctness 49
4.8.1 Split MS Monitor 49
4.8.2 Functional Correctness—Multiple Steps 52
4.8.3 Evaluation Context Lemmas 54
4.8.4 Functional Correctness—Single-step 55
4.8.5 Functional Correctness-Primitive Step 58
4.9 Connection Lemma 68
4.10 MS Preservation 68
4.10.1 MS—Multiple Steps 71
4.11 Non-MS Preservation 77

1 Memory Safety Monitor

$$
\begin{aligned}
\text{Address } & a \in \mathbb{N} \\
\text{Color tag } & c \in \mathbb{C} \\
\text{Shade tag } & s \in \mathbb{S} \\
\text{Shading fun. } & \phi : \{1..n-1\} \to \mathbb{S} \\
\text{Event } & \alpha = \epsilon \mid \texttt{read}(a^c) \mid \texttt{write}(a^c) \mid \texttt{alloc}(n, a^c, \phi) \mid \texttt{sfree}\,(a^c) \\
\text{Events } & \overline{\alpha} = \epsilon \mid \alpha \cdot \overline{\alpha} \\
\text{Abstract Segment } & T \in (\mathbb{N} \rightharpoonup \{A(c,s), F(c,s)\}, \times Y) \\
\text{History} & Y = \overline{\alpha}
\end{aligned}
$$

$$\frac{}{\vdash T \overset{\epsilon}{\rightsquigarrow} T}\ \text{(MS-Empty)}$$

$$\frac{T(a) = A(c,s)}{\vdash T \overset{\texttt{read}(a^{(c,s)})}{\rightsquigarrow} T}\ \text{(MS-Read)} \qquad \frac{T(a) = A(c,s)}{\vdash T \overset{\texttt{write}(a^{(c,s)})}{\rightsquigarrow} T}\ \text{(MS-Write)}$$

$$\frac{\begin{array}{cc}\texttt{fresh}\,(c) & \forall j \in \{0..n-1\}.\, T(a+j) = F(\_,\_) \\ T' = T[a+i \mapsto A(c,\phi(i)) \mid i \in \{0..n-1\}] & T'.Y = T.Y \cdot \texttt{alloc}(n, a^c, \phi)\end{array}}{\vdash T \overset{\texttt{alloc}(n,a^c,\phi)}{\rightsquigarrow} T'}\ \text{(MS-Alloc)}$$

$$\frac{\begin{array}{cc}T.Y = {\alpha_1}^* \cdot \texttt{alloc}(n, a^c, \phi) \cdot {\alpha_2}^* & \texttt{sfree}\,(a^c) \notin {\alpha_2}^* \\ T' = T[i \mapsto F(c, s_i) \mid i \mapsto A(c, s_i) \in T] & T'.Y = T.Y \cdot \texttt{sfree}\,(a^c)\end{array}}{\vdash T \overset{\texttt{sfree}(a^c)}{\rightsquigarrow} T'}\ \text{(MS-Free)}$$

We map alloc/frees to single events since we believe this is the closest to a programmer's mental model, while reads and writes are no longer segments.

MS-Wasm memory safety

$$T_0 = ([i \mapsto F(c_0, s_0) | i \in 0..n-1], []) \qquad \text{for size } n$$

$$\texttt{MS}\,(\overline{\alpha}) \overset{\textbf{def}}{=} \texttt{MS}\,(\overline{\alpha}, T_0)$$

$$\texttt{MS}\,(\overline{\alpha}, T) \overset{\textbf{def}}{=} \exists T'. \vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T'$$

$$\frac{}{\overline{\alpha} \vdash T \overset{\epsilon}{\rightsquigarrow} T}\ \text{(Empty)} \qquad \frac{\vdash T_1 \overset{\alpha}{\rightsquigarrow} T_1' \quad \vdash T_1' \overset{\overline{\alpha}}{\rightsquigarrow} T_2}{\vdash T_1 \overset{\alpha \cdot \overline{\alpha}}{\rightsquigarrow} T_2}\ \text{(Cons)}$$

Memory safety monitor for multiple events $\vdash T \overset{\alpha}{\rightsquigarrow} T'$.

Lemma 1 (Split Abstract MS Monitor)**.** Let $\vdash T \overset{\overline{\alpha}_1 \cdot \overline{\alpha}_2}{\rightsquigarrow} T''$. If $\vdash T \overset{\overline{\alpha}_1}{\rightsquigarrow} T'$ then $\vdash T' \overset{\overline{\alpha}_2}{\rightsquigarrow} T''$.

Proof. Immediate from (Cons) □

Lemma 2 (Split Abstract Segment)**.** If

1. $\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T''$ and
2. $\exists T', T' \subseteq T''$

then $\exists \overline{\alpha}_1, \overline{\alpha}_2$ such that

- $\overline{\alpha} = \overline{\alpha}_1 \cdot \overline{\alpha}_2$ and
- $\vdash T \overset{\overline{\alpha}_1}{\rightsquigarrow} T'$ and
- $\vdash T' \overset{\overline{\alpha}_2}{\rightsquigarrow} T''$

Proof. Immediate from (Cons) □

2 Source Language

A C-like language.

2.1 Syntax

Additions

$$
\begin{aligned}
\text{Modules } \mathsf{P}, \mathsf{M} &::= \mathsf{I}, \mathsf{D}, \mathsf{F}, \mathsf{n}_{\mathsf{hs}} \\
\text{Imports } \mathsf{I} &::= \varnothing \mid \mathsf{I}; (\mathsf{w}\ \mathsf{g}(\mathsf{x} : \mathsf{w})) \\
\text{Struct. Defs. } \mathsf{D} &::= \varnothing \mid \mathsf{s} \mapsto \mathsf{S} \\
\text{Structs } \mathsf{S} &::= \mathsf{f} : \tau \mid \mathsf{f} : \tau; \mathsf{S} \\
\text{Types } \mathsf{w} &::= \tau \mid \mathsf{struct}\ \mathsf{s} \mid \mathsf{array}\ \tau \\
\text{Word Types } \tau &::= \mathsf{int} \mid \mathsf{ptr}\ \mathsf{w} \\
\text{Functions } \mathsf{F} &::= \varnothing \mid \tau\ \mathsf{g}(\mathsf{x} : \tau) \mapsto ((\mathsf{y} : \tau)^*, \mathsf{e}); \mathsf{F} \\
\text{Function Types } \sigma &::= \tau \to \tau \\
\text{Expressions } \mathsf{e} &::= \mathsf{n} \mid \mathsf{n}^{(\mathsf{n},\mathsf{n},\mathsf{w},\mathsf{n}_{\mathsf{id}})} \mid \mathsf{x} \mid \mathsf{e}; \mathsf{e} \\
&\quad \mid \mathsf{malloc}(\tau, \mathsf{e}) \mid \mathsf{malloc}(\mathsf{w}) \mid \mathsf{e} \oplus \mathsf{e} \\
&\quad \mid \mathsf{e}[\mathsf{e}] := \mathsf{e} \mid \&\mathsf{e} \to \mathsf{f} \mid *\mathsf{e} \mid \mathsf{e}[\mathsf{e}] \\
&\quad \mid \mathsf{x} := \mathsf{e} \mid *\mathsf{e} := \mathsf{e} \mid \mathsf{x} := \mathsf{call}\ \mathsf{g}\ (\mathsf{e}) \\
&\quad \mid \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ \mathsf{e}\ \mathsf{else}\ \mathsf{e} \mid \mathsf{free}(\mathsf{e}) \\
\text{Values } \mathsf{v} &::= \mathsf{n} \mid \mathsf{n}^{(\mathsf{n},\mathsf{n},\mathsf{w},\mathsf{n}_{\mathsf{id}})} \\
\text{Slots } \mathsf{s} &::= (\mathsf{n}, \mathsf{n}, \mathsf{w}, \mathsf{n}_{\mathsf{id}}) \\
\text{Allocator States } \mathsf{A} &::= (\mathsf{s}^*, \mathsf{s}^*, \mathsf{Y}) \\
\text{Allocator Histories } \mathsf{Y} &::= \overline{\alpha} \\
\text{Heaps } \mathsf{H} &::= \mathsf{v}^* \\
\text{Eval. Contexts } \mathsf{E} &::= [\cdot] \mid \mathsf{E}; \mathsf{e} \mid \mathsf{E} \oplus \mathsf{e} \mid \mathsf{v} \oplus \mathsf{E} \\
&\quad \mid \&\mathsf{E} \to \mathsf{f} \mid *\mathsf{E} \mid \mathsf{x} := \mathsf{E} \mid *\mathsf{E} := \mathsf{e} \mid *\mathsf{v} := \mathsf{E} \\
&\quad \mid \mathsf{E}[\mathsf{e}] \mid \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n}_{\mathsf{id}})}[\mathsf{E}] \mid \mathsf{E}[\mathsf{e}_2] := \mathsf{e}_3 \\
&\quad \mid \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n}_{\mathsf{id}})}[\mathsf{E}] := \mathsf{e} \mid \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n}_{\mathsf{id}})}[\mathsf{n}] := \mathsf{E} \\
&\quad \mid \mathsf{malloc}(\tau, \mathsf{E}) \mid \mathsf{x} := \mathsf{call}\ \mathsf{g}\ (\mathsf{E}) \\
&\quad \mid \mathsf{if}\ \mathsf{E}\ \mathsf{then}\ \mathsf{e}\ \mathsf{else}\ \mathsf{e} \mid \mathsf{free}(\mathsf{E}) \\
\text{Environments } \Gamma &::= \varnothing \mid \Gamma; (\mathsf{x} : \tau) \\
\text{Struct Envs. } \Sigma &::= \varnothing \mid \Sigma; (\mathsf{n}^{\tau}) \\
\text{Stack Frames } \theta &::= \varnothing \mid \mathsf{x} \mapsto \mathsf{v}; \theta \\
\text{Stacks } \mathsf{K} &::= \varnothing \mid \theta_{\mathsf{f}}; \mathsf{K} \\
\text{Op. Sem. Prog. States } \Omega &::= \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright (\mathsf{e}, \mathsf{E}^*) \\
\text{Substitutions } \gamma &::=
\end{aligned}
$$

$$\text{Traces } \overline{\alpha} ::= \varnothing \mid \alpha \cdot \overline{\alpha}$$
$$\text{Actions } \alpha ::= \epsilon \mid \mathsf{alloc}(\mathsf{a}^{(\mathsf{n},\mathsf{n},\mathsf{w},\mathsf{n}_{\mathsf{id}})}) \mid \mathsf{read}(\mathsf{n}^{(\mathsf{n},\mathsf{n},\mathsf{w},\mathsf{n}_{\mathsf{id}})}, \mathsf{v}) \mid \mathsf{write}(\mathsf{n}^{(\mathsf{n},\mathsf{n},\mathsf{w},\mathsf{n}_{\mathsf{id}})}, \mathsf{v}) \mid \mathsf{fread}(\mathsf{n}, \mathsf{v}) \mid \mathsf{fwrite}(\mathsf{n}, \mathsf{v}) \mid \mathsf{free}(\mathsf{n}^{(\mathsf{n},\mathsf{n},\mathsf{w},\mathsf{n}_{\mathsf{id}})}) \mid \mathsf{ffree}\ \mathsf{n}$$

2.2 Typing

$\Gamma_{\mathsf{f}} = (\tau_{\mathsf{r}}, \{\mathsf{x} : \tau_{\mathsf{a}}, \mathsf{y}_1 : \tau_1, \mathsf{y}_2 : \tau_2, \dots\})$ where $\tau_{\mathsf{r}}\ \mathsf{f}(\mathsf{x} : \tau_{\mathsf{a}}) \mapsto ((\mathsf{y}_1 : \tau_1)^*, \mathsf{e}_{\mathsf{f}}) \in \mathsf{P.F}$
$\Delta(\mathsf{f}) = \Gamma_{\mathsf{f}}$

Typing judgments are as follows:

Typing expressions	$\mathsf{P}, \Gamma \vdash \mathsf{e} : \tau$
Typing evaluation contexts	$\mathsf{P}, \Gamma \vdash \mathsf{E} : \mathsf{w} \Rightarrow \mathsf{w}'$
Typing functions	$\mathsf{P} \vdash \mathsf{w}_{\mathsf{r}}\ \mathsf{f}(\mathsf{x} : \mathsf{w}_{\mathsf{a}}) \mapsto \mathsf{e}_{\mathsf{f}} : \mathsf{w}_{\mathsf{a}} \to \mathsf{w}_{\mathsf{r}}$
Typing programs	$\vdash \mathsf{P} : \mathsf{wt}$
Typing configurations	$\vdash \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright (\mathsf{e}_{\mathsf{f}}, \mathsf{E}^*)$

(T-Config)
$$\frac{\mathsf{P}, \Gamma_{\mathsf{f}} \vdash \mathsf{e} : \mathsf{w} \quad \mathsf{P}, \Delta \vdash \mathsf{E}^* : \mathsf{w} \Rightarrow \mathsf{w}' \quad \mathsf{P}, \Delta \vdash \mathsf{K}}{\vdash \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright (\mathsf{e}_{\mathsf{f}}, \mathsf{E}^*)}$$

(T-Prog)
$$\frac{\forall \tau_{\mathsf{r}}\ \mathsf{f}(\mathsf{x} : \tau_{\mathsf{a}}) \mapsto ((\mathsf{y} : \tau)^*, \mathsf{e}_{\mathsf{f}}) \in \mathsf{P.F}, \mathsf{P} \vdash \tau_{\mathsf{r}}\ \mathsf{f}(\mathsf{x} : \tau_{\mathsf{a}}) \mapsto ((\mathsf{y} : \tau)^*, \mathsf{e}_{\mathsf{f}}) : \tau_{\mathsf{a}} \to \tau_{\mathsf{r}}}{\vdash \mathsf{P} : \mathsf{wt}}$$

Typing Programs and Configurations

(T-Stack-Frame-Nil)
$$\frac{}{\mathsf{P}, \Gamma_{\mathsf{f}} \vdash \varnothing}$$

(T-Stack-Frame)
$$\frac{\mathsf{P}, \Gamma_{\mathsf{f}} \vdash \mathsf{x} : \mathsf{w} \quad \mathsf{P}, \Gamma_{\mathsf{f}} \vdash \mathsf{v} : \mathsf{w} \quad \mathsf{P}, \Gamma_{\mathsf{f}} \vdash \theta_{\mathsf{f}}}{\mathsf{P}, \Gamma_{\mathsf{f}} \vdash (\mathsf{x} \mapsto \mathsf{v}); \theta_{\mathsf{f}}}$$

(T-Stack-Nil)
$$\frac{}{\mathsf{P}, \Delta \vdash \varnothing}$$

(T-Stack)
$$\frac{\mathsf{P}, \Delta(\mathsf{f}) \vdash \theta_{\mathsf{f}} \quad \mathsf{P}, \Delta \vdash \mathsf{K}}{\mathsf{P}, \Delta \vdash \theta_{\mathsf{f}}; \mathsf{K}}$$

(T-Ctx-Stack-Nil)
$$\frac{\mathsf{P}, \Gamma_{\mathsf{f}} \vdash \mathsf{E}_{\mathsf{f}} : \mathsf{w}' \Rightarrow \mathsf{w}}{\mathsf{P} \vdash \mathsf{E}_{\mathsf{f}}; \varnothing : \mathsf{w}' \Rightarrow \mathsf{w}}$$

(T-Ctx-Stack)
$$\frac{\mathsf{P}, \Delta(\mathsf{f}) \vdash \mathsf{E}_{\mathsf{f}} : \mathsf{w}_1 \Rightarrow \mathsf{w}_2 \quad \mathsf{P}, \Delta \vdash \mathsf{E}^* : \mathsf{w}_2 \Rightarrow \mathsf{w}_3}{\mathsf{P}, \Delta \vdash \mathsf{E}_{\mathsf{f}}; \mathsf{E}^* : \mathsf{w}_1 \Rightarrow \mathsf{w}_3}$$

Typing Stack Frames, Stack and a list of Eval Contexts $\mathsf{P}, \Delta \vdash \mathsf{E}_{\mathsf{f}}; \mathsf{E}^* : \mathsf{w}' \Rightarrow \mathsf{w}$

(T-Fun)
$$\frac{\mathsf{P}, (\mathsf{x} : \mathsf{w}_{\mathsf{a}}, (\mathsf{y} : \mathsf{w})^*) \vdash \mathsf{e}_{\mathsf{f}} : \mathsf{w}_{\mathsf{r}}}{\mathsf{P} \vdash \mathsf{w}_{\mathsf{r}}\ \mathsf{f}(\mathsf{x} : \mathsf{w}_{\mathsf{a}}) \mapsto ((\mathsf{y} : \mathsf{w})^*, \mathsf{e}_{\mathsf{f}}) : \mathsf{w}_{\mathsf{a}} \to \mathsf{w}_{\mathsf{r}}}$$

Typing Functions

(T-Constant-int)
$$\frac{}{P, \Gamma \vdash n : \text{int}}$$

(T-Raw-Ptr)
$$\frac{}{P, \Gamma \vdash n : \text{ptr}\, w}$$

(T-Authentic-Ptr)
$$\frac{w \neq \text{array}\ \tau}{P, \Gamma \vdash n^{(n_b, 1, w, n_{id})} : \text{ptr}\, w}$$

(T-Authentic-Arr-Ptr)
$$\frac{n_\ell > 1}{P, \Gamma \vdash n^{(n_b, n_\ell, \tau, n_{id})} : \text{ptr array}\ \tau}$$

(T-Var)
$$\frac{\Gamma(x) = w}{P, \Gamma \vdash x : w}$$

(T-BinOp)
$$\frac{P, \Gamma \vdash e_j : \text{int} \quad j \in \{1, 2\}}{P, \Gamma \vdash e_1 \oplus e_2 : \text{int}}$$

(T-Ptr-Arith)
$$\frac{P, \Gamma \vdash e_1 : \text{ptr array}\ \tau \quad P, \Gamma \vdash e_2 : \text{int}}{P, \Gamma \vdash e_1 \oplus e_2 : \text{ptr array}\ \tau}$$

(T-Malloc-Array)
$$\frac{P, \Gamma \vdash e : \text{int}}{P, \Gamma \vdash \text{malloc}(\tau, e) : \text{ptr array}\ \tau}$$

(T-Malloc)
$$\frac{}{P, \Gamma \vdash \text{malloc}(w) : \text{ptr}\, w}$$

(T-Var-Assign)
$$\frac{P, \Gamma \vdash x : \tau \quad P, \Gamma \vdash e : \tau}{P, \Gamma \vdash x := e : \text{int}}$$

(T-Assign)
$$\frac{P, \Gamma \vdash e_1 : \text{ptr}\, \tau \quad P, \Gamma \vdash e_2 : \tau}{P, \Gamma \vdash *e_1 := e_2 : \text{int}}$$

(T-Array-Assign)
$$\frac{P, \Gamma \vdash e_1 : \text{ptr array}\ \tau \quad P, \Gamma \vdash e_3 : \tau \quad P, \Gamma \vdash e_2 : \text{int}}{P, \Gamma \vdash e_1[e_2] := e_3 : \text{int}}$$

(T-Seq)
$$\frac{P, \Gamma \vdash e_1 : w_1 \quad P, \Gamma \vdash e_2 : w_2}{P, \Gamma \vdash e_1; e_2 : w_2}$$

(T-Deref)
$$\frac{P, \Gamma \vdash e : \text{ptr}\, \tau}{P, \Gamma \vdash *e : \tau}$$

(T-Arr-Deref)
$$\frac{P, \Gamma \vdash e_1 : \text{ptr array}\ \tau \quad P, \Gamma \vdash e_2 : \text{int}}{P, \Gamma \vdash e_1[e_2] : \tau}$$

(T-Struct-field)
$$\frac{P, \Gamma \vdash e : \text{ptr struct}\ s \quad P.D(s) = \ldots; \tau_f\ f; \ldots}{P, \Gamma \vdash \&e \to f : \text{ptr}\, \tau_f}$$

(T-If)
$$\frac{P, \Gamma \vdash e_0 : \text{int} \quad P, \Gamma \vdash e_1 : w \quad P, \Gamma \vdash e_2 : w}{P, \Gamma \vdash \text{if}\ e_0\ \text{then}\ e_1\ \text{else}\ e_2 : w}$$

(T-Call)
$$\frac{w_r\ g(x : w_1) \mapsto ((y_j : w_j)^*, e_g) \in P.F \quad P, \Gamma \vdash e_1 : w_1 \quad P, \Gamma \vdash y : w_r}{P, \Gamma \vdash y := \text{call}\ g\ (e_1) : w_r}$$

(T-Free)
$$\frac{P, \Gamma \vdash e : \text{ptr}\, w}{P, \Gamma \vdash \text{free}(e) : \text{int}}$$

Source Typing expressions

(T-Ctx-BinOp1)
$$\frac{P, \Gamma \vdash E : w \Rightarrow \text{int} \quad P, \Gamma \vdash e_2 : \text{int}}{P, \Gamma \vdash E \oplus e_2 : w \Rightarrow \text{int}}$$

(T-Ctx-BinOp2)
$$\frac{P, \Gamma \vdash E : w \Rightarrow \text{int} \quad P, \Gamma \vdash v_1 : \text{int}}{P, \Gamma \vdash v_1 \oplus E : w \Rightarrow \text{int}}$$

(T-Ctx-Malloc)
$$\frac{P, \Gamma \vdash E : w' \Rightarrow \text{int}}{P, \Gamma \vdash \text{malloc}(\tau, E) : w' \Rightarrow \text{array}\ \tau}$$

(T-Ctx-Var-Assign)
$$\frac{P, \Gamma \vdash E : w' \Rightarrow w \quad P, \Gamma \vdash x : w}{P, \Gamma \vdash x := E : w' \Rightarrow w}$$

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{ptr}\,\mathsf{w} \quad \mathsf{P},\Gamma \vdash \mathsf{e}_2 : \mathsf{w}}{\mathsf{P},\Gamma \vdash *\mathsf{E} := \mathsf{e}_2 : \mathsf{w}' \Rightarrow \mathsf{int}}\ (\text{T-Ctx-Assign1})$$

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{w} \quad \mathsf{P},\Gamma \vdash \mathsf{v} : \mathsf{ptr}\,\mathsf{w}}{\mathsf{P},\Gamma \vdash *\mathsf{v} := \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{int}}\ (\text{T-Ctx-Assign2})$$

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{ptr}\,\mathsf{array}\ \tau \quad \mathsf{P},\Gamma \vdash \mathsf{e}_2 : \mathsf{int} \quad \mathsf{P},\Gamma \vdash \mathsf{e}_3 : \tau}{\mathsf{P},\Gamma \vdash \mathsf{E}[\mathsf{e}_2] := \mathsf{e}_3 : \mathsf{w}' \Rightarrow \mathsf{int}}\ (\text{T-Ctx-Arr-Assign1})$$

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w} \Rightarrow \mathsf{int} \quad \mathsf{P},\Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w}',\mathsf{n}_{\mathsf{id}})} : \mathsf{ptr}\,\mathsf{array}\ \tau \quad \mathsf{P},\Gamma \vdash \mathsf{e} : \tau}{\mathsf{P},\Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\tau,\mathsf{n}_{\mathsf{id}})}[\mathsf{E}] := \mathsf{e} : \mathsf{w} \Rightarrow \mathsf{int}}\ (\text{T-Ctx-Arr-Assign2})$$

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w} \Rightarrow \tau \quad \mathsf{P},\Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\tau,\mathsf{n}_{\mathsf{id}})} : \mathsf{ptr}\,\mathsf{array}\ \tau \quad \mathsf{P},\Gamma \vdash \mathsf{n} : \mathsf{int}}{\mathsf{P},\Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\tau,\mathsf{n}_{\mathsf{id}})}[\mathsf{n}] := \mathsf{E} : \mathsf{w} \Rightarrow \mathsf{int}}\ (\text{T-Ctx-Arr-Assign3})$$

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{w}_1 \quad \mathsf{P},\Gamma \vdash \mathsf{e} : \mathsf{w}}{\mathsf{P},\Gamma \vdash \mathsf{E};\mathsf{e} : \mathsf{w}' \Rightarrow \mathsf{w}}\ (\text{T-Ctx-Seq})$$

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{ptr}\,\mathsf{w}}{\mathsf{P},\Gamma \vdash *\mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{w}}\ (\text{T-Ctx-Deref})$$

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w} \Rightarrow \mathsf{ptr}\,\mathsf{array}\ \tau \quad \mathsf{P},\Gamma \vdash \mathsf{e} : \mathsf{int}}{\mathsf{P},\Gamma \vdash \mathsf{E}[\mathsf{e}] : \mathsf{w} \Rightarrow \tau}\ (\text{T-Ctx-Arr-Deref1})$$

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\tau,\mathsf{n}_{\mathsf{id}})} : \mathsf{ptr}\,\mathsf{array}\ \tau \quad \mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w} \Rightarrow \mathsf{int}}{\mathsf{P},\Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\tau,\mathsf{n}_{\mathsf{id}})}[\mathsf{E}] : \mathsf{w} \Rightarrow \tau}\ (\text{T-Ctx-Arr-Deref2})$$

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w} \Rightarrow \mathsf{ptr}\,\mathsf{struct}\ \mathsf{s} \quad \mathsf{P.D}(\mathsf{s}) = \ldots;\tau_{\mathsf{f}}\ \mathsf{f};\ldots}{\mathsf{P},\Gamma \vdash \&\mathsf{E} \to \mathsf{f} : \mathsf{w} \Rightarrow \mathsf{ptr}\,\tau_{\mathsf{f}}}\ (\text{T-Ctx-Struct-field})$$

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{int} \quad \mathsf{P},\Gamma \vdash \mathsf{e}_1 : \mathsf{w} \quad \mathsf{P},\Gamma \vdash \mathsf{e}_2 : \mathsf{w}}{\mathsf{P},\Gamma \vdash \mathsf{if}\ \mathsf{E}\ \mathsf{then}\ \mathsf{e}_1\ \mathsf{else}\ \mathsf{e}_2 : \mathsf{w}' \Rightarrow \mathsf{w}}\ (\text{T-Ctx-Branch})$$

$$\frac{\mathsf{w}_{\mathsf{r}}\ \mathsf{g}(\mathsf{x} : \mathsf{w}_1) \mapsto (\mathsf{y}_{\mathsf{j}} : \mathsf{w}_{\mathsf{j}})^*, \mathsf{e}_{\mathsf{g}}) \in \mathsf{P.F} \quad \mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w} \Rightarrow \mathsf{w}_1 \quad \mathsf{P},\Gamma \vdash \mathsf{y} : \mathsf{w}_{\mathsf{r}}}{\mathsf{P},\Gamma \vdash \mathsf{y} := \mathsf{call}\ \mathsf{g}\ (\mathsf{E}) : \mathsf{w} \Rightarrow \mathsf{w}_{\mathsf{r}}}\ (\text{T-Ctx-Call})$$

Source Typing Evaluation Contexts $\mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w} \Rightarrow \mathsf{w}'$

2.3 Source Semantics

The semantics follow these judgements:

$\mathsf{M} \vdash \mathsf{P},\mathsf{K},\mathsf{H},\mathsf{A} \triangleright \mathsf{e} \xrightarrow{\alpha}\!\!\!\!\rightarrow \mathsf{P},\mathsf{K}',\mathsf{H}',\mathsf{A}' \triangleright \mathsf{e}'$ primitive, labelled reductions

$\Omega \xRightarrow{\alpha} \Omega'$ single top-level small-step reduction

$\Omega \xRightarrow{\overline{\alpha}*} \Omega'$ reflexive-transitive closure of $\Longrightarrow$

$\mathsf{P},\mathsf{K},\mathsf{H},\mathsf{A} \triangleright \mathsf{e} \Downarrow_{\overline{\alpha}} \mathsf{P},\mathsf{K}',\mathsf{H}',\mathsf{A}' \triangleright \mathsf{v}$ big step trace semantics

Stacks work as follows: when a call is done, stuff is pushed from the current pointer, erasing the next entry When a return is done, the current pointer is shifted back by one. A stack contains a list of mappings from vars to values.

$$\begin{aligned} \mathsf{K} &::= \varnothing \mid \theta; \mathsf{K} \\ \theta &::= \varnothing \mid \mathsf{x} \mapsto \mathsf{v}; \theta \\ \Omega &::= \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright (\mathsf{e}, \mathsf{E}^*) \end{aligned}$$

Some helpers. The initial state has an infinite stack.

$$\frac{(\mathsf{w}\ \mathsf{main}(\mathsf{x} : \mathsf{w})) \mapsto (\mathsf{y} : \mathsf{w})^*, \mathsf{e} \in \mathsf{P}.\mathsf{F}}{\Omega_0\,(\mathsf{P}) = \mathsf{P}, \theta_0, \mathsf{H}_0, \mathsf{A} \triangleright (\mathsf{e}, \varnothing)}\ (\text{Initial})$$

$$\frac{\mathsf{K} = \theta; \mathsf{K}^* \qquad \mathsf{K}' = \theta[\mathsf{x} \,/\, \mathsf{n}]; \mathsf{K}^*}{\mathsf{K} + \mathsf{x} \mapsto \mathsf{n} = \mathsf{K}'}\ (\text{Stack-top-update})$$

(S-Var)

$$\frac{x \mapsto n' \in \theta}{M \vdash P, \theta; K, H, A \triangleright x \xrightarrow{\epsilon}\!\!\!\!\rightarrow\!\!\!\!\rightarrow P, \theta; K, H, A \triangleright n'}$$

(S-BinOp)

$$\frac{n = n_1 \oplus n_2}{M \vdash P, K, H, A \triangleright n_1 \oplus n_2 \xrightarrow{\epsilon}\!\!\!\!\rightarrow\!\!\!\!\rightarrow P, K, H, A \triangleright n}$$

(S-Ptr-Arith)

$$\frac{}{M \vdash P, K, H, A \triangleright a^{(b,\ell,w,n_{id})} \oplus n \xrightarrow{\epsilon}\!\!\!\!\rightarrow\!\!\!\!\rightarrow P, K, H, A \triangleright (a+n)^{(b,\ell,w,n_{id})}}$$

(S-Var-Assign)

$$\frac{K' = K + x \mapsto v}{M \vdash P, K, H, A \triangleright x := v \xrightarrow{\epsilon}\!\!\!\!\rightarrow\!\!\!\!\rightarrow P, K', H, A \triangleright v}$$

(S-Assign)

$$\frac{H' = H[a \mapsto v]}{M \vdash P, K, H, A \triangleright *a^{(b,\ell,w,n_{id})} := v \xrightarrow{write(a^{(b,\ell,w,n_{id})},v)}\!\!\!\!\rightarrow\!\!\!\!\rightarrow P, K, H', A \triangleright 0}$$

(S-Assign-Forge)

$$\frac{H' = H[a \mapsto v]}{M \vdash P, K, H, A \triangleright *a := v \xrightarrow{fwrite(a,v)}\!\!\!\!\rightarrow\!\!\!\!\rightarrow P, K, H', A \triangleright 0}$$

(S-Arr-Assign)

$$\frac{H' = H[a + n \mapsto v]}{M \vdash P, K, H, A \triangleright a^{(b,\ell,w,n_{id})}[n] := v \xrightarrow{write((a+n)^{(b,\ell,w,n_{id})},v)}\!\!\!\!\rightarrow\!\!\!\!\rightarrow P, K, H', A \triangleright 0}$$

(S-Sequence)

$$\frac{}{M \vdash P, K, H, A \triangleright v; e \xrightarrow{\epsilon}\!\!\!\!\rightarrow\!\!\!\!\rightarrow P, K, H', A \triangleright e}$$

(S-Ife)

$$\frac{i = \begin{cases} 1 & \text{if } n = 1 \\ 2 & \text{if } n = 0 \end{cases}}{M \vdash P, K, H, A \triangleright \text{if } n \text{ then } e_1 \text{ else } e_2 \xrightarrow{\epsilon}\!\!\!\!\rightarrow\!\!\!\!\rightarrow P, K, H, A \triangleright e_i}$$

(S-Dereference)

$$\frac{H[a] = v}{M \vdash P, K, H, A \triangleright *a^{(b,\ell,w,n_{id})} \xrightarrow{read(a^{(b,\ell,w,n_{id})},v)}\!\!\!\!\rightarrow\!\!\!\!\rightarrow P, K, H, A \triangleright v}$$

(S-Dereference-Forge)

$$\frac{H[a] = v}{M \vdash P, K, H, A \triangleright *a \xrightarrow{fread(a,v)}\!\!\!\!\rightarrow\!\!\!\!\rightarrow P, K, H, A \triangleright v}$$

(S-Arr-Dereference)

$$\frac{H[a + n] = v}{M \vdash P, K, H, A \triangleright a^{(b,\ell,w,n_{id})}[n] \xrightarrow{read((a+n)^{(b,\ell,w,n_{id})},v)}\!\!\!\!\rightarrow\!\!\!\!\rightarrow P, K, H, A \triangleright v}$$

(S-Struct-Field)

$$\frac{D(s) = \{f_1 : \tau_1, \ldots f : \tau_k \ldots f_n : \tau_n\} \quad o_1 = k - 1}{M \vdash P, K, H, A \triangleright \&a^{(b,1,\text{struct } s,n_{id})} \rightarrow f_k \xrightarrow{\epsilon}\!\!\!\!\rightarrow\!\!\!\!\rightarrow P, K, H, A \triangleright (a+o_1)^{(b+o_1,1,\tau_k,n_{id})}}$$

(Struct-Field-Forge)

$$\frac{D(s) = \{f_1 : \tau_1 \ldots f_k : \tau_k \ldots\}}{M \vdash P, K, H, A \triangleright \&a \to f_k \xrightarrow{\epsilon} P, K, H, A \triangleright a + k}$$

(S-Malloc-Single)

$$\frac{w \neq \text{array} \quad \_ \quad \langle H, A\rangle \xhookrightarrow{\text{alloc}(a^{(a,1,w,n_{id})})} \langle H', A'\rangle}{M \vdash P, K, H, A \triangleright \text{malloc}(w) \xrightarrow{\text{alloc}(a^{(a,1,w,n_{id})})} P, K, H', A' \triangleright a^{(a,1,w,n_{id})}}$$

(S-Malloc-Array)

$$\frac{\langle H, A\rangle \xhookrightarrow{\text{alloc}(a^{(a,\ell,\tau,n_{id})})} \langle H', A'\rangle}{M \vdash P, K, H, A \triangleright \text{malloc}(\tau, \ell) \xrightarrow{\text{alloc}(a^{(a,\ell,\tau,n_{id})})} P, K, H', A' \triangleright a^{(a,\ell,\tau,n_{id})}}$$

(S-Free)

$$\frac{\langle H, A\rangle \xhookrightarrow{\text{free}(a^{(b,\ell,w,n_{id})})} \langle H', A'\rangle}{M \vdash P, K, H, A \triangleright \text{free}(a^{(b,\ell,w,n_{id})}) \xrightarrow{\text{free}(a^{(b,\ell,w,n_{id})})} P, K, H', A' \triangleright 0}$$

(S-Free-Nop)

$$\frac{\langle H, A\rangle \xhookrightarrow{\text{free}(a^{(b,\ell,w,n_{id})})} \langle H, A\rangle}{M \vdash P, K, H, A \triangleright \text{free}(a^{(b,\ell,w,n_{id})}) \xrightarrow{\text{free}(a^{(b,\ell,w,n_{id})})} P, K, H, A \triangleright 0}$$

(S-Free-Forge)

$$\frac{\langle H, A\rangle \xhookrightarrow{\text{ffree } a} \langle H', A'\rangle}{M \vdash P, K, H, A \triangleright \text{free}(a) \xrightarrow{\text{ffree } a} P, K, H', A' \triangleright 0}$$

Redex step: $M \vdash P, K, H, A \triangleright e \xrightarrow{\alpha} P, K', H', A' \triangleright e'$

(S-Step)

$$\frac{M \vdash P, K, H, A \triangleright e \xrightarrow{\alpha} P, K', H', A' \triangleright e'}{P, K, H, A \triangleright (E[e], E^*) \xRightarrow{\alpha} P, K', H', A' \triangleright (E[e'], E^*)}$$

(S-Call)

$$\frac{(w\ g(x' : w')) \mapsto ((y_j : w_j)^*, e_b) \in P.F \quad K' = K; \{x' \mapsto n', y_0 \mapsto 0, y_1 \mapsto 0, \ldots\}}{P, K, H, A \triangleright (E_1[x := \text{call } g\ (n')], E^*) \xRightarrow{\epsilon} P, K', H, A \triangleright (e_b, E_1[x := [\cdot]] : E^*)}$$

(S-Return)

$$\frac{K' = K; \{y \mapsto \ldots\}}{P, K', H, A \triangleright (v, E_1 : E^*) \xRightarrow{\epsilon} P, K, H, A \triangleright (E_1[v], E^*)}$$

Source Top-level step: $P, K, H, A \triangleright (e, E^*) \xRightarrow{\alpha} P, K', H', A' \triangleright (e', E'^*)$

(S-Base)

$$\frac{}{\Omega \xrightarrow{\epsilon}{}^{*} \Omega}$$

(S-Steps)

$$\frac{\Omega \xrightarrow{\alpha} \Omega' \quad \Omega' \xrightarrow{\overline{\alpha}}{}^{*} \Omega''}{\Omega \xrightarrow{\alpha \cdot \overline{\alpha}}{}^{*} \Omega''}$$

Source Trace Semantics

(Src-Alloc)

$$\frac{\begin{array}{c} s_F^* = s_1^* \cdot (a, \ell', w', n'_{id}) \cdot s_2^* \qquad 0 < \ell \leq \ell' \qquad a + \ell < |H| \\ H' = H[a + j \mapsto 0 \mid j \in 0..\ell - 1] \qquad Y' = Y \cdot \mathsf{alloc}\left(a^{(a,\ell,w,n_{id})}\right) \\ s_F'^* = s_1^* \cdot (a + n, \ell' - \ell, w', n'_{id}) \cdot s_2^* \qquad \mathsf{fresh}(n_{id}) \qquad s_A'^* = (a, \ell, w, n_{id}) \cdot s_U^* \end{array}}{\langle H, (s_A^*, s_F^*, Y)\rangle \xrightarrow{\mathsf{alloc}\left(a^{(a,\ell,w,n_{id})}\right)} \langle H', (s_A'^*, s_F'^*, Y')\rangle}$$

(Src-Free)

$$\frac{\begin{array}{c} s_A^* = s_1^* \cdot (a, \ell, w, n_{id}) \cdot s_2^* \qquad s_F'^* = (a, \ell, w, n_{id}) \cdot s_F^* \\ s_A'^* = s_1^* \cdot s_2^* \qquad Y' = Y \cdot \mathsf{free}\left(a^{(a,\ell,w,n_{id})}\right) \end{array}}{\langle H, (s_A^*, s_F^*, Y)\rangle \xrightarrow{\mathsf{free}\left(a^{(a,\ell,w,n_{id})}\right)} \langle H, (s_A'^*, s_F'^*, Y')\rangle}$$

(Src-Free-Nop)

$$\frac{(\nexists (a, \_, \_, n_{id}) \in s_A^*) \vee (\mathsf{free}(a^{(b,\ell,w,n_{id})}) \in Y)}{\langle H, (s_A^*, s_F^*, Y)\rangle \xrightarrow{\mathsf{free}(a^{(b,\ell,w,n_{id})})} \langle H, (s_A^*, s_F^*, Y)\rangle}$$

(Src-Free-Forged)

$$\frac{s_A^* = s_1^* \cdot (a, \_, \_, \_) \cdot s_2^* \qquad s_F'^* = (a, \_, \_, \_) \cdot s_F^* \qquad s_A'^* = s_1^* \cdot s_2^*}{\langle H, (s_A^*, s_F^*, Y)\rangle \xrightarrow{\mathsf{ffree}\ a} \langle H, (s_A^*, s_F^*, Y)\rangle}$$

(Src-Free-Forged-Nop)

$$\frac{(a, \_, \_, \_) \notin s_A^*}{\langle H, (s_A^*, s_F^*, Y)\rangle \xrightarrow{\mathsf{ffree}\ a} \langle H, (s_A'^*, s_F'^*, Y')\rangle}$$

Source allocator semantics

2.4 Source Examples

Example 1. The below snippet of code allocates a pointer to array of 10 integers and then assigns 42 to array index 2.

```
1  x := malloc(ptr (array int));   /* Allocate a pointer to array */
2  y := malloc(int, 10);           /* Allocate an array of size 10 int */
3  *x := y;                        /* Contents of pointer point to the above array */
4  **x ⊕ 2 := 42;                  /* y[2] := 42 */
5  **x ⊕ 2                         /* Output y[2] */
```

The type of x is ptr (array int); type of y is array int and the type of $*(*x \oplus 2)$ is int.

2.5 Source Memory Safety

Definition 1 (Source Color Assignment)**.** The source color assignment, δ, is a map from source address to the abstract memory location and the tuple of color ($\mathbb{N}$) and shades ($\mathbb{N}$). Without loss of generality, we use shade 0 for array allocations.

$$\delta : \mathsf{Address} \times \mathsf{Tid} \to \mathbb{N} \times \{\mathbb{N}, \mathbb{N}\}$$

Definition 2 (Valid δ)**.** A $\delta : \mathsf{Address} \times \mathsf{Tid} \to \mathbb{N} \times \{\mathbb{N}, \mathbb{N}\}$ is a valid coloring scheme, denoted $\vdash \delta$, if the mapped colors are unique.

Definition 3 (Trace Agreement ($\alpha =_\delta \alpha$))**.**

(Tr-Concatenate)
$$\frac{\alpha =_\delta \overline{\alpha}_1 \quad \overline{\alpha} =_\delta \overline{\alpha}_2}{\alpha \cdot \overline{\alpha} =_\delta \overline{\alpha}_1 \cdot \overline{\alpha}_2}$$

(Tr-Read-Authentic)
$$\frac{n = \texttt{srcsz}\,(\mathsf{w}) \quad \delta(\mathsf{b}, \mathsf{n_{id}}) = b^{(c,s)} \quad \mathsf{o} = \mathsf{a} - \mathsf{b} \quad a = b + (\mathsf{o} * n)}{\mathsf{read}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}, \mathsf{v}) =_\delta \texttt{read}(a^{(c,s)}) \cdot \texttt{read}((a+1)^{(c,s)}) \cdot \ldots \cdot \texttt{read}((a+n-1)^{(c,s)})}$$

(Tr-Write-Authentic)
$$\frac{n = \texttt{srcsz}\,(\mathsf{w}) \quad \delta(\mathsf{b}, \mathsf{n_{id}}) = b^{(c,s)} \quad \mathsf{o} = \mathsf{a} - \mathsf{b} \quad a = b + (\mathsf{o} * n)}{\mathsf{write}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}, \mathsf{v}) =_\delta \texttt{write}(a^{(c,s)}) \cdot \texttt{write}((a+1)^{(c,s)}) \cdot \ldots \cdot \texttt{write}((a+n-1)^{(c,s)})}$$

(Tr-SAlloc)
$$\frac{n = \texttt{srcsz}\,(\mathsf{w}) * \mathsf{n} \quad \delta(\mathsf{a}, \mathsf{n_{id}}) = a^{(c,0)} \quad \phi_0 = \lambda\_.0}{\mathsf{salloc}(\mathsf{a}, \mathsf{n}, \tau, \mathsf{n_{id}}) =_\delta \texttt{alloc}(n, a^c, \phi_0)}$$

(Tr-SAlloc-Struct)
$$\frac{\begin{array}{c}\mathsf{D}(\mathsf{s}) = \{\mathsf{f}_0 : \tau_0, \ldots \mathsf{f}_{n-1} : \tau_{n-1}\} \\ \forall i \in \{0..\mathsf{n}-1\}.\, \phi_i = [\Sigma_{k=0}^{i-1} \texttt{srcsz}\,(\tau_k) + j \mapsto i \mid j \in \{0..\texttt{srcsz}\,(\tau_i) - 1\}] \\ \phi = \bigcup_i \phi_i \quad \forall i \in \{0..\mathsf{n}-1\}.\, \delta(\mathsf{a}+i, \mathsf{n_{id}}) = (a + \Sigma_{k=0}^{i-1} \texttt{srcsz}\,(\tau_k))^{(c, \phi(\Sigma_{k=0}^{i-1} \texttt{srcsz}(\tau_k)))}\end{array}}{\mathsf{salloc}(\mathsf{a}, 1, \mathsf{struct\ s}, \mathsf{n_{id}}) =_\delta \texttt{alloc}(\texttt{srcsz}\,(\mathsf{struct\ s}), a^c, \phi)}$$

(Tr-SFree)
$$\frac{\delta(\mathsf{b}, \mathsf{n_{id}}) = a^{(c,\_)}}{\mathsf{free}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}) =_\delta \texttt{sfree}\,(a^c)}$$

Lemma 3 (Empty Source Trace)**.** If $\alpha =_\delta \overline{\alpha}$ and $\overline{\alpha} = \epsilon$, then $\alpha = \epsilon$.

Proof. Case analysis on $\alpha =_\delta \overline{\alpha}$. □

Lemma 4 (Splitting Source Trace Equivalence)**.** If $\alpha \cdot \overline{\alpha} =_\delta \overline{\alpha}$ then $\exists \overline{\alpha}', \overline{\alpha}''$,

1. $\overline{\alpha} = \overline{\alpha}' \cdot \overline{\alpha}''$ and
2. $\alpha =_\delta \overline{\alpha}'$ and
3. $\overline{\alpha} =_\delta \overline{\alpha}''$

Proof. Inverting rule (Tr-Concatenate) gives us the required proof. □

Lemma 5 (Monotonicity of Address Map-1)**.** If $\alpha =_\delta \overline{\alpha}$ and $\delta' \supseteq \delta$, then $\alpha =_{\delta'} \overline{\alpha}$.

Proof. Case analysis on $\alpha =_\delta \overline{\alpha}$. Note that the coloring scheme of δ is preserved in δ', hence the proof follows. □

Definition 4 (Segment Agreement $T =_\delta \mathsf{A}$)**.**

$$\frac{\begin{array}{c}\text{(Allocator-Shadow-Memory-Agreement)}\\ \forall(\mathsf{b}, \mathsf{n_{id}}, a^{(c,s)}) \in \delta \\ \forall X \in \{A, F\} \\ \exists \tau, \ell.(\mathsf{b}, \ell, \tau, \mathsf{n_{id}}) \in \mathsf{s_X}^* \iff \forall i \in \{0..\ell \times \texttt{srcsz}\,(\tau) - 1\}. T(a+i) = X(c,s) \\ \vee \\ \exists \mathsf{s}.(\mathsf{b}, 1, \mathsf{struct\ s}, \mathsf{n_{id}}) \in \mathsf{s_X}^* \wedge \mathsf{D(s)} = \{\mathsf{f}_1 : \tau_1, \mathsf{f}_2 : \tau_2 \ldots \mathsf{f_k} : \tau_\mathsf{k}\} \iff \\ \forall i \in \{0..k-1\}. a_i^{(c,s_i)} = \delta(\mathsf{b}+i, \mathsf{n_{id}}) \wedge \\ a_i = a_0 + \Sigma_{h=0}^{i=1} \texttt{srcsz}\,(\tau_h) \wedge \\ \forall j \in 0..\texttt{srcsz}\,(\tau_i) - 1. T(a_i + j) = X(c, s_i) \\ \mathsf{Y} =_\delta T.Y\end{array}}{T =_\delta \mathsf{s_A}^*, \mathsf{s_F}^*, \mathsf{Y}}$$

Definition 5 (Top-level Memory Safety for C-like)**.** $\mathtt{MS}(\overline{\alpha}) \stackrel{\text{def}}{=} \exists \overline{\alpha}, \delta,$

1. $\overline{\alpha} =_\delta \overline{\alpha}$ and
2. $\mathtt{MS}\,(\overline{\alpha})$

3 Target Language: MSWasm

We first describe the basic Wasm language semantics and then incrementally add MSWasm features.

3.1 Wasm Syntax

$$
\begin{aligned}
&*\text{Indexes } \mathbf{n} \in \mathbb{N} \\
&*\text{Value Types } \tau ::= \mathbf{u32} \mid \mathbf{s32} \mid ... \\
&*\text{Instr. Types } \rho ::= \tau^* \to \tau^* \\
&+\text{Import Types } \beta ::= \rho^* \\
&+\text{Module Types } \mu ::= \beta \to \beta \\
&*\text{Typing Ctx. } \mathbf{\Gamma} ::= \{\mathbf{funcs}\ \rho^*, \mathbf{locals}\ \tau^*, \mathbf{return}\ \tau^*\} \\
&*\text{Instructions } \mathbf{i} ::= \mathbf{nop} \mid \mathbf{trap} \mid \tau.\mathbf{const}\ \mathbf{c} \mid \boldsymbol{\tau}.\otimes \mid \mathbf{get}\ \mathbf{n} \mid \mathbf{set}\ \mathbf{n} \\
&\qquad\qquad \mid \tau.\mathbf{load}\ \mathbf{n} \mid \tau.\mathbf{store}\ \mathbf{n} \mid \mathbf{dup} \mid \mathbf{drop} \mid \mathbf{branch}\ \mathbf{i}^*\ \mathbf{i}^* \\
&\qquad\qquad \mid \mathbf{call}\ \mathbf{n} \mid \mathbf{return} \mid \mathbf{alloc}\ \mathbf{n} \mid \mathbf{free}\ \mathbf{n} \\
&*\text{Functions } \mathbf{f} ::= \{\mathbf{var}\ \tau^*, \mathbf{body}\ \mathbf{i}^*\} \\
&*\text{Imports } \mathbf{Imp} ::= \{\mathbf{funcs}\ \mathbf{n}^*\} \\
&*\text{Functions } \mathbf{\Phi}^* ::= (\mathbf{f} : \rho)^* \\
&*\text{Module Defs. } \mathbf{M} ::= \{\mathbf{heaps}\ \mathbf{u32}, \mathbf{funcs}\ \mathbf{\Phi}^*, \mathbf{import}\ \mathbf{Imp}\}
\end{aligned}
$$

Modules Syntax.

$$
\begin{aligned}
&*\text{Module Id. } \mathbf{m} \\
&*\text{Funs. and Memories Id. } \pi ::= \mathbf{n} \\
&*\text{Bytes } \mathbf{b} \in \{0, ..., 2^8 - 1\} \\
&*\text{Heaps } \mathbf{H} ::= \mathbf{b}^* \\
&*\text{Values } \mathbf{v} ::= \mathbf{c} \mid \cdots \\
&+\text{Local Maps } \theta \in \mathbb{N} \rightharpoonup \mathbf{Value} \\
&*\text{Local Frames } \mathbf{F} ::= (\theta, \mathbf{i}^*, \mathbf{v}^*) \\
&*\text{Stack Frames } \mathbf{S} ::= \mathbf{F}_\pi \\
&\text{Slots } \mathbf{s} ::= (\mathbf{n}, \mathbf{n}) \\
&\text{Allocator States } \mathbf{A} ::= (\mathbf{s}^*, \mathbf{s}^*) \\
&\text{Module Stores } \mathbf{\Sigma}, \mathbf{I} \in \langle \mathbf{H}, \mathbf{A} \rangle \\
&+\text{Local Configurations } \mathbf{L} ::= \langle \mathbf{\Sigma}, \mathbf{F} \rangle \\
&*\text{Configurations } \mathbf{\Omega} ::= \langle \mathbf{\Sigma}, \mathbf{S}^* \rangle \\
&\text{Local Events } \gamma ::= \mathbf{read}_{\boldsymbol{\tau}}\ (\mathbf{a}) \mid \mathbf{write}_{\boldsymbol{\tau}}\ (\mathbf{a}) \mid \mathbf{alloc}(\mathbf{a}, \mathbf{n}) \mid \mathbf{free}(\mathbf{a})
\end{aligned}
$$

$$\begin{aligned} & \mid \mathbf{call}(\pi, \mathbf{v}^*, \mathbf{i}^*) \mid \mathbf{return}(\mathbf{v}^*) \mid \mathbf{trap} \mid \epsilon \\ \text{Events } \alpha ::= {} & \mathbf{read}_{\boldsymbol{\tau}}(\mathbf{a}) \mid \mathbf{write}_{\boldsymbol{\tau}}(\mathbf{a}) \mid \mathbf{alloc}(\mathbf{a}, \mathbf{n}) \mid \mathbf{free}(\mathbf{a}) \\ & \mid \mathbf{trap} \mid \epsilon \end{aligned}$$

Runtime instances.

3.2 Wasm Static Semantics

$$\frac{}{\Gamma \vdash \mathbf{nop} : [] \to []}\ (\text{Nop}) \qquad \frac{}{\Gamma \vdash \mathbf{trap} : \tau_1^* \to \tau_2^*}\ (\text{Trap})$$

$$\frac{\tau \neq \mathbf{handle}}{\Gamma \vdash \tau.\mathbf{const}\ \mathbf{c} : [] \to [\tau]}\ (\text{Const}) \qquad \frac{\tau \neq \mathbf{handle}}{\Gamma \vdash \tau.\otimes : [\tau, \tau] \to [\tau]}\ (\text{Bop})$$

$$\frac{\Gamma.\mathbf{local}[\mathbf{n}] = \tau}{\Gamma \vdash \mathbf{get}\ \mathbf{n} : [] \to [\tau]}\ (\text{Get}) \qquad \frac{\Gamma.\mathbf{local}[\mathbf{n}] = \tau}{\Gamma \vdash \mathbf{set}\ \mathbf{n} : [\tau] \to []}\ (\text{Set})$$

$$\frac{\mathbf{n} < \Gamma.\mathbf{heaps}}{\Gamma \vdash \tau.\mathbf{load}\ \mathbf{n} : [\mathbf{i32}] \to [\tau]}\ (\text{Load}) \qquad \frac{\mathbf{n} < \Gamma.\mathbf{heaps}}{\Gamma \vdash \tau.\mathbf{write}\ \mathbf{n} : [\mathbf{i32}, \tau] \to []}\ (\text{Store})$$

$$\frac{\mathbf{n} < \Gamma.\mathbf{heaps}}{\Gamma \vdash \mathbf{alloc}\ \mathbf{n} : [\mathbf{i32}] \to [\mathbf{i32}]}\ (\text{Alloc}) \qquad \frac{\mathbf{n} < \Gamma.\mathbf{heaps}}{\Gamma \vdash \mathbf{free}\ \mathbf{n} : [\mathbf{i32}] \to []}\ (\text{Free})$$

$$\frac{\Gamma.\mathbf{funcs}[\mathbf{n}] = \rho}{\Gamma \vdash \mathbf{call}\ \mathbf{n} : \rho}\ (\text{Call}) \qquad \frac{\Gamma.\mathbf{return} = \tau_1^*}{\Gamma \vdash \mathbf{return} : \tau_1^* \tau^* \to \tau_2^*}\ (\text{Return})$$

$$\frac{\mathbf{j} \in \{\mathbf{1}, \mathbf{2}\} \quad \Gamma \vdash \mathbf{i}_\mathbf{j}^* : [] \to []}{\Gamma \vdash \mathbf{branch}\ \mathbf{i}_\mathbf{1}^*\ \mathbf{i}_\mathbf{2}^* : [\mathbf{i32}] \to []}\ (\text{If}) \qquad \frac{}{\Gamma \vdash \epsilon : \tau^* \to \tau^*}\ (\text{Instr-Empty})$$

$$\frac{\Gamma \vdash \mathbf{i}_\mathbf{1}^* : \tau_1^* \to \tau_2^* \quad \Gamma \vdash \mathbf{i}_\mathbf{2}^* : \tau_2^* \to \tau_3^*}{\Gamma \vdash \mathbf{i}_\mathbf{1}^*\ \mathbf{i}_\mathbf{2}^* : \tau_1^* \to \tau_3^*}\ (\text{Instr-Seq}) \qquad \frac{\Gamma \vdash \mathbf{i}^* : \tau_1^* \to \tau_2^*}{\Gamma \vdash \mathbf{i}^* : \tau_1^* \tau \to \tau_2^* \tau}\ (\text{Instr-Weaken})$$

Wasm Instructions: $\Gamma \vdash \mathbf{i} : \rho$ and $\Gamma \vdash \mathbf{i}^* : \rho$.

$$\frac{\Gamma\{\mathbf{local} = \tau^\mathbf{a} \tau^\mathbf{v}\ \mathbf{return} = \tau^*\} \vdash \mathbf{i}^* : [] \to \tau^*}{\Gamma \vdash \{\mathbf{var}\ \tau^\mathbf{v}, \mathbf{body}\ \mathbf{i}^*\} : \tau^\mathbf{a} \to \tau^*}\ (\text{Fun})$$

$$\frac{\begin{array}{cc}\Gamma = \{\mathbf{funcs} = \rho_\mathbf{I}^* \rho^\mathbf{n}\} & |\mathbf{\Phi}^*| = \mathbf{n} \qquad \forall \mathbf{f_j} \in \mathbf{\Phi}^*.\Gamma \vdash \mathbf{f_j} : \rho_\mathbf{j} \\ \mathbf{Imp} = \{\mathbf{funcs}\ \mathbf{m}^*\} & \rho_\mathbf{I}^* = [\Gamma.\mathbf{funcs}[\mathbf{x}] \mid \mathbf{x} \in \mathbf{m}^*]\end{array}}{\vdash \{\mathbf{heaps}\ \mathbf{u32}, \mathbf{funcs}\ \mathbf{\Phi}^*, \mathbf{import}\ \mathbf{Imp}\}}\ (\text{Module})$$

Functions $\Gamma \vdash \mathbf{f} : \rho$, module definitions $\vdash \mathbf{M}$.

Instance Types.

Instance Ctx. $\mathbf{\Delta} \in \pi \to \mathbf{\Gamma}$

Other types and contexts.

(Stack-Empty)
$$\frac{}{\vdash \epsilon : []}$$

(Handle)
$$\frac{}{\vdash \langle \mathbf{n_{base}}, \mathbf{n_{offset}}, \mathbf{n_{bound}}, \mathbf{b_{valid}} \rangle : \mathbf{handle}}$$

(Stack-Cons)
$$\frac{\vdash \mathbf{v} : \tau \qquad \vdash \mathbf{v}^* : \tau^*}{\vdash \mathbf{v}\mathbf{v}^* : \tau\tau^*}$$

(Local-Map)
$$\frac{\forall \tau_{\mathbf{i}} \in \tau^*.\ \theta\,(\mathbf{i}) = \tau_{\mathbf{i}}}{\vdash \theta : \tau^*}$$

Values stacks $\vdash \mathbf{v}^* : \tau^*$, and local maps $\vdash \theta : \tau^*$.

(Local-Frame)
$$\frac{\mathbf{F} = \langle \theta, \mathbf{i}^*, \mathbf{v}^* \rangle \qquad \vdash \theta : \mathbf{\Gamma.locals} \qquad \vdash \mathbf{v}^* : \tau_{\mathbf{v}}^* \qquad \mathbf{\Gamma} \vdash \mathbf{i}^* : \rho \qquad \rho = \tau_{\mathbf{v}}^* \to \tau_{\mathbf{2}}^*}{\mathbf{\Gamma} \vdash \mathbf{F} : \tau_{\mathbf{v}}^* \to \tau_{\mathbf{2}}^*}$$

(Stack-Frame)
$$\frac{\mathbf{\Delta}(\mathbf{n}) \vdash \mathbf{F} : \rho}{\mathbf{\Delta} \vdash \mathbf{F_n} : \rho}$$

(FStack-Empty)
$$\frac{}{\mathbf{\Gamma} \vdash \epsilon : [] \to []}$$

(FStack-Cons)
$$\frac{\mathbf{\Delta} \vdash \mathbf{F_n} : \tau_{\mathbf{1}}^* \to \tau_{\mathbf{2}}^* \qquad \mathbf{\Delta} \vdash \mathbf{S}^* : \tau_{\mathbf{2}}^* \to \tau_{\mathbf{3}}^*}{\mathbf{\Delta} \vdash \mathbf{F_n}\mathbf{S}^* : \tau_{\mathbf{1}}^* \to \tau_{\mathbf{3}}^*}$$

Frame stacks frame stacks $\mathbf{\Delta} \vdash \mathbf{S}^* : \rho$, frames $\mathbf{\Delta} \vdash \mathbf{S} : \rho$, and local frames $\mathbf{\Delta} \vdash \mathbf{S}^* : \rho$.

(MInst)
$$\frac{\mathbf{\Sigma} = \langle \mathbf{H}, \mathbf{A} \rangle \quad \text{What about typing functions?}}{\mathbf{\Delta} \vdash \mathbf{\Sigma}}$$

Instance typing: $\vdash \mathbf{\Sigma} : \beta$.

(Stack)
$$\frac{\mathbf{\Delta} \vdash \mathbf{S}^* : \rho}{\mathbf{\Delta} \vdash \mathbf{S}^*}$$

(Config)
$$\frac{\vdash \mathbf{\Sigma} \qquad \mathbf{\Delta} \vdash \mathbf{S}^*}{\mathbf{\Delta} \vdash \langle \mathbf{\Sigma}, \mathbf{S}^* \rangle}$$

Typing for stores $\vdash \mathbf{\Sigma}$, Stacks and configurations $\mathbf{\Delta} \vdash \mathbf{\Omega}$

3.3 Wasm Dynamic Semantics

- Semantics for a basic block-based allocator $(\mathbf{H}, \mathbf{A}) \overset{\alpha}{\rightsquigarrow} (\mathbf{H}, \mathbf{A})$.
- Intra-function semantics $\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \mathbf{F} \rangle \xrightarrow{\gamma} \langle \mathbf{\Sigma}, \mathbf{F} \rangle$.
- Top-level semantics $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \xrightarrow{\alpha} \mathbf{\Omega}$.
- Trace semantics $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \xRightarrow{\overline{\alpha}} \mathbf{\Omega}$.

(Nop)

$$\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{nop} : \mathbf{i}^*, \mathbf{v}^* \rangle \longrightarrow \langle \mathbf{\Sigma}, \theta, \mathbf{i}^*, \mathbf{v}^* \rangle$$

(Const)

$$\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \boldsymbol{\tau}\mathbf{.const\ c} : \mathbf{i}^*, \mathbf{v}^* \rangle \longrightarrow \langle \mathbf{\Sigma}, \theta, \mathbf{i}^*, \mathbf{c} : \mathbf{v}^* \rangle$$

(Bop)

$$\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \boldsymbol{\tau}.\otimes : \mathbf{i}^*, \mathbf{v1} : \mathbf{v2} : \mathbf{v}^* \rangle \longrightarrow \langle \mathbf{\Sigma}, \theta, \mathbf{i}^*, [\![\mathbf{v1} \oplus \mathbf{v2}]\!] : \mathbf{v}^* \rangle$$

(Get)

$$\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{get\ x} : \mathbf{i}^*, \mathbf{v}^* \rangle \longrightarrow \langle \mathbf{\Sigma}, \theta, \mathbf{i}^*, \theta(\mathbf{x}) : \mathbf{v}^* \rangle$$

(Set)

$$\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{set\ x} : \mathbf{i}^*, \mathbf{v} : \mathbf{v}^* \rangle \longrightarrow \langle \mathbf{\Sigma}, \theta[\mathbf{x} \mapsto \mathbf{v}], \mathbf{i}^*, \mathbf{v}^* \rangle$$

(Load)

$$\frac{\mathbf{H} = \mathbf{\Sigma}.\mathbf{H} \qquad \mathbf{a} + |\tau| < |\mathbf{H}| \qquad \mathbf{H}[\mathbf{a} : \mathbf{a} + |\tau|] = \mathbf{b}^{|\tau|} \qquad \mathbf{v} = \tau\mathbf{.unpack}(\mathbf{b}^{|\tau|})}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \boldsymbol{\tau}\mathbf{.loadn} : \mathbf{i}^*, \mathbf{a} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{read}_\tau(\mathbf{a})} \langle \mathbf{\Sigma}, \theta, \mathbf{i}^*, \mathbf{v} : \mathbf{v}^* \rangle}$$

(Store)

$$\frac{\mathbf{H} = \mathbf{\Sigma}.\mathbf{H} \qquad \mathbf{a} + |\tau| < |\mathbf{H}| \qquad \mathbf{b}^{|\tau|} = \tau\mathbf{.pack}(\mathbf{v}) \qquad \mathbf{H}' = \mathbf{H}[\mathbf{a} + \mathbf{j} \mapsto \mathbf{b_j} \mid \mathbf{j} \in \mathbf{0}..|\tau| - \mathbf{1}] \qquad \mathbf{\Sigma}' = \mathbf{\Sigma}_{\psi.\mathbf{H}}[(\pi.\mathbf{m}, \mathbf{n}) \mapsto \mathbf{H}']}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \boldsymbol{\tau}\mathbf{.store\ n} : \mathbf{i}^*, \mathbf{v} : \mathbf{a} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{write}_\tau(\mathbf{a})} \langle \mathbf{\Sigma}', \theta, \mathbf{i}^*, \mathbf{v}^* \rangle}$$

Wasm Intra-function semantics: $\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \mathbf{F} \rangle \xrightarrow{\gamma} \langle \mathbf{\Sigma}, \mathbf{F} \rangle$.

(If-True)

$$\frac{\mathbf{c} = \mathbf{0}}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{branch\ i}_1^*\ \mathbf{i}_2^*\ : \mathbf{i}^*, \mathbf{c} : \mathbf{v}^* \rangle \longrightarrow \langle \mathbf{\Sigma}, \theta, \mathbf{i}_1^* \mathbf{i}^*, \mathbf{v}^* \rangle}$$

(If-False)

$$\frac{\mathbf{c} \neq \mathbf{0}}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{branch\ i}_1^*\ \mathbf{i}_2^*\ : \mathbf{i}^*, \mathbf{c} : \mathbf{v}^* \rangle \longrightarrow \langle \mathbf{\Sigma}, \theta, \mathbf{i}_2^* \mathbf{i}^*, \mathbf{v}^* \rangle}$$

(Trap)

$$\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{trap} : \mathbf{i}^*, \mathbf{v}^* \rangle \xrightarrow{\mathbf{trap}} \langle \mathbf{\Sigma}, \theta, \epsilon, \epsilon \rangle$$

(Load-Trap)

$$\frac{\mathbf{H} = \mathbf{\Sigma.H} \qquad \mathbf{a} + |\tau| \geq |\mathbf{H}|}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \boldsymbol{\tau}\mathbf{.loadn}\ \mathbf{i}^*, \mathbf{a} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{trap}} \langle \mathbf{\Sigma}, \theta, \epsilon, \epsilon \rangle}$$

(Store-Trap)

$$\frac{\mathbf{H} = \mathbf{\Sigma.H} \qquad \mathbf{a} + |\tau| \geq |\mathbf{H}|}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \boldsymbol{\tau}\mathbf{.store}\ \mathbf{n}\ \mathbf{i}^*, \mathbf{v} : \mathbf{a} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{trap}} \langle \mathbf{\Sigma}, \theta, \epsilon, \epsilon \rangle}$$

Wasm Intra-function semantics (cont).

(Call)

$$\frac{\mathbf{\Sigma}_\phi(\mathbf{n}) = \{\mathbf{var}\ \tau^{\mathbf{j}}, \mathbf{body}\ \mathbf{i}_1^*\} : \rho \qquad \rho = \tau^{\mathbf{k}} \to \tau^* \qquad \mathbf{v}_1^* = \mathbf{v}^{\mathbf{k}}\mathbf{0}^{\mathbf{j}} \qquad \pi_1 = \mathbf{n}}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{call}\ \mathbf{n} : \mathbf{i}^*, \mathbf{v}^{\mathbf{k}}\mathbf{v}^* \rangle \xrightarrow{\mathbf{call}(\pi_1, \mathbf{i}_1^*, \mathbf{v}_1^*)} \langle \mathbf{\Sigma}, \theta, \mathbf{i}^*, \mathbf{v}^* \rangle}$$

(Return-Explicit)

$$\frac{\mathbf{\Sigma}_\phi(\pi) = \{\_\} : \tau^* \to \tau^{\mathbf{k}}}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{return} : \mathbf{i}^*, \mathbf{v}^{\mathbf{k}}\mathbf{v}^* \rangle \xrightarrow{\mathbf{return}(\mathbf{v}^{\mathbf{k}})} \langle \mathbf{\Sigma}, \theta, \mathbf{i}^*, \mathbf{v}^* \rangle}$$

(Return-Implicit)

$$\frac{\mathbf{\Sigma}_\phi(\pi) = \{\_\} : \tau^* \to \tau^{\mathbf{k}}}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \epsilon, \mathbf{v}^{\mathbf{k}}\mathbf{v}^* \rangle \xrightarrow{\mathbf{return}(\mathbf{v}^{\mathbf{k}})} \langle \mathbf{\Sigma}, \theta, \epsilon, \mathbf{v}^* \rangle}$$

Wasm Function call and return (intra-function semantics).

(Ctx)

$$\frac{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \mathbf{F} \rangle \xrightarrow{\gamma} \langle \mathbf{\Sigma}', \mathbf{F}' \rangle \qquad \gamma \notin \{\mathbf{trap}, \mathbf{call}(\_, \_, \_), \mathbf{return}(\_)\}}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \mathbf{F} : \mathbf{S}^* \rangle \xrightarrow{\gamma} \langle \mathbf{\Sigma}', \mathbf{F}' : \mathbf{S}^* \rangle}$$

(Ctx-Call)

$$\frac{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \mathbf{F} \rangle \xrightarrow{\mathbf{call}(\pi_2, \mathbf{i}^*, \mathbf{v}^*)} \langle \mathbf{\Sigma}', \mathbf{F}' \rangle \qquad \theta = [\ \mathbf{j} \mapsto \mathbf{v_j} \mid \mathbf{v_j} \in \mathbf{v}^*\ ] \qquad \mathbf{F}'' = \langle \theta, \mathbf{i}^*, \epsilon \rangle}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \mathbf{F} : \mathbf{S}^* \rangle \longrightarrow \langle \mathbf{\Sigma}', \mathbf{F}'' : \mathbf{F}' : \mathbf{S}^* \rangle}$$

(Ctx-Return)

$$\frac{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \mathbf{F} \rangle \xrightarrow{\mathbf{return}(\mathbf{v}^{\mathbf{k}})} \langle \mathbf{\Sigma}', \mathbf{F}' \rangle \qquad \mathbf{F}''_{\mathbf{b}} = \langle \theta, \mathbf{i}^*, \mathbf{v}^* \rangle \qquad \mathbf{F}''_{\mathbf{a}} = \langle \theta, \mathbf{i}^*, \mathbf{v}^{\mathbf{k}}\mathbf{v}^* \rangle}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \mathbf{F}' : \mathbf{F}''_{\mathbf{b}} : \mathbf{S}^* \rangle \longrightarrow \langle \mathbf{\Sigma}', \mathbf{F}''_{\mathbf{a}} : \mathbf{S}^* \rangle}$$

(Exit)

$$\frac{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \mathbf{F} \rangle \xrightarrow{\mathbf{return}(\epsilon)} \langle \mathbf{\Sigma}', \mathbf{F}' \rangle}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \mathbf{F_n} : [] \rangle \longrightarrow \langle \mathbf{\Sigma}', [] \rangle}$$

(Trap)

$$\frac{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \mathbf{F} \rangle \xrightarrow{\mathbf{trap}} \langle \mathbf{\Sigma}, \mathbf{F}' \rangle}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \mathbf{F_n} : \mathbf{S}^* \rangle \xrightarrow{\mathbf{trap}} \langle \mathbf{\Sigma}, [] \rangle}$$

Wasm Inter-function Semantics: $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \xrightarrow{\alpha} \mathbf{\Omega}$

(Base)

$$\frac{}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \mathbf{S}^* \rangle \stackrel{\epsilon}{\Longrightarrow} \langle \mathbf{\Sigma}, \mathbf{S}^* \rangle}$$

(Step)

$$\frac{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \mathbf{S}^* \rangle \stackrel{\alpha}{\longrightarrow} \langle \mathbf{\Sigma}', \mathbf{S}'^* \rangle \qquad \mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}', \mathbf{S}'^* \rangle \stackrel{\overline{\alpha}}{\Longrightarrow} \langle \mathbf{\Sigma}'', \mathbf{S}''^* \rangle}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \mathbf{S}^* \rangle \stackrel{\alpha\overline{\alpha}}{\Longrightarrow} \langle \mathbf{\Sigma}'', \mathbf{S}''^* \rangle}$$

Wasm Trace Semantics. $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \stackrel{\overline{\alpha}}{\Longrightarrow} \mathbf{\Omega}$

3.4 MSWasm Extensions

We now add MSWasm specific extensions to the Wasm language described above.

$$
\begin{aligned}
\text{Value Types } \tau &::= \cdots \mid \mathbf{handle} \\
\text{Values } \mathbf{v} &::= \cdots \mid \mathbf{h} \\
\text{Handles } \mathbf{h} &::= \langle \mathbf{base\ n}, \mathbf{offset\ n}, \mathbf{bound\ n}, \mathbf{valid\ b}, \mathbf{id\ n_{id}} \rangle \\
\text{Instructions } \mathbf{i} &::= \cdots \mid \boldsymbol{\tau}\mathbf{.segload} \mid \boldsymbol{\tau}\mathbf{.segstore} \mid \mathbf{slice} \\
&\quad \mid \mathbf{segalloc} \mid \mathbf{segfree} \mid \mathbf{handle.add} \\
\text{Heap cell tag } \mathbf{t} &::= \mathbf{D} \mid \mathbf{H} \\
\text{Tagged Heaps } \mathbf{T} &::= (\mathbf{b}, \mathbf{t})^* \\
\text{Module Defs. } \mathbf{M} &::= \{\mathbf{heaps\ u32}^*, \mathbf{segments\ u32}^*, \mathbf{funcs\ f}^*, \mathbf{export\ E}\} \\
\text{Module Instances/Stores } \mathbf{I}, \boldsymbol{\Sigma} &::= (\mathbf{heap\ H}, \mathbf{segment\ T}, \mathbf{alloc\ A}) \\
\text{Local Events } \gamma &::= \cdots \mid \mathbf{salloc(h)} \mid \mathbf{sfree(h)} \mid \mathbf{read}_{\boldsymbol{\tau}}\,\mathbf{(h)} \mid \mathbf{write}_{\boldsymbol{\tau}}\,\mathbf{(h)} \\
\text{Events } \alpha &::= \cdots \mid \mathbf{salloc(h)} \mid \mathbf{sfree(h)} \mid \mathbf{read}_{\boldsymbol{\tau}}\,\mathbf{(h)} \mid \mathbf{write}_{\boldsymbol{\tau}}\,\mathbf{(h)}
\end{aligned}
$$

$$
\begin{aligned}
\text{Slots } \mathbf{s} &::= (\mathbf{n}, \mathbf{n}, \mathbf{n_{id}}) \\
\text{Allocator States } \mathbf{A} &::= \mathbf{s}^*, \mathbf{s}^*, \mathbf{Y} \\
\text{Allocator Histories } \mathbf{Y} &::= \overline{\alpha}
\end{aligned}
$$

$$\frac{}{\boldsymbol{\Gamma} \vdash \boldsymbol{\tau}\mathbf{.segload} : [\mathbf{handle}] \to [\tau]}\ \text{(Load-Handle)}$$

$$\frac{}{\boldsymbol{\Gamma} \vdash \boldsymbol{\tau}\mathbf{.segstore} : [\mathbf{handle}, \tau] \to []}\ \text{(Store-Handle)}$$

$$\frac{}{\boldsymbol{\Gamma} \vdash \mathbf{slice} : [\mathbf{handle}, \mathbf{i32}, \mathbf{i32}] \to [\mathbf{handle}]}\ \text{(Slice)}$$

$$\frac{}{\boldsymbol{\Gamma} \vdash \mathbf{segalloc} : [\mathbf{i32}] \to [\mathbf{handle}]}\ \text{(Alloc-Handle)}$$

$$\frac{}{\boldsymbol{\Gamma} \vdash \mathbf{handle.add} : [\mathbf{handle}, \mathbf{i32}] \to [\mathbf{handle}]}\ \text{(Add)}$$

$$\frac{}{\boldsymbol{\Gamma} \vdash \mathbf{segfree} : [\mathbf{handle}] \to []}\ \text{(Free)}$$

Typing for MSWasm Extensions

$$\mathbf{MS}(\mathbf{v}, \tau) = \mathbf{o} \geq \mathbf{0} \wedge \mathbf{o} + |\tau| < \mathbf{n_2} \qquad \text{where } \mathbf{v} = \langle \mathbf{n_1}, \mathbf{o}, \mathbf{n_2}, \_ \rangle$$

Memory safety and SFI checks.

(H-Load)

$$\frac{\begin{array}{c}\tau \neq \mathbf{handle} \\ \mathbf{v_1} = \langle \mathbf{n_1}, \mathbf{o}, \mathbf{n_2}, \mathbf{1}, \mathbf{n_{id}} \rangle \quad (\mathbf{H}, \mathbf{T}, \mathbf{A}, \_) = \mathbf{\Sigma} \quad \mathbf{MS}(\mathbf{v_1}, \tau) \quad \mathbf{n_{id}} \in \mathbf{A.allocated} \\ \mathbf{a} = \mathbf{n_1} + \mathbf{o} \quad (\mathbf{b}^*, \mathbf{t}^*) = [\ \mathbf{T}[\mathbf{a}+\mathbf{j}] \mid \mathbf{j} \in \{\mathbf{0}..|\tau|-\mathbf{1}\}\ ] \\ \mathbf{v_2} = \tau.\mathbf{unpack}(\mathbf{b}^*)\end{array}}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \boldsymbol{\tau}.\mathbf{segload} : \mathbf{i}^*, \mathbf{v_1} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{read}_\tau(\mathbf{v_1})} \langle \mathbf{\Sigma}, \theta, \mathbf{i}^*, \mathbf{v_2} : \mathbf{v}^* \rangle}$$

(H-Load-Handle)

$$\frac{\begin{array}{c}\tau = \mathbf{handle} \\ \mathbf{v_1} = \langle \mathbf{n_1}, \mathbf{o}, \mathbf{n_2}, \mathbf{1}, \mathbf{n_{id}} \rangle \quad (\mathbf{H}, \mathbf{T}, \mathbf{A}, \_) = \mathbf{\Sigma} \quad \mathbf{MS}(\mathbf{v_1}, \tau) \quad \mathbf{n_{id}} \in \mathbf{A.allocated} \\ \mathbf{a} = (\mathbf{n_1} + \mathbf{o})\%|\mathbf{handle}| \quad (\mathbf{b}^*, \mathbf{t}^*) = [\ \mathbf{T}[\mathbf{a}+\mathbf{j}] \mid \mathbf{j} \in \{\mathbf{0}..|\tau|-\mathbf{1}\}\ ] \\ \mathbf{c} = \bigwedge_{\mathbf{t} \in \mathbf{t}^*} (\mathbf{t} = \square) \quad \mathbf{v_2} = \mathbf{handle.unpack}(\mathbf{b}^*) = \langle \mathbf{n_1'}, \mathbf{a'}, \mathbf{n_2'}, \mathbf{c'}, \mathbf{n_{id}} \rangle \\ \mathbf{v_2'} = \langle \mathbf{n_1'}, \mathbf{a'}, \mathbf{n_2'}, \mathbf{c} \wedge \mathbf{c'}, \mathbf{n_{id}} \rangle\end{array}}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \boldsymbol{\tau}.\mathbf{segload} : \mathbf{i}^*, \mathbf{v_1} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{read}_\tau(\mathbf{v_1})} \langle \mathbf{\Sigma}, \theta, \mathbf{i}^*, \mathbf{v_2'} : \mathbf{v}^* \rangle}$$

(H-Store)

$$\frac{\begin{array}{c}\tau \neq \mathbf{handle} \quad \mathbf{v_1} = \langle \mathbf{n_1}, \mathbf{o}, \mathbf{n_2}, \mathbf{1}, \mathbf{n_{id}} \rangle \\ (\mathbf{H}, \mathbf{T}, \mathbf{A}, \_) = \mathbf{\Sigma} \quad \mathbf{MS}(\mathbf{v_1}, \tau) \quad \mathbf{n_{id}} \in \mathbf{A.allocated} \\ \mathbf{b}^* = \tau.\mathbf{pack}(\mathbf{v_2}) \quad \mathbf{t} = \bigcirc \quad \mathbf{a} = (\mathbf{n_1} + \mathbf{o}) \\ \mathbf{T'} = \mathbf{T}[\mathbf{a}+\mathbf{j} \mapsto (\mathbf{b_j}, \mathbf{t}) \mid \mathbf{j} \in \{\mathbf{0}..|\tau|-\mathbf{1}\}] \quad \mathbf{\Sigma'} = (\mathbf{H}, \mathbf{T'}, \mathbf{A})\end{array}}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \boldsymbol{\tau}.\mathbf{segstore} : \mathbf{i}^*, \mathbf{v_2} : \mathbf{v_1} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{write}_\tau(\mathbf{v1})} \langle \mathbf{\Sigma'}, \theta, \mathbf{i}^*, \mathbf{v}^* \rangle}$$

(H-Store-Handle)

$$\frac{\begin{array}{c}\tau = \mathbf{handle} \quad \mathbf{v_1} = \langle \mathbf{n_1}, \mathbf{o}, \mathbf{n_2}, \mathbf{1}, \mathbf{n_{id}} \rangle \\ (\mathbf{H}, \mathbf{T}, \mathbf{A}, \mathbf{f}^*) = \mathbf{\Sigma} \quad \mathbf{MS}(\mathbf{v_1}, \tau) \quad \mathbf{n_{id}} \in \mathbf{A.allocated} \\ \mathbf{b}^* = \tau.\mathbf{pack}(\mathbf{v_2}) \quad \mathbf{t} = \square \quad \mathbf{a} = (\mathbf{n_1} + \mathbf{o})\%|\mathbf{handle}| \\ \mathbf{T'} = \mathbf{T}[\mathbf{a}+\mathbf{j} \mapsto (\mathbf{b_j}, \mathbf{t}) \mid \mathbf{j} \in \{\mathbf{0}..|\tau|-\mathbf{1}\}] \quad \mathbf{\Sigma'} = (\mathbf{H}, \mathbf{T'}, \mathbf{A}, \mathbf{f}^*)\end{array}}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \boldsymbol{\tau}.\mathbf{segstore} : \mathbf{i}^*, \mathbf{v_2} : \mathbf{v_1} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{write}_\tau(\mathbf{v1})} \langle \mathbf{\Sigma'}, \theta, \mathbf{i}^*, \mathbf{v}^* \rangle}$$

(H-Load-Trap)

$$\frac{\begin{array}{c}\mathbf{v_1} = \langle \mathbf{n_1}, \mathbf{o}, \mathbf{n_2}, \mathbf{c}, \mathbf{n_{id}} \rangle \quad (\mathbf{H}, \mathbf{T}, \mathbf{A}, \_) = \mathbf{\Sigma} \\ \mathbf{c} = \mathbf{false} \vee \neg \mathbf{MS}(\mathbf{v_1}, \tau \vee \mathbf{n_{id}} \notin \mathbf{A.allocated})\end{array}}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \boldsymbol{\tau}.\mathbf{segload} : \mathbf{i}^*, \mathbf{v_1} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{trap}} \langle \mathbf{\Sigma}, \theta, \epsilon, \epsilon \rangle}$$

(H-Store-Trap)

$$\frac{\begin{array}{c}\mathbf{v_1} = \langle \mathbf{n_1}, \mathbf{o}, \mathbf{n_2}, \mathbf{c} \rangle \quad (\mathbf{H}, \mathbf{T}, \mathbf{A}, \_) = \mathbf{\Sigma} \\ \mathbf{c} = \mathbf{false} \vee \neg \mathbf{MS}(\mathbf{v_1}, \tau) \vee \mathbf{n_{id}} \notin \mathbf{A.allocated}\end{array}}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \boldsymbol{\tau}.\mathbf{segstore} : \mathbf{i}^*, \mathbf{v_2} : \mathbf{v_1} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{trap}} \langle \mathbf{\Sigma}, \theta, \epsilon, \epsilon \rangle}$$

(H-Alloc)

$$\frac{\begin{array}{c}(\mathbf{H}, \mathbf{T}, \mathbf{A}) = \mathbf{\Sigma} \quad \langle \mathbf{T}, \mathbf{A} \rangle \xhookrightarrow{\mathbf{salloc}(\mathbf{a},\mathbf{n},\mathbf{n_{id}})} \langle \mathbf{T'}, \mathbf{A'} \rangle \\ \mathbf{v} = \langle \mathbf{a}, \mathbf{0}, \mathbf{n}, \mathbf{1}, \mathbf{n_{id}} \rangle \quad \mathbf{\Sigma'} = (\mathbf{H}, \mathbf{T'}, \mathbf{A'})\end{array}}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{segalloc} : \mathbf{i}^*, \mathbf{n} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{salloc}(\mathbf{v})} \langle \mathbf{\Sigma'}, \theta, \mathbf{i}^*, \mathbf{v} : \mathbf{v}^* \rangle}$$

(H-Free)

$$\frac{(\mathbf{H},\mathbf{T},\mathbf{A}) = \mathbf{\Sigma} \quad \mathbf{v} = \langle \mathbf{a}, \mathbf{0}, \mathbf{n}', \mathbf{1}, \mathbf{n_{id}} \rangle \quad \langle \mathbf{T}, \mathbf{A} \rangle \xhookrightarrow{\mathbf{sfree(a,n_{id})}} \langle \mathbf{T}', \mathbf{A}' \rangle \quad \mathbf{\Sigma}' = (\mathbf{H},\mathbf{T}',\mathbf{A}')}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{segfree} : \mathbf{i}^*, \mathbf{v} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{sfree(v)}} \langle \mathbf{\Sigma}', \theta, \mathbf{i}^*, \mathbf{v}^* \rangle}$$

(H-Free-trap)

$$\frac{\mathbf{v} = \langle \mathbf{a}, \mathbf{n}, \mathbf{n}', \mathbf{0}, \mathbf{n_{id}} \rangle \quad \vee \quad \big(\mathbf{v} = \langle \mathbf{a}, \mathbf{0}, \mathbf{n}', \mathbf{1}, \mathbf{n_{id}} \rangle \wedge \langle \mathbf{T}, \mathbf{A} \rangle \xhookrightarrow{\mathbf{sfree(a,n_{id})}} \langle \mathbf{T}, \mathbf{A} \rangle\big) \quad \vee \quad \big(\mathbf{v} = \langle \mathbf{a}, \mathbf{n}, \mathbf{n}', \mathbf{1}, \mathbf{n_{id}} \rangle \wedge \mathbf{n} \neq \mathbf{0}\big) \vee \mathbf{n_{id}} \notin \mathbf{A}.\mathbf{allocated}}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{segfree} : \mathbf{i}^*, \mathbf{v} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{trap}} \langle \mathbf{\Sigma}, \theta, \epsilon, \epsilon \rangle}$$

(Slice)

$$\frac{\mathbf{v} = \langle \mathbf{n_1}, \mathbf{o}, \mathbf{n_2}, \mathbf{b}, \mathbf{n_{id}} \rangle \quad \mathbf{o_1} \geq \mathbf{0} \quad \mathbf{o_2} \geq \mathbf{0} \quad \mathbf{o_1} < \mathbf{n_2} \quad \mathbf{v}' = \langle \mathbf{n_1} + \mathbf{o_1}, \mathbf{o}, \mathbf{n_2} - \mathbf{o_2}, \mathbf{b}, \mathbf{n_{id}} \rangle}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{slice} : \mathbf{i}^*, \mathbf{o_2} : \mathbf{o_1} : \mathbf{v} : \mathbf{v}^* \rangle \twoheadrightarrow \langle \mathbf{\Sigma}, \theta, \mathbf{i}^*, \mathbf{v}' : \mathbf{v}^* \rangle}$$

(Slice-Trap)

$$\frac{\mathbf{v} = \langle \mathbf{n_1}, \mathbf{o}, \mathbf{n_2}, \mathbf{b}, \mathbf{n_{id}} \rangle \quad \mathbf{o_1} < \mathbf{0} \vee \mathbf{o_1} \geq \mathbf{n_2} \vee \mathbf{o_2} < \mathbf{0} \vee \mathbf{n_{id}} \notin \mathbf{A}.\mathbf{allocated}}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{slice} : \mathbf{i}^*, \mathbf{v} : \mathbf{o_1} : \mathbf{o_2} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{trap}} \langle \mathbf{\Sigma}, \theta, \epsilon, \epsilon \rangle}$$

(Handle-Add)

$$\frac{\mathbf{v} = \langle \mathbf{n_1}, \mathbf{o}, \mathbf{n_2}, \mathbf{b}, \mathbf{n_{id}} \rangle \quad \mathbf{v}' = \langle \mathbf{n_1}, \mathbf{o} + \mathbf{n}, \mathbf{n_2}, \mathbf{b}, \mathbf{n_{id}} \rangle}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{handle.add} : \mathbf{i}^*, \mathbf{n} : \mathbf{v} : \mathbf{v}^* \rangle \twoheadrightarrow \langle \mathbf{\Sigma}, \theta, \mathbf{i}^*, \mathbf{v}' : \mathbf{v}^* \rangle}$$

MSWasm Semantics.

(Seg-Alloc)

$$\frac{\mathbf{s_f^*} = \mathbf{s_1^*} \cdot (\mathbf{a}, \mathbf{n}', \mathbf{n}'_{\mathbf{id}}) \cdot \mathbf{s_2^*} \quad \mathbf{0} < \mathbf{n} \leq \mathbf{n}' \quad \mathbf{a} + \mathbf{n} < |\mathbf{T}| \quad \mathbf{aligned(a)} \quad \mathbf{T}' = \mathbf{T}[\mathbf{a} + \mathbf{j} \mapsto (\mathbf{0}, \mathbf{D}) \mid \mathbf{j} \in \mathbf{0..n-1}] \quad \mathbf{Y}' = \mathbf{Y} \cdot \mathbf{salloc}\,(\mathbf{a}, \mathbf{n}, \mathbf{n_{id}}) \quad \mathbf{s_f'^*} = \mathbf{s_1^*} \cdot (\mathbf{a} + \mathbf{n}, \mathbf{n}' - \mathbf{n}, \mathbf{n}'_{\mathbf{id}}) \cdot \mathbf{s_2^*} \quad \mathbf{fresh(n_{id})} \quad \mathbf{s_A'^*} = (\mathbf{a}, \mathbf{n}, \mathbf{n_{id}}) \cdot \mathbf{s_A^*}}{\langle \mathbf{T}, (\mathbf{s_A^*}, \mathbf{s_f^*}, \mathbf{Y}) \rangle \xhookrightarrow{\mathbf{salloc(a,n,n_{id})}} \langle \mathbf{T}', (\mathbf{s_A'^*}, \mathbf{s_f'^*}, \mathbf{Y}') \rangle}$$

(Seg-Free)

$$\frac{\mathbf{s_A^*} = \mathbf{s_1^*} \cdot (\mathbf{a}, \mathbf{n}, \mathbf{n_{id}}) \cdot \mathbf{s_2^*} \quad \mathbf{s_f'^*} = (\mathbf{a}, \mathbf{n}, \mathbf{n_{id}}) \cdot \mathbf{s_f^*} \quad \mathbf{s_A'^*} = \mathbf{s_1^*} \cdot \mathbf{s_2^*} \quad \mathbf{Y}' = \mathbf{Y} \cdot \mathbf{sfree}\,(\mathbf{a}, \mathbf{n_{id}})}{\langle \mathbf{T}, (\mathbf{s_A^*}, \mathbf{s_f^*}, \mathbf{Y}) \rangle \xhookrightarrow{\mathbf{sfree(a,n_{id})}} \langle \mathbf{T}, (\mathbf{s_A'^*}, \mathbf{s_f'^*}, \mathbf{Y}') \rangle}$$

(Seg-Free-nop)

$$\frac{(\nexists(\mathbf{a}, \_, \_) \in \mathbf{s_A^*}) \vee (\mathbf{sfree}\,(\mathbf{a}, \mathbf{n_{id}}) \in \mathbf{Y})}{\langle \mathbf{T}, (\mathbf{s_A^*}, \mathbf{s_f^*}, \mathbf{Y}) \rangle \xhookrightarrow{\mathbf{sfree(a,n_{id})}} \langle \mathbf{T}, (\mathbf{s_A^*}, \mathbf{s_f^*}, \mathbf{Y}) \rangle}$$

MSWasm Allocator Semantics: $(\mathbf{T}, \mathbf{A}) \overset{\alpha}{\rightsquigarrow} (\mathbf{T}, \mathbf{A})$.

3.5 MSWasm Memory Safety

In this section, we develop the machinery necessary to define the memory safety for MSWasm. Similar to source, the memeory safety of MSWasm is defined in terms of the abstract MS monitor.

Definition 6 (Target Color Assignment)**.** The target color assignment, δ , is a map from source address to the tuple of color ($\mathbb{C}$) and shades ($\mathbb{S}$).

$$\delta : \mathbf{Address} \times \mathbf{Tid} \to \mathbb{N} \times \mathbb{C} \times \mathbb{S}$$

Definition 7 (Trace Agreement ($\alpha =_\delta \alpha$))**.**

$$\frac{}{\epsilon =_\delta \epsilon}\ \text{(Tr-Empty)} \qquad \frac{\alpha =_\delta \alpha \quad \overline{\alpha} =_\delta \overline{\alpha}}{\alpha \cdot \overline{\alpha} =_\delta \alpha \cdot \overline{\alpha}}\ \text{(Tr-Concatenate)}$$

$$\frac{n = \mathtt{sz}(\tau) \quad \delta(\mathbf{b}, \mathbf{n_{id}}) = b^{(c,s)} \quad a = b + \mathbf{o}}{\mathbf{read}_\tau\ (\langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}} \rangle) =_\delta \mathtt{read}(a^{(c,s)}) \cdot \mathtt{read}((a+1)^{(c,s)}) \ldots \mathtt{read}((a+n-1)^{(c,s)})}\ \text{(Tr-Read)}$$

$$\frac{n = \mathtt{sz}(\tau) \quad \delta(\mathbf{b}, \mathbf{n_{id}}) = b^{(c,s)} \quad a = b + \mathbf{o}}{\mathbf{write}_\tau\ (\langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}} \rangle) =_\delta \mathtt{write}(a^{(c,s)}) \cdot \mathtt{write}((a+1)^{(c,s)}) \ldots \mathtt{write}((a+n-1)^{(c,s)})}\ \text{(Tr-Write)}$$

$$\frac{\mathbf{h} = \langle \mathbf{a}, \_, \mathbf{n}, \_, \mathbf{n_{id}} \rangle \quad n = \mathbf{n} \quad \forall i \in \{0 \ldots n-1\}, \delta(\mathbf{a}+i, \mathbf{n_{id}}) = a^{(c,\phi(i))}}{\mathbf{salloc}(\mathbf{h}) =_\delta \mathtt{alloc}(n, a^c, \phi)}\ \text{(Tr-SAlloc)}$$

$$\frac{\mathbf{h} = \langle \mathbf{b}, \_, \mathbf{n}, \_, \mathbf{n_{id}} \rangle \quad \delta(\mathbf{b}, \mathbf{n_{id}}) = a^{(c,\_)}}{\mathbf{sfree}(\mathbf{h}, \mathbf{n_{id}}) =_\delta \mathtt{sfree}\,(a^c)}\ \text{(Tr-Sfree)} \qquad \frac{}{\mathbf{trap} =_\delta \epsilon}\ \text{(Tr-Trap)}$$

Definition 8 (Safe Memory Allocator)**.** A memory allocator is *safe* if the set of allocated addresses are disjoint from the free addresses and all individual slots are non-overlapping.

$$\frac{\mathbf{s_A}^* \cap \mathbf{s_F}^* = \varnothing}{(\mathbf{s_A}^*, \mathbf{s_F}^*, \_)\ \bigstar}\ \text{(Safe Allocator)}$$

Definition 9 (Segment Agreement)**.**

$$\frac{\begin{array}{c}\forall (\mathbf{b}, \ell, \mathbf{n_{id}}) \in \mathbf{s_A}^*.\ \text{let } a^{(c,\_)} = \delta(\mathbf{b}, \mathbf{n_{id}}) \text{ in } \forall i \in 0..\ell - 1.\, T(a+i) = A(c, \_) \\ \forall (\mathbf{b}, \ell, \mathbf{n_{id}}) \in \mathbf{s_F}^*.\ \text{let } a^{(c,\_)} = \delta(\mathbf{b}, \mathbf{n_{id}}) \text{ in } \forall i \in 0..\ell - 1.\, T(a+i) = F(c, \_) \\ \mathbf{Y} =_\delta T.Y\end{array}}{T =_\delta \mathbf{s_A}^*, \mathbf{s_F}^*, \mathbf{Y}}\ \text{(Segment Agreement)}$$

Lemma 6 (Monotonicity of Address Map)**.** Let $T =_\delta \mathbf{A}$. If $\delta' \supseteq \delta$ then T $=_{\delta'} \mathbf{A}$.

Proof. Trivial. □

Definition 10 (Top-level Memory Safety for C-like)**.** $\mathtt{MS}(\overline{\alpha}) \stackrel{\text{def}}{=} \exists \overline{\alpha}, \delta,$

1. $\overline{\alpha} =_\delta \overline{\alpha}$ and
2. $\mathtt{MS}\,(\overline{\alpha})$

3.6 MSWasm Language Properties

In this section, we prove a few important properties of well-typed MSWasm programs. Specifically, we prove the following:

1. Type preservation.
2. Well-typedness implies robust memory safety (Def 16).

We first define what it means for a configuration to be valid and then prove that validity and well-typedness of configurations is preserved during the step.

$$\frac{}{\mathbf{A} \vdash \mathbf{isValid}\ \mathbf{c}}\ \text{(const)}$$

$$\frac{(\mathbf{n_b}, \mathbf{n_l}, \mathbf{n_{id}}) \in \mathbf{A}}{\mathbf{A} \vdash \mathbf{isValid}\ \langle \mathbf{base}\ \mathbf{n_b}, \mathbf{offset}\ \mathbf{n_o}, \mathbf{bound}\ \mathbf{n_l}, \mathbf{valid}\ \mathbf{1}, \mathbf{n_{id}} \rangle}\ \text{(cor-handle)}$$

$$\frac{}{\mathbf{A} \vdash \mathbf{isValid}\ \langle \mathbf{base}\ \mathbf{n_b}, \mathbf{offset}\ \mathbf{n_o}, \mathbf{bound}\ \mathbf{n_l}, \mathbf{valid}\ \mathbf{0}, \mathbf{n_{id}} \rangle}\ \text{(incor-handle)}$$

$$\frac{\mathbf{A} \vdash \mathbf{isValid}\ \mathbf{v}^* \qquad \mathbf{A} \vdash \mathbf{isValid}\ \mathbf{S}^*}{\mathbf{A} \vdash \mathbf{isValid}\ (\_, \_, \mathbf{v}^*)_\mathbf{n} : \mathbf{S}^*}\ \text{(cons-stack-frame)}$$

$$\frac{\mathbf{A} \vdash \mathbf{isValid}\ \mathbf{v}' \qquad \mathbf{A} \vdash \mathbf{isValid}\ (\_, \_, \mathbf{v}^*)_\mathbf{n} : \mathbf{S}^*}{\mathbf{A} \vdash \mathbf{isValid}\ (\_, \_, \mathbf{v}' : \mathbf{v}^*)_\mathbf{n} : \mathbf{S}^*}\ \text{(cons-data-stack)}$$

$$\frac{}{\mathbf{A} \vdash \mathbf{isValid}\ (\_, \_, \varnothing)_\mathbf{n} : \mathbf{S}^*}\ \text{(empty-data-stack)}$$

$$\frac{\mathbf{t^1} = \square \wedge \mathbf{t^2} = \square \wedge \cdots \wedge \mathbf{t}^{|\mathbf{handle}|} \implies \mathbf{h} = \mathbf{handle.unpack}(\mathbf{b^1} \ldots \mathbf{b}^{|\mathbf{handle}|}) \implies \mathbf{A} \vdash \mathbf{isValid}\ \mathbf{h} \qquad \mathbf{A} \vdash \mathbf{isValid}\ \mathbf{H}}{\mathbf{A} \vdash \mathbf{isValid}\ (\mathbf{b^1}, \mathbf{t^1}) \ldots (\mathbf{b}^{|\mathbf{handle}|}, \mathbf{t}^{|\mathbf{handle}|}) : \mathbf{H}}\ \text{(segment)}$$

$$\frac{\boldsymbol{\Sigma} = \langle \mathbf{H}, \mathbf{T}, \mathbf{A} \rangle \qquad \mathbf{A} \vdash \mathbf{isValid}\ \mathbf{T} \qquad \mathbf{A} \vdash \mathbf{isValid}\ \mathbf{S}^*}{\vdash \mathbf{isValid}\ \langle \boldsymbol{\Sigma}, \mathbf{S}^* \rangle}\ \text{(conf)}$$

Valid Stack and Segment Configurations

3.6.1 Type Preservation

Lemma 7 (Type and Validity Preservation for Configurations-Trace Semantics)**.** If

1. $\mathbf{\Delta} \vdash \mathbf{\Omega}$
2. $\vdash \mathbf{isValid}\ \mathbf{\Omega}$
3. $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \overset{\overline{\alpha}}{\Longrightarrow} \mathbf{\Omega}'$

then

1. $\mathbf{\Delta} \vdash \mathbf{\Omega}'$
2. $\vdash \mathbf{isValid}\ \mathbf{\Omega}'$

Proof. Proof is by induction on $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \overset{\overline{\alpha}}{\Longrightarrow} \mathbf{\Omega}'$. For the case (Step), we have:

$$\mathbf{\Phi}^* \vdash \mathbf{\Omega} \xrightarrow{\alpha} \mathbf{\Omega}' \tag{1}$$

$$\mathbf{\Phi}^* \vdash \mathbf{\Omega}' \overset{\overline{\alpha}}{\Longrightarrow} \mathbf{\Omega}'' \tag{2}$$

Applying Lemma 8 (Type and Validity Preservation for Configurations-Inter) on (1) we have

$$\mathbf{\Delta} \vdash \mathbf{\Omega}' \tag{3}$$

$$\vdash \mathbf{isValid}\ \mathbf{\Omega}' \tag{4}$$

This gives us necessary premises to apply I.H to (2), and we have the required proof.

□

Lemma 8 (Type and Validity Preservation for Configurations-Inter)**.** If

1. $\mathbf{\Delta} \vdash \mathbf{\Omega}$
2. $\vdash \mathbf{isValid}\ \mathbf{\Omega}$ and
3. $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \xrightarrow{\alpha} \mathbf{\Omega}'$

then

1. $\mathbf{\Delta} \vdash \mathbf{\Omega}'$
2. $\vdash \mathbf{isValid}\ \mathbf{\Omega}'$

Proof. Proof is by induction on $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \xrightarrow{\alpha} \mathbf{\Omega}'$. For the case (Ctx), invoke Lemma 9 (Type and Validity Preservation for Configurations-Intra). The other cases (Ctx-Call), (Ctx-Return), (Exit) and (Trap) are relatively straightforward. □

Lemma 9 (Type and Validity Preservation for Configurations-Intra)**.** If

1. $\mathbf{\Delta} \vdash \mathbf{\Omega}$
2. $\vdash \mathbf{isValid}\ \mathbf{\Omega}$ and
3. $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \xrightarrow{\alpha}\!\!\!\!\rightarrow_{\mathbf{n}} \mathbf{\Omega}'$

then

1. $\mathbf{\Delta} \vdash \mathbf{\Omega}'$
2. $\vdash \mathbf{isValid}\ \mathbf{\Omega}'$

Proof. Proof is by induction on $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \xrightarrow{\alpha}\!\!\!\!\rightarrow_{\mathbf{n}} \mathbf{\Omega}'$. The proof for type preservation is straightforward. Below we only describe the interesting cases for proving the validity of the configuration.

Case (H-Alloc): Let $\mathbf{\Sigma} = \langle \mathbf{H}, \mathbf{T}, \mathbf{A} \rangle$ and $\mathbf{\Sigma} = \langle \mathbf{H}, \mathbf{T}', \mathbf{A}' \rangle$.

Given $\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, (\theta, \mathbf{segalloc} : \mathbf{i}^*, \mathbf{n}' : \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^* \rangle \xrightarrow{\mathbf{alloc(v)}}\!\!\!\!\rightarrow_{\mathbf{n}} \langle \mathbf{\Sigma}', (\theta, \mathbf{i}^*, \mathbf{v}' : \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^* \rangle$.
We have to prove

$$\mathbf{A}' \vdash \mathbf{isValid}\ (\theta, \mathbf{i}^*, \mathbf{v}' : \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^*$$

where $\mathbf{v} = \langle \mathbf{a}, \mathbf{0}, \mathbf{a} + \mathbf{n}'', \mathbf{1}, \mathbf{n_{id}} \rangle$ and $\mathbf{n}'' = \mathbf{n}' * \mathtt{sz}\,(\tau)$.

We are given $\mathbf{A} \vdash \mathbf{isValid}\ (\theta, \mathbf{segalloc}; \mathbf{i}^*, \mathbf{n}' : \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^*$. Inverting (cons-data-stack), we have $\mathbf{A} \vdash \mathbf{isValid}\ (\theta, \mathbf{segalloc}; \mathbf{i}^*, \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^*$. This implies

$$\mathbf{A} \vdash \mathbf{isValid}\ (\theta, \mathbf{i}^*, \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^* \tag{5}$$

Since $(\mathbf{a}, \mathbf{a} + \mathbf{n}'') \in \lfloor \mathbf{A}' \rfloor_1$, from (cor-handle), we have

$$\mathbf{A}' \vdash \mathbf{isValid}\ \mathbf{v}' \tag{6}$$

Combining (5) and (6) yield the required premises for (cons-data-stack).

We still have to prove $\mathbf{A}' \vdash \mathbf{isValid}\ \mathbf{T}'$. From (Seg-Alloc), we have that $\mathbf{T}' = \mathbf{T}[\mathbf{a} + \mathbf{j} \mapsto (\mathbf{0}, \bigcirc) \mid \mathbf{j} \in \mathbf{0}..\mathbf{n}'' - \mathbf{1}]$. That is, no new handle has been allocated in the segment. Since $\lfloor \mathbf{A} \rfloor_1 \subseteq \lfloor \mathbf{A}' \rfloor_1$, $\mathbf{A} \vdash \mathbf{isValid}\ \mathbf{T}$ implies $\mathbf{A}' \vdash \mathbf{isValid}\ \mathbf{T}'$.

Case (H-Load) and (H-Load-Handle): Let $\mathbf{\Sigma} = \langle \mathbf{H}, \mathbf{T}, \mathbf{A} \rangle$ and $\mathbf{\Sigma} = \langle \mathbf{H}, \mathbf{T}', \mathbf{A} \rangle$.

Given $\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, (\theta, \tau.\mathbf{segload} : \mathbf{i}^*, \mathbf{v_1} : \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^* \rangle \xrightarrow{\mathbf{read}_\tau(\mathbf{v1},\mathbf{v2})}\!\!\!\!\rightarrow_{\mathbf{n}} \langle \mathbf{\Sigma}', (\theta, \mathbf{i}^*, \mathbf{v_2} : \mathbf{v}^*) \rangle_{\mathbf{n}} : \mathbf{S}_0^*$.
We have to prove $\mathbf{A} \vdash \mathbf{isValid}\ (\theta, \mathbf{i}^*, \mathbf{v_2} : \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^*$ where

$\mathbf{v_2} = \tau.\mathbf{unpack}([\ \mathbf{T}[\mathbf{a} + \mathbf{j}] \mid \mathbf{j} \in \{\mathbf{0}..\mathtt{sz}\,(\tau) - \mathbf{1}\}\ ])$. We are given $\mathbf{A} \vdash \mathbf{isValid}\ (\theta, \tau.\mathbf{segload}; \mathbf{i}^*, \mathbf{v_1} : \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^*$. Inverting (cons-data-stack), we have $\mathbf{A} \vdash \mathbf{isValid}\ (\theta, \tau.\mathbf{segload}; \mathbf{i}^*, \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^*$. This implies

$$\mathbf{A} \vdash \mathbf{isValid}\ (\theta, \mathbf{i}^*, \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^* \tag{7}$$

Let $\tau = \mathbf{handle}$. Since we have $\mathbf{A} \vdash \mathbf{isValid}\ \mathbf{T}$, then $\mathbf{A} \vdash \mathbf{isValid}\ \mathbf{v_2}$. From (7) and (cons-data-stack), we thus have $\mathbf{A} \vdash \mathbf{isValid}\ (\theta, \mathbf{i}^*, \mathbf{v_2} : \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^*$. Since $\mathbf{T}' = \mathbf{T}$, we also have $\mathbf{A} \vdash \mathbf{isValid}\ \mathbf{T}'$.

Case (H-Load-Trap): Given $\Phi^* \vdash \langle \Sigma, \theta, \tau.\mathbf{segload} : \mathbf{i}^*, \mathbf{v_1} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{trap}} \twoheadrightarrow \langle \Sigma, \theta, \epsilon, \epsilon \rangle$. Since stack is emptied, from rule (empty-data-stack), we have that the final stack is valid. Since Σ is not changed, we have $\vdash \mathbf{isValid}\ \Omega'$ where $\Omega' = \langle \Sigma, \theta, \epsilon, \epsilon \rangle$.

Case (H-Store) and (H-Store-Handle): Almost similar to (H-Load) and (H-Load-Handle) case except that if a handle is being written, i.e., $\tau = \mathbf{handle}$, we have that the handle is valid (because initial stack is valid).

Case (H-Store-Trap): Similar to the above (H-Load-Trap) case.

Case (H-Free) Given the reduction, we have to prove

$$\mathbf{A}' \vdash \mathbf{isValid}\ (\theta, \mathbf{i}^*, \mathbf{v}' : \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^*$$

The only change is in $\mathbf{A}$, but the validity of handles still follows from Rule (cor-handle).

Case (H-Free-trap): Trivial, since this is a pop.

□

3.6.2 Memory Safety Properties

We develop the machinery necessary to define the attacker model and what it means for a MSWasm module to be robustly memory safe.

Definition 11 (Module instantiation)**.** Let $\mathbf{M} = \{\mathbf{heaps}\ \mathbf{h_s}, \mathbf{segments}\ \mathbf{t_s}, \mathbf{funcs}\ \mathbf{f}^*, \mathbf{import}\ \mathbf{Imp}\}$. Define $\mathtt{instantiate}(\mathbf{M}) \stackrel{\text{def}}{=} \langle \mathbf{H_0}, \mathbf{T_0}, \mathbf{A_0} \rangle$ such that $\mathbf{H_0}$ and $\mathbf{T_0}$ are heap and segment memory of size $\mathbf{h_s}$ and $\mathbf{t_s}$, respectively, and are initialized to $\mathbf{0}$ whereas allocator $\mathbf{A_0}$ is safe memory allocator with no used slots, that is, $(\epsilon, (0, \mathbf{t_s}))$.

Definition 12 (Initial Config)**.** $\Omega_0\ (\mathbf{M}, \mathbf{n_{entry}}) = \langle \mathtt{instantiate}(\mathbf{M}), (\theta_{\mathbf{entry}}, \mathbf{i}^*, \epsilon)_{\mathbf{n_{entry}}} \rangle$ where

$$\mathbf{n_{entry}} = \{\mathbf{var}\ \tau^{\mathbf{v}}, \mathbf{body}\ \mathbf{i}^*\}$$
$$\theta_{\mathbf{entry}} = \{\mathbf{j} \mapsto \mathbf{0} \mid \mathbf{j} \in \{\mathbf{1} \ldots \mathbf{v}\}\}$$

Definition 13 (Attacker)**.**

$$\mathbf{M} \vdash \mathbf{A} : \mathbf{attacker} \stackrel{\text{def}}{=} \ \vdash \mathbf{A} \text{ and } \mathbf{A} = \{\mathbf{0}, \mathbf{0}, [\mathbf{main}; \mathbf{f_a^*}], \mathbf{I_a}\}$$
$$\text{where } \mathbf{M} = \{\_, \_, \mathbf{f_m^*}, \mathbf{I_m}\} \text{ and } \mathtt{dom}\,(\mathbf{I_a}) = \mathtt{dom}\,(\mathbf{f_m^*}) \text{ and } \mathtt{dom}\,(\mathbf{I_m}) = \mathtt{dom}\,(\mathbf{main}; \mathbf{f_a^*})$$

A composition (or module join) is union of all heaps, segments and functions.

Definition 14 (Linking)**.**

$$\mathbf{M^1} \circ \mathbf{M^2} \stackrel{\text{def}}{=} \{\mathbf{h^{1*}h^{2*}}, \mathbf{s^{1*}s^{2*}}, [\mathbf{f_1^1}; \cdots; \mathbf{f_n^1}; \mathbf{f_1^2}; \cdots \mathbf{f_m^2}], \varnothing\}$$
$$\text{where } \mathbf{M^1} \stackrel{\text{def}}{=} \{\mathbf{h^{1*}}, \mathbf{s^{1*}}, [\mathbf{f_1^1}; \cdots; \mathbf{f_n^1}], [\mathbf{f_1^2}; \cdots \mathbf{f_m^2}]\} \text{ and } \mathbf{M^2} \stackrel{\text{def}}{=} \{\mathbf{h^{2*}}, \mathbf{s^{2*}}, [\mathbf{f_1^2}; \cdots; \mathbf{f_m^2}], [\mathbf{f_1^1}; \cdots \mathbf{f_n^1}]\}$$

Definition 15 (Module is memory-safe)**.** Module $\mathbf{M}$ is memory-safe, denoted $\vdash \mathtt{MS}(\mathbf{M})$, if $\mathbf{\Phi}^* \vdash \mathbf{\Omega_0}\,(\mathbf{M}, \mathbf{n_{main}}) \stackrel{\overline{\alpha}}{\Longrightarrow} \_$. then $\mathtt{MS}(\overline{\alpha})$.

Definition 16 (Module is memory-safe robustly)**.**

$$\vdash \mathbf{RMS}\,(\mathbf{M}) \stackrel{\mathsf{def}}{=} \forall \mathbf{A}, \mathbf{M} \vdash \mathbf{A} : \mathbf{attacker} \implies \vdash \mathtt{MS}(\mathbf{M} \circ \mathbf{A})$$

Lemma 10 (Memory Safety Preservation-Intra)**.** If

1. $\mathbf{\Delta} \vdash \mathbf{\Omega}$ and
2. $\vdash \mathbf{isValid}\ \mathbf{\Omega}$ and
3. $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \xrightarrow{\alpha}\!\!\!\!\rightarrow_{\mathbf{n}} \mathbf{\Omega}'$ and
4. $T =_{\delta} \mathbf{\Omega.A}$

then $\exists \overline{\alpha}, \delta', T'$ such that

- $\delta' \supseteq \delta$ and
- $T' =_{\delta'} \mathbf{\Omega'.A}$ and
- $\alpha =_{\delta'} \overline{\alpha}$ and
- $\vdash T \stackrel{\overline{\alpha}}{\rightsquigarrow} T'$

Proof. Proof is by induction on the single evaluation step. We first prove for interesting cases that contribute to traces.

Case (H-Alloc): Given:

$\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, (\theta, \mathbf{segalloc} : \mathbf{i}^*, \mathbf{n}' : \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^* \rangle \xrightarrow{\mathbf{alloc}(\mathbf{v}')}\!\!\!\!\rightarrow_{\mathbf{n_f}} \langle \mathbf{\Sigma}', (\theta, \mathbf{i}^*, \mathbf{v}' : \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^* \rangle$.
Let $\mathbf{\Omega}' = \langle \mathbf{\Sigma}', \mathbf{S}_0'^* \rangle$ and $\mathbf{\Sigma}' = \langle \mathbf{H}', \mathbf{T}', \mathbf{A}' \rangle$. We have $\alpha = \mathbf{alloc}\,(\mathbf{a}, \mathbf{n}', \mathbf{n_{id}})$.
We are also given that $T =_{\delta} \mathbf{A}$. We first have to prove

$$\exists \delta', \delta' \supseteq \delta \tag{8}$$
$$\exists \overline{\alpha}, \alpha =_{\delta'} \overline{\alpha} \tag{9}$$

Let $\delta' = \delta[\mathbf{a}, \mathbf{n_{id}} \mapsto (a, (c, 0)) \ldots \mathbf{a} + n' - 1, \mathbf{n_{id}} \mapsto (a + n' - 1, (c, 0))]$ where c is a fresh color and $\mathbf{n}' = n'$. Thus from rule (Tr-SAlloc), we have $\overline{\alpha} = \mathtt{alloc}(n', a^c, \lambda\_.0)$.

We now have to prove that

$$\exists T', T' =_{\delta'} \mathbf{A}' \tag{10}$$

From rule (H-Alloc), we have $\langle \mathbf{T}, \mathbf{s}_{\mathbf{A}}^*, \mathbf{s}_{\mathbf{f}}^* \rangle \xhookrightarrow{\mathbf{salloc}(\mathbf{a}, \mathbf{n}', \mathbf{n_{id}})} \langle \mathbf{T}', \mathbf{s}_{\mathbf{A}}'^*, \mathbf{s}_{\mathbf{F}}'^* \rangle$.

Thus $\mathbf{A}' = (\mathbf{s}'^{*}_{\mathbf{A}}, \mathbf{s}'^{*}_{\mathbf{F}})$. From rule (Seg-Alloc), we have

$$\mathbf{s}^{*}_{\mathbf{f}} = \mathbf{s}^{*}_{\mathbf{1}} \cdot (\mathbf{a}, \mathbf{n}'', \mathbf{n}'_{\mathbf{id}}) \cdot \mathbf{s}^{*}_{\mathbf{2}} \tag{11}$$
$$\mathbf{0} < \mathbf{n}' \leq \mathbf{n}'' \tag{12}$$
$$\mathbf{a} + \mathbf{n}' < |\mathbf{T}| \tag{13}$$
$$\mathbf{aligned}(\mathbf{a}) \tag{14}$$
$$\mathbf{T}' = \mathbf{T}[\mathbf{a} + \mathbf{j} \mapsto (\mathbf{0}, \mathbf{D}) \mid \mathbf{j} \in \mathbf{0}..\mathbf{n}' - \mathbf{1}] \tag{15}$$
$$\mathbf{Y}' = \mathbf{Y} \cdot \mathbf{salloc}\,(\mathbf{a}, \mathbf{n}', \mathbf{n_{id}}) \tag{16}$$
$$\mathbf{s}'^{*}_{\mathbf{f}} = \mathbf{s}^{*}_{\mathbf{1}} \cdot (\mathbf{a} + \mathbf{n}', \mathbf{n}'' - \mathbf{n}', \mathbf{n}'_{\mathbf{id}}) \cdot \mathbf{s}^{*}_{\mathbf{2}} \tag{17}$$
$$\mathbf{fresh}(\mathbf{n_{id}}) \tag{18}$$
$$\mathbf{s}'^{*}_{\mathbf{A}} = (\mathbf{a}, \mathbf{n}', \mathbf{n_{id}}) \cdot \mathbf{s}^{*}_{\mathbf{A}} \tag{19}$$

Let $T' = (T[a + i \mapsto (c, 0) \mid i \in \{0, \ldots, n' - 1\}], T.Y.\texttt{alloc}(n', a^c, \lambda\_.0))$ where $n' = \mathbf{n}'$. Note that c is picked to be fresh in $\boldsymbol{\delta}'$, thus it is also fresh in T' since $\boldsymbol{\delta}' \supseteq \boldsymbol{\delta}$ and $T =_{\boldsymbol{\delta}} \boldsymbol{\Omega}'.\mathbf{A}$. From Definition 9 (Segment Agreement), we thus have $T' =_{\boldsymbol{\delta}'} \boldsymbol{\Omega}'.\mathbf{A}$.

From the action relation, we also get the history relation $T'.Y =_{\boldsymbol{\delta}'} \mathbf{Y}'$.

Finally, from rule (MS-Alloc) we have $\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T'$.

Case (H-Load) and (H-Load-Handle): Given

$$\boldsymbol{\Phi}^{*} \vdash \langle \boldsymbol{\Sigma}, (\theta, \boldsymbol{\tau}.\mathbf{segload} : \mathbf{i}^{*}, \mathbf{h} : \mathbf{v}^{*})_{\mathbf{n}} : \mathbf{S}^{*}_{\mathbf{0}} \rangle \xrightarrow{\mathbf{read}_{\boldsymbol{\tau}}(\mathbf{h})}{}_{\mathbf{n_f}} \langle \boldsymbol{\Sigma}, (\theta, \mathbf{i}^{*}, \mathbf{v}' : \mathbf{v}^{*}) \rangle_{\mathbf{n}} : \mathbf{S}^{*}_{\mathbf{0}}.$$

Let $\boldsymbol{\Sigma}' = \langle \mathbf{H}, \mathbf{T}', \mathbf{A}' \rangle$, $\mathbf{h} = \langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}} \rangle$. Given $\alpha = \mathbf{read}_{\boldsymbol{\tau}}\,(\mathbf{a})$ and $T =_{\boldsymbol{\delta}} \mathbf{A}$. We first have to prove

$$\exists \boldsymbol{\delta}', \boldsymbol{\delta}' \supseteq \boldsymbol{\delta}$$
$$\exists \overline{\alpha}, \alpha =_{\boldsymbol{\delta}'} \overline{\alpha}$$

From the segment agreement between $\mathbf{A}$ and T (rule (Segment Agreement)), we have $\boldsymbol{\delta}(\mathbf{b}, \mathbf{n_{id}}) = b^{(c,s)}$.

Let $\boldsymbol{\delta}' = \boldsymbol{\delta}$ and $\overline{\alpha} = \texttt{read}(a^{(c,s)}) \cdot \texttt{read}((a+1)^{(c,s)}) \ldots \texttt{read}((a+n-1)^{(c,s)})$ where $n = \mathbf{sz}\,(\tau)$ and $a = b + \mathbf{o}$.

From rule (Tr-Read), we have $\mathbf{read}_{\boldsymbol{\tau}}\,(\langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}} \rangle) =_{\boldsymbol{\delta}'} \texttt{read}(a^{(c,s)}) \cdot \texttt{read}((a+1)^{(c,s)}) \ldots \texttt{read}((a+n-1)^{(c,s)})$.

From (H-Load), we have $\mathbf{A}' = \mathbf{A}$. Let $T' = T$. It follows that $T' =_{\boldsymbol{\delta}'} \mathbf{A}'$.

We still have to prove that $\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T'$.

From the segment agreement we know that the slots are in the allocated part of T', so from $\alpha =_{\boldsymbol{\delta}'} \overline{\alpha}$ and $T' =_{\boldsymbol{\delta}'} \mathbf{A}'$.

However, in order to conclude that $\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T'$, we need to prove $\vdash T \overset{\texttt{read}(a+i^{(c,s)})}{\rightsquigarrow} T$ for all $i \in \{0, \ldots, n-1\}$. That is, we have to prove the premises of rule (MS-Read). Specifically, we have to show that $T(a) = A(c,s) \cdots T(a+n-1) = A(c,s)$. We derive that from (1)$\vdash$ $\mathbf{isValid\ \Omega}$ (the handle is valid and correspond to an allocator slot), (2) $T =_\delta \mathbf{\Omega.A}$ (colors for locations accessed are in agreement with abstract memory), and (3) bounds and tid checks from the rule (H-Load) (or rule (H-Load-Handle)) (access is in bounds, i.e., within the slot, therefore all locations have the same color). It follows that $\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T'$ (from rule (MS-Read) and rule (Cons)) with $T' = T$.

Case (H-Store) and (H-Store-Handle): Similar to the above (H-Load) case.

Case (H-Load-Trap): Given $\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \tau.\mathbf{segload} : \mathbf{i}^*, \mathbf{v_1} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{trap}} \langle \mathbf{\Sigma}, \theta, \epsilon, \epsilon \rangle$. Let $\delta' = \delta$ and $T' = T$. Since Σ has not changed during the step, we trivially have $T' =_{\delta'} \mathbf{\Sigma.A}$. Also, from rule (Tr-Trap), we have $\mathbf{trap} =_\delta \epsilon$. Thus $\overline{\alpha} = \epsilon$. From rule (Empty), we have $\vdash T \overset{\epsilon}{\rightsquigarrow} T$.

Case (H-Store-Trap): Similar to the above (H-Load-Trap) case.

Case (H-Free) We have this reduction

$$\frac{(\mathbf{H}, \mathbf{T}, \mathbf{A}) = \mathbf{\Sigma} \quad \mathbf{v} = \langle \mathbf{a}, \mathbf{n}, \mathbf{n}', \mathbf{b}, \mathbf{n_{id}} \rangle \quad \langle \mathbf{T}, \mathbf{A} \rangle \xrightarrow{\mathbf{sfree(a)}} \langle \mathbf{T}', \mathbf{A}' \rangle \quad \mathbf{\Sigma}' = (\mathbf{H}, \mathbf{T}', \mathbf{A}')}{\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{segfree} : \mathbf{i}^*, \mathbf{v} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{sfree(v)}} \langle \mathbf{\Sigma}', \theta, \mathbf{i}^*, \mathbf{v}^* \rangle}\ \text{(H-Free-full)}$$

Let $\alpha = \texttt{sfree}\,(a^c)$ for $\mathbf{a}, \mathbf{n_{id}} =_\delta (a, c)$.

Let $\delta' = \delta$.

Let $T' = T[a+i \mapsto F(c, s_i) | i \in 0..n-1]$, given that $n = |\texttt{rng}\,(T) \cap A(c, \_)|$ and $\forall i \in 0..n-1.\, T[a+i] = A(c,s)$

From the segment agreement we get that $\delta(\mathbf{a}, \mathbf{n_{id}}) = a^{(c,\_)}$, so from Rule (Tr-Sfree) the traces are related.

To prove $T' =_{\delta'} \mathbf{A}'$ we need to show that the freed slot is tagged F in T', which is the case.

Then, from the action relation, we also get the history relation $T'.Y =_{\delta'} \mathbf{\Omega'.A.Y}$.

From Rule (MS-Free) the monitor steps to T'.

Remaining Cases: Do not emit any traces related to the memory safety monitor. Hence, they step on empty events (see rule (Empty)).

□

Lemma 11 (Memory Safety Preservation-Inter)**.** If

1. $\mathbf{\Delta} \vdash \mathbf{\Omega}$ and
2. $\vdash \mathbf{isValid}\ \mathbf{\Omega}$ and
3. $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \xrightarrow{\alpha} \mathbf{\Omega}'$ and
4. $T =_{\delta} \mathbf{\Omega.A}$

then $\exists \overline{\alpha}, \delta', T'$ such that

- $\delta' \supseteq \delta$ and
- $T' =_{\delta'} \mathbf{\Omega'.A}$ and
- $\alpha =_{\delta'} \alpha$ and
- $\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T'$

Proof. Proof by induction on the step relation: $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \xrightarrow{\alpha} \mathbf{\Omega}'$. The only interesting case is rule (Ctx). Lift intra-function semantics to inter-function semantics and invoke Lemma 10 (Memory Safety Preservation-Intra).

For rule (Ctx-Call), rule (Ctx-Return), rule (Exit) and rule (Trap), the events do not affect the memory safety monitor and use the monitor step rule (Empty). □

Lemma 12 (Memory Safety for Trace Semantics)**.** If

1. $\mathbf{\Delta} \vdash \mathbf{\Omega}$ and
2. $\vdash \mathbf{isValid}\ \mathbf{\Omega}$ and
3. $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \xRightarrow{\overline{\alpha}} \mathbf{\Omega}'$ and
4. $T =_{\delta} \mathbf{\Omega.A}$

then $\exists \overline{\alpha}, \delta', T'$ such that

- $\delta' \supseteq \delta$ and
- $T' =_{\delta'} \mathbf{\Omega'.A}$ and
- $\overline{\alpha} =_{\delta'} \overline{\alpha}$ and
- $\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T'$

Proof. Proof is by induction on the trace step.

Base: Given $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \xRightarrow{\epsilon} \mathbf{\Omega}$. $\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T'$ follows from rule (Empty), with $T = T'$.

Case (Step): Given

$$\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, (\theta, \mathbf{i}^*, \mathbf{v}^*)_{\mathbf{n}} : \mathbf{S}_0^* \rangle \xrightarrow{\alpha \cdot \overline{\alpha}} \langle \mathbf{\Sigma}', \mathbf{S}'^* \rangle \tag{20}$$

where $\mathbf{\Sigma} = (\mathbf{H}, \mathbf{T}, \mathbf{A})$ and $\mathbf{\Sigma} = (\mathbf{H}', \mathbf{T}', \mathbf{A}')$. Inverting the rule (Step), we have the

$$\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, (\theta, \mathbf{i}^*, \mathbf{v}^*) : \mathbf{S}_0^* \rangle \xrightarrow{\alpha} \langle \mathbf{\Sigma}'', \mathbf{S}''^* \rangle \tag{21}$$

$$\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}'', \mathbf{S}''^* \rangle \xrightarrow{\overline{\alpha}} \langle \mathbf{\Sigma}', \mathbf{S}'^* \rangle \tag{22}$$

such that $\mathbf{\Sigma}'' = (\mathbf{H}'', \mathbf{T}'', \mathbf{A}'')$.

We are given $T =_{\boldsymbol{\delta}} \mathbf{A}$. Since we have required premises, applying Lemma 11 (Memory Safety Preservation-Inter) on (20) give

$$\boldsymbol{\delta}'' \supseteq \boldsymbol{\delta} \tag{23}$$

$$\alpha =_{\boldsymbol{\delta}''} \overline{\alpha} \tag{24}$$

$$T'' =_{\boldsymbol{\delta}''} \mathbf{A}'' \tag{25}$$

$$\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T'' \tag{26}$$

From Lemma 8 (Type and Validity Preservation for Configurations-Inter), we further have

$$\mathbf{\Delta} \vdash \langle \mathbf{\Sigma}'', \mathbf{S}''^* \rangle \tag{27}$$

$$\vdash \mathbf{isValid}\ \langle \mathbf{\Sigma}'', \mathbf{S}''^* \rangle \tag{28}$$

Applying I.H. on (22) using (27), (28) and (25),

$$\boldsymbol{\delta}' \supseteq \boldsymbol{\delta} \tag{29}$$

$$\overline{\alpha} =_{\boldsymbol{\delta}'} \overline{\alpha} \tag{30}$$

$$T' =_{\boldsymbol{\delta}'} \mathbf{\Sigma}'.\mathbf{A} \tag{31}$$

$$\vdash T'' \overset{\overline{\alpha}''}{\rightsquigarrow} T' \tag{32}$$

From (23) and (29), we have $\boldsymbol{\delta}' \supseteq \boldsymbol{\delta}$.

Combining (24) and (30) using rule (Tr-Concatenate) we have $\alpha \cdot \overline{\alpha} = \overline{\alpha} \cdot \overline{\alpha}''$.

Combining (26) and (32) using the monitor step (Cons) we can conclude $\vdash T \overset{\overline{\alpha} \cdot \overline{\alpha}''}{\rightsquigarrow} T'$.

Hence proved.

□

Lemma 13 (Instantiated Config is Valid)**.** If $\mathbf{\Omega_0}\ (\mathbf{M}, \mathbf{n_f})$ then $\vdash \mathbf{isValid}\ \mathbf{\Omega}$.

Proof. Follows from the validity of initialized segment (rule (segment)) and empty stack (rule (empty-data-stack)). □

Theorem 1 (Robust Memory-Safety)**.** If $\vdash \mathbf{M}$ then $\vdash \mathbf{RMS}\ (\mathbf{M} \circ \mathbf{A})$.

Proof Sketch. Let $\mathcal{A}$ be some attacker such that $\mathbf{main} = \{\mathbf{var}\ \tau^{\mathbf{v}}, \mathbf{body}\ \mathbf{i}^*\}$. Let the following:

$$\mathbf{\Omega} = \mathbf{\Omega_0}\ (\mathcal{A} \circ \mathbf{M}, \mathbf{n_{main}}) \tag{33}$$

$$\mathbf{\Phi}^* \vdash \mathbf{\Omega} \overset{\overline{\alpha}}{\Longrightarrow} \mathbf{\Omega}' \tag{34}$$

$$\boldsymbol{\delta} = \varnothing \tag{35}$$

We need to show $\mathtt{MS}(\overline{\alpha})$. We will invoke Lemma 12 (Memory Safety for Trace Semantics) to get the required proof. However, we need the following:

$$\mathbf{\Delta} \vdash \mathbf{\Omega} \tag{36}$$

$$\vdash \mathbf{isValid}\ \mathbf{\Omega} \tag{37}$$

$$\mathbf{\Phi}^* \vdash \mathbf{\Omega} \overset{\overline{\alpha}}{\Longrightarrow} \mathbf{\Omega}' \tag{38}$$

$$T =_{\boldsymbol{\delta}} \mathbf{\Omega.A} \tag{39}$$

Applying Lemma 13 (Instantiated Config is Valid) to $\mathbf{\Omega_0}\ (\mathcal{A} \circ \mathbf{M}, \mathbf{n_{main}})$, we have (37).

Given $\vdash \mathcal{A} \circ \mathbf{M}$. From rule (Module) and rule (Fun), we have

$$\mathbf{\Gamma}\{\mathbf{local} = \tau^{\mathbf{a}}\tau^{\mathbf{v}}\ \mathbf{return} = \tau^*\} \vdash \mathbf{i}^* : [] \to \tau^* \tag{40}$$

From rule (Stack), we have $\mathbf{\Delta} \vdash (\theta, \mathbf{i}^*, \epsilon)_{\mathbf{n_{main}}}$. Thus, we have (36).

Since the initial allocator has no used slots, i.e., $\mathbf{\Omega.A} = (\epsilon, \_)$, we have (39).

Invoking Lemma 12 (Memory Safety for Trace Semantics) using (36) to (39), we have

$$\boldsymbol{\delta}' \supseteq \boldsymbol{\delta} \tag{41}$$

$$T' =_{\boldsymbol{\delta}'} \mathbf{A}' \tag{42}$$

$$\overline{\alpha} =_{\boldsymbol{\delta}'} \overline{\alpha} \tag{43}$$

$$\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T' \tag{44}$$

Since T is empty, (43) and (44) together imply $\mathtt{MS}(\overline{\alpha})$. Hence proved.

□

4 Compiler

We define the formal compilation from source C language to MSWasm.

4.1 Compilation of Programs

The compilation judgment for types, environments, program, functions and expressions:

$$[\![\mathsf{w}]\!] = \tau$$
$$[\![\mathsf{P}, F_{\mathsf{map}}]\!]^{\mathsf{glob}} = \mathbf{\Delta}$$
$$[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M}$$
$$[\![\mathsf{w_r}\ \mathsf{f}(\mathsf{x} : \mathsf{w_a}) \mapsto \mathsf{e}]\!]^{\mathsf{fun}} = \mathbf{i}$$
$$[\![\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{i}$$

(Integers)
$$\frac{}{[\![\mathsf{int}]\!] = \mathbf{u32}}$$

(C-Pointers)
$$\frac{}{[\![\mathsf{ptr\ w}]\!] = \mathbf{handle}}$$

(C-Array)
$$\frac{}{[\![\mathsf{array}\ \tau]\!] = \mathbf{handle}}$$

(C-T-All-FuncTypes)
$$\frac{\forall k \in |P.F|, \tau_{\mathsf{rk}}\ \mathsf{f_k}(\mathsf{x} : \tau_{\mathsf{ak}}) \mapsto \_ \in \mathsf{P.F} \implies [\![\tau_{\mathsf{ak}}]\!] = \tau_{\mathbf{ak}} \implies [\![\tau_{\mathsf{rk}}]\!] = \tau_{\mathbf{rk}} \implies \rho[k] = \tau_{\mathbf{ak}} \to \tau_{\mathbf{rk}}}{[\![\mathsf{P.F}]\!]^{\mathsf{all}} = \rho^*}$$

(C-T-Env)
$$\frac{\begin{array}{c}\tau_{\mathsf{r}}\ \mathsf{f}(\mathsf{x} : \tau_{\mathsf{a}}) \mapsto ((\mathsf{y_j} : \tau_{\mathsf{j}})^*, \mathsf{e_f}) \in \mathsf{P.F} \\ [\![\tau_{\mathsf{a}}]\!] = \tau_{\mathbf{a}} \qquad [\![\tau_{\mathsf{r}}]\!] = \tau_{\mathbf{r}} \qquad [\![\tau_{\mathsf{j}}]\!] = \tau_{\mathbf{j}} \\ [\![\mathsf{P.F}]\!]^{\mathsf{all}} = \rho^* \\ \mathbf{\Gamma_f} = \{\mathbf{funcs}\ \rho^*, \mathbf{locals}\ \tau_{\mathbf{a}}\tau_{\mathbf{1}}\tau_{\mathbf{2}} \ldots, \mathbf{return}\ \tau_{\mathbf{r}}\}\end{array}}{[\![\mathsf{P}, \Gamma_{\mathsf{f}}]\!]^{\mathsf{env}} = \mathbf{\Gamma_f}}$$

(C-T-Global-Env)
$$\frac{\forall i, [\![\mathsf{P}, \Delta[i]]\!]^{\mathsf{env}} = \mathbf{\Delta}[i]}{[\![\mathsf{P}, \Delta]\!]^{\mathsf{glob}} = \mathbf{\Delta}}$$

Compilation of types and environments

(C-Function)
$$\frac{\begin{array}{c}F_{\mathsf{map}}(\mathsf{f}) = \mathbf{n_f} \\ [\![\tau_{\mathsf{a}}]\!] = \tau_{\mathbf{a}} \qquad [\![\tau_{\mathsf{r}}]\!] = \tau_{\mathbf{r}} \qquad \forall i, [\![\tau_{\mathsf{i}}]\!] = \tau_{\mathbf{i}} \qquad \tau^{\mathsf{j}} = \tau_{\mathbf{a}}; \tau_{\mathbf{1}}; \tau_{\mathbf{2}}; \ldots \\ [\![\mathsf{P}, [\mathsf{x} \mapsto \tau_{\mathsf{a}}] \vdash \mathsf{e} : \tau_{\mathsf{r}}]\!]^{\mathsf{exp}} = \mathbf{i}^*\end{array}}{[\![\tau_{\mathsf{r}}\ \mathsf{f}(\mathsf{x} : \tau_{\mathsf{a}}) \mapsto ((\mathsf{y_j} : \tau_{\mathsf{j}})^*, \mathsf{e})]\!]^{\mathsf{fun}} = \langle \mathbf{var}\ \tau^{\mathsf{j}}, \mathbf{i}^* \rangle}$$

(Program)
$$\frac{\begin{array}{c}\forall k, \mathsf{P.F}[k] = \mathsf{f_k}, [\![\mathsf{w_{rk}}\ \mathsf{f_k}(\mathsf{x} : \mathsf{w_{ak}}) \mapsto ((\mathsf{y_{jk}} : \mathsf{w_{jk}})^*, \mathsf{e_k}]\!]^{\mathsf{fun}} = \mathbf{f_k} \\ |\mathbf{s}| = \mathtt{sz}\,(\mathsf{P.n_{hs}}) \\ \mathbf{M} = \{\mathbf{heap}\ \mathbf{0}, \mathbf{segment}\ \mathbf{s}, \mathbf{funcs}\ \mathbf{f_0 f_1} \ldots \mathbf{f_n}\}\end{array}}{[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M}}$$

Compilation of function and program

(C-Constant-int)
$$\frac{|\mathsf{n}| = |\mathbf{n}|}{[\![\mathsf{P}, \Gamma \vdash \mathsf{n} : \mathsf{int}]\!]^{\mathsf{exp}} = \mathbf{u32.const\ n}}$$

(C-Var)
$$\frac{}{[\![\mathsf{P}, \Gamma \vdash \mathsf{x} : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{get\ x}}$$

(C-Ptr-Arith)
$$\frac{\begin{array}{c}[\![\mathsf{P}, \Gamma \vdash \mathsf{e}_1 : \mathsf{ptr\ array}\ \tau]\!]^{\mathsf{exp}} = \mathbf{i}_1^* \\ [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_2 : \mathsf{int}]\!]^{\mathsf{exp}} = \mathbf{i}_2^* \\ \mathbf{n} = \mathbf{sz}\,(\mathsf{w})\end{array}}{[\![\mathsf{P}, \Gamma \vdash \mathsf{e}_1 \oplus \mathsf{e}_2 : \mathsf{ptr\ array}\ \tau]\!]^{\mathsf{exp}} = \mathbf{i}_1^*; \mathbf{i}_2^*; \mathbf{u32.const\ n}; \mathbf{u32.}\times; \mathbf{handle.add}}$$

(C-BinOp)
$$\frac{\begin{array}{c}[\![\mathsf{P}, \Gamma \vdash \mathsf{e}_j : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{i}_j^* \\ \mathbf{j} \in \{1, 2\} \\ [\![\tau]\!] = \tau\end{array}}{[\![\mathsf{P}, \Gamma \vdash \mathsf{e}_1 \oplus \mathsf{e}_2 : \tau]\!]^{\mathsf{exp}} = \mathbf{i}_1^*; \mathbf{i}_2^*; \boldsymbol{\tau}.\otimes}$$

(C-Malloc-Array)
$$\frac{\begin{array}{c}[\![\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{int}]\!]^{\mathsf{exp}} = \mathbf{i}^* \\ \mathbf{n}' = \mathbf{sz}\,(\tau)\end{array}}{[\![\mathsf{P}, \Gamma \vdash \mathsf{malloc}(\tau, \mathsf{e}) : \mathsf{ptr\ array}\ \tau]\!]^{\mathsf{exp}} = \mathbf{i}^*; \mathbf{u32.const\ n}'; \mathbf{u32.mult}; \mathbf{segalloc}}$$

(C-Malloc-Single)
$$\frac{\mathbf{n} = \mathbf{sz}\,(\mathsf{w})}{[\![\mathsf{P}, \Gamma \vdash \mathsf{malloc}(\mathsf{w}) : \mathsf{ptr\ w}]\!]^{\mathsf{exp}} = \mathbf{u32.const\ n}; \mathbf{segalloc}}$$

(C-Free)
$$\frac{[\![\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{ptr\ w}]\!]^{\mathsf{exp}} = \mathbf{i}^*}{[\![\mathsf{P}, \Gamma \vdash \mathsf{free(e)} : \mathsf{int}]\!]^{\mathsf{exp}} = \mathbf{i}^*; \mathbf{free}\ ; \mathbf{u32.const\ 0}}$$

Compilation of expressions

(C-Deref)
$$\frac{[\![\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{ptr}\ \tau]\!]^{\mathsf{exp}} = \mathbf{i}^*}{[\![\mathsf{P}, \Gamma \vdash *\mathsf{e} : \tau]\!]^{\mathsf{exp}} = \mathbf{i}^*; \boldsymbol{\tau}.\mathbf{segload}}$$

(C-Struct-field)
$$\frac{\begin{array}{c}[\![\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{ptr\ struct\ s}]\!]^{\mathsf{exp}} = \mathbf{i}^* \\ \mathsf{D(s)} = \{\mathsf{f}_1 : \tau_1, \ldots \mathsf{f}_k : \tau_k \ldots \mathsf{f}_n : \tau_n\} \\ (\mathbf{o_1}, \mathbf{o_2}) = (\Sigma_i^{k-1}\mathbf{sz}\,(\tau_i), (\mathbf{sz}\,(\mathsf{struct\ s}) - \mathbf{sz}\,(\tau_k)))\end{array}}{[\![\mathsf{P}, \Gamma \vdash \&\mathsf{e} \to \mathsf{f}_k : \mathsf{ptr}\ \tau_k]\!]^{\mathsf{exp}} = \mathbf{i}^*; \mathbf{u32.const\ o_1}; \mathbf{u32.const\ o_2}, \mathbf{slice}}$$

(C-Arr-Deref)
$$\frac{\begin{array}{c}[\![\mathsf{P}, \Gamma \vdash \mathsf{e}_1 : \mathsf{array}\ \tau]\!]^{\mathsf{exp}} = \mathbf{i}_1^* \\ [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_2 : \mathsf{int}]\!]^{\mathsf{exp}} = \mathbf{i}_2^* \\ \mathbf{n} = \mathbf{sz}\,(\tau)\end{array}}{[\![\mathsf{P}, \Gamma \vdash \mathsf{e}_1[\mathsf{e}_2] : \mathsf{ptr\ array}\ \tau]\!]^{\mathsf{exp}} = \mathbf{i}_1^*; \mathbf{i}_2^*; \mathbf{u32.const\ n}; \mathbf{u32.}\times; \mathbf{handle.add}; \boldsymbol{\tau}.\mathbf{segload}}$$

(C-Var-Assign)

$$\frac{[\![\tau]\!] = \tau \quad [\![\mathsf{P}, \Gamma \vdash \mathsf{e} : \tau]\!]^{\mathsf{exp}} = \mathbf{i}^*}{[\![\mathsf{P}, \Gamma \vdash \mathsf{x} := \mathsf{e} : \mathsf{int}]\!]^{\mathsf{exp}} = \mathbf{i}^*; \mathbf{set\ x}; \mathbf{u32.const\ 0}}$$

(C-Assign)

$$\frac{[\![\tau]\!] = \tau \quad [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_1 : \tau]\!]^{\mathsf{exp}} = \mathbf{i}_1^* \quad [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_2 : \tau]\!]^{\mathsf{exp}} = \mathbf{i}_2^*}{[\![\mathsf{P}, \Gamma \vdash *\mathsf{e}_1 := \mathsf{e}_2 : \mathsf{int}]\!]^{\mathsf{exp}} = \mathbf{i}_1^*; \mathbf{i}_2^*; \boldsymbol{\tau}\mathbf{.segstore}; \mathbf{u32.const\ 0}}$$

(C-Array-Assign)

$$\frac{\begin{array}{c}[\![\tau]\!] = \tau \quad [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_1 : \mathsf{array}\ \tau]\!]^{\mathsf{exp}} = \mathbf{i}_1^* \\ [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_2 : \mathsf{int}]\!]^{\mathsf{exp}} = \mathbf{i}_2^* \quad [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_3 : \tau]\!]^{\mathsf{exp}} = \mathbf{i}_3^* \quad \mathbf{n} = \mathbf{sz}\,(\tau)\end{array}}{[\![\mathsf{P}, \Gamma \vdash \mathsf{e}_1[\mathsf{e}_2] := \mathsf{e}_3 : \mathsf{int}]\!]^{\mathsf{exp}} = \mathbf{i}_1^*; \mathbf{i}_2^*; \mathbf{u32.const\ n}; \mathbf{u32.}\times; \mathbf{handle.add}; \mathbf{i}_3^*; \boldsymbol{\tau}\mathbf{.segstore}; \mathbf{u32.const\ 0}}$$

(C-Branch)

$$\frac{[\![\mathsf{P}, \Gamma \vdash \mathsf{e}_0 : \mathsf{int}]\!]^{\mathsf{exp}} = \mathbf{i}_0^* \quad [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_1 : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{i}_1^* \quad [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_2 : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{i}_2^*}{[\![\mathsf{P}, \Gamma \vdash \mathsf{if}\ \mathsf{e}_0\ \mathsf{then}\ \mathsf{e}_1\ \mathsf{else}\ \mathsf{e}_2 : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{i}_0^*; \mathbf{branch\ i}_1^*\ \mathbf{i}_2^*}$$

(C-Seq)

$$\frac{[\![\mathsf{P}, \Gamma \vdash \mathsf{e}_1 : \mathsf{w}_1]\!]^{\mathsf{exp}} = \mathbf{i}_1^* \quad [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_2 : \mathsf{w}']\!]^{\mathsf{exp}} = \mathbf{i}_2^*}{[\![\mathsf{P}, \Gamma \vdash \mathsf{e}_1; \mathsf{e}_2 : \mathsf{w}']\!]^{\mathsf{exp}} = \mathbf{i}_1^*; \mathbf{drop}; \mathbf{i}_2^*}$$

(C-Call)

$$\frac{\begin{array}{c}F_{\mathsf{map}}(\mathsf{g}) = \mathbf{n_g} \quad V_{\mathsf{map}}(\mathsf{x}) = \mathbf{n_x} \\ \mathrm{flookup}(\mathsf{P}, \mathsf{g}) = \mathsf{w}_2\ \mathsf{g}(\mathsf{x} : \mathsf{w}_1) \mapsto ((\mathsf{y_j} : \mathsf{w_j})^*, \mathsf{e_g}) \\ [\![\mathsf{w}_2\ \mathsf{g}(\mathsf{x} : \mathsf{w}_1) \mapsto ((\mathsf{y_j} : \mathsf{w_j})^*, \mathsf{e_g})]\!]^{\mathsf{fun}} = \langle \mathbf{var}\ \tau^{\mathsf{j}}, \mathbf{i}^* \rangle \\ [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_1 : \mathsf{w}_1]\!]^{\mathsf{exp}} = \mathbf{i}_1^*\end{array}}{[\![\mathsf{P}, \Gamma \vdash \mathsf{x} := \mathsf{call\ g}\,(\mathsf{e}_1) : \mathsf{w}']\!]^{\mathsf{exp}} = \mathbf{i}_1^*; \mathbf{call\ n_g}; \mathbf{set\ n_x}}$$

Compilation of expressions

4.2 Evaluation Context Compilation

Evaluation context compilation is the following relation:

$$\mathsf{P}, \Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{w} \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle$$

(Ctx-Hole)

$$\frac{}{\mathsf{P}, \Gamma \vdash [\cdot] : \mathsf{w} \Rightarrow \mathsf{w} \sim_\delta \langle [], [] \rangle}$$

(Ctx-Op1)

$$\frac{[\![w]\!] = \tau \quad [\![P, \Gamma \vdash e : w]\!]^{exp} = \mathbf{i}^* \quad P, \Gamma \vdash E : w' \Rightarrow w \sim_\delta \langle \mathbf{i}'^*, \mathbf{v}^* \rangle}{P, \Gamma \vdash E \oplus e : w' \Rightarrow w \sim_\delta \langle \mathbf{i}'^*; \mathbf{i}^*; \boldsymbol{\tau}.\otimes, \mathbf{v}^* \rangle}$$

(Ctx-Op2)

$$\frac{[\![w]\!] = \tau \quad P, \Gamma \vdash v' : w \sim_\delta \mathbf{v}' \quad P, \Gamma \vdash E : w' \Rightarrow w \sim_\delta \langle \mathbf{i}'^*, \mathbf{v}^* \rangle}{P, \Gamma \vdash v' \oplus E : w' \Rightarrow w \sim_\delta \langle \mathbf{i}'^*; \boldsymbol{\tau}.\otimes, (\mathbf{v}^*; \mathbf{v}') \rangle}$$

(Ctx-Field)

$$\frac{[\![P, \Gamma \vdash E : w' \Rightarrow \mathsf{ptr}\,\mathsf{struct}\,s]\!]^{exp} = \langle \mathbf{i}'^*, \mathbf{v}^* \rangle \quad (\mathbf{o_1}, \mathbf{o_2}) = \mathrm{offset}(s, f)}{P, \Gamma \vdash \&E \to f : w' \Rightarrow \mathsf{ptr}\,\tau_f \sim_\delta \langle \mathbf{i}'^*; \mathbf{u32.const}\ \mathbf{o_1}; \mathbf{u32.const}\ \mathbf{o_2}; \mathbf{slice}, \mathbf{v}^* \rangle}$$

(Ctx-Malloc)

$$\frac{P, \Gamma \vdash E : w' \Rightarrow \mathsf{int} \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle \quad \mathbf{n} = \mathbf{sz}\,(\tau)}{P, \Gamma \vdash \mathrm{malloc}(\tau, E) : w' \Rightarrow \mathsf{array}\ w \sim_\delta \langle \mathbf{i}^*; \mathbf{u32.const}\ \mathbf{n}; \mathbf{u32}.\times; \mathbf{segalloc}, \mathbf{v}^* \rangle}$$

(Ctx-Deref)

$$\frac{[\![w]\!] = \tau \quad P, \Gamma \vdash E : w' \Rightarrow \mathsf{ptr}\,w \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle}{P, \Gamma \vdash *E : w' \Rightarrow w \sim_\delta \langle \mathbf{i}^*; \boldsymbol{\tau}.\mathbf{segload}, \mathbf{v}^* \rangle}$$

(Ctx-Arr-Deref)

$$\frac{[\![\tau]\!] = \tau \quad P, \Gamma \vdash E : w \Rightarrow \mathsf{ptr}\,\mathsf{array}\ \tau \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle \quad P, \Gamma \vdash n : \mathsf{int} \sim_\delta \mathbf{n} \quad \mathbf{n}' = \mathbf{sz}\,(\tau)}{P, \Gamma \vdash E[n] : w \Rightarrow \tau \sim_\delta \langle \mathbf{i}^*; \mathbf{u32.const}\ \mathbf{n}; \mathbf{u32.const}\ \mathbf{n}'; \mathbf{u32}.\times; \mathbf{handle.add}; \boldsymbol{\tau}.\mathbf{segload}, \mathbf{v}^* \rangle}$$

(Ctx-Arr-Deref1)

$$\frac{[\![\tau]\!] = \tau \quad P, \Gamma \vdash E : w \Rightarrow \mathsf{int} \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle \quad P, \Gamma \vdash a^{(b,\ell,\tau,n_{id})} : \mathsf{ptr}\,\mathsf{array}\ \tau \sim_\delta \mathbf{a} \quad \mathbf{n}' = \mathbf{sz}\,(\tau)}{P, \Gamma \vdash a^{(b,\ell,\tau,n_{id})}[E] : w \Rightarrow \tau \sim_\delta \langle \mathbf{i}^*; \mathbf{u32.const}\ \mathbf{n}'; \mathbf{u32}.\times; \mathbf{handle.add}; \boldsymbol{\tau}.\mathbf{segload}, \mathbf{v}^*; \mathbf{a} \rangle}$$

(Ctx-Var-Assign)

$$\frac{[\![\tau]\!] = \tau \quad P, \Gamma \vdash E : w' \Rightarrow \tau \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle}{P, \Gamma \vdash x := E : w' \Rightarrow \tau \sim_\delta \langle \mathbf{i}^*; \mathbf{dup}; \mathbf{set}\ \mathbf{x}, \mathbf{v}^* \rangle}$$

(Ctx-Assign)

$$\frac{[\![\tau]\!] = \tau \quad P, \Gamma \vdash E : w' \Rightarrow \mathsf{ptr}\,\tau \sim_\delta \langle \mathbf{i}_1^*, \mathbf{v}^* \rangle \quad [\![P, \Gamma \vdash e_2 : \tau]\!]^{exp} = \mathbf{i}_2^*}{P, \Gamma \vdash *E := e_2 : w' \Rightarrow \tau \sim_\delta \langle \mathbf{i}_1^*; \mathbf{i}_2^*; \boldsymbol{\tau}.\mathbf{segstore}; \mathbf{u32.const}\ \mathbf{0}, \mathbf{v}^* \rangle}$$

(Ctx-Assign2)

$$\frac{[\![\mathsf{w}]\!] = \tau \quad \mathsf{P},\Gamma \vdash \mathsf{v}' : \mathsf{ptr}\,\tau \sim_\delta \mathbf{v}' \quad \mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \tau \sim_\delta \langle \mathbf{i}_2^*, \mathbf{v}^* \rangle}{\mathsf{P},\Gamma \vdash *\mathsf{v}' := \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{int} \sim_\delta \langle \mathbf{i}_2^*; \boldsymbol{\tau}.\mathbf{segstore}; \mathbf{u32.const\ 0}, (\mathbf{v}^*; \mathbf{v}') \rangle}$$

(Ctx-Arr-Assign1)

$$\frac{[\![\tau]\!] = \tau \quad \mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{ptr}\,\tau \sim_\delta \langle \mathbf{i}_1^*, \mathbf{v}^* \rangle \quad [\![\mathsf{P},\Gamma \vdash \mathsf{e}_2 : \tau]\!]^{\mathsf{exp}} = \mathbf{i}_2^*}{\mathsf{P},\Gamma \vdash *\mathsf{E} := \mathsf{e}_2 : \mathsf{w}' \Rightarrow \tau \sim_\delta \langle \mathbf{i}_1^*; \mathbf{i}_2^*; \boldsymbol{\tau}.\mathbf{segstore}; \mathbf{u32.const\ 0}, \mathbf{v}^* \rangle}$$

(Ctx-Arr-Assign2)

$$\frac{[\![\tau]\!] = \tau \quad \mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{ptr}\,\tau \sim_\delta \langle \mathbf{i}_1^*, \mathbf{v}^* \rangle \quad [\![\mathsf{P},\Gamma \vdash \mathsf{e}_2 : \tau]\!]^{\mathsf{exp}} = \mathbf{i}_2^*}{\mathsf{P},\Gamma \vdash *\mathsf{E} := \mathsf{e}_2 : \mathsf{w}' \Rightarrow \tau \sim_\delta \langle \mathbf{i}_1^*; \mathbf{i}_2^*; \boldsymbol{\tau}.\mathbf{segstore}; \mathbf{u32.const\ 0}, \mathbf{v}^* \rangle}$$

(Ctx-Arr-Assign3)

$$\frac{[\![\tau]\!] = \tau \quad \mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{ptr}\,\tau \sim_\delta \langle \mathbf{i}_1^*, \mathbf{v}^* \rangle \quad [\![\mathsf{P},\Gamma \vdash \mathsf{e}_2 : \tau]\!]^{\mathsf{exp}} = \mathbf{i}_2^*}{\mathsf{P},\Gamma \vdash *\mathsf{E} := \mathsf{e}_2 : \mathsf{w}' \Rightarrow \tau \sim_\delta \langle \mathbf{i}_1^*; \mathbf{i}_2^*; \boldsymbol{\tau}.\mathbf{segstore}; \mathbf{u32.const\ 0}, \mathbf{v}^* \rangle}$$

(Ctx-Seq)

$$\frac{[\![\mathsf{P},\Gamma \vdash \mathsf{e} : \mathsf{w}_2]\!]^{\mathsf{exp}} = \mathbf{i}_2^* \quad \mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{w}_1 \sim_\delta \langle \mathbf{i}_1^*, \mathbf{v}^* \rangle}{\mathsf{P},\Gamma \vdash \mathsf{E};\mathsf{e} : \mathsf{w}' \Rightarrow \mathsf{w}_2 \sim_\delta \langle \mathbf{i}_1^*; \mathbf{i}_2^*, \mathbf{v}^* \rangle}$$

(Ctx-Branch)

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{int} \sim_\delta \langle \mathbf{i}_0^*, \mathbf{v}^* \rangle \quad [\![\mathsf{P},\Gamma \vdash \mathsf{e}_1 : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{i}_1^* \quad [\![\mathsf{P},\Gamma \vdash \mathsf{e}_2 : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{i}_2^*}{\mathsf{P},\Gamma \vdash \mathsf{if\ E\ then\ e}_1\ \mathsf{else\ e}_2 : \mathsf{w}' \Rightarrow \mathsf{w} \sim_\delta \langle \mathbf{i}_0; \mathbf{branch\ i}_1^*\ \mathbf{i}_2^*\ , \mathbf{v}^* \rangle}$$

(Ctx-Call)

$$\frac{F_{\mathsf{map}}(\mathsf{g}) = \mathbf{n_g} \quad V_{\mathsf{map}}(\mathsf{x}) = \mathbf{n_x} \quad \mathrm{flookup}(\mathsf{P},\mathsf{g}) = \mathsf{w}_2\ \mathsf{g}(\mathsf{x} : \mathsf{w}_1) \mapsto \mathsf{e}_\mathsf{g} \quad \mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w}'' \Rightarrow \mathsf{w}_1 \sim_\delta \langle \mathbf{i}_1^*, \mathbf{v}^* \rangle \quad [\![\mathsf{P},(\Gamma,\mathsf{x} : \mathsf{w}_2) \vdash \mathsf{e}_2 : \mathsf{w}']\!]^{\mathsf{exp}} = \mathbf{i_2}}{\mathsf{P},\Gamma \vdash \mathsf{x} := \mathsf{call\ g\ (E)} : \mathsf{w}'' \Rightarrow \mathsf{w}' \sim_\delta \langle \mathbf{i}_1^*; \mathbf{call\ n_g}; \mathbf{set\ n_x}; \mathbf{i}_2^*, \mathbf{v}^* \rangle}$$

$$\boxed{\mathsf{P},\Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{w} \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle}$$

(Ctx-Stack)

$$\frac{\forall \mathsf{j} \in |\mathsf{E}_\mathsf{f}^*|.\mathsf{P},\Gamma_\mathsf{f} \vdash \mathsf{E}_\mathsf{f} : \mathsf{w}_{1\mathsf{f}} \Rightarrow \mathsf{w}_{2\mathsf{f}} \sim_\delta \langle \mathbf{i}_\mathbf{j}^*, \mathbf{v}_\mathbf{j}^* \rangle \implies \mathsf{K}[\mathsf{j}] \sim_{\delta,V_{\mathsf{map}}} \langle \theta_\mathbf{j}, \mathbf{i}_\mathbf{j}^*, \mathbf{v}_\mathbf{j}^* \rangle \implies \mathbf{S}[\mathbf{j}] = \langle \theta_\mathbf{j}, \mathbf{i}_\mathbf{j}^*, \mathbf{v}_\mathbf{j}^* \rangle}{\langle \mathsf{P},\mathsf{K},\mathsf{E}_\mathsf{f}^* \rangle \sim_\delta \mathbf{S}^*}$$

$$\boxed{\langle \mathsf{P},\mathsf{K},\mathsf{E}_\mathsf{f}^* \rangle \sim_\delta \mathbf{S}^*}$$

4.3 Trace Relation

Definition 17 (Functional relation source and target events $\alpha \ =_\delta \ \alpha$)**.**

$$\text{(Tr-Rel-Read)}\quad \frac{\vdash \mathsf{v} : \tau \qquad [\![\tau]\!] = \tau \qquad \mathsf{w} \neq \mathsf{struct}\ \_ \qquad \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} \sim_\delta \langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}} \rangle}{\mathsf{read}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}, \mathsf{v}) \ =_\delta \ \mathbf{read}_\tau\ (\langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}})}$$

$$\text{(Tr-Rel-Write)}\quad \frac{\vdash \mathsf{v} : \tau \qquad [\![\tau]\!] = \tau \qquad \mathsf{w} \neq \mathsf{struct}\ \_ \qquad \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} \sim_\delta \langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}} \rangle}{\mathsf{write}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}, \mathsf{v}) \ =_\delta \ \mathbf{write}_\tau\ (\langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}})}$$

$$\text{(Tr-Rel-Allocate)}\quad \frac{(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}) \sim_\delta \langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}} \rangle}{\mathsf{salloc}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}) \ =_\delta \ \mathbf{salloc}(\langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}} \rangle)}$$

$$\text{(Tr-Rel-Free)}\quad \frac{(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}) \sim_\delta \langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}} \rangle}{\mathsf{free}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}) \ =_\delta \ \mathbf{free}(\langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}} \rangle)} \qquad \text{(Tr-Rel-Read-Forge)}\quad \frac{}{\mathsf{fread}(\mathsf{a}, \mathsf{v}) \ =_\delta \ \mathbf{trap}}$$

$$\text{(Tr-Rel-Write-Forge)}\quad \frac{}{\mathsf{fwrite}(\mathsf{a}, \mathsf{v}) \ =_\delta \ \mathbf{trap}} \qquad \text{(Tr-Rel-Free-Forge)}\quad \frac{}{\mathsf{ffree}\ \mathsf{a} \ =_\delta \ \mathbf{trap}}$$

Definition 18 (Functional relation source and target traces $\overline{\alpha} \ =_\delta \ \overline{\alpha}$)**.**

$$\text{(Empty)}\quad \frac{}{\epsilon \ =_\delta \ \epsilon} \qquad \text{(Tr-Concat)}\quad \frac{\alpha \ =_\delta \ \alpha \qquad \overline{\alpha} \ =_\delta \ \overline{\alpha}}{\alpha \cdot \overline{\alpha} \ =_\delta \ \alpha \cdot \overline{\alpha}}$$

4.4 Equivalence Relation for State

Definition 19 (Address Map δ)**.** Address map, $\delta : \mathbb{N} \times \mathsf{n_{id}} \to \mathbb{N} \times \mathbf{n_{id}}$, maps a source address to a target address.

Definition 20 (Heap Equivalence)**.**

$$\text{(Heap-Equiv)}\quad \frac{\begin{array}{c}\forall \mathsf{b} \in \mathbf{sz}\,(\mathsf{H} - 1).\ \text{let}\ \mathsf{H}[\mathsf{b}] = \mathsf{v} : \tau \\ (\mathsf{a}, \mathsf{n}, \_, \mathsf{n_{id}}) \in \lfloor \mathsf{A} \rfloor_1 \qquad \mathsf{b} \in [\mathsf{a}..\mathsf{a} + \mathsf{n}) \\ \text{let}\ \mathbf{b}, \mathbf{n_{id}} = \delta(\mathsf{b}, \mathsf{n_{id}}) \qquad [\![\tau]\!] = \tau \qquad s = \mathbf{sz}\,(\tau) \\ \mathbf{T}[\mathbf{b}]..\mathbf{T}[\mathbf{b} + s - 1] = (\mathbf{b}, \mathbf{t})_0..(\mathbf{b}, \mathbf{t})_{s-1} \\ \mathsf{v} \sim_\delta (\mathbf{b}, \mathbf{t})_0..(\mathbf{b}, \mathbf{t})_{s-1}\end{array}}{\mathsf{H} \sim_{\delta,\mathsf{A}} \mathbf{T}}$$

Definition 21 (Stack Frame Equivalence)**.** Let $\mathbf{F}_{\mathbf{n_f}} = (\theta, \mathbf{i}*, \mathbf{v}*)_{\mathbf{n_f}}$. Let $\mathsf{B_f}$ and $\mathbf{F}_{\mathbf{n_f}}$ be source and target stack frames such that $F_{\mathsf{map}}\ [\mathsf{f}] = \mathbf{n_f}$. Stack frames $\mathsf{B_f}$ and $\mathbf{F}_{\mathbf{n_f}}$ are equivalent, denoted as $\mathsf{B_f} \sim_\delta \mathbf{F}_{\mathbf{n_f}}$, if for all x, $\mathsf{B}[\mathsf{x}] \sim_\delta \theta[\mathsf{x}]$.

Definition 22 (Stack Equivalence)**.** Stacks K and $\mathbf{S}^*$ are equivalent, denoted as $\mathsf{K} \sim_\delta \mathbf{S}$, if all stack frames are equivalent. That is, for all i, $\mathsf{K}[i] \sim_\delta \mathbf{S}[i]$

Definition 23 (Slot Equivalence)**.**

$$\frac{\mathsf{n} \times \mathtt{srcsz}\,(\tau) = \mathbf{n} \qquad \forall \mathsf{i} \in \{0..\mathsf{n}-1\}.\ \delta(\mathsf{a}+\mathsf{i}, \mathsf{n_{id}}) = (\mathbf{a} + \mathsf{i} \times \mathtt{srcsz}\,(\tau), \mathbf{n_{id}})}{(\mathsf{a}, \mathsf{n}, \tau, \mathsf{n_{id}}) \sim_\delta (\mathbf{a}, \mathbf{n}, \mathbf{n_{id}})}\ \text{(Slot-Eq-Prim)}$$

$$\frac{\begin{array}{c}\mathtt{srcsz}\,(\mathsf{s}) = \mathbf{n} \qquad \mathsf{D}(\mathsf{s}) = \{\mathsf{f}_1 : \tau_1 \ldots \mathsf{f}_\mathsf{n} : \tau_\mathsf{n}\} \\ \forall \mathsf{i} \in 0..\mathsf{n}-1.\ \delta(\mathsf{a}+\mathsf{i}, \mathsf{n_{id}}) = (\mathbf{a} + \Sigma_{k=0}^{\mathsf{i}-1} \mathtt{srcsz}\,(\tau_{\mathsf{k}+1}), \mathbf{n_{id}})\end{array}}{(\mathsf{a}, 1, \mathsf{struct\ s}, \mathsf{n_{id}}) \sim_\delta (\mathbf{a}, \mathbf{n}, \mathbf{n_{id}})}\ \text{(Slot-Eq-Struct)}$$

Definition 24 (Allocator Equivalence)**.**

$$\frac{\forall X \in \{A, F\}, \mathsf{s_X}^* \sim_\delta \mathbf{s_X}^* \qquad \mathsf{Y} =_\delta \mathbf{Y}}{\mathsf{s_A}^*, \mathsf{s_F}^*, \mathsf{Y} \sim_\delta \mathbf{s_A}, \mathbf{s_F}^*, \mathbf{Y}^*}\ \text{(Alloc-Eq)}$$

Definition 25 (State Equivalence)**.** The source configuration $\langle \mathsf{H}, \mathsf{K}, \mathsf{A} \rangle$ and the target configuration $\langle \mathbf{T}, \mathbf{A}, \mathbf{S} \rangle$ are equivalent with respect to an address map δ, denoted as $\langle \mathsf{H}, \mathsf{K}, \mathsf{A} \rangle \sim_\delta \langle \mathbf{T}, \mathbf{A}, \mathbf{S} \rangle$ if

- $\mathsf{H} \sim_\delta \mathbf{T}$
- $\mathsf{A} \sim_\delta \mathbf{A}$
- $\mathsf{K} \sim_\delta \mathbf{S}$

4.5 Runtime Compilation

Since the target language uses a stack-based semantics, we need to relate the runtime configurations for source and target. Note that this only proof artifact and does not exist during the actual runtime.

Runtime Compilation of Values (Data)

$$\frac{|\mathsf{n}| = |\mathbf{n}|}{\mathsf{n} \sim_\delta \mathbf{n}}\ \text{(C-data-int)} \qquad \frac{\begin{array}{c}\vdash \mathsf{v} : \tau \\ \mathsf{v} \sim_\delta \mathbf{unpack}(\mathbf{b_0} \ldots \mathbf{b_{s-1}}) \\ s = \mathtt{sz}\,(\tau)\end{array}}{\mathsf{v} \sim_\delta (\mathbf{b}, \mathbf{t})_0 \ldots (\mathbf{b}, \mathbf{t})_{s-1}}\ \text{(Heap-Value)} \qquad \frac{}{\mathsf{a} \sim_\delta \langle \mathbf{b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{0}, \mathbf{n_{id}} \rangle}\ \text{(C-data-Ptr-corrupt)}$$

(C-data-Valid-Ptr-Array)

$$\frac{\delta(\mathsf{b}, \mathsf{n_{id}}) = \mathbf{b}, \mathbf{n_{id}} \qquad \ell * \mathbf{sz}(\tau) = \mathbf{n_l} \qquad (\mathsf{a} - \mathsf{b}) * \mathbf{sz}(\tau) = \mathbf{n_o}}{\mathsf{a}^{(\mathsf{b},\ell,\tau,\mathsf{n_{id}})} \sim_\delta \langle \mathbf{b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle}$$

(C-data-Valid-Ptr-Single)

$$\frac{\mathsf{w} \neq \mathsf{array}\ \_ \qquad \delta(\mathsf{b}, \mathsf{n_{id}}) = \mathbf{b}, \mathbf{n_{id}} \qquad \mathbf{sz}(\mathsf{w}) = \mathbf{n_l} \qquad (\mathsf{a} - \mathsf{b}) = \mathbf{n_o} = 0}{\mathsf{a}^{(\mathsf{b},1,\mathsf{w},\mathsf{n_{id}})} \sim_\delta \langle \mathbf{b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle}$$

Runtime Compilation of Data: $\mathsf{P}, \Gamma \vdash \mathsf{v} : \mathsf{w} \sim_\delta \mathbf{v}$

Runtime Compilation of expressions

(Rt-Value)

$$\frac{\mathsf{v} \sim_\delta \mathbf{v}}{\mathsf{P}, \Gamma \vdash \mathsf{v} : \tau \sim_\delta \langle [], \mathbf{v} \rangle}$$

(Rt-Ptr-Arith)

$$\frac{\begin{array}{c}\mathsf{P}, \Gamma \vdash \mathsf{v_1} : \mathsf{ptr\ array}\ \tau \sim_\delta \langle [], \mathbf{v_1} \rangle \\ \mathsf{P}, \Gamma \vdash \mathsf{v_2} : \mathsf{int} \sim_\delta \langle [], \mathbf{v_2} \rangle \\ \mathbf{n} = \mathbf{sz}(\tau) * \mathbf{v_2}\end{array}}{\mathsf{P}, \Gamma \vdash \mathsf{v_1} \oplus \mathsf{v_2} : \mathsf{array}\ \tau \sim_\delta \langle \mathbf{handle.add}, \mathbf{n}; \mathbf{v_1} \rangle}$$

(Rt-Ptr-Arith-1)

$$\frac{\begin{array}{c}\mathsf{P}, \Gamma \vdash \mathsf{v_1} : \mathsf{ptr\ array}\ \tau \sim_\delta \langle [], \mathbf{v_1} \rangle \\ \mathsf{P}, \Gamma \vdash \mathsf{e_2} : \mathsf{int} \sim_\delta \langle \mathbf{i_2^*}, \mathbf{v_2^*} \rangle \\ \mathbf{s} = \mathbf{sz}(\tau)\end{array}}{\mathsf{P}, \Gamma \vdash \mathsf{v_1} \oplus \mathsf{e_2} : \mathsf{array}\ \tau \sim_\delta \langle \mathbf{i_2^*}; \mathbf{u32.const\ s}; \mathbf{u32.}\times; \mathbf{handle.add}, \mathbf{v_2^*}; \mathbf{v_1} \rangle}$$

(Rt-Ptr-Arith-2)

$$\frac{\begin{array}{c}\mathsf{P}, \Gamma \vdash \mathsf{e_1} : \mathsf{ptr\ array}\ \mathsf{w} \sim_\delta \langle \mathbf{i_1^*}, \mathbf{v_1^*} \rangle \\ [\![\mathsf{P}, \Gamma \vdash \mathsf{e_2} : \mathsf{int}]\!]^{\mathsf{exp}} = \mathbf{i_2^*} \\ \mathbf{s} = \mathbf{sz}(\tau)\end{array}}{\mathsf{P}, \Gamma \vdash \mathsf{e_1} \oplus \mathsf{e_2} : \mathsf{array}\ \tau \sim_\delta \langle \mathbf{i_1^*}; \mathbf{i_2^*}; \mathbf{u32.const\ s}; \mathbf{u32.}\times; \mathbf{handle.add}, \mathbf{v_1^*} \rangle}$$

(Rt-BinOp)

$$\frac{\begin{array}{c}[\![\mathsf{int}]\!] = \tau \\ \mathsf{P}, \Gamma \vdash \mathsf{v_1} : \mathsf{int} \sim_\delta \mathbf{v_1} \\ \mathsf{P}, \Gamma \vdash \mathsf{v_2} : \mathsf{int} \sim_\delta \mathbf{v_2}\end{array}}{\mathsf{P}, \Gamma \vdash \mathsf{v_1} \oplus \mathsf{v_2} : \mathsf{int} \sim_\delta \langle \boldsymbol{\tau}.\otimes, \mathbf{v_2}; \mathbf{v_1} \rangle}$$

(Rt-BinOp-1)

$$\frac{\begin{array}{c}[\![\mathsf{int}]\!] = \tau \\ \mathsf{P}, \Gamma \vdash \mathsf{e_1} : \mathsf{int} \sim_\delta \langle \mathbf{i_1^*}, \mathbf{v_1^*} \rangle \\ [\![\mathsf{P}, \Gamma \vdash \mathsf{e_2} : \mathsf{int}]\!]^{\mathsf{exp}} = \mathbf{i_2^*}\end{array}}{\mathsf{P}, \Gamma \vdash \mathsf{e_1} \oplus \mathsf{e_2} : \mathsf{int} \sim_\delta \langle \mathbf{i_1^*}; \mathbf{i_2^*} \boldsymbol{\tau}.\otimes, \mathbf{v_1^*} \rangle}$$

(Rt-BinOp-2)

$$\frac{\begin{array}{c}[\![\mathsf{int}]\!] = \tau \\ \mathsf{P}, \Gamma \vdash \mathsf{e_2} : \mathsf{int} \sim_\delta \langle \mathbf{i_2^*}, \mathbf{v_2^*} \rangle \\ \mathsf{P}, \Gamma \vdash \mathsf{v_1} : \mathsf{int} \sim_\delta \langle [], \mathbf{v_1} \rangle\end{array}}{\mathsf{P}, \Gamma \vdash \mathsf{v_1} \oplus \mathsf{e_2} : \mathsf{int} \sim_\delta \langle \mathbf{i_2^*} \boldsymbol{\tau}.\otimes, (\mathbf{v_2^*}; \mathbf{v_1}) \rangle}$$

(Rt-Var-Assign)

$$\frac{[\![\tau]\!] = \tau \quad \mathsf{P}, \Gamma \vdash \mathsf{e} : \tau \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle}{\mathsf{P}, \Gamma \vdash \mathsf{x} := \mathsf{e} : \tau \sim_\delta \langle \mathbf{set\ x}, \mathbf{v}^* \rangle}$$

(Rt-Var-Assign-1)

$$\frac{[\![\tau]\!] = \tau \quad \mathsf{P}, \Gamma \vdash \mathsf{v} : \tau \sim_\delta \langle [], \mathbf{v} \rangle}{\mathsf{P}, \Gamma \vdash \mathsf{x} := \mathsf{v} : \tau \sim_\delta \langle \mathbf{set\ x}, \mathbf{v} \rangle}$$

(Rt-Assign)

$$\frac{[\![\mathsf{w}]\!] = \tau \quad \mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} : \mathsf{ptr\ w} \sim_\delta \langle [], \mathbf{a} \rangle \quad \mathsf{P}, \Gamma \vdash \mathsf{v} : \mathsf{w} \sim_\delta \langle [], \mathbf{v} \rangle}{\mathsf{P}, \Gamma \vdash *\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} := \mathsf{v} : \mathsf{int} \sim_\delta \langle \boldsymbol{\tau}\mathbf{.segstore}; \mathbf{u32.const\ 0}, (\mathbf{v}; \mathbf{a}) \rangle}$$

(Rt-Assign-forge)

$$\frac{[\![\mathsf{w}]\!] = \tau \quad \mathsf{P}, \Gamma \vdash \mathsf{a} : \mathsf{ptr\ w} \sim_\delta \langle [], \mathbf{a} \rangle \quad \mathsf{P}, \Gamma \vdash \mathsf{v} : \mathsf{w} \sim_\delta \langle [], \mathbf{v} \rangle}{\mathsf{P}, \Gamma \vdash *\mathsf{a} := \mathsf{v} : \mathsf{int} \sim_\delta \langle \boldsymbol{\tau}\mathbf{.segstore}; \mathbf{u32.const\ 0}, (\mathbf{v}; \mathbf{a}) \rangle}$$

(Rt-Assign-1)

$$\frac{[\![\mathsf{w}]\!] = \tau \quad \mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} : \mathsf{ptr\ w} \sim_\delta \langle [], \mathbf{a} \rangle \quad \mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{w} \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle}{\mathsf{P}, \Gamma \vdash *\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} := \mathsf{e} : \mathsf{int} \sim_\delta \langle \mathbf{i}^*; \boldsymbol{\tau}\mathbf{.segstore}; \mathbf{u32.const\ 0}, \mathbf{v}^*; \mathbf{a} \rangle}$$

(Rt-Assign-forge-1)

$$\frac{[\![\mathsf{w}]\!] = \tau \quad \mathsf{P}, \Gamma \vdash \mathsf{a} : \mathsf{ptr\ w} \sim_\delta \langle [], \mathbf{a} \rangle \quad \mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{w} \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle}{\mathsf{P}, \Gamma \vdash *\mathsf{a} := \mathsf{e} : \mathsf{int} \sim_\delta \langle \mathbf{i}^*; \boldsymbol{\tau}\mathbf{.segstore}; \mathbf{u32.const\ 0}, \mathbf{v}^*; \mathbf{a} \rangle}$$

(Rt-Assign-2)

$$\frac{[\![\tau]\!] = \tau \quad \mathsf{P}, \Gamma \vdash \mathsf{e}_1 : \mathsf{ptr}\,\tau \sim_\delta \langle \mathbf{i}_1^*, \mathbf{v}_1^* \rangle \quad [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_2 : \tau]\!]^{\mathsf{exp}} = \mathbf{i}_2^*}{\mathsf{P}, \Gamma \vdash *\mathsf{e}_1 := \mathsf{e}_2 : \tau \sim_\delta \langle \mathbf{i}_1^*; \mathbf{i}_2^*; \boldsymbol{\tau}\mathbf{.segstore}; \mathbf{u32.const\ 0}, \mathbf{v}_1^* \rangle}$$

(Rt-Arr-Assign)

$$\frac{\begin{array}{c}[\![\tau]\!] = \tau \quad \mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} : \mathsf{ptr\ array}\ \tau \sim_\delta \langle [], \mathbf{a} \rangle \quad \mathsf{P}, \Gamma \vdash \mathsf{n} : \mathsf{int} \sim_\delta \langle [], \mathbf{n} \rangle \\ \mathsf{P}, \Gamma \vdash \mathsf{v} : \tau \sim_\delta \langle [], \mathbf{v} \rangle \quad \mathbf{s} = \mathbf{sz}\ (\tau) \\ \text{TODO: Incorrect order of operands}\end{array}}{\mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}[\mathsf{n}] := \mathsf{v} : \mathsf{int} \sim_\delta \langle \mathbf{u32}.\times; \mathbf{handle.add}; \boldsymbol{\tau}\mathbf{.segstore}; \mathbf{u32.const\ 0}, (\mathbf{s}; \mathbf{v}; \mathbf{a}) \rangle}$$

(Rt-Arr-Assign1)

$$\frac{\begin{array}{c}[\![\tau]\!] = \tau \\ \mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n}_{\mathsf{id}})} : \mathsf{ptr}\,\mathsf{array}\ \tau \sim_\delta \langle [], \mathbf{a} \rangle \\ \mathsf{P}, \Gamma \vdash \mathsf{n} : \mathsf{int} \sim_\delta \langle [], \mathbf{n} \rangle \\ [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_3 : \tau]\!]^{\mathsf{exp}} = i_3^* \\ \mathbf{s} = \mathbf{sz}\,(\tau)\end{array}}{\mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n}_{\mathsf{id}})}[\mathsf{n}] := \mathsf{e}_3 : \mathsf{int} \sim_\delta \langle \mathbf{u32.const\ s; u32.\times; handle.add; i_3^*; \tau.segstore; u32.const\ 0, n; a} \rangle}$$

(Rt-Arr-Assign2)

$$\frac{\begin{array}{c}[\![\tau]\!] = \tau \\ \mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n}_{\mathsf{id}})} : \mathsf{ptr}\,\mathsf{array}\ \tau \sim_\delta \langle [], \mathbf{a} \rangle \\ \mathsf{P}, \Gamma \vdash \mathsf{e}_2 : \mathsf{int} \sim_\delta \langle \mathbf{i_2^*, v^*} \rangle \\ [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_3 : \tau]\!]^{\mathsf{exp}} = i_3^* \\ \mathbf{s} = \mathbf{sz}\,(\tau)\end{array}}{\mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n}_{\mathsf{id}})}[\mathsf{e}_2] := \mathsf{e}_3 : \mathsf{int} \sim_\delta \langle \mathbf{i_2^*; u32.const\ s; u32.\times; handle.add; i_3^*; \tau.segstore; u32.const\ 0, a} \rangle}$$

(Rt-Arr-Assign3)

$$\frac{\begin{array}{c}[\![\tau]\!] = \tau \\ \mathsf{P}, \Gamma \vdash \mathsf{e}_1 : \mathsf{ptr}\,\mathsf{array}\ \tau \sim_\delta \langle \mathbf{i_1^*, v^*} \rangle \\ [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_2 : \mathsf{int}]\!]^{\mathsf{exp}} = i_2^* \\ [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_3 : \tau]\!]^{\mathsf{exp}} = i_3^* \\ \mathbf{s} = \mathbf{sz}\,(\tau)\end{array}}{\mathsf{P}, \Gamma \vdash \mathsf{e}_1[\mathsf{e}_2] := \mathsf{e}_3 : \mathsf{int} \sim_\delta \langle \mathbf{i_1^*; i_2^*; u32.const\ s; u32.\times; handle.add; i_3^*; \tau.segstore; u32.const\ 0, v^*} \rangle}$$

(Rt-Seq)

$$\frac{[\![\mathsf{P}, \Gamma \vdash \mathsf{e}_2 : \mathsf{w}_2]\!]^{\mathsf{exp}} = \mathbf{i_2^*}}{\mathsf{P}, \Gamma \vdash \mathsf{v}_1; \mathsf{e}_2 : \mathsf{w} \sim_\delta \langle \mathbf{i_2^*}, [] \rangle}$$

(Rt-Seq-1)

$$\frac{\begin{array}{c}\mathsf{P}, \Gamma \vdash \mathsf{e}_1 : \mathsf{w} \sim_\delta \langle \mathbf{i_1^*, v^*} \rangle \\ [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_2 : \mathsf{w}_2]\!]^{\mathsf{exp}} = \mathbf{i_2^*}\end{array}}{\mathsf{P}, \Gamma \vdash \mathsf{e}_1; \mathsf{e}_2 : \mathsf{w} \sim_\delta \langle \mathbf{i_1^*; drop; i_2^*, v^*} \rangle}$$

(Rt-Field)

$$\frac{\begin{array}{c}\mathsf{P}, \Gamma \vdash \mathsf{v} : \mathsf{ptr}\,\mathsf{struct}\ \mathsf{s} \sim_\delta \langle [], \mathbf{v} \rangle \\ \mathsf{D}(\mathsf{s}) = \{\mathsf{f}_1 : \tau_1, \ldots \mathsf{f}_\mathsf{k} : \tau_\mathsf{k} \ldots \mathsf{f}_\mathsf{n} : \tau_\mathsf{n}\} \\ (\mathbf{o_1}, \mathbf{o_2}) = (\Sigma_\mathsf{i}^{\mathsf{k}-1}\mathbf{sz}\,(\tau_\mathsf{i}), (\mathbf{sz}\,(\mathsf{struct\ s}) - \mathbf{sz}\,(\tau_\mathsf{k})))\end{array}}{\mathsf{P}, \Gamma \vdash \&\mathsf{v} \to \mathsf{f}_\mathsf{k} : \mathsf{ptr}\,\tau_\mathsf{k} \sim_\delta \langle \mathbf{slice, o_2; o_1; v} \rangle}$$

(Rt-Field-1)

$$\frac{\begin{array}{c}\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{ptr}\,\mathsf{struct}\ \mathsf{s} \sim_\delta \langle \mathbf{i^*, v^*} \rangle \\ \mathsf{D}(\mathsf{s}) = \{\mathsf{f}_1 : \tau_1, \ldots \mathsf{f} : \tau_\mathsf{k} \ldots \mathsf{f}_\mathsf{n} : \tau_\mathsf{n}\} \\ (\mathbf{o_1}, \mathbf{o_2}) = (\Sigma_\mathsf{i}^{\mathsf{k}-1}\mathbf{sz}\,(\tau_\mathsf{i}), (\mathbf{sz}\,(\mathsf{struct\ s}) - \mathbf{sz}\,(\tau_\mathsf{k})))\end{array}}{\mathsf{P}, \Gamma \vdash \&\mathsf{e} \to \mathsf{f}_\mathsf{k} : \mathsf{ptr}\,\tau_\mathsf{k} \sim_\delta \langle \mathbf{i^*; u32.const\ o_1; u32.const\ o_2; slice, v^*} \rangle}$$

(Rt-Malloc-Single)

$$\frac{\mathbf{n'} = \mathbf{sz}\,(\mathsf{w})}{\mathsf{P}, \Gamma \vdash \mathsf{malloc(w)} : \mathsf{ptr\,w} \sim_\delta \langle \mathbf{segalloc, n'} \rangle}$$

(Rt-Malloc-Array-1)

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{n} : \mathsf{int} \sim_\delta \langle [], \mathbf{n} \rangle \quad \mathbf{n}' = \mathbf{n} * \mathbf{sz}\,(\tau)}{\mathsf{P},\Gamma \vdash \mathsf{malloc}(\tau, \mathsf{n}) : \mathsf{array\ w} \sim_\delta \langle \mathbf{segalloc}, \mathbf{n}' \rangle}$$

(Rt-Malloc-Array)

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{e} : \mathsf{int} \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle \quad \mathbf{n}' = \mathbf{sz}\,(\tau)}{\mathsf{P},\Gamma \vdash \mathsf{malloc}(\tau, \mathsf{e}) : \mathsf{array}\ \tau \sim_\delta \langle \mathbf{i}^*; \mathbf{u32.const\ n}'; \mathbf{u32.mult}; \mathbf{segalloc}, \mathbf{v}^* \rangle}$$

(Rt-Deref)

$$\frac{[\![\tau]\!] = \tau \quad \mathsf{P},\Gamma \vdash \mathsf{v} : \mathsf{ptr}\ \tau \sim_\delta \langle [], \mathbf{v} \rangle}{\mathsf{P},\Gamma \vdash *\mathsf{v} : \tau \sim_\delta \langle \boldsymbol{\tau}\mathbf{.segload}, \mathbf{v} \rangle}$$

(Rt-Deref-1)

$$\frac{[\![\tau]\!] = \tau \quad \mathsf{P},\Gamma \vdash \mathsf{e} : \mathsf{ptr}\ \tau \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle}{\mathsf{P},\Gamma \vdash *\mathsf{e} : \tau \sim_\delta \langle \mathbf{i}^*; \boldsymbol{\tau}\mathbf{.segload}, \mathbf{v}^* \rangle}$$

(Rt-Arr-Deref)

$$\frac{[\![\tau]\!] = \tau \quad \mathsf{P},\Gamma \vdash \mathsf{n} : \mathsf{int} \sim_\delta \langle [], \mathbf{n} \rangle \quad \mathsf{P},\Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\tau,\mathsf{n_{id}})} : \mathsf{ptr\ array}\ \tau \sim_\delta \langle [], \mathbf{a} \rangle \quad \mathbf{n}' = \mathbf{sz}\,(\tau) * \mathbf{n}}{\mathsf{P},\Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\tau,\mathsf{n_{id}})}[\mathsf{n}] : \tau \sim_\delta \langle \mathbf{handle.add}; \boldsymbol{\tau}\mathbf{.segload}, \mathbf{n}'; \mathbf{a} \rangle}$$

(Rt-Arr-Deref1)

$$\frac{[\![\tau]\!] = \tau \quad \mathsf{P},\Gamma \vdash \mathsf{e} : \mathsf{int} \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle \quad \mathsf{P},\Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\tau,\mathsf{n_{id}})} : \mathsf{ptr\ array}\ \tau \sim_\delta \langle [], \mathbf{a} \rangle \quad \mathbf{n} = \mathbf{sz}\,(\tau)}{\mathsf{P},\Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\tau,\mathsf{n_{id}})}[\mathsf{e}] : \tau \sim_\delta \langle \mathbf{i}^*; \mathbf{u32.const\ n}; \mathbf{u32.mult}; \mathbf{handle.add}; \boldsymbol{\tau}\mathbf{.segload}, \mathbf{v}^*; \mathbf{a} \rangle}$$

(Rt-Arr-Deref2)

$$\frac{[\![\tau]\!] = \tau \quad \mathsf{P},\Gamma \vdash \mathsf{e}_1 : \mathsf{int} \sim_\delta \langle \mathbf{i}_1^*, \mathbf{v}^* \rangle \quad \mathsf{P},\Gamma \vdash \mathsf{e}_2 : \mathsf{ptr\ array}\ \tau \sim_\delta i_2^* \quad \mathbf{n} = \mathbf{sz}\,(\tau)}{\mathsf{P},\Gamma \vdash \mathsf{e}_1[\mathsf{e}_2] : \tau \sim_\delta \langle \mathbf{i}_1^*; \mathbf{i}_2^*; \mathbf{u32.const\ n}; \mathbf{u32.mult}; \mathbf{handle.add}; \boldsymbol{\tau}\mathbf{.segload}, \mathbf{v}^* \rangle}$$

(Rt-Free)

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} : \mathsf{ptr\ w} \sim_\delta \langle [], \mathbf{v} \rangle}{\mathsf{P},\Gamma \vdash \mathsf{free}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}) : \mathsf{int} \sim_\delta \langle \mathbf{segfree}; \mathbf{u32.const\ 0}, \mathbf{v} \rangle}$$

(Rt-Free-Forge)

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{a} : \mathsf{ptr\ w} \sim_\delta \langle [], \mathbf{v} \rangle}{\mathsf{P},\Gamma \vdash \mathsf{free}(\mathsf{a}) : \mathsf{int} \sim_\delta \langle \mathbf{segfree}; \mathbf{u32.const\ 0}, \mathbf{v} \rangle}$$

(Rt-Free-1)

$$\frac{\mathsf{P},\Gamma \vdash \mathsf{e} : \mathsf{ptr\ w} \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle}{\mathsf{P},\Gamma \vdash \mathsf{free}(\mathsf{e}) : \mathsf{int} \sim_\delta \langle \mathbf{i}^*; \mathbf{segfree}; \mathbf{u32.const\ 0}, \mathbf{v}^* \rangle}$$

(Rt-Branch)

$$\frac{\mathsf{P}, \Gamma \vdash \mathsf{v} : \mathsf{int} \sim_\delta \langle [], \mathbf{v} \rangle \quad [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_1 : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{i}_1^* \quad [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_2 : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{i}_2^*}{\mathsf{P}, \Gamma \vdash \mathsf{if\ v\ then\ e_1\ else\ e_2} : \mathsf{w} \sim_\delta \langle \mathbf{branch\ i_1^*\ i_2^*}, \mathbf{v} \rangle}$$

(Rt-Branch-1)

$$\frac{\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{int} \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle \quad [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_1 : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{i}_1^* \quad [\![\mathsf{P}, \Gamma \vdash \mathsf{e}_2 : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{i}_2^*}{\mathsf{P}, \Gamma \vdash \mathsf{if\ e\ then\ e_1\ else\ e_2} : \mathsf{w} \sim_\delta \langle \mathbf{i}^*; \mathbf{branch\ i_1^*\ i_2^*}, \mathbf{v}^* \rangle}$$

(Rt-Call)

$$\frac{F_{\mathsf{map}}(\mathsf{g}) = \mathbf{n_g} \quad V_{\mathsf{map}}(\mathsf{x}) = \mathbf{n_x} \quad \mathsf{flookup}(\mathsf{P}, \mathsf{g}) = \mathsf{w}_2\ \mathsf{g}(\mathsf{x} : \mathsf{w}_1) \mapsto \mathsf{e_g} \quad \mathsf{P}, \Gamma \vdash \mathsf{v}_1 : \mathsf{w}_1 \sim_\delta \langle [], \mathbf{v_1} \rangle}{\mathsf{P}, \Gamma \vdash \mathsf{x} := \mathsf{call\ g\ (v_1)} : \mathsf{w}' \sim_\delta \langle \mathbf{call\ n_g; set\ n_x}, \mathbf{v_1} \rangle}$$

(Rt-Call-1)

$$\frac{F_{\mathsf{map}}(\mathsf{g}) = \mathbf{n_g} \quad V_{\mathsf{map}}(\mathsf{x}) = \mathbf{n_x} \quad \mathsf{flookup}(\mathsf{P}, \mathsf{g}) = \mathsf{w}_2\ \mathsf{g}(\mathsf{x} : \mathsf{w}_1) \mapsto \mathsf{e_g} \quad \mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{w}_1 \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle}{\mathsf{P}, \Gamma \vdash \mathsf{x} := \mathsf{call\ g\ (e)} : \mathsf{w}' \sim_\delta \langle \mathbf{i}^*; \mathbf{call\ n_g; set\ n_x}, \mathbf{v}^* \rangle}$$

Runtime Compilation of Expressions: $\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{w} \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle$

4.6 Configuration Compilation

(Rt-Config)

$$\frac{\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{w} \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle \quad \langle \mathsf{P}, \mathsf{K}, \mathsf{E}^* \rangle \sim_\delta \mathbf{S}^* \quad \mathsf{H} \sim_\delta \mathbf{\Sigma.T} \quad \mathsf{A} \sim_\delta \mathbf{\Sigma.A} \quad \theta_\mathsf{f} \sim_\delta \theta_\mathbf{f}}{\mathsf{P}, (\theta_\mathsf{f}; \mathsf{K}), \mathsf{H}, \mathsf{A} \triangleright (\mathsf{e_f}, \mathsf{E}^*) \sim_\delta \langle \mathbf{\Sigma}, (\theta_\mathbf{f}, \mathbf{i}^*, \mathbf{v}^*) : \mathbf{S}^* \rangle}$$

(Compile-Config)

$$\frac{[\![\mathsf{P}, \Gamma_\mathsf{f} \vdash \mathsf{e_f} : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{i}^* \quad [\![\mathsf{P}, \Delta]\!]^{\mathsf{glob}} = \mathbf{\Delta} \quad \langle \mathsf{P}, \mathsf{K}, \mathsf{E}^* \rangle \sim_\delta \mathbf{S}^* \quad \mathsf{H} \sim_\delta \mathbf{\Sigma.T} \quad \mathsf{A} \sim_\delta \mathbf{\Sigma.A} \quad \theta_\mathsf{f} \sim_\delta \theta_\mathbf{f} \quad \mathbf{\Delta} \vdash \langle \mathbf{\Sigma}, (\theta, \mathbf{i}^*, \mathbf{v}^*) : \mathbf{S}^* \rangle}{[\![\mathsf{P}, (\theta_\mathsf{f}; \mathsf{K}), \mathsf{H}, \mathsf{A} \triangleright (\mathsf{e_f}, \mathsf{E}^*), \delta]\!] = \langle \mathbf{\Sigma}, (\theta_\mathbf{f}, \mathbf{i}^*, \mathbf{v}^*) : \mathbf{S}^* \rangle}$$

4.7 Type Preserving Compilation

We prove that the compilation preserves well-typedness.

Theorem 2 (Type-Preservation of a Compiled Program)**.** If

- $\vdash \mathsf{P} : \mathsf{wt}$
- $[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M}$

- $[\![\mathsf{P}, \Delta]\!]^{\mathsf{glob}} = \mathbf{\Delta}$

then $\mathbf{\Delta} \vdash \mathbf{M}$.

Proof. From rule (Program), we have

$$\forall k, \mathsf{P.F}[k] = \mathsf{f_k}, [\![\mathsf{w_{rk}}\ \mathsf{f_k}(\mathsf{x} : \mathsf{w_{ak}}) \mapsto ((\mathsf{y_{jk}} : \mathsf{w_{jk}})^*, \mathsf{e_k})]\!]^{\mathsf{fun}} = \mathbf{f_k} \tag{45}$$

$$|\mathbf{s}| = \mathtt{sz}\,(\mathsf{P.n_{hs}}) \tag{46}$$

$$\mathbf{M} = \{\mathbf{heap\ 0}, \mathbf{segment\ s}, \mathbf{funcs\ f_0 f_1 \ldots f_n}\} \tag{47}$$

Applying Lemma 14 (Type-Preservation of a Compiled Function) to the (45), we have all the required premises for rule (Module).

□

Lemma 14 (Type-Preservation of a Compiled Function)**.** If

- $\mathsf{P} \vdash \tau_\mathsf{r}\ \mathsf{f}(\mathsf{x} : \tau_\mathsf{a}) \mapsto ((\mathsf{y_j} : \tau_\mathsf{j})^*, \mathsf{e_f}) : \tau_\mathsf{a} \to \tau_\mathsf{r}$
- $[\![\tau_\mathsf{a}]\!] = \tau_\mathbf{a}$
- $[\![\tau_\mathsf{r}]\!] = \tau_\mathbf{r}$
- $[\![\tau_\mathsf{j}]\!] = \tau_\mathbf{j}$
- $[\![\tau_\mathsf{r}\ \mathsf{f}(\mathsf{x} : \tau_\mathsf{a}) \mapsto ((\mathsf{y_j} : \tau_\mathsf{j})^*, \mathsf{e})]\!]^{\mathsf{fun}} = \langle \mathbf{var}\ \tau_\mathbf{a} \tau^\mathbf{v}, \mathbf{i}^* \rangle$

then $\{\mathbf{funcs}\ \rho^*, \mathbf{locals} = \tau_\mathbf{a}\tau_\mathbf{1}\tau_\mathbf{2}\ldots, \mathbf{return} = \tau_\mathbf{r}\} \vdash \{\mathbf{var}\ \tau_\mathbf{a}\tau^\mathbf{v}, \mathbf{body}\ \mathbf{i}^*\} : \tau^\mathbf{a} \to \tau_\mathbf{r}$.

Proof. We have $\Gamma_\mathsf{f} = (\tau_\mathsf{r}, \{\mathsf{x} : \tau_\mathsf{a}, \mathsf{y_1} : \tau_1, \mathsf{y_2} : \tau_2, \ldots\})$. By lifting the compilation of types to typing environments, we have $[\![\mathsf{P}, \Gamma_\mathsf{f}]\!]^{\mathsf{env}} = \mathbf{\Gamma_f}$ where $\mathbf{\Gamma_f} = \{\mathbf{funcs}\ \rho^*, \mathbf{locals}\ \tau_\mathbf{a}\tau_\mathbf{1}\tau_\mathbf{2}\ldots, \mathbf{return}\ \tau_\mathbf{r}\}$ Applying Corollary 1 (Type-Preservation of a Compiled Expression) we have the proof. □

Lemma 15 (Type-Preservation of a Compiled Expression)**.** If

- $\mathsf{P}, \Gamma_\mathsf{f} \vdash \mathsf{e} : \mathsf{w}$,
- $[\![\mathsf{P}, \Gamma_\mathsf{f}]\!]^{\mathsf{env}} = \mathbf{\Gamma_f}$
- $[\![\mathsf{w}]\!] = \tau$
- $[\![\mathsf{P}, \Gamma_\mathsf{f} \vdash \mathsf{e} : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{i}^*$,

then for all τ'^*, $\mathbf{\Gamma_f} \vdash \mathbf{i}^* : \tau'^* \to \tau\tau'^*$.

Proof. Proof is by induction on the compilation judgment. □

Corollary 1 (Type-Preservation of a Compiled Expression)**.** If

- $\mathsf{P}, \Gamma_\mathsf{f} \vdash \mathsf{e} : \mathsf{w}$,
- $[\![\mathsf{P}, \Gamma_\mathsf{f}]\!]^{\mathsf{env}} = \mathbf{\Gamma_f}$

- $[\![\mathsf{w}]\!] = \tau$
- $[\![\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{i}^*$,

we have $\mathbf{\Gamma_f} \vdash \mathbf{i} : [] \to \tau$.

Proof. Instantiate τ^* with $[]$ in Lemma 15. □

4.8 Functional Correctness

We prove that the compilation preserves functional correctness.

4.8.1 Split MS Monitor

Before we prove functional correctness, we first need to establish that when the source takes multiple steps and results in MS monitor taking several steps, then they can be split. We will use this result to prove that when the source configuration (in related source and target configurations) takes multiple steps that are memory safe, then the compiled configuration takes one or more steps safely, and moreover the output source and target configurations will be in the same relation.

Lemma 16 (MS Monitor Split)**.** If

1. $\Omega \xRightarrow{\alpha} \Omega'$ and
2. $\Omega' \xRightarrow{\overline{\alpha}^*} \Omega''$ and
3. $\delta \subseteq \delta''$ and
4. $T =_\delta \Omega.\mathsf{A}$ and
5. $\alpha \cdot \overline{\alpha} =_{\delta''} \overline{\alpha}$ and
6. $\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T''$ and
7. $T'' =_{\delta''} \Omega''.\mathsf{A}$

then $\exists \delta', \overline{\alpha}_1, \overline{\alpha}_2$ and T',

(a) $\overline{\alpha} = \overline{\alpha}_1 \cdot \overline{\alpha}_2$

(b)

$\delta \subseteq \delta'$
$\delta \subseteq \delta'$
$\alpha =_{\delta'} \overline{\alpha}_1$
$\vdash T \overset{\overline{\alpha}_1}{\rightsquigarrow} T'$
$T' =_{\delta'} \Omega'.\mathsf{A}$

(c)

$$\delta' \subseteq \delta''$$
$$\overline{\alpha} =_{\delta''} \overline{\alpha}_2$$
$$\vdash T' \overset{\overline{\alpha}_2}{\rightsquigarrow} T''$$
$$T'' =_{\delta''} \Omega''.\mathsf{A}$$

Proof. Given:

$$\Omega \overset{\alpha}{\Longrightarrow} \Omega' \tag{48}$$
$$\Omega' \overset{\overline{\alpha}}{\Longrightarrow}{}^{*} \Omega'' \tag{49}$$
$$\delta \subseteq \delta'' \tag{50}$$
$$T =_{\delta} \Omega.\mathsf{A} \tag{51}$$
$$\alpha \cdot \overline{\alpha} =_{\delta''} \overline{\alpha} \tag{52}$$
$$\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T'' \tag{53}$$
$$T'' =_{\delta''} \Omega''.\mathsf{A} \tag{54}$$

We have to prove

$$\delta \subseteq \delta' \tag{55}$$
$$\alpha =_{\delta'} \overline{\alpha}_1 \tag{56}$$
$$\vdash T \overset{\overline{\alpha}_1}{\rightsquigarrow} T' \tag{57}$$
$$T' =_{\delta'} \Omega'.\mathsf{A} \tag{58}$$
$$\delta' \subseteq \delta'' \tag{59}$$
$$\overline{\alpha} =_{\delta''} \overline{\alpha}_2 \tag{60}$$
$$\vdash T' \overset{\overline{\alpha}_2}{\rightsquigarrow} T'' \tag{61}$$
$$T'' =_{\delta''} \Omega''.\mathsf{A} \tag{62}$$

If $\overline{\alpha}$ is ϵ, then $\alpha \cdot \overline{\alpha}$ is ϵ as well (Lemma 3 (Empty Source Trace)).

Let $\overline{\alpha} \neq \epsilon$. Then, applying Lemma 4 (Splitting Source Trace Equivalence), we have $\overline{\alpha} = \overline{\alpha}_1 \cdot \overline{\alpha}_2$ for some $\overline{\alpha}_1$ and $\overline{\alpha}_2$.

Inverting (52) using (Tr-Concatenate), we have

$$\alpha =_{\delta''} \overline{\alpha}_1 \tag{63}$$
$$\overline{\alpha} =_{\delta''} \overline{\alpha}_2 \tag{64}$$

Case analysis on (63)

Case (Tr-Read-Authentic): Given:

$$\mathsf{read}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}, \mathsf{v}) =_{\delta''} \mathtt{read}(a^c) \cdot \mathtt{read}((a+1)^c) \ldots \mathtt{read}((a+n-1)^c)$$

Thus $\overline{\alpha}_1 = \mathtt{read}(a^c) \cdot \mathtt{read}((a+1)^c) \ldots \mathtt{read}((a+n-1)^c)$. From (Cons) and (MS-Read), we have $\vdash T \overset{\overline{\alpha}_1}{\rightsquigarrow} T$. Thus $T = T'$. From Lemma 1 (Split Abstract MS Monitor), we have

$$\vdash T' \overset{\overline{\alpha}_2}{\rightsquigarrow} T'' \tag{65}$$

Since $T = T'$, it follows that $\Omega.\mathsf{A} = \Omega'.\mathsf{A}$. Let $\delta' = \delta''$. Thus

$$T' =_{\delta'} \Omega'.\mathsf{A} \tag{66}$$

Thus, we have proved (67) to (74).

Case (Tr-Write-Authentic): Similar to the above case.

Case (Tr-SAlloc): Given:

$$\mathsf{salloc}(\mathsf{a}, \mathsf{n}, \mathsf{w}) =_{\delta''} \mathtt{alloc}\,(n, a^c)$$

(We prove for $\mathsf{w} = \tau$. The proof for the case $\mathsf{w} = \mathsf{struct\ s}$ is analogous.)

We have $\overline{\alpha}_1 = \mathtt{alloc}(n, a^c, \phi)$. We need to prove:

$$\delta \subseteq \delta' \tag{67}$$
$$\alpha =_{\delta'} \overline{\alpha}_1 \tag{68}$$
$$\vdash T \overset{\overline{\alpha}_1}{\rightsquigarrow} T' \tag{69}$$
$$T' =_{\delta'} \Omega'.\mathsf{A} \tag{70}$$
$$\delta' \subseteq \delta'' \tag{71}$$
$$\overline{\alpha} =_{\delta''} \overline{\alpha}_2 \tag{72}$$
$$\vdash T' \overset{\overline{\alpha}_2}{\rightsquigarrow} T'' \tag{73}$$
$$T'' =_{\delta''} \Omega''.\mathsf{A} \tag{74}$$

We first prove $\vdash T \overset{\overline{\alpha}_1}{\rightsquigarrow} T'$. We are already given $\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T''$. That is, $\vdash T \overset{\mathtt{alloc}(n,a^c,\phi)\cdot\overline{\alpha}_2}{\rightsquigarrow} T''$. From rule (Cons), we have $\vdash T \overset{\mathtt{alloc}(n,a^c,\phi)}{\rightsquigarrow} T'$. This implies $\vdash T \overset{\overline{\alpha}_1}{\rightsquigarrow} T'$.

From (Tr-SAlloc), we have that $\delta''(\mathsf{a}, \mathsf{n_{id}}) = a^{(c,0)}$. Inverting (MS-Alloc), we have that c is fresh. Let $T' = T[a+i \mapsto A(c, \phi(i)) \mid i \in \{0..n-1\}]$ and $T'.Y = T.Y \cdot \mathtt{alloc}(n, a^c, \phi)$. We are already given $T =_\delta \Omega.\mathsf{A}$. Let $\delta' = \delta''$. Thus from Definition 4 (Segment Agreement $T =_\delta \mathsf{A}$), we just need

to prove that the newly allocated buffer agrees with the abstract memory. From (48) and (Src-Alloc), we have $\Omega'.\mathsf{A}.\mathsf{s}^*_{\mathsf{F}} = \mathsf{s}^*_1 \cdot (\mathsf{a} + \mathsf{n}, \ell' - \ell, \mathsf{w}', \mathsf{n}'_{\mathsf{id}}) \cdot \mathsf{s}^*_2$ and $\Omega'.\mathsf{A}.\mathsf{s}^*_{\mathsf{A}} = (\mathsf{a}, \ell, \mathsf{w}, \mathsf{n}_{\mathsf{id}}) \cdot \mathsf{s}^*_{\mathsf{U}}$ where $\mathsf{fresh}(\mathsf{n}_{\mathsf{id}})$. We thus have $T' =_{\delta'} \Omega'.\mathsf{A}$.

Applying Lemma 1 (Split Abstract MS Monitor) to $\vdash T \overset{\overline{\alpha}_1 \cdot \overline{\alpha}_2}{\rightsquigarrow} T''$ and $\vdash T \overset{\overline{\alpha}_1}{\rightsquigarrow} T'$, we have

$$\vdash T' \overset{\overline{\alpha}_2}{\rightsquigarrow} T'' \tag{75}$$

Hence the proof.

Case (Tr-SFree): Given:

$$\mathsf{free}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n}_{\mathsf{id}})}) =_\delta \texttt{sfree}\,(a^c)$$

We have $\overline{\alpha}_1 = \texttt{sfree}\,(a^c)$. We first need to prove $\vdash T \overset{\overline{\alpha}_1}{\rightsquigarrow} T'$ and then apply Lemma 1 (Split Abstract MS Monitor) to $\vdash T \overset{\overline{\alpha}_1 \cdot \overline{\alpha}_2}{\rightsquigarrow} T''$ and $\vdash T \overset{\overline{\alpha}_1}{\rightsquigarrow} T'$. Since we are already given $\vdash T' \overset{\overline{\alpha}_1 \cdot \overline{\alpha}_2}{\rightsquigarrow} T''$ (from (53)), inverting (Cons), we have $\vdash T \overset{\overline{\alpha}_1}{\rightsquigarrow} T'$ and so $\vdash T \overset{\overline{\alpha}_1}{\rightsquigarrow} T'$ (trivially from (Empty)). The remaining proof is along the lines of the above alloc case.

□

4.8.2 Functional Correctness—Multiple Steps

Lemma 17 (Monotone δ)**.** If $\Omega \sim_\delta \mathbf{\Omega}$ and $\delta \subseteq \delta'$ then $\Omega \sim_{\delta'} \mathbf{\Omega}$.

Proof. Follows from (Rt-Config). □

Lemma 18 (Functional Correctness—Multiple Steps)**.** If

1. $\Omega \overset{\overline{\alpha}}{\Longrightarrow^*} \Omega'$ and
2. $[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M}$ and
3. $[\![\mathsf{P}, \Delta]\!]^{\mathsf{glob}} = \mathbf{\Delta}$ and
4. $\Omega \sim_\delta \mathbf{\Omega}$ and
5. $\mathbf{\Delta} \vdash \mathbf{\Omega}$ and
6. $\vdash \mathbf{isValid}\ \mathbf{\Omega}$ and
7. $T =_\delta \Omega.\mathsf{A}$
8. $\delta \subseteq \delta'$ and

9. $\overline{\alpha} =_{\delta'} \overline{\alpha}$ and
10. $\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T'$ and
11. $T' =_{\delta'} \Omega'.\mathsf{A}$

then

(i) $\exists \delta' \supseteq \delta$ and $\boldsymbol{\Omega}', \Omega' \sim_{\delta'} \boldsymbol{\Omega}'$

(ii) $\exists \overline{\alpha}, \mathbf{M.funcs} \vdash \boldsymbol{\Omega} \xRightarrow{\overline{\alpha}} \boldsymbol{\Omega}'$ and

(iii) $\overline{\alpha} =_{\delta'} \overline{\alpha}$ and

(iv) $\vdash \mathbf{isValid}\ \boldsymbol{\Omega}'$

Proof. Proof is by induction on the relation $\Omega \xRightarrow{\overline{\alpha}^*} \Omega'$.

(S-Base): Trivial.

(S-Steps): We have

$$\Omega \xRightarrow{\alpha} \Omega' \tag{76}$$
$$\Omega' \xRightarrow{\overline{\alpha}^*} \Omega'' \tag{77}$$

We are also given:

$$[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M} \tag{78}$$
$$[\![\mathsf{P}, \Delta]\!]^{\mathsf{glob}} = \boldsymbol{\Delta} \tag{79}$$
$$\Omega \sim_{\delta} \boldsymbol{\Omega}'' \tag{80}$$
$$\boldsymbol{\Delta} \vdash \boldsymbol{\Omega} \tag{81}$$
$$\vdash \mathbf{isValid}\ \boldsymbol{\Omega} \tag{82}$$
$$T =_{\delta} \Omega.\mathsf{A} \tag{83}$$
$$\delta \subseteq \delta'' \tag{84}$$
$$\overline{\alpha} =_{\delta''} \overline{\alpha} \tag{85}$$
$$\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T'' \tag{86}$$
$$T'' =_{\delta'} \Omega''.\mathsf{A} \tag{87}$$

Applying Lemma 16 (MS Monitor Split) to (76), (77) and (83) to (87), gives:

$$\overline{\alpha} = \overline{\alpha}_1 \cdot \overline{\alpha}_2 \tag{88}$$
$$\delta \subseteq \delta' \tag{89}$$

$$\alpha =_{\delta'} \overline{\alpha}_1 \tag{90}$$

$$\vdash T \overset{\overline{\alpha}_1}{\rightsquigarrow} T' \tag{91}$$

$$T' =_{\delta'} \Omega'.\mathsf{A} \tag{92}$$

$$\delta' \subseteq \delta'' \tag{93}$$

$$\overline{\alpha} =_{\delta''} \overline{\alpha}_2 \tag{94}$$

$$\vdash T' \overset{\overline{\alpha}_2}{\rightsquigarrow} T'' \tag{95}$$

$$T'' =_{\delta''} \Omega''.\mathsf{A} \tag{96}$$

That gives us necessary premises to invoke Lemma 21 (Functional Correctness—Single-step) on (76), (78) to (82) and (89) to (92). We get

$$\exists \delta' \supseteq \delta \text{ and } \boldsymbol{\Omega}', \Omega' \sim_{\delta'} \boldsymbol{\Omega}' \tag{97}$$

$$\mathbf{M.funcs} \vdash \boldsymbol{\Omega} \overset{\alpha}{\Longrightarrow} \boldsymbol{\Omega}' \tag{98}$$

$$\alpha =_{\delta'} \alpha \tag{99}$$

$$\vdash \mathbf{isValid}\ \boldsymbol{\Omega}' \tag{100}$$

Applying Lemma 7 (Type and Validity Preservation for Configurations-Trace Semantics) to (76), gives

$$\boldsymbol{\Delta} \vdash \boldsymbol{\Omega}'$$

Above combined with (78), (79), (97) to (100), (93) to (96) gives us necessary premises to invoke I.H on (77). Applying I.H we have the proof.

□

4.8.3 Evaluation Context Lemmas

Lemma 19 (Context Composition)**.** If

1. $\mathsf{e} = \mathsf{E}[\mathsf{e}']$
2. $\mathsf{P}, \Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{w} \sim_{\delta} \langle \mathbf{i}^*_{\mathbf{E}[\cdot]}, \mathbf{v}^*_{\mathbf{E}[\cdot]} \rangle$
3. $\mathsf{P}, \Gamma \vdash \mathsf{e}' : \mathsf{w}' \sim_{\delta} \langle \mathbf{i}'^*, \mathbf{v}'^* \rangle$

then $\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{w} \sim_{\delta} \langle \mathbf{i}'^*; \mathbf{i}^*_{\mathbf{E}[\cdot]}, \mathbf{v}'^*; \mathbf{v}^*_{\mathbf{E}[\cdot]} \rangle$.

Proof. Proof is by induction on the compilation of evaluation context, that is, $\mathsf{P}, \Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{w} \sim_{\delta} \langle \mathbf{i}^*_{\mathbf{E}[\cdot]}, \mathbf{v}^*_{\mathbf{E}[\cdot]} \rangle$ □

Lemma 20 (Context Decomposition)**.** If

1. $\mathsf{e} = \mathsf{E}[\mathsf{e}']$

2. $\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{w} \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle$

then there exists $\mathbf{i}'^*$, $\mathbf{i}^*_{\mathbf{E}[\cdot]}$, $\mathbf{v}'^*$, $\mathbf{v}^*_{\mathbf{E}[\cdot]}$, such that:

(i) $\mathbf{i}^* = \mathbf{i}'^* \; ; \; \mathbf{i}^*_{\mathbf{E}[\cdot]}$

(ii) $\mathbf{v}^* = \mathbf{v}'^* \; ; \; \mathbf{v}^*_{\mathbf{E}[\cdot]}$

(iii) $\mathsf{P}, \Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{w} \sim_\delta \langle \mathbf{i}^*_{\mathbf{E}[\cdot]}, \mathbf{v}^*_{\mathbf{E}[\cdot]} \rangle$

(iv) $\mathsf{P}, \Gamma \vdash \mathsf{e}' : \mathsf{w}' \sim_\delta \langle \mathbf{i}'^*, \mathbf{v}'^* \rangle$

Proof. Proof is by induction on $\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{w} \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle$. □

4.8.4 Functional Correctness—Single-step

Lemma 21 (Functional Correctness—Single-step)**.** If

1. $\Omega \overset{\alpha}{\Longrightarrow} \Omega'$ and
2. $[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M}$ and
3. $[\![\mathsf{P}, \Delta]\!]^{\mathsf{glob}} = \mathbf{\Delta}$ and
4. $\Omega \sim_\delta \mathbf{\Omega}$ and
5. $\mathbf{\Delta} \vdash \mathbf{\Omega}$ and
6. $\vdash \mathbf{isValid}\ \mathbf{\Omega}$ and
7. $T =_\delta \Omega.\mathsf{A}$
8. $\delta \subseteq \delta'$ and
9. $\alpha =_{\delta'} \overline{\alpha}$ and
10. $T \xrightarrow{\overline{\alpha}} T'$ and
11. $T' =_{\delta'} \Omega'.\mathsf{A}$

then

(i) $\exists \delta' \supseteq \delta$ and $\mathbf{\Omega}'$, $\Omega' \sim_{\delta'} \mathbf{\Omega}'$ and

(ii) $\exists \alpha$, $\mathbf{M.funcs} \vdash \mathbf{\Omega} \overset{\alpha}{\Longrightarrow} \mathbf{\Omega}'$ and

(iii) $\alpha =_{\delta'} \alpha$ and

(iv) $\vdash \mathbf{isValid}\ \mathbf{\Omega}'$

Proof. Proof by induction on the $\Omega \overset{\alpha}{\Longrightarrow} \Omega'$. Interesting cases are those which emit non-empty events and manipulate pointers.

Case (S-Step): We are given:

$$\mathsf{P}, (\theta; \mathsf{K}), \mathsf{H}, \mathsf{A} \triangleright (\mathsf{E}[\mathsf{e}], \mathsf{E}^*) \xRightarrow{\alpha} \mathsf{P}, (\theta', \mathsf{K}), \mathsf{H}', \mathsf{A}' \triangleright (\mathsf{E}[\mathsf{e}'], \mathsf{E}^*)$$

First, we invert $\Omega \sim_\delta \mathbf{\Omega} = \langle \mathbf{\Sigma}, (\theta, \mathbf{i}^*, \mathbf{v}^*) : \mathbf{S}^* \rangle$ via rule (Rt-Config), and obtain $\mathsf{P}, \Gamma \vdash \mathsf{E}[\mathsf{e}] : \mathsf{w} \sim_\delta \langle \mathbf{i}^*, \mathbf{v}^* \rangle$. Then, we apply Lemma 20 to the above judgment and decompose the target instruction list $\mathbf{i}^*$ and value stack $\mathbf{v}^*$ in the parts that are related to evaluation context E and to the redex e respectively, i.e., there exists $\mathbf{i}'^*$, $\mathbf{i}^*_{\mathbf{E}[\cdot]}$, $\mathbf{v}'^*$, $\mathbf{v}^*_{\mathbf{E}[\cdot]}$, such that:

$$\mathbf{i}^* = \mathbf{i}'^*; \mathbf{i}^*_{\mathbf{E}[\cdot]} \tag{101}$$
$$\mathbf{v}^* = \mathbf{v}'^*; \mathbf{v}^*_{\mathbf{E}[\cdot]} \tag{102}$$
$$\mathsf{P}, \Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{w} \sim_\delta \langle \mathbf{i}^*_{\mathbf{E}[\cdot]}, \mathbf{v}^*_{\mathbf{E}[\cdot]} \rangle \tag{103}$$
$$\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{w}' \sim_\delta \langle \mathbf{i}'^*, \mathbf{v}'^* \rangle \tag{104}$$

Next, we invert the single source step (rule (S-Step)) and obtain the primitive reduction:

$$\mathsf{M} \vdash \mathsf{P}, \theta, \mathsf{H}, \mathsf{A} \triangleright \mathsf{e} \xrightarrow{\alpha}\!\!\!\!\rightarrow \mathsf{P}, \theta', \mathsf{H}', \mathsf{A}' \triangleright \mathsf{e}' \tag{105}$$

Let $\Omega_1 = \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright \mathsf{e}$, $\Omega'_1 = \mathsf{P}, \mathsf{K}', \mathsf{H}', \mathsf{A}' \triangleright \mathsf{e}'$, and $\mathbf{\Omega_1} = \langle \mathbf{\Sigma}, (\theta, \mathbf{i}'^*, \mathbf{v}'^*) \rangle$. Since $\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{w}' \sim_\delta \langle \mathbf{i}'^*, \mathbf{v}'^* \rangle$, we have that $\Omega_1 \sim_\delta \mathbf{\Omega_1}$ (rule (Rt-Config)), thus we simulate the primitive reduction via Lemma 22 (Functional Correctness—Primitive Step), i.e., there exists an extended bijection $\delta' \supseteq \delta$, an event α, such that $\alpha =_{\delta'} \alpha$, a target configuration $\mathbf{\Omega'_1} = \langle \mathbf{\Sigma}', (\theta', \mathbf{i}''^*, \mathbf{v}''^*) \rangle$, such that $\Omega'_1 \sim_{\delta'} \mathbf{\Omega'_1}$ and $\mathbf{M.funcs} \vdash \mathbf{\Omega_1} \xRightarrow{\alpha} \mathbf{\Omega'_1}$. Then, we plug the result of the primitive reduction in $\mathbf{\Omega'_1}$ into the hole of the target context identified above in $\mathbf{\Omega}$ (103), i.e., let $\mathbf{\Omega}' = \langle \mathbf{\Sigma}', (\theta', \mathbf{i}''^*; \mathbf{i}^*_{\mathbf{E}[\cdot]}, \mathbf{v}''^*; \mathbf{v}^*_{\mathbf{E}[\cdot]}) \rangle$. Since $\Omega'_1 \sim_{\delta'} \mathbf{\Omega'_1}$, we have that $\mathsf{P}, \Gamma \vdash \mathsf{e}' : \mathsf{w}' \sim_{\delta'} \langle \mathbf{i}''^*, \mathbf{v}''^* \rangle$ (rule (Rt-Config)), thus, we can apply Lemma 19 to that and $\mathsf{P}, \Gamma \vdash \mathsf{E} : \mathsf{w}' \Rightarrow \mathsf{w} \sim_{\delta'} \langle \mathbf{i}^*_{\mathbf{E}[\cdot]}, \mathbf{v}^*_{\mathbf{E}[\cdot]} \rangle$ (103 lifted by monotonicity of $\delta' \supseteq \delta$), and obtain $\mathsf{P}, \Gamma \vdash \mathsf{E}[\mathsf{e}'] : \mathsf{w} \sim_{\delta'} \langle \mathbf{i}''^*; \mathbf{i}^*_{\mathbf{E}[\cdot]}, \mathbf{v}''^*; \mathbf{v}^*_{\mathbf{E}[\cdot]} \rangle$, from which have $\Omega' \sim_{\delta'} \mathbf{\Omega}'$. Finally, notice that the target reduction $\mathbf{M.funcs} \vdash \mathbf{\Omega_1} \xRightarrow{\alpha} \mathbf{\Omega'_1}$ is independent from the target context (just like in the source language), thus the same reduction can be lifted into the original configuration, giving us $\mathbf{M.funcs} \vdash \mathbf{\Omega} \xRightarrow{\alpha} \mathbf{\Omega}'$, as desired.

Case (S-Call): Given $\mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright (\mathsf{E}_1[\mathsf{x} := \mathsf{call}\ \mathsf{g}\ (\mathsf{n}')], \mathsf{E}^*) \xRightarrow{\epsilon} \mathsf{P}, \mathsf{K}', \mathsf{H}, \mathsf{A} \triangleright (\mathsf{e}_\mathsf{b}, \mathsf{E}_1[\mathsf{x} := [\cdot]] : \mathsf{E}^*)$. Inverting rule (S-Call), we have

$$(\mathsf{w}\ \mathsf{g}(\mathsf{x}' : \mathsf{w}')) \mapsto ((\mathsf{y}_\mathsf{j} : \mathsf{w}_\mathsf{j})^*, \mathsf{e}_\mathsf{b}) \in \mathsf{P.F} \tag{106}$$
$$\mathsf{K}' = \mathsf{K}; \{\mathsf{x}' \mapsto \mathsf{n}', \mathsf{y}_0 \mapsto 0, \mathsf{y}_1 \mapsto 0, \dots\} \tag{107}$$

From rule (C-Call), we have $[\![\mathsf{P}, \Gamma \vdash \mathsf{x} := \mathsf{call}\ \mathsf{g}\ (\mathsf{n}') : \mathsf{w}']\!]^{\mathsf{exp}} = \mathbf{u32.const\ n'; call\ n_g; set\ n_x}$.

We are also given $\mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright (\mathsf{E}_1\,[\mathsf{x} := \mathsf{call}\ \mathsf{g}\ (\mathsf{n}')]\,, \mathsf{E}^*) \sim_\delta \langle \mathbf{\Sigma}, (\theta, \mathbf{u32.const\ n'} : \mathbf{call\ n} : \mathbf{i}^*, \mathbf{v}^*) : \mathbf{S}^* \rangle$. Inverting rule (Rt-Config), we have

$$\mathsf{P}, \Gamma \vdash \mathsf{E}_1\,[\mathsf{x} := \mathsf{call}\ \mathsf{g}\ (\mathsf{n}')] : \mathsf{w} \sim_\delta \langle \mathbf{u32.const\ n'} : \mathbf{call\ n} : \mathbf{i}^*, \mathbf{v}^* \rangle \quad (108)$$

$$\langle \mathsf{P}, \mathsf{K}[1\ldots], \mathsf{E}^* \rangle \sim_\delta \mathbf{S}^* \quad (109)$$

$$\mathsf{H} \sim_\delta \mathbf{\Sigma.T} \quad (110)$$

$$\mathsf{A} \sim_\delta \mathbf{\Sigma.A} \quad (111)$$

$$\mathsf{K}[0] \sim_\delta \theta \quad (112)$$

From rule (Const) and rule (Call), we have

$$\mathbf{M.funcs} \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{u32.const\ n'} : \mathbf{call\ n} : \mathbf{i}^*, \mathbf{v}^* \rangle \xrightarrow{\mathbf{call}(\pi_1, \mathbf{i}_1^*, \mathbf{v}_1^*)} \langle \mathbf{\Sigma}, \theta, \mathbf{i}^*, \mathbf{v}^* \rangle$$

where $\mathbf{\Sigma}_\phi(\mathbf{n}) = \{\mathbf{var}\ \tau^{\mathbf{j}}, \mathbf{body}\ \mathbf{i}_1^*\} : \rho$, $\rho = \tau \to \tau^*$ and $\mathbf{v}_1^* = \mathbf{n'o^j}$. From rule (Ctx-Call), we further have

$$\mathbf{M.funcs} \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{call\ n} : \mathbf{i}^*, \mathbf{n'} : \mathbf{v}^* \rangle \longrightarrow \langle \mathbf{\Sigma}, ([\ \mathbf{k} \mapsto \mathbf{v_k} \mid \mathbf{v_k} \in \mathbf{v}_1^*\ ], \mathbf{i}_1^*, \epsilon) : (\theta, \mathbf{i}^*, \mathbf{v}^*) : \mathbf{S}^* \rangle$$

We have to prove that

$$\mathsf{P}, \mathsf{K}', \mathsf{H}, \mathsf{A} \triangleright (\mathsf{e_b}, \mathsf{E}_1\,[\mathsf{x} := [\cdot]] : \mathsf{E}^*) \sim_\delta \langle \mathbf{\Sigma}, ([\ \mathbf{k} \mapsto \mathbf{v_k} \mid \mathbf{v_k} \in \mathbf{v}_1^*\ ], \mathbf{i}_1^*, \epsilon) : (\theta, \mathbf{i}^*, \mathbf{v}^*) : \mathbf{S}^* \rangle$$

From rule (Rt-Config), we have to prove

$$\mathsf{P}, \Gamma \vdash \mathsf{e_b} : \mathsf{w} \sim_\delta \langle \mathbf{i}_1^*, \epsilon \rangle \quad (113)$$

$$\langle \mathsf{P}, \mathsf{K}, \mathsf{E}_1\,[\mathsf{x} := [\cdot]] : \mathsf{E}^* \rangle \sim_\delta (\theta, \mathbf{i}^*, \mathbf{v}^*) : \mathbf{S}^* \quad (114)$$

$$\mathsf{H} \sim_\delta \mathbf{\Sigma.T} \quad (115)$$

$$\mathsf{A} \sim_\delta \mathbf{\Sigma.A} \quad (116)$$

$$\{\mathsf{x}' \mapsto \mathsf{n}', \mathsf{y}_0 \mapsto 0, \mathsf{y}_1 \mapsto 0, \ldots\} \sim_\delta [\ \mathbf{k} \mapsto \mathbf{v_k} \mid \mathbf{v_k} \in \mathbf{n'o^j}\ ] \quad (117)$$

We have (113) from rule (Program) and target step rule (Call). Invoking Lemma 20 (Context Decomposition) on (108) along with (112), (109) we have (114). Derivation of remaining premises from (110) and (111) is straightforward.

The step does not emit any trace and so we trivially have $\alpha \ =_{\delta'}\ \alpha$.

Heap is not modified during the target step. The new stack frame is valid from rule (empty-data-stack). From rule (cons-data-stack) we thus have $\vdash \mathbf{isValid}\ \mathbf{\Omega}'$.

Case (S-Return): Proof similar to the above case.

□

4.8.5 Functional Correctness-Primitive Step

Lemma 22 (Functional Correctness—Primitive Step)**.** If

1. $\mathsf{M} \vdash \Omega \xrightarrow{\alpha}\!\!\!\!\rightarrow \Omega'$ and
2. $[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M}$ and
3. $[\![\mathsf{P}, \Delta]\!]^{\mathsf{glob}} = \boldsymbol{\Delta}$ and
4. $\Omega \sim_\delta \boldsymbol{\Omega}$ and
5. $\boldsymbol{\Delta} \vdash \boldsymbol{\Omega}$ and
6. $\vdash \mathbf{isValid}\ \boldsymbol{\Omega}$ and
7. $T =_\delta \Omega.\mathsf{A}$
8. $\delta \subseteq \delta'$ and
9. $\alpha =_{\delta'} \overline{\alpha}$ and
10. $T \xrightarrow{\overline{\alpha}} T'$ and
11. $T' =_{\delta'} \Omega'.\mathsf{A}$

then $\exists \delta', \boldsymbol{\Omega}', \alpha$, such that

(i) $\delta' \supseteq \delta$,

(ii) $\Omega' \sim_{\delta'} \boldsymbol{\Omega}'$ and

(iii) $\mathbf{M.funcs} \vdash \boldsymbol{\Omega} \overset{\alpha}{\Longrightarrow} \boldsymbol{\Omega}'$ and

(iv) $\alpha =_{\delta'} \alpha$ and

(v) $\vdash \mathbf{isValid}\ \boldsymbol{\Omega}'$

Proof. Proof by induction on the $\mathsf{M} \vdash \Omega \xrightarrow{\alpha}\!\!\!\!\rightarrow \Omega'$. Interesting cases are those which emit non-empty events and manipulate pointers.

Case (S-Ptr-Arith): We have to prove

$$\exists \delta' \supseteq \delta \text{ and } \boldsymbol{\Omega}', \Omega' \sim_{\delta'} \boldsymbol{\Omega}' \tag{118}$$

$$\mathbf{M.funcs} \vdash \boldsymbol{\Omega} \overset{\alpha}{\Longrightarrow} \boldsymbol{\Omega}' \tag{119}$$

$$\alpha =_{\delta'} \alpha \tag{120}$$

Given the following:

$$\mathsf{M} \vdash \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright \mathsf{a}^{(\mathsf{b}, \ell, \mathsf{w}, \mathsf{n}_{\mathsf{id}})} \oplus \mathsf{n} \xrightarrow{\epsilon}\!\!\!\!\rightarrow \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright (\mathsf{a} + \mathsf{n})^{(\mathsf{b}, \ell, \mathsf{w}, \mathsf{n}_{\mathsf{id}})} \tag{121}$$

$$[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M} \tag{122}$$

$$[\![\mathsf{P}, \Delta]\!]^{\mathsf{glob}} = \boldsymbol{\Delta} \tag{123}$$
$$\Omega \sim_\delta \boldsymbol{\Omega} \tag{124}$$
$$\boldsymbol{\Delta} \vdash \boldsymbol{\Omega} \tag{125}$$
$$\vdash \mathbf{isValid}\ \boldsymbol{\Omega} \tag{126}$$
$$\mathsf{MS{+}{+}}(\alpha, \Omega.\mathsf{A}) \tag{127}$$

From (Rt-Config), we have

$$\Omega = \mathsf{P}, (\theta_{\mathsf{f}}; \mathsf{K}), \mathsf{H}, \mathsf{A} \triangleright (\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n}_{\mathsf{id}})} \oplus \mathsf{n}, \mathsf{E}^*) \tag{128}$$
$$\Omega' = \mathsf{P}, (\theta_{\mathsf{f}}; \mathsf{K}), \mathsf{H}, \mathsf{A} \triangleright ((\mathsf{a} + \mathsf{n})^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n}_{\mathsf{id}})}, \mathsf{E}^*) \tag{129}$$
$$\mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n}_{\mathsf{id}})} \oplus \mathsf{n} : \mathsf{array}\ \tau \sim_\delta \langle \mathbf{handle.add}, \mathbf{n}; \langle \mathbf{b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}} \rangle \tag{130}$$
$$\boldsymbol{\Omega} = \langle \boldsymbol{\Sigma}, (\theta_{\mathbf{f}}, \mathbf{handle.add}, \mathbf{n}; \langle \mathbf{b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}}) : \mathbf{S}^* \rangle \tag{131}$$
$$\langle \mathsf{P}, \mathsf{K}, \mathsf{E}^* \rangle \sim_\delta \mathbf{S}^* \tag{132}$$
$$\mathsf{H} \sim_\delta \boldsymbol{\Sigma}.\mathbf{T} \tag{133}$$
$$\mathsf{A} \sim_\delta \boldsymbol{\Sigma}.\mathbf{A} \tag{134}$$
$$\theta_{\mathsf{f}} \sim_\delta \theta_{\mathbf{f}} \tag{135}$$

From (Rt-Ptr-Arith), we have:

$$\mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n}_{\mathsf{id}})} : \mathsf{array}\ \tau \sim_\delta \langle [], \langle \mathbf{b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}} \rangle \tag{136}$$
$$\mathsf{P}, \Gamma \vdash \mathsf{n} : \mathsf{int} \sim_\delta \langle [], \mathbf{v} \rangle \tag{137}$$
$$\mathbf{n} = \mathbf{sz}\,(\tau) * \mathbf{v} \tag{138}$$

Inverting (137) using (C-data-int), we have $|\mathbf{v}| = |\mathsf{n}|$. Thus (138) becomes

$$\mathbf{n} = \mathbf{sz}\,(\tau) * \mathsf{n} \tag{139}$$

Proof for $\Omega' \sim_{\delta'} \boldsymbol{\Omega}'$: From (Handle-Add), we have

$$\boldsymbol{\Phi}^* \vdash \langle \boldsymbol{\Sigma}, (\theta_{\mathbf{f}}, \mathbf{handle.add}, \mathbf{n}; \langle \mathbf{b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}}) : \mathbf{S}^* \rangle \longrightarrow\!\!\!\!\!\rightarrow \langle \boldsymbol{\Sigma}, (\theta_{\mathbf{f}}, [], \langle \mathbf{b}, \mathbf{n_o} + \mathbf{n}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}}) : \mathbf{S}^* \rangle$$

Let $\boldsymbol{\Omega}' = \langle \boldsymbol{\Sigma}, (\theta_{\mathbf{f}}, [], \langle \mathbf{b}, \mathbf{n_o} + \mathbf{n}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}}) : \mathbf{S}^* \rangle$ and $\delta' = \delta$. From (Rt-Value), it suffices to show that

$$(\mathsf{a} + \mathsf{n})^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n}_{\mathsf{id}})} \sim_\delta \langle [], \langle \mathbf{b}, \mathbf{n_o} + \mathbf{n}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}} \rangle$$

Inverting (136) using (Rt-Value), we have

$$\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n}_{\mathsf{id}})} \sim_\delta \langle \mathbf{b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}} \tag{140}$$

From (C-data-Valid-Ptr-Array) we have

$$\delta(\mathsf{b}, \mathsf{n}_{\mathsf{id}}) = (\mathbf{b}, \mathbf{n_{id}}) \tag{141}$$

$$\ell * \mathtt{sz}(\mathsf{w}) = \mathbf{n_l} \tag{142}$$
$$(\mathsf{a} - \mathsf{b}) * \mathtt{sz}(\mathsf{w}) = \mathbf{n_o} \tag{143}$$

From (139), we have

$$((\mathsf{a} - \mathsf{b}) * \mathtt{sz}(\mathsf{w})) + \mathsf{n} * \mathtt{sz}(\mathsf{w}) = \mathbf{n_o} + \mathbf{n}$$

This implies (from (C-data-Valid-Ptr-Array)), $(\mathsf{a} + \mathsf{n})^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} \sim_\delta \langle [], \langle \mathbf{b}, \mathbf{n_o} + \mathbf{n}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}} \rangle$. Since $\delta' = \delta$, we thus have $\Omega' \sim_{\delta'} \mathbf{\Omega'}$.

Proof for $\mathbf{M.funcs} \vdash \mathbf{\Omega} \overset{\alpha}{\Longrightarrow} \mathbf{\Omega'}$: From (Handle-Add), we have that $\mathbf{M.funcs} \vdash \mathbf{\Omega} \overset{\epsilon}{\Longrightarrow} \mathbf{\Omega'}$ and $\alpha = \epsilon$.

Proof for $\alpha =_{\delta'} \alpha$: We have $\alpha = \epsilon$. From (Empty), we have $\alpha =_{\delta'} \alpha$.

Case (S-Assign): Given the following:

$$\mathsf{M} \vdash \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright *\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} := \mathsf{v} \xrightarrow{\mathsf{write}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})},\mathsf{v})} \mathsf{P}, \mathsf{K}, \mathsf{H}', \mathsf{A} \triangleright 0 \tag{144}$$
$$[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M} \tag{145}$$
$$[\![\mathsf{P}, \Delta]\!]^{\mathsf{glob}} = \mathbf{\Delta} \tag{146}$$
$$\Omega \sim_\delta \mathbf{\Omega} \tag{147}$$
$$\mathbf{\Delta} \vdash \mathbf{\Omega} \tag{148}$$
$$\vdash \mathbf{isValid}\ \mathbf{\Omega} \tag{149}$$
$$\mathsf{MS{+}{+}}(\alpha, \Omega.\mathsf{A}) \tag{150}$$

From (Rt-Config) and (Rt-Assign), we have:

$$\Omega = \mathsf{P}, (\theta_\mathsf{f}; \mathsf{K}), \mathsf{H}, \mathsf{A} \triangleright (*\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} := \mathsf{v}, \mathsf{E}^*) \tag{151}$$
$$\Omega' = \mathsf{P}, (\theta_\mathsf{f}; \mathsf{K}), \mathsf{H}', \mathsf{A} \triangleright (0, \mathsf{E}^*) \tag{152}$$
$$[\![\tau]\!] = \tau \tag{153}$$
$$\mathsf{P}, \Gamma \vdash *\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} := \mathsf{v} : \mathsf{w} \sim_\delta \langle \boldsymbol{\tau}.\mathbf{segstore}; \mathbf{u32.const\ 0}, (\mathbf{v}; \langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}}) \rangle \tag{154}$$
$$\mathbf{\Omega} = \langle \mathbf{\Sigma}, (\theta, \boldsymbol{\tau}.\mathbf{segstore}; \mathbf{u32.const\ 0}, (\mathbf{v}; \langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}})) : \mathbf{S}^* \rangle \tag{155}$$
$$\langle \mathsf{P}, \mathsf{K}, \mathsf{E}^* \rangle \sim_\delta \mathbf{S}^* \tag{156}$$
$$\mathsf{H} \sim_\delta \mathbf{\Sigma.T} \tag{157}$$
$$\mathsf{A} \sim_\delta \mathbf{\Sigma.A} \tag{158}$$
$$\theta_\mathsf{f} \sim_\delta \theta_\mathbf{f} \tag{159}$$

Let

- $\delta' = \delta$,

- $\mathbf{\Omega'} = \langle \mathbf{\Sigma'}, (\theta, \mathbf{u32.const\ 0}, []) : \mathbf{S}^* \rangle$, and
- $\alpha = \mathbf{write}_{\tau}\ (\langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}})$.

Then:

Proof for $\mathbf{M.funcs} \vdash \mathbf{\Omega} \xRightarrow{\alpha} \mathbf{\Omega'}$**:** To show that the target store operation succeeds, we use the assumption that the source execution is memory safe. In particular, to apply rule (H-Store) (or (H-Store-Handle), depending on the type $[\![\tau]\!] = \tau$), we only need to show that $\mathbf{n_o} > \mathbf{0}$, $\mathbf{n_o} + |\tau| < \mathbf{n_l}$, and $\mathbf{n_{id}} \in \mathbf{A.allocated}$. First, we invert $\alpha =_{\delta'} \overline{\alpha}$ with rule (src-tr-write-auth) and deduce that $\overline{\alpha} = \mathtt{write}(a^{(c,s)}) \cdot \mathtt{write}((a+1)^{(c,s)}) \ldots \mathtt{write}((a+n-1)^{(c,s)})$ where $n = \mathtt{sz}(\tau) = |\tau|$ and $\delta(\mathsf{b}, \mathsf{n_{id}}) = b^{(c,s)}$. Since $T \xrightarrow{\overline{\alpha}} T'$, we know that these memory accesses are memory safe, i.e., $T[a+i] = A(c,s)$ for $i \in \{0, \ldots n-1\}$. Thus, by $T =_{\delta} \Omega.\mathsf{A}$ (Definition 4), we know that the pointer $\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}$ is *not* dangling, i.e., $\mathsf{n_{id}} \in \mathsf{A.allocated}$, and in-bounds, i.e., $\mathsf{a} \geq \mathsf{b}$ and $(\mathsf{a} - \mathsf{b}) < \ell$. We now leverage the source to target equivalence relation to derive corresponding facts about the target handle. First, the source pointer and target handle are related, thus we have $\delta(\mathsf{b}, \mathsf{n_{id}}) = \mathbf{b}, \mathbf{n_{id}}$, $\ell * \mathtt{sz}(\tau) = \mathbf{n_l}$ and $(\mathsf{a} - \mathsf{b}) * \mathtt{sz}(\tau) = \mathbf{n_o}$, from $\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} \sim_{\delta} \langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}}$ ((Rt-Value) and (C-data-Valid-Ptr-Array)). Then, we apply Lemma 25 to these, $\mathsf{a} \geq \mathsf{b}$, and $(\mathsf{a} - \mathsf{b}) < \ell$, and obtain $\mathbf{o} \geq \mathbf{0}$ and $\mathbf{o} + |\tau| < \ell$. Lastly, since $\mathsf{A} \sim_{\delta} \mathbf{\Sigma.A}$ and $\mathsf{n_{id}} \in \mathsf{A.allocated}$, we have $\mathbf{n_{id}} \in \mathbf{A.allocated}$. Thus, by rule (H-Store) and then (Const), we construct the reduction $\mathbf{M.funcs} \vdash \mathbf{\Omega} \xRightarrow{\alpha} \mathbf{\Omega'}$, where $\alpha = \mathbf{write}_{\tau}\ (\langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}})$.

Proof for $\Omega' \sim_{\delta} \mathbf{\Omega'}$ **:** First, we relate the source value 0 resulting from the assignment to the target value $\mathbf{0}$ pushed on the stack by instruction $\mathbf{const\ 0}$, via rule (C-data-int). Then, to show $\Omega' \sim_{\delta} \mathbf{\Omega'}$, it suffices to show that the resulting stores are related, i.e., $\mathsf{H'} \sim_{\delta} \mathbf{\Sigma'.T}$. The initial stores are related (157), therefore, by (Heap-Equiv) we only need to show that the byte-string encoding of value v is written in the target store at a related location. Formally, we have $\mathbf{\Sigma'.T}[(\mathbf{n_b} + \mathbf{n_o}) \ldots (\mathbf{n_b} + \mathbf{n_0}) + \mathbf{s} - \mathbf{1}] = (\mathbf{b}, \mathbf{t})_0 \ldots (\mathbf{b}, \mathbf{t})_{s-1} = \tau.\mathbf{pack}(\mathbf{v})$, where $s = \mathtt{sz}(\tau)$. Since the values written at the source and target are related, i.e., $\mathsf{v} \sim_{\delta} \mathbf{v}$ from $\Omega \sim_{\delta} \mathbf{\Omega}$, we easily construct $\mathsf{v} \sim_{\delta} (\mathbf{b_0}, \mathbf{t})_0 \ldots (\mathbf{b_{s-1}}, \mathbf{t})_{s-1}$ by rule (Heap-Value). Lastly, we invert $\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} \sim_{\delta} \langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle$ ((154) and (Rt-Value)), and obtain $\delta(\mathsf{b}, \mathsf{n_{id}}) = (\mathbf{n_b}, \mathbf{n_{id}} \mathbf{0})$, i.e., the source and target addresses that are written are related.

Proof for $\alpha =_{\delta'} \alpha$**:** We have $\alpha = \mathsf{write}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}, \mathsf{v})$ and $\alpha = \mathbf{write}_{\tau}\ (\langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}})$. From $\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} \sim_{\delta} \langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle$ and (153), we have all the premises required for (Tr-Rel-Write), and thus $\alpha =_{\delta'} \alpha$.

Case (S-Arr-Assign): Similar to above case.

Case (S-Dereference): Given the following:

$$\mathsf{M} \vdash \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright *\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} \xrightarrow{\mathsf{read}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})},\mathsf{v})} \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright \mathsf{v} \tag{160}$$

$$[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M} \tag{161}$$

$$[\![\mathsf{P}, \Delta]\!]^{\mathsf{glob}} = \boldsymbol{\Delta} \tag{162}$$

$$\Omega \sim_\delta \boldsymbol{\Omega} \tag{163}$$

$$\boldsymbol{\Delta} \vdash \boldsymbol{\Omega} \tag{164}$$

$$\vdash \mathbf{isValid}\ \boldsymbol{\Omega} \tag{165}$$

$$T =_\delta \Omega.\mathsf{A} \tag{166}$$

$$\delta \subseteq \delta' \tag{167}$$

$$\alpha =_{\delta'} \overline{\alpha} \tag{168}$$

$$T \xrightarrow{\overline{\alpha}} T' \tag{169}$$

$$T' =_{\delta'} \Omega'.\mathsf{A} \tag{170}$$

From (Rt-Config) and (Rt-Deref), we have:

$$\Omega = \mathsf{P}, (\theta_\mathsf{f}; \mathsf{K}), \mathsf{H}, \mathsf{A} \triangleright (*\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}, \mathsf{E}^*) \tag{171}$$

$$\Omega' = \mathsf{P}, (\theta_\mathsf{f}; \mathsf{K}), \mathsf{H}', \mathsf{A} \triangleright (\mathsf{v}, \mathsf{E}^*) \tag{172}$$

$$[\![\tau]\!] = \tau \tag{173}$$

$$\mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} : \mathsf{ptr}\,\tau \sim_\delta \langle [], \langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_\hbar \rangle \tag{174}$$

$$\boldsymbol{\Omega} = \langle \boldsymbol{\Sigma}, (\theta, \boldsymbol{\tau}.\mathbf{segload}, \langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_\hbar) : \mathbf{S}^* \rangle \tag{175}$$

$$\langle \mathsf{P}, \mathsf{K}, \mathsf{E}^* \rangle \sim_\delta \mathbf{S}^* \tag{176}$$

$$\mathsf{H} \sim_\delta \boldsymbol{\Sigma}.\mathbf{T} \tag{177}$$

$$\mathsf{A} \sim_\delta \boldsymbol{\Sigma}.\mathbf{A} \tag{178}$$

$$\theta_\mathsf{f} \sim_\delta \theta_\mathbf{f} \tag{179}$$

Proof for $\mathbf{M}.\mathbf{funcs} \vdash \boldsymbol{\Omega} \stackrel{\alpha}{\Longrightarrow} \boldsymbol{\Omega}'$: The configuration steps via rule (H-Load) (resp. (H-Load-Handle)), where the premises $\mathbf{n_o} > \mathbf{0}$, $\mathbf{n_o} + |\tau| < \mathbf{n_l}$, and $\mathbf{n_{id}} \in \mathbf{A}.\mathbf{allocated}$ hold following the same argument used in case (S-Assign).

Thus, we have: $\boldsymbol{\Phi}^* \vdash \langle \boldsymbol{\Sigma}, (\theta, \boldsymbol{\tau}.\mathbf{segload}, \langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_\mathbf{F}) : \mathbf{S}^* \rangle \xrightarrow{\alpha} \boldsymbol{\Omega}'$ where

$$\boldsymbol{\Omega}' = \langle \boldsymbol{\Sigma}, (\theta, [], \mathbf{v}) : \mathbf{S}^* \rangle \tag{180}$$

$$\mathbf{v} = \tau.\mathbf{unpack}(\mathbf{b}^*) \tag{181}$$

$$(\mathbf{b}^*, \mathbf{t}^*) = [\ \mathbf{T}[\mathbf{a} + \mathbf{j}] \mid \ \mathbf{j} \in \{\mathbf{0}..|\tau| - \mathbf{1}\}\ ] \tag{182}$$

$$\mathbf{a} = \mathbf{n_b} + \mathbf{n_o} \tag{183}$$

$$\alpha = \mathbf{read}_\tau\ (\langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_\mathbf{F}) \tag{184}$$

Proof for $\Omega' \sim_{\delta'} \mathbf{\Omega'}$: Let $\mathbf{\Omega'} = \langle \mathbf{\Sigma}, (\theta, [], \mathbf{v}) : \mathbf{S}^* \rangle$ and $\delta' = \delta$. To prove $\Omega' \sim_{\delta'} \mathbf{\Omega'}$, it suffices to show $\mathsf{P}, \Gamma \vdash \mathsf{v} : \tau \sim_{\delta} \langle [], \mathbf{v} \rangle$, where $\mathsf{v} = \mathsf{H}[\mathsf{a}]$ from 160 (rule (S-Dereference)). Intuitively, $\mathbf{v}$ is related to v because $\mathbf{v}$ is obtained by reading an handle related to the source pointer from a segment memory store that is related to the source heap. Formally, from the equivalence of the stores, i.e., (177) and rule (Heap-Equiv), we have that:

$$\mathsf{H}[\mathsf{a}] = \mathsf{v} \tag{185}$$
$$\vdash \mathsf{v} : \tau \tag{186}$$
$$[\![\tau]\!] = \tau \tag{187}$$
$$s = \mathtt{sz}\,(\tau) \tag{188}$$
$$(\mathsf{b}, \mathsf{n}, \tau, \mathsf{n_{id}}) \in \lfloor \mathsf{A} \rfloor_1 \tag{189}$$
$$\mathsf{a} \in [\mathsf{b}..\mathsf{b} + \mathsf{n}) \tag{190}$$
$$\mathbf{a}, \mathbf{n'_{id}} = \delta(\mathsf{a}, \mathsf{n_{id}}) \tag{191}$$
$$\mathbf{T}[\mathbf{a}] \dots \mathbf{T}[\mathbf{a} + \mathbf{s} - \mathbf{1}] = (\mathbf{b}, \mathbf{t})_0 (\mathbf{b}, \mathbf{t})_1 \dots (\mathbf{b}, \mathbf{t})_{s-1} \tag{192}$$
$$\mathsf{v} \sim_{\delta} (\mathbf{b}, \mathbf{t})_0 (\mathbf{b}, \mathbf{t})_1 \dots (\mathbf{b}, \mathbf{t})_{s-1} \tag{193}$$

Next, we need to show that the byte string corresponding to v ((192) and (193)) is the same byte string read in the target step via handle $\langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_F$, i.e. $\mathbf{a} = \mathbf{n_b} + \mathbf{n_o}$ and $\mathbf{n'_{id}} = \mathbf{n_{id}}$. Since the handle is valid (165), there exist an allocated slot in the target allocator state, i.e., $(\mathbf{n_b}, \mathbf{n_l}, \mathbf{n_{id}}) \in \lfloor \mathbf{\Omega.A} \rfloor_1$ by rule (cor-handle). Furthermore, since the source and target allocators are related (178), this slot is related to the slot from 189, i.e., $(\mathsf{b}, \mathsf{n}, \tau, \mathsf{n_{id}}) \sim_{\delta} (\mathbf{n_b}, \mathbf{n_l}, \mathbf{n_{id}})$. By inverting this equivalence via rule (Slot-Eq-Prim), we obtain:

$$\forall \mathsf{i}, 0 \leq \mathsf{i} < \mathsf{n}, \delta(\mathsf{b} + \mathsf{i}, \mathsf{n_{id}}) = (\mathbf{n_b} + \mathsf{i} * \mathtt{sz}\,(\tau), \mathbf{n_{id}})$$

which we instantiate with $\mathsf{i} = \mathsf{a} - \mathsf{b}$ and obtain $\delta(\mathsf{a}, \mathsf{n_{id}}) = (\mathbf{n_b} + (\mathsf{b} - \mathsf{a}) * \mathtt{sz}\,(\tau), \mathbf{n_{id}})$.[1] Furthermore, since the pointer and the handle are related ((174) and rule (C-data-Valid-Ptr-Array)), we that $(\mathsf{a} - \mathsf{b}) * \mathtt{sz}\,(\tau) = \mathbf{n_o}$, so we can rewrite the equation above to:

$$\delta(\mathsf{a}, \mathsf{n_{id}}) = (\mathbf{n_b} + \mathbf{n_o}, \mathbf{n_{id}}) = (\mathbf{a}, \mathbf{tid'})$$

which implies $\mathbf{a} = \mathbf{n_b} + \mathbf{n_o}$ and $\mathbf{n_{id}} = \mathbf{n'_{id}}$, as desired. Finally, from (192), (193) and rule (Heap-Value), we have $\mathsf{v} \sim_{\delta} \mathbf{unpack}(\mathbf{b_0} \dots \mathbf{b_{s-1}}) = \mathbf{v}$.

Proof for $\alpha \ =_{\delta} \ \alpha$: We have $\alpha = \mathsf{read}(\mathsf{a}^{(\mathsf{b}, \ell, \mathsf{w}, \mathsf{n_{id}})}, \mathsf{v})$ and $\alpha = \mathbf{read}_{\tau}\,(\langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1} \rangle_{\mathbf{F}})$. From (173) and (174), we have all premises related to (Tr-Rel-Read). Thus, we have $\alpha \ =_{\delta'} \ \alpha$.

Case (S-Arr-Dereference): Similar to above case.

[1] Notice that $0 \leq \mathsf{b} - \mathsf{a} < \mathsf{n}$ because the source execution is memory safe by assumption.

Case (S-Struct-Field): Given the following:

$$\mathsf{M} \vdash \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright \&\mathsf{a} \to \mathsf{f_k} \xrightarrow{\epsilon} \!\!\!\!\rightarrow \quad \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright \mathsf{a} + \mathsf{o}_1^{(\mathsf{b}+\mathsf{o}_1, 1, \tau_k, \mathsf{n_{id}})} \tag{194}$$

$$[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M} \tag{195}$$

$$[\![\mathsf{P}, \Delta]\!]^{\mathsf{glob}} = \mathbf{\Delta} \tag{196}$$

$$\Omega \sim_\delta \mathbf{\Omega} \tag{197}$$

$$\mathbf{\Delta} \vdash \mathbf{\Omega} \tag{198}$$

$$\vdash \mathbf{isValid}\ \mathbf{\Omega} \tag{199}$$

$$\mathsf{MS{+}{+}}(\alpha, \Omega.\mathsf{A}) \tag{200}$$

From (Rt-Config) and (Rt-Field), we have:

$$\Omega = \mathsf{P}, (\theta_\mathsf{f}; \mathsf{K}), \mathsf{H}, \mathsf{A} \triangleright (\&\mathsf{a} \to \mathsf{f_k}, \mathsf{E}^*) \tag{201}$$

$$\Omega' = \mathsf{P}, (\theta_\mathsf{f}; \mathsf{K}), \mathsf{H}', \mathsf{A} \triangleright (\mathsf{a} + \mathsf{o}_1^{(\mathsf{b}+\mathsf{o}_1, 1, \tau_k, \mathsf{n_{id}})}, \mathsf{E}^*) \tag{202}$$

$$\mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{b}, \ell, \mathsf{struct\ s}, \mathsf{n_{id}})} : \mathsf{ptr\,struct\ s} \sim_\delta \langle [], \langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_\mathbf{F} \rangle \tag{203}$$

$$(\mathbf{o_1}, \mathbf{o_2}) = (\Sigma_\mathsf{i}^{\mathsf{k}-1} \mathbf{sz}\,(\tau_\mathsf{i}), (\mathbf{sz}\,(\mathsf{struct\ s}) - \mathbf{sz}\,(\tau_\mathsf{k}))) \tag{204}$$

$$\mathbf{\Omega} = \langle \mathbf{\Sigma}, (\theta, \mathbf{slice}; \mathbf{o_2}; \mathbf{o_1}; \langle \mathbf{n_b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle_\mathbf{F}) : \mathbf{S}^* \rangle \tag{205}$$

$$\langle \mathsf{P}, \mathsf{K}, \mathsf{E}^* \rangle \sim_\delta \mathbf{S}^* \tag{206}$$

$$\mathsf{H} \sim_\delta \mathbf{\Sigma.T} \tag{207}$$

$$\mathsf{A} \sim_\delta \mathbf{\Sigma.A} \tag{208}$$

$$\theta_\mathsf{f} \sim_\delta \theta_\mathbf{f} \tag{209}$$

Proof for $\mathbf{M.funcs} \vdash \mathbf{\Omega} \xRightarrow{\alpha} \mathbf{\Omega'}$**:** Let $\mathbf{\Omega'} = \langle \mathbf{\Sigma}, (\theta, [], \langle \mathbf{n_b} + \mathbf{o_1}, \mathbf{n_o}, \mathbf{n_l} - \mathbf{o_2}, \mathbf{1}, \mathbf{n_{id}} \rangle_\mathbf{F}) : \mathbf{S}^* \rangle$.

To show that the target configuration steps with rule (Slice), we first need to show that the offsets are not negative. From (204), we trivially have

$$\mathbf{o_1} \geq \mathbf{0} \tag{210}$$

$$\mathbf{o_2} \geq \mathbf{0} \tag{211}$$

Then, we only need to show that that offset $\mathbf{o_1}$ is bounded by the length of the segment $\mathbf{n_l}$, i.e., $\mathbf{o_1} < \mathbf{n_l}$. Since 194 involves a struct, we know that $\ell = 1$ and we can invert 203 with rule rule (C-data-Valid-Ptr-Single), from which we obtain:

$$\delta(\mathsf{b}, \mathsf{n_{id}}) = \mathbf{n_b}, \mathbf{n_{id}} \tag{212}$$

$$\mathbf{sz}\,(\mathsf{struct\ s}) = \mathbf{n_l} \tag{213}$$

$$(\mathsf{a} - \mathsf{b}) = \mathbf{n_o} = 0 \tag{214}$$

Then, we rewrite the equation above using 213 and 204, from which we have: $\mathbf{o_1} = \Sigma_{\mathsf{i}=1}^{\mathsf{k}-1} \mathbf{sz}\,(\tau_\mathsf{i}) < \Sigma_{\mathsf{i}=1}^{\mathsf{n}} \mathbf{sz}\,(\tau_\mathsf{i}) = \mathbf{sz}\,(\mathsf{struct\ s}) = \mathbf{n_l}$, as desired.

Proof for $\Omega' \sim_{\delta'} \mathbf{\Omega'}$: To prove $\Omega' \sim_{\delta'} \mathbf{\Omega'}$, it suffices to show that $\mathsf{P}, \Gamma \vdash \mathsf{a} + \mathsf{o}_1^{(\mathsf{b}+\mathsf{o}_1, 1, \tau_k, \mathsf{n}_{id})} : \mathsf{ptr}\,\tau_k \sim \langle [], \langle \mathbf{n_b} + \mathbf{o_1}, \mathbf{n_o}, \mathbf{n_l} - \mathbf{o_2}, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}} \rangle$. In order to apply (C-data-Valid-Ptr-Array), we need to show:

$$\delta(\mathsf{b} + \mathsf{o}_1, \mathsf{n}_{id}) = (\mathbf{n_b} + \mathbf{o_1}, \mathbf{n_{id}}) \tag{215}$$

$$\mathbf{sz}\,(\tau_k) = \mathbf{n_l} - \mathbf{o_2} \tag{216}$$

$$(\mathsf{a} + \mathsf{o}_1 - \mathsf{b} - \mathsf{o}_1) * \mathbf{sz}\,(\tau_k) = \mathbf{n_o} \tag{217}$$

Since the source and target allocators are related (208), from rule (Slot-Eq-Struct), we have:

$$\delta(\mathsf{b}, \mathsf{n}_{id}) = (\mathbf{n_b}, \mathbf{n_{id}}) \tag{218}$$

$$\mathsf{D}(\mathsf{s}) = \{\mathsf{f}_1 : \tau_1 \ldots \mathsf{f}_n : \tau_n\} \tag{219}$$

$$\forall \mathsf{i}, 0 \leq \mathsf{i} < \mathsf{n}, \delta(\mathsf{b} + \mathsf{i}, \mathsf{n}_{id}) = (\mathbf{n_b} + \Sigma_{\mathsf{j}=1}^{\mathsf{i}} \mathbf{sz}\,(\tau_j), \mathbf{n_{id}}) \tag{220}$$

Then, to obtain (215), we instantiate (220) with $\mathsf{i} = \mathsf{o}_1 = \mathsf{k} - 1$ and obtain $\delta(\mathsf{b} + \mathsf{o}_1, \mathsf{n}_{id}) = (\mathbf{n_b} + \Sigma_{\mathsf{j}=1}^{\mathsf{k}-1} \mathbf{sz}\,(\tau_j), \mathbf{n_{id}}) = (\mathbf{n_b} + \mathbf{o_1}, \mathbf{n_{id}})$ by $\mathbf{o_1} = \Sigma_{\mathsf{i}=1}^{\mathsf{k}-1} \mathbf{sz}\,(\tau_i)$ (204). Equation (216) follows directly by rewriting with $\mathbf{sz}\,(\mathsf{struct\ s}) = \mathbf{n_l}$ (213) and $\mathbf{o_2} = \mathbf{sz}\,(\mathsf{struct\ s}) - \mathbf{sz}\,(\tau_k)$ (204):

$$\mathbf{sz}\,(\tau_k) = \mathbf{n_l} - \mathbf{o_2} = \mathbf{sz}\,(\mathsf{struct\ s}) - (\mathbf{sz}\,(\mathsf{struct\ s}) - \mathbf{sz}\,(\tau_k)) = \mathbf{sz}\,(\tau_k)$$

(204), we have $\mathbf{o_2} = \mathbf{sz}\,(\mathsf{struct\ s}) - \mathbf{sz}\,(\tau_k)$. Combined with (213), we have (216). Finally, we trivially have $(\mathsf{a} - \mathsf{b}) * \mathbf{sz}\,(\tau_k) = \mathbf{n_o}$ (217) from $(\mathsf{a} - \mathsf{b}) = \mathbf{n_o} = 0$ (214).

Proof for $\alpha \ =_{\delta'} \ \alpha$: Since α and α are empty, they are trivially related.

Case (S-Malloc-Array): Given the following:

$$\mathsf{M} \vdash \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright \mathsf{malloc}(\tau, \mathsf{n}) \xrightarrow{\mathsf{alloc}(\mathsf{a}^{(\mathsf{a},\mathsf{n},\tau,\mathsf{n}_{id})})} \mathsf{P}, \mathsf{K}, \mathsf{H}', \mathsf{A}' \triangleright \mathsf{a}^{(\mathsf{a},\mathsf{n},\tau,\mathsf{n}_{id})} \tag{221}$$

$$[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M} \tag{222}$$

$$[\![\mathsf{P}, \Delta]\!]^{\mathsf{glob}} = \mathbf{\Delta} \tag{223}$$

$$\Omega \sim_{\delta} \mathbf{\Omega} \tag{224}$$

$$\mathbf{\Delta} \vdash \mathbf{\Omega} \tag{225}$$

$$\vdash \mathbf{isValid}\ \mathbf{\Omega} \tag{226}$$

$$\mathsf{MS{+}{+}}(\alpha, \Omega.\mathsf{A}) \tag{227}$$

From (Rt-Config) and (Rt-Malloc-Array-1), we have:

$$\Omega = \mathsf{P}, (\theta_f; \mathsf{K}), \mathsf{H}, \mathsf{A} \triangleright (\mathsf{malloc}(\tau, \mathsf{n}), \mathsf{E}^*) \tag{228}$$

$$\Omega' = \mathsf{P}, (\theta_f; \mathsf{K}), \mathsf{H}', \mathsf{A} \triangleright (\mathsf{a}^{(\mathsf{a},\mathsf{n},\tau,\mathsf{n}_{id})}, \mathsf{E}^*) \tag{229}$$

$$\mathsf{fresh}(\mathsf{n_{id}}) \tag{230}$$
$$\mathsf{P}, \Gamma \vdash \mathsf{n} : \mathsf{int} \sim_\delta \langle [], \mathbf{n} \rangle \tag{231}$$
$$[\![\tau]\!] = \tau \tag{232}$$
$$\mathbf{\Omega} = \langle \mathbf{\Sigma}, (\theta, \mathbf{segalloc}, \mathbf{n}') : \mathbf{S}^* \rangle \tag{233}$$
$$\mathbf{n}' = \mathbf{n} * \mathtt{sz}(\tau) \tag{234}$$
$$\langle \mathsf{P}, \mathsf{K}, \mathsf{E}^* \rangle \sim_\delta \mathbf{S}^* \tag{235}$$
$$\mathsf{H} \sim_\delta \mathbf{\Sigma.T} \tag{236}$$
$$\mathsf{A} \sim_\delta \mathbf{\Sigma.A} \tag{237}$$
$$\theta_{\mathsf{f}} \sim_\delta \theta_{\mathbf{f}} \tag{238}$$

Proof for $\mathbf{M.funcs} \vdash \mathbf{\Omega} \overset{\alpha}{\Longrightarrow} \mathbf{\Omega}'$**:** From (H-Alloc), we have

$\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, (\theta, \mathbf{segalloc}, \mathbf{n}') : \mathbf{S}^* \rangle \xrightarrow{\mathbf{salloc(v)}} \langle \mathbf{\Sigma}', (\theta, [], \mathbf{v}) : \mathbf{S}^* \rangle$ where $\mathbf{\Sigma}' = (\mathbf{\Sigma.H}, \mathbf{T}', \mathbf{A}')$ such that

$$\mathbf{v} = \langle \mathbf{n_b}, \mathbf{0}, \mathbf{n}', \mathbf{true}, \mathbf{n_{id}} \rangle \tag{239}$$
$$\mathbf{a} + \mathbf{n}' < |\mathbf{T}| \tag{240}$$
$$\mathbf{T}' = \mathbf{T}[\mathbf{a} + \mathbf{j} \mapsto (\mathbf{0}, \mathbf{D}) \mid \mathbf{j} \in \mathbf{0}..\mathbf{n}' - \mathbf{1}] \tag{241}$$
$$\mathbf{A}' = (\mathbf{s}'^*_{\mathbf{U}}, \mathbf{s}'^*_{\mathbf{F}}) \tag{242}$$
$$\mathbf{s}'^*_{\mathbf{F}} = \mathbf{s}^*_{\mathbf{1}} \cdot (\mathbf{a} + \mathbf{n}', \mathbf{n}'' - \mathbf{n}', \_) \cdot \mathbf{s}^*_{\mathbf{2}} \tag{243}$$
$$\mathbf{s}'^*_{\mathbf{U}} = (\mathbf{a}, \mathbf{n}', \mathbf{n_{id}}) \cdot \mathbf{s}^*_{\mathbf{U}} \tag{244}$$
$$\mathbf{fresh}(\mathbf{n_{id}}) \tag{245}$$

Note that (240) holds from rule (Program).

Proof for $\Omega' \sim_{\delta'} \mathbf{\Omega}'$ **:** Let $\mathbf{\Omega}' = \langle \mathbf{\Sigma}', (\theta, [], \langle \mathbf{n_b}, \mathbf{0}, \mathbf{n}', \mathbf{true}, \mathbf{n_{id}} \rangle) : \mathbf{S}^* \rangle$ and $\delta' = \delta[(\mathsf{a} + \mathsf{i}, \mathsf{n_{id}}) \mapsto (\mathbf{n_b} + \mathsf{i} \times \mathtt{sz}(\tau), \mathbf{n_{id}}) \mid \mathsf{i} \in \{0 \ldots \mathsf{n} - 1\}]$. Notice that δ' is still a bijection because $\mathsf{n_{id}}$ and $\mathbf{n_{id}}$ are both fresh identifiers (245), (230) and that $\delta' \supseteq \delta$.

To prove $\Omega' \sim_{\delta'} \mathbf{\Omega}'$, it suffices to show that $\mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{a},\mathsf{n},\tau,\mathsf{n_{id}})} : \mathsf{array}\ \tau \sim_{\delta'} \langle [], \langle \mathbf{n_b}, \mathbf{0}, \mathbf{n}', \mathbf{true}, \mathbf{n_{id}} \rangle \rangle$, $\mathsf{H}' \sim_{\delta'} \mathbf{\Sigma}'.\mathbf{T}$ and $\mathsf{A}' \sim_{\delta'} \mathbf{\Sigma}'.\mathbf{A}$.[2]

To prove $\mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{a},\mathsf{n},\tau,\mathsf{n_{id}})} : \mathsf{array}\ \tau \sim_{\delta'} \langle [], \langle \mathbf{n_b}, \mathbf{0}, \mathbf{n}', \mathbf{true}, \mathbf{n_{id}} \rangle \rangle$ via (C-data-Valid-Ptr-Array), we must show (i) $\delta'(\mathsf{a}, \mathsf{n_{id}}) = (\mathbf{n_b}, \mathbf{n_{id}})$, which we have by definition from our choice of δ' , (ii) $\ell * \mathtt{sz}(\tau) = \mathbf{n}'$, which we have by (234) and $\mathsf{n} = \mathbf{n}$, and (iii) $(\mathsf{a} - \mathsf{a}) * \mathtt{sz}(\tau) = \mathbf{0}$, which trivially holds.

We will now proceed to prove $\mathsf{A}' \sim_{\delta'} \mathbf{\Sigma}'.\mathbf{A}$. We just need to prove the slot-equivalence for the newly allocated slot since the history equivalence is given by the relatedness of traces. That is, $(\mathsf{a}, \mathsf{n}, \mathsf{array}\ \tau, \mathsf{n_{id}}) \sim_{\delta'} (\mathbf{a}, \mathbf{n}', \mathbf{n_{id}})$. From the construction of δ' , (232), (234), we have

[2]The state components that are unchanged in this step are related up to δ (e.g., 235), therefore they are also related up to $\delta' \supseteq \delta$ by Lemma 17.

$$\delta'(\mathsf{a}, \mathsf{n_{id}}) = (\mathbf{a}, \mathbf{n_{id}}) \tag{246}$$
$$\forall 0 \leq \mathsf{i} < \mathsf{n}, \delta(\mathsf{a} + \mathsf{i}, \mathsf{n_{id}}) = (\mathbf{a} + \mathsf{i} * \mathbf{sz}(\tau), \mathbf{n_{id}}) \tag{247}$$
$$|\mathsf{n}| * \mathbf{sz}(\tau) = |\mathbf{n}'| \tag{248}$$

We have all the premises for (Slot-Eq-Prim) and thus we have the required proof.

Finally, we prove $\mathsf{H}' \sim_{\delta'} \mathbf{\Sigma}'.\mathbf{T}$. Notice that $\mathsf{H} \sim_{\delta} \mathbf{\Sigma}.\mathbf{T}$ from (236), therefore we only need to relate the final memories with respect to the initialization of the allocated buffer. In particular, we observe that the source allocator initializes the array as $\mathsf{H}' = \mathsf{H}[\mathsf{a} + \mathsf{j} \mapsto 0 \mid \mathsf{j} \in 0..\mathsf{n} - 1]$ via rule (Src-Alloc), and similarly the target allocator initializes the corresponding bytes in segment memory via rule (Seg-Alloc) , i.e., $\mathbf{T}' = \mathbf{T}[\mathbf{a} + \mathbf{j} \mapsto (\mathbf{0}, \mathbf{D}) \mid \mathbf{j} \in \mathbf{0}..\mathbf{n}' - \mathbf{1}]$ (241). Thus, we apply rule (Heap-Equiv) to (232), (247), and $0 \sim_{\delta} (\mathbf{0}, \mathbf{D})_0 \ldots (\mathbf{0}, \mathbf{D})_{\mathbf{sz}(\tau)-1}$ ((Heap-Value)) to each element of the array and obtain $\mathsf{H}' \sim_{\delta'} \mathbf{\Sigma}'.\mathbf{T}$, as desired.

Proof for $\alpha =_{\delta'} \alpha$: We show $\mathsf{alloca}^{(\mathsf{a},\mathsf{n},\tau,\mathsf{n_{id}})} =_{\delta'} \mathbf{salloc}(\langle \mathbf{n_b}, \mathbf{0}, \mathbf{n}', \mathbf{true}, \mathbf{n_{id}} \rangle)$ via rule (Tr-Rel-Allocate), where the pointer and the handle are allocated as shown above.

Case (S-Malloc-Single): Similar to the above case.

Case (S-Free): We first invert the judgment $\Omega \sim_{\delta} \mathbf{\Omega}$ via rule (Rt-Config) and rule (Rt-Free) and obtain that the source pointer and target handle are related, i.e., $\mathsf{v} = \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} \sim_{\delta} \langle \mathbf{b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle = \mathbf{v}$, where

$$\delta(\mathsf{b}, \mathsf{n_{id}}) = \mathbf{b}, \mathbf{n_{id}}$$
$$\ell * \mathbf{sz}(\tau) = \mathbf{n_l}$$
$$(\mathsf{a} - \mathsf{b}) * \mathbf{sz}(\tau) = \mathbf{n_o}$$

and the next instructions in the target configuration are $\mathbf{free}\ ; \mathbf{u32.const}\ \mathbf{0}$.[3] Then, to show that the target steps via rule (H-Free), we need to show that a segment with base address $\mathbf{b}$ is allocated. Since the source configuration generates event $\mathsf{alloc}\left(\mathsf{b}^{(\mathsf{a},\mathsf{n},\mathsf{w},\mathsf{n_{id}})}\right)$, we know that the source pointer points to the first address of an allocated buffer, i.e., $\mathsf{b} = \mathsf{a}$ and $(\mathsf{b}, \mathsf{n}, \mathsf{w}, \mathsf{n_{id}}) \in \mathsf{A.Allocated}$, by allocator rule (Src-Free). Then, because source and target allocator are related, i.e., $\mathsf{A} \sim_{\delta} \mathbf{A}$, we can invert the judgment via rule rule (Alloc-Eq) and rule (Slot-Eq-Prim) (the case for rule (Slot-Eq-Struct) is similar), and deduce that the corresponding target segment is also allocated, i.e., $(\mathbf{b}, \mathbf{n}, \mathbf{n_{id}}) \in \mathbf{A.Allocated}$. Thus, the target configuration steps first via rule (H-Free), generating event $\alpha = \mathbf{sfree}(\mathbf{v})$, which is related up to bijection $\delta' = \delta$ to the source event $\alpha = \mathsf{sfree}(\mathsf{v})$, by rule rule (Tr-Rel-Free) applied to $\mathsf{v} \sim_{\delta} \mathbf{v}$, which we have from

[3] The handle $\mathbf{v}$ is valid because the source execution is memory safe.

$\Omega \sim_\delta \mathbf{\Omega}$. Finally, the allocators step with related events and the new histories are equivalent by the relatedness of traces, thus the resulting allocator states are related, i.e., $\mathsf{A}' \sim_\delta \mathbf{A}'$, and the target configuration steps via rule rule (Const), consuming instruction **u32.const 0** and pushing **0** on the stack, which leads to a related configuration $\mathbf{\Omega}' \sim_\delta \Omega'$.

Case (S-Assign-Forge): Impossible due to non-ms step.

Case (S-Dereference-Forge): Impossible due to non-ms step.

Case (Struct-Field-Forge): Impossible due to non-ms step.

Case (S-Free-Forge): Impossible due to non-ms step.

Case (Src-Free-Nop): Impossible due to non-ms step. □

4.9 Connection Lemma

Lemma 23 (Connection Lemma)**.** If $[\![\Omega, \delta]\!] = \mathbf{\Omega}$ then $\exists\, \mathbf{\Omega}'$ such that $\mathbf{\Phi}^* \vdash \mathbf{\Omega} \xRightarrow{\epsilon} \mathbf{\Omega}'$ and $\Omega \sim_\delta \mathbf{\Omega}'$.

Proof. Invert rule (Compile-Config) and induct on $[\![\mathsf{P}, \Gamma_\mathsf{f} \vdash \mathsf{e_f} : \mathsf{w}]\!]^{\mathsf{exp}} = \mathbf{i}^*$. □

4.10 MS Preservation

We are now ready to prove that the compilation preserves memory safety: if the source configuration takes a memory safe step, then the compiled configuration also takes a memory safe step.

Theorem 3 (MS Preservation—Weak)**.** Let $\vdash \mathsf{P} : \mathsf{wt}$. If $[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M}$ then $\vdash \mathbf{RMS}\,(\mathbf{M})$.

Proof. From Theorem 2 (Type-Preservation of a Compiled Program), we have $\vdash \mathbf{M}$. Invoking Theorem 1 (Robust Memory-Safety), we have the proof. □

Theorem 4 (MS Preservation—Strong)**.** If

1. $\vdash \mathsf{P} : \mathsf{wt}$
2. $[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M}$
3. $[\![\mathsf{P}, \Delta]\!]^{\mathsf{glob}} = \mathbf{\Delta}$
4. $\Omega_0\,(\mathsf{P}) \xRightarrow{\overline{\alpha}^*} \Omega$
5. $\mathtt{MS}(\overline{\alpha})$

then

- $\mathbf{M.funcs} \vdash \mathbf{\Omega_0}\,(\mathbf{M}, \mathbf{n_f}) \xRightarrow{\overline{\alpha}} \mathbf{\Omega}$ and

- $\exists \delta, \Omega \sim_{\delta} \mathbf{\Omega}$ and
- $\overline{\alpha} =_{\delta} \overline{\alpha}$ and
- $\mathtt{MS}(\overline{\alpha})$

Proof. Applying Lemma 24 (Initial Configurations) to the first two given premises gives $[\![\Omega_0(\mathsf{P}), \varnothing]\!] = \mathbf{\Omega_0}(\mathbf{M}, \mathbf{n_f})$. Further applying Lemma 23 (Connection Lemma), we have $\mathbf{\Phi}^* \vdash \mathbf{\Omega_0}(\mathbf{M}, \mathbf{n_f}) \overset{\epsilon}{\Longrightarrow} \bar{\mathbf{\Omega}}$ and $\Omega_0(\mathsf{P}) \sim_{\varnothing} \bar{\Omega}$ where $\mathbf{\Phi}^* = \mathbf{M.funcs}$.

We have to prove

$$\mathbf{\Phi}^* \vdash \mathbf{\Omega_0}(\mathbf{M}, \mathbf{n_f}) \overset{\overline{\alpha}}{\Longrightarrow} \mathbf{\Omega} \tag{249}$$

$$\exists \delta, \Omega \sim_{\delta} \mathbf{\Omega} \tag{250}$$

$$\overline{\alpha} =_{\delta'} \overline{\alpha} \tag{251}$$

$$\mathtt{MS}(\overline{\alpha}) \tag{252}$$

Applying Theorem 2 (Type-Preservation of a Compiled Program) to the first three premises in the hypothesis, we have $\mathbf{\Delta} \vdash \mathbf{M}$. From the target config typing rule, ((Config)), we have $\mathbf{\Delta} \vdash \mathbf{\Omega_0}(\mathbf{M}, \mathbf{n_f})$. Let T, δ° and δ° be empty. Then, we trivially have $T =_{\delta^{\circ}} \Omega_0.\mathsf{A}$. From $\mathtt{MS}(\overline{\alpha})$ we have

$$\overline{\alpha} =_{\delta} \overline{\alpha} \tag{253}$$

$$\mathtt{MS}(\overline{\alpha}) \tag{254}$$

Expanding the definition of $\mathtt{MS}(\overline{\alpha})$ we have

$$\vdash T \overset{\overline{\alpha}}{\Longrightarrow} T' \tag{255}$$

Note that $\delta \supseteq \varnothing$. Also, (253) together with (255) and $\Omega_0(\mathsf{P}) \overset{\overline{\alpha}}{\Longrightarrow}{}^{*} \Omega$ gives $T' =_{\delta} \Omega.\mathsf{A}$. We now have the premises necessary to invoke Lemma 18 (Functional Correctness—Multiple Steps) and Lemma 28 (MS Preservation Strong—Multiple Steps). Thus,

$$\exists \delta \text{ and } \mathbf{\Omega}, \Omega \sim_{\delta} \mathbf{\Omega} \tag{256}$$

$$\mathbf{\Phi}^* \vdash \mathbf{\Omega_0} \overset{\overline{\alpha}}{\Longrightarrow} \mathbf{\Omega} \tag{257}$$

$$\overline{\alpha} =_{\delta} \overline{\alpha} \tag{258}$$

$$\vdash \mathbf{isValid}\ \mathbf{\Omega} \tag{259}$$

$$\boldsymbol{\delta} = \delta \circ \delta^{-1} \tag{260}$$

$$\overline{\alpha} =_{\boldsymbol{\delta}} \overline{\alpha} \tag{261}$$

$$T' =_{\boldsymbol{\delta}} \mathbf{\Omega.A} \tag{262}$$

To prove $\mathsf{MS}(\overline{\alpha})$, we need

$$\overline{\alpha} =_{\delta} \overline{\alpha} \tag{263}$$

$$\vdash T \overset{\overline{\alpha}}{\rightsquigarrow} T' \tag{264}$$

which we have from (261) and (255). Hence proved. □

Lemma 24 (Initial Configurations)**.** If

1. $\vdash \mathsf{P} : \mathsf{wt}$
2. $[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M}$

then $[\![\Omega_0\,(\mathsf{P})\,, \varnothing]\!] = \mathbf{\Omega_0}\,(\mathbf{M}, \mathbf{n_f})$

Proof. Expand the definitions and the proof follows from (Compile-Config). □

Corollary 2 (Functional Correctness)**.** If

1. $\vdash \mathsf{P} : \mathsf{wt}$
2. $[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M}$
3. $\Omega_0\,(\mathsf{P}) \overset{\overline{\alpha}}{\Longrightarrow}{}^{*} \Omega$
4. $\mathsf{MS}(\overline{\alpha})$

then

- $\mathbf{M.funcs} \vdash \mathbf{\Omega_0}\,(\mathbf{M}, \mathbf{n_f}) \overset{\overline{\alpha}}{\Longrightarrow} \mathbf{\Omega}$ and
- $\exists \delta$, $\Omega \sim_{\delta} \mathbf{\Omega}$
- $\overline{\alpha} =_{\delta} \overline{\alpha}$

Proof. From Theorem 4 (MS Preservation—Strong). □

Lemma 25 (Offsets out-of-bounds)**.** Let $\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} \sim_{\delta} \langle \mathbf{b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle$.

1. If $\mathsf{a} - \mathsf{b} \geq \ell$ then $\mathbf{n_o} \geq \mathbf{n_l}$.
2. If $\mathsf{a} - \mathsf{b} < \ell$ then $\mathbf{n_o} < \mathbf{n_l}$.
3. If $\mathsf{a} \geq \mathsf{b}$ then $\mathbf{n_0} \geq \mathbf{0}$.
4. If $\mathsf{a} < \mathsf{b}$ then $\mathbf{n_0} < \mathbf{0}$.

Proof. From the premises of (C-data-Valid-Ptr-Array), we have

$$[\![\mathsf{w}]\!] = \tau \tag{265}$$
$$\delta(\mathsf{b}, \mathsf{n_{id}}) = \mathbf{b}, \mathbf{n_{id}} \tag{266}$$
$$\ell * \mathtt{sz}\,(\tau) = \mathbf{n_l} \tag{267}$$
$$(\mathsf{a} - \mathsf{b}) * \mathtt{sz}\,(\tau) = \mathbf{n_o} \tag{268}$$

Let $(\mathsf{a} - \mathsf{b}) \geq \ell$. So, $(\mathsf{a} - \mathsf{b}) * \mathtt{sz}\,(\tau) \geq \ell * \mathtt{sz}\,(\tau)$. From (267) and (268), we have $\mathbf{n_o} \geq \mathbf{n_l}$. Let $(\mathsf{a} - \mathsf{b}) < \ell$. So, $(\mathsf{a} - \mathsf{b}) * \mathtt{sz}\,(\tau) \leq \ell * \mathtt{sz}\,(\tau)$. From (267) and (268), we have $\mathbf{n_o} < \mathbf{n_l}$. Proof for $\mathbf{n_0} \geq \mathbf{0}$ (resp. $\mathbf{n_0} < \mathbf{0}$) follows trivially from (268) and by the fact that $\mathtt{sz}\,(\tau) > 0$. □

4.10.1 MS—Multiple Steps

Lemma 26 (Trace Conversion)**.** If

1. $\overline{\alpha} =_\delta \overline{\alpha}$ and
2. $\overline{\alpha} \;=_\delta\; \overline{\alpha}$

then $\overline{\alpha} =_{\boldsymbol{\delta}} \overline{\alpha}$ where $\boldsymbol{\delta} = \delta \circ \delta^{-1}$.

Proof. Induction on $\overline{\alpha} \;=_\delta\; \overline{\alpha}$.

Case (Empty): Trivial.

Case (Tr-Concat): We have

$$\alpha \cdot \overline{\alpha} =_\delta \overline{\alpha} \tag{269}$$
$$\alpha \cdot \overline{\alpha}' \;=_\delta\; \alpha \cdot \overline{\alpha}' \tag{270}$$

We need to prove

$$\alpha \cdot \overline{\alpha} =_{\boldsymbol{\delta}} \overline{\alpha}$$

Applying Lemma 4 (Splitting Source Trace Equivalence) to (269), we have

$$\overline{\alpha} = \overline{\alpha}' \cdot \overline{\alpha}'' \tag{271}$$
$$\alpha =_\delta \overline{\alpha}' \tag{272}$$
$$\overline{\alpha} =_\delta \overline{\alpha}'' \tag{273}$$

Inverting (270) using (Tr-Concat), we have

$$\alpha \;=_\delta\; \alpha \tag{274}$$
$$\overline{\alpha} \;=_\delta\; \overline{\alpha} \tag{275}$$

Applying I.H to (273) we have that: If

- $\overline{\alpha} =_\delta \overline{\alpha}''$ and

- $\overline{\alpha} =_\delta \overline{\alpha}$

then $\overline{\alpha} =_\delta \overline{\alpha}''$.

Applying (273) and (275), we thus have

$$\overline{\alpha} =_\delta \overline{\alpha}'' \tag{276}$$

We still need to prove

$$\alpha =_\delta \overline{\alpha}' \tag{277}$$

Case analysis on (274) gives the following:

Case (Tr-Rel-Read): Given $\mathsf{read}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}, \mathsf{v}) =_\delta \mathbf{read}_\tau\,(\langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}}\rangle_{\mathbf{F}})$.
From the premises, we have:

$$\vdash \mathsf{v} : \tau \tag{278}$$
$$[\![\tau]\!] = \tau \tag{279}$$
$$\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} \sim_\delta \langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}}\rangle_{\mathbf{F}} \tag{280}$$

We have $\alpha = \mathsf{read}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}, \mathsf{v})$ and $\alpha = \mathbf{read}_\tau\,(\langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}}\rangle_{\mathbf{F}})$. From (292) and (Tr-Read-Authentic), we have

$$\mathsf{read}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}, \mathsf{v}) =_\delta \mathtt{read}(a^c) \cdot \mathtt{read}((a+1)^c) \ldots \mathtt{read}((a+n-1)^c) \tag{281}$$

Inverting (281) using (Tr-Read-Authentic), we have

$$n = \mathtt{sz}\,(\mathsf{v}) \tag{282}$$
$$\delta(\mathsf{b}) = b^c \tag{283}$$
$$\mathsf{o} = \mathsf{a} - \mathsf{b} \tag{284}$$
$$a = b + (\mathsf{o} * n) \tag{285}$$

Our goal is to prove

$$\mathbf{read}_\tau\,(\langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}\rangle_{\mathbf{F}}) =_\delta \mathtt{read}(a^c) \cdot \mathtt{read}((a+1)^c) \ldots \mathtt{read}((a+n-1)^c) \tag{286}$$

That is, inverting (286) using ((Tr-Read)), we need the following:

$$n = \mathtt{sz}\,(\tau) \tag{287}$$
$$\boldsymbol{\delta}(\mathbf{b}) = b^c \tag{288}$$
$$a = b + \mathbf{o} \tag{289}$$

From (278) and (279), we have (282) implies (287). Consider the following:

$$\delta(\mathsf{b}) \implies \delta \circ \delta^{-1}(\mathsf{b}) \implies \delta(\mathsf{b}) = b^c \qquad \text{(from (283))}$$

Thus (288) holds.

Inverting (280) using rule (C-data-Valid-Ptr-Array), we have

$$\delta(\mathsf{b}) = \mathbf{b} \tag{290}$$
$$\ell * \mathtt{sz}\,(\mathsf{w}) = \ell \tag{291}$$
$$(\mathsf{a} - \mathsf{b}) * \mathtt{sz}\,(\mathsf{w}) = \mathbf{o} \tag{292}$$

Rewriting (285), we have

$$\begin{aligned} & a = b + (\mathsf{a} - \mathsf{b}) * n \\ \implies & a = b + \mathbf{o} \iff \mathtt{sz}\,(\mathsf{w}) = n \qquad \text{(from (292))} \end{aligned}$$

It remains to prove that $\mathtt{sz}\,(\mathsf{w}) = n$. Note that from rule (Tr-Rel-Read), we have $\mathsf{w} \neq \mathsf{struct}\ \_$. Then $\mathsf{w} = \tau$. (see rule (S-Malloc-Single) and rule (S-Malloc-Array)). Since $\mathtt{sz}\,(\tau) = \mathtt{sz}\,(\mathsf{v})$, we have $\mathtt{sz}\,(\mathsf{w}) = n$. Thus we have (289) which inturn implies that (286).

Case (Tr-Rel-Write) : Given $\mathsf{write}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}, \mathsf{v}) =_\delta \mathbf{write}_\tau\,(\langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}} \rangle_{\mathbf{F}})$. Proof similar to the above case.

Case (Tr-Rel-Allocate): Given $\mathsf{salloc}(\mathsf{a}, \mathsf{n}, \mathsf{w}, \mathsf{n_{id}}) =_\delta \mathbf{salloc}(\mathbf{a}, \mathbf{n}, \mathbf{n_{id}})$. From the premises, we have:

$$\delta(\mathsf{a}) = \mathbf{a} \tag{293}$$
$$\mathsf{n} * \mathtt{sz}\,(\mathsf{w}) = \mathbf{n} \tag{294}$$

Depending on whether w is $\mathsf{struct}\ \_$, we have:

$$\mathsf{salloc}(\mathsf{a}, \mathsf{n}, \mathsf{w}, \mathsf{n_{id}}) =_\delta \mathtt{alloc}(n, a^c, \phi_0)$$

or

$$\mathsf{salloc}(\mathsf{a}, 1, \mathsf{struct\ s}, \mathsf{n_{id}}) =_\delta \mathtt{alloc}(\mathtt{srcsz}\,(\mathsf{struct\ s}), a^c, \phi)$$

For the first case, we need to prove

$$\mathbf{salloc}(\mathbf{a}, \mathbf{n}, \mathbf{n_{id}}) =_\delta \mathtt{alloc}(n, a^c, \phi)$$

For the second case, we need to prove

$$\mathbf{salloc}(\mathbf{a}, \mathbf{n}, \mathbf{n_{id}}) =_\delta \mathtt{alloc}(n, a^c, \phi)$$

These conclusions follow from (Tr-SAlloc) and (Tr-SAlloc) ((Tr-SAlloc-Struct) and (Tr-SAlloc), respectively).

Case (Tr-Rel-Free): Given

$$\mathsf{free}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}) =_\delta \mathtt{sfree}\,(a^c) \tag{295}$$

$$\mathsf{free}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}) =_\delta \mathbf{free}(\langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}} \rangle) \tag{296}$$

We need to prove

$$\mathbf{sfree}(\mathbf{h}, \mathbf{n_{id}}) =_{\boldsymbol{\delta}} \mathtt{sfree}\,(a^c)$$

From (296) and (Tr-Rel-Free), we have

$$(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}) \sim_\delta \langle \mathbf{b}, \mathbf{o}, \ell, \mathbf{1}, \mathbf{n_{id}} \rangle$$

We have

$$\delta^{-1}(\mathbf{b}, \mathbf{n_{id}}) = (\mathsf{b}, \mathsf{n_{id}}) \tag{297}$$

$$\delta(\mathsf{b}, \mathsf{n_{id}}) = a^{(c,\_)} \tag{298}$$

Rule (Tr-Sfree) is immediate from (298).

Case (Tr-Rel-Read-Forge) : Impossible case as there is no $\overline{\alpha}$ such that $\mathsf{fread}(\mathsf{a}, \mathsf{v}) =_\delta \overline{\alpha}$.

Case (Tr-Rel-Write-Forge): Impossible case as there is no $\overline{\alpha}$ such that $\mathsf{fwrite}(\mathsf{a}, \mathsf{v}) =_\delta \overline{\alpha}$.

Case (Tr-Rel-Free-Forge): Impossible case as there is no $\overline{\alpha}$ such that $\mathsf{ffree\ a} =_\delta \overline{\alpha}$.

□

Lemma 27 (Alloc Conversion)**.** If

1. $T =_\delta \Omega.\mathsf{A}$ and
2. $\Omega \sim_\delta \boldsymbol{\Omega}$ and

then $T =_{\boldsymbol{\delta}} \boldsymbol{\Omega}.\mathbf{A}$ where $\boldsymbol{\delta} = \delta \circ \delta^{-1}$.

Proof. Given:

$$T =_\delta \Omega.\mathsf{A} \tag{299}$$

$$\Omega \sim_\delta \boldsymbol{\Omega} \tag{300}$$

$$\boldsymbol{\delta} = \delta \circ \delta^{-1} \tag{301}$$

Case $\mathsf{w} \neq \mathsf{struct\_}$**:** From (299), we have

$$\forall(\mathsf{a}, \mathsf{n}, \mathsf{w}) \in \lfloor \mathsf{A} \rfloor_1, n = \mathsf{n} * \mathtt{sz}\,(\mathsf{w}) \tag{302}$$

$$\delta(\mathsf{a}) = a^{(c,\perp)} \ldots \delta(\mathsf{a} + \mathsf{n} - 1) = (a + n - 1)^{(c,\perp)} \tag{303}$$

$$T(a) = \ldots T(a+n-1) = (c, \bot) \tag{304}$$

From (300), we have $\Omega.\mathsf{A} \sim_\delta \mathbf{\Omega.A}$. From (Alloc-Eq), we have

$$\forall (\mathsf{a}, \mathsf{n}, \mathsf{w}) \in \lfloor \Omega.\mathsf{A} \rfloor_1, (\mathsf{a}, \mathsf{n}, \mathsf{w}) \sim_\delta (\mathbf{a}, \mathbf{n}) \wedge (\mathbf{a}, \mathbf{n}) \in \lfloor \mathbf{\Omega.A} \rfloor_1 \tag{305}$$

Thus, we have $(\mathsf{a}, \mathsf{n}, \mathsf{w}) \sim_\delta (\mathbf{a}, \mathbf{n})$. Inverting using (Slot-Eq-Prim) or **??**,

$$\delta(\mathsf{a}) = \mathbf{a} \tag{306}$$
$$|\mathsf{n}| * \mathtt{sz}(\mathsf{w}) = |\mathbf{n}| \tag{307}$$

To prove $T =_\delta \mathbf{\Omega.A}$, we need the following:

$$\forall (\mathbf{a}, \mathbf{n}) \in \lfloor \mathbf{\Omega.A} \rfloor_1, n = \mathbf{n} \tag{308}$$
$$\forall i \in \{0, n-1\}, \boldsymbol{\delta}(\mathbf{a}+i) = a^{(c,\_)} \tag{309}$$
$$T(\boldsymbol{\delta}(\mathbf{a})) = \ldots T(\boldsymbol{\delta}(\mathbf{a}) + n - 1) = (c, \_) \tag{310}$$

This implies,

$$\forall (\mathbf{a}, \mathbf{n}) \in \lfloor \mathbf{\Omega.A} \rfloor_1, n = \mathbf{n} \tag{311}$$
$$\forall i \in \{0, n-1\}, \delta \circ \delta^{-1}(\mathbf{a}+i) = a^{(c,\_)} \tag{312}$$
$$T(\delta \circ \delta^{-1}(\mathbf{a})) = \ldots T(\delta \circ \delta^{-1}(\mathbf{a}) + n - 1) = (c, \_) \tag{313}$$

From (306), we have $\delta^{-1}(\mathbf{a}) = \mathsf{a}$. Thus, $\delta \circ \delta^{-1}(\mathbf{a}) = \delta(\mathsf{a})$. Rewriting the above, we need to prove:

$$\forall (\mathbf{a}, \mathbf{n}) \in \lfloor \mathbf{\Omega.A} \rfloor_1, n = \mathbf{n} \tag{314}$$
$$\delta(\mathsf{a}) = a^{(c,\_)} \ldots \delta(\mathsf{a}+n-1) = (a+n-1)^{(c,\_)} \tag{315}$$
$$T(a) = \ldots T(a+n-1) = (c, \_) \tag{316}$$

We already have (315) and (316) from (303) and (304). (302) and (307) together imply (314). Thus, we have the proof for $T =_\delta \mathbf{\Omega.A}$.

Case $\mathsf{w} = \mathsf{struct\ s}$**:** From (299), we have

$$\forall (\mathsf{a}, 1, \mathsf{struct\ s}) \in \lfloor \mathsf{A} \rfloor_1, \mathsf{D(s)} = \{\mathsf{f}_1 : \tau_1, \mathsf{f}_2 : \tau_2 \ldots \mathsf{f}_k : \tau_k\} \implies$$
$$n_0 = 0 \implies n_i = \mathtt{srcsz}(\tau_i) \implies$$
$$\delta(\mathsf{a}+i) = (a + \Sigma_i n_i)^{(c, s_i)} \wedge$$
$$T(a + \Sigma_i n_i) = T(a + \Sigma_i n_i + 1) = \cdots = T(a + \Sigma_i n_{i+1} - 1) = (c, s_i)$$

Note that all fields have same color but different shades. When $\boldsymbol{\delta}{=}\delta \circ \delta^{-1}$, we get different shades for the addresses corresponding to fields. However, the definition $T =_\delta \mathbf{\Omega.A}$ ignores the shades. Thus following an argument similar to the previous case, we have the proof.

□

Lemma 28 (MS Preservation Strong—Multiple Steps)**.** If

1. $\Omega \overset{\overline{\alpha}}{\Longrightarrow}{}^{*} \Omega'$ and
2. $[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M}$ and
3. $[\![\mathsf{P}, \Delta]\!]^{\mathsf{glob}} = \boldsymbol{\Delta}$ and
4. $\Omega \sim_{\delta} \boldsymbol{\Omega}$ and
5. $\boldsymbol{\Delta} \vdash \boldsymbol{\Omega}$ and
6. $\vdash \mathbf{isValid}\ \boldsymbol{\Omega}$ and
7. $T =_{\delta} \Omega.\mathsf{A}$ and
8. $\delta \subseteq \delta'$ and
9. $\overline{\alpha} =_{\delta'} \overline{\alpha}$ and
10. $T \xrightarrow{\overline{\alpha}} T'$ and
11. $T' =_{\delta'} \Omega'.\mathsf{A}$

then

(i) $\boldsymbol{\delta'} = \delta' \circ \delta'^{-1}$

(ii) $\overline{\alpha} =_{\boldsymbol{\delta'}} \overline{\alpha}$ and

(iii) $T' =_{\boldsymbol{\delta'}} \boldsymbol{\Omega'}.\mathbf{A}$

Proof. Given:

$$\Omega \overset{\overline{\alpha}}{\Longrightarrow}{}^{*} \Omega' \tag{317}$$
$$[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M} \tag{318}$$
$$[\![\mathsf{P}, \Delta]\!]^{\mathsf{glob}} = \boldsymbol{\Delta} \tag{319}$$
$$\Omega \sim_{\delta} \boldsymbol{\Omega} \tag{320}$$
$$\boldsymbol{\Delta} \vdash \boldsymbol{\Omega} \tag{321}$$
$$\vdash \mathbf{isValid}\ \boldsymbol{\Omega} \tag{322}$$
$$T =_{\delta} \Omega.\mathsf{A} \tag{323}$$
$$\delta \subseteq \delta' \tag{324}$$
$$\overline{\alpha} =_{\delta'} \overline{\alpha} \tag{325}$$
$$T \xrightarrow{\overline{\alpha}} T' \tag{326}$$
$$T' =_{\delta'} \Omega'.\mathsf{A} \tag{327}$$

We have to prove

$$\boldsymbol{\delta'} = \delta' \circ \delta'^{-1} \tag{328}$$

$$\overline{\alpha} =_{\boldsymbol{\delta'}} \overline{\alpha} \tag{329}$$

$$T' =_{\boldsymbol{\delta'}} \boldsymbol{\Omega'}.\mathbf{A} \tag{330}$$

Applying Lemma 18 (Functional Correctness—Multiple Steps) from premises (317) to (327), we have

$$\exists \delta' \supseteq \delta \text{ and } \boldsymbol{\Omega'}, \Omega' \sim_{\boldsymbol{\delta'}} \boldsymbol{\Omega'} \tag{331}$$

$$\mathbf{M.funcs} \vdash \boldsymbol{\Omega} \overset{\overline{\alpha}}{\Longrightarrow} \boldsymbol{\Omega'} \tag{332}$$

$$\overline{\alpha} =_{\delta'} \overline{\alpha} \tag{333}$$

Let $\boldsymbol{\delta} = \delta \circ \delta^{-1}$ and $\boldsymbol{\delta'} = \delta' \circ \delta'^{-1}$.

Applying Lemma 26 (Trace Conversion) to (325) and (333), we have (328).

Applying Lemma 27 (Alloc Conversion) to (327) and (331), we have (329). □

4.11 Non-MS Preservation

We also prove that the compilation preserves non-memory safety in a safe way: if the source configuration takes a step that is memory unsafe, then the compiled configuration traps.

Theorem 5 (Non-MS Preservation—Top Level)**.** If

1. $\vdash \mathsf{P} : \mathsf{wt}$ and
2. $[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M}$ and
3. $\Omega_0(\mathsf{P}) \overset{\overline{\alpha}}{\Longrightarrow}^* \Omega$ and
4. $\mathtt{MS}(\overline{\alpha})$ and
5. $\Omega \overset{\alpha}{\Longrightarrow} \Omega'$ and
6. $\neg\, \mathtt{MS}(\overline{\alpha} \cdot \alpha)$

then

- $\mathbf{M.funcs} \vdash \boldsymbol{\Omega_0}(\mathbf{M}, \mathbf{n_f}) \overset{\overline{\alpha}}{\Longrightarrow} \boldsymbol{\Omega}$ and
- $\exists \delta, \Omega \sim_\delta \boldsymbol{\Omega}$ and
- $\overline{\alpha} =_\delta \overline{\alpha}$ and
- $\mathtt{MS}(\overline{\alpha})$ and

- $\mathbf{M.funcs} \vdash \mathbf{\Omega} \xRightarrow{\mathbf{trap}} \mathbf{\Omega'}$

Proof. Given:

$$\vdash \mathsf{P} : \mathsf{wt} \tag{334}$$
$$[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M} \tag{335}$$
$$\Omega_0\,(\mathsf{P}) \xRightarrow{\overline{\alpha}^*} \Omega \tag{336}$$
$$\mathsf{MS}(\overline{\alpha}) \tag{337}$$
$$\Omega \xRightarrow{\alpha} \Omega' \tag{338}$$
$$\neg\mathsf{MS}(\overline{\alpha} \cdot \alpha) \tag{339}$$

Invoke Theorem 4 (MS Preservation—Strong) on (334) to (337) we have

$$\mathbf{M.funcs} \vdash \mathbf{\Omega_0}\,(\mathbf{M}, \mathbf{n_f}) \xRightarrow{\overline{\alpha}} \mathbf{\Omega} \tag{340}$$
$$\exists \delta, \Omega \sim_\delta \mathbf{\Omega} \tag{341}$$
$$\overline{\alpha} =_\delta \overline{\alpha} \tag{342}$$
$$\mathsf{MS}(\overline{\alpha}) \tag{343}$$

Now induct on on $\Omega \xRightarrow{\alpha} \Omega'$. Cases (S-Call) and (S-Return) are invalid as they do not emit any relevant events. Only (S-Step) is valid. Invoke Lemma 29 (Non-MS Preservation—Primitive Step) on the premise of (S-Step) to get the required proof. □

Lemma 29 (Non-MS Preservation—Primitive Step)**.** If

1. $[\![\mathsf{P}]\!]^{\mathsf{prog}} = \mathbf{M}$ and
2. $\mathsf{M} \vdash \Omega \xrightarrow{\alpha}\!\!\!\!\rightarrow \Omega'$ and
3. $T =_\delta \Omega.\mathsf{A}$ and
4. (a) $\forall \overline{\alpha}, \alpha \neq_\delta \overline{\alpha}$ or

   (b) $\exists \overline{\alpha}, \alpha =_\delta \overline{\alpha}$ and $\vdash T \overset{\overline{\alpha}}{\not\Rightarrow}$

   and
5. $\Omega \sim_\delta \mathbf{\Omega}$ and
6. $\vdash \mathbf{isValid}\ \mathbf{\Omega}$

then

- $\mathbf{M.funcs} \vdash \mathbf{\Omega} \xRightarrow{\mathbf{trap}} \mathbf{\Omega'}$

Proof. We prove by case analysis on $\mathsf{M} \vdash \Omega \xrightarrow{\alpha} \Omega'$. Interesting cases are (S-Assign), (S-Arr-Assign), (S-Assign-Forge), (S-Dereference), (S-Arr-Dereference), (S-Dereference-Forge), (S-Free), (Src-Free-Nop) and (S-Free-Forge). We prove for assignment and free; proof for remaining cases (including dereference) follows analogously.

Case (S-Assign):

$$\mathsf{M} \vdash \mathsf{P}, \theta_{\mathsf{f}}; \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright *\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} := \mathsf{v} \xrightarrow{\mathsf{write}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})},\mathsf{v})} \mathsf{P}, \theta_{\mathsf{f}}; \mathsf{K}, \mathsf{H}', \mathsf{A} \triangleright \mathsf{v}$$

We have $\alpha = \mathsf{write}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}, \mathsf{v})$. Let T and δ be such that $T =_\delta \mathsf{A}$ and $\mathsf{write}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}, \mathsf{v}) =_\delta \overline{\alpha}$. Inverting (Tr-Write-Authentic), we have $\overline{\alpha} = \mathtt{write}(a^{c,s}) \cdot \mathtt{write}((a+1)^{c,s}) \ldots \mathtt{write}((a+n-1)^{c,s})$ where $n = \mathtt{sz}(\mathsf{v})$ and $\delta(\mathsf{b}, \mathsf{n_{id}}) = b^{c,s}$ and $a = b + (\mathsf{o} * n)$.

We also have that $\vdash T \overset{\overline{\alpha}}{\rightsquigarrow}$. From (MS-Write), we then have that there exists a' such that

$$T(a') \neq A(c,s) \tag{344}$$

This implies either a' is allocated or freed. Consider the first case. Without loss of generality assume that $a' = a + n - 1$. From the given condition $T =_\delta \mathsf{A}$, we have $T(b) = \cdots = T(b + (\ell * n) - 1) = (c,s)$. This implies, $a' \notin (b, b + (\ell * n) - 1)$. Thus

$$a' < b$$

or

$$a' > b + (\ell * n) - 1$$

Once again, without loss of generality, assume that the latter holds. Thus,

$$\begin{aligned} & a + n - 1 > b + (\ell * n) - 1 \\ \implies & b + (\mathsf{o} * n) + n - 1 > b + (\ell * n) - 1 \\ \implies & \mathsf{o} > \ell - 1 \\ \implies & \mathsf{o} \geq \ell \end{aligned}$$

We are already given $\mathsf{P}, \theta_{\mathsf{f}}; \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright *\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} := \mathsf{v} \sim_\delta \langle \mathbf{\Sigma}, (\theta_{\mathbf{f}}, \boldsymbol{\tau}.\mathbf{segstore}; \mathbf{u32.const\ 0}, (\mathbf{v}; \mathbf{a})) : \mathbf{S}^* \rangle$. From (Rt-Config) and (Rt-Assign), we have

$$\mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} : \mathsf{ptr}\,\mathsf{w} \sim_\delta \langle [], \mathbf{a} \rangle \tag{345}$$

$$\mathsf{P}, \Gamma \vdash \mathsf{v} : \mathsf{w} \sim_\delta \langle [], \mathbf{v} \rangle \tag{346}$$

$$\mathsf{A} \sim_\delta \mathbf{\Sigma.A} \tag{347}$$

From (C-data-Valid-Ptr-Array), we have $\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} \sim_\delta \langle \mathbf{b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle$. Applying Lemma 25 (Offsets out-of-bounds) to $\mathsf{o} \geq \ell$, we have that

$\mathbf{n_o} \geq \mathbf{n_l}$. This implies that $\mathbf{n_o} + |\tau| > \mathbf{n_l}$ and thus takes the target step (H-Store-Trap).

Consider the latter case where a' is freed. That is,

$$T(a') = F(c', s') \tag{348}$$

This implies a belongs to a freed slot in source allocator A. From (C-data-Valid-Ptr-Array), we have $\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} \sim_\delta \langle \mathbf{b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle$. From (347) and rule (Alloc-Eq), we get that the target handle belongs to a free slot in the target allocator $\mathbf{\Sigma.A}$. Thus, the target takes the step (H-Store-Trap) due to the premise $\mathbf{n_{id}} \in \mathbf{A.freed}$.

Hence proved.

Case (S-Assign-Forge): Given:

$$\mathsf{M} \vdash \mathsf{P}, \theta_\mathsf{f}; \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright *\mathsf{a} := \mathsf{v} \xrightarrow{\mathsf{fwrite(a,v)}} \mathsf{P}, \theta_\mathsf{f}; \mathsf{K}, \mathsf{H}', \mathsf{A} \triangleright \mathsf{v}$$

We have $\alpha = \mathsf{fwrite(a, v)}$. Thus, $\exists \overline{\alpha}$, such that $\mathsf{fwrite(a, v)} =_\delta \overline{\alpha}$.

We are already given $\mathsf{P}, \theta_\mathsf{f}; \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright *\mathsf{a} := \mathsf{v} \sim_\delta \langle \mathbf{\Sigma}, (\theta_\mathbf{f}, \tau\mathbf{.segstore}; \mathbf{u32.const\ 0}, (\mathbf{v}; \mathbf{a})) : \mathbf{S}^* \rangle$. From (Rt-Config) and (Rt-Assign-forge), we have

$$\mathsf{P}, \Gamma \vdash \mathsf{a} : \mathsf{ptr\,w} \sim_\delta \langle [], \mathbf{a} \rangle \tag{349}$$
$$\mathsf{P}, \Gamma \vdash \mathsf{v} : \mathsf{w} \sim_\delta \langle [], \mathbf{v} \rangle \tag{350}$$
$$\mathsf{A} \sim_\delta \mathbf{\Sigma.A} \tag{351}$$

Consider (349). From (C-data-Ptr-corrupt), we have $\mathbf{a} = \langle \mathbf{b}, \mathbf{o}, \mathbf{l}, \mathbf{0}, \mathbf{n_{id}} \rangle$. Hence, from (H-Store-Trap), we thus have $\mathbf{M.funcs} \vdash \mathbf{\Omega} \xRightarrow{\mathbf{trap}} \mathbf{\Omega'}$.

Case (S-Free-Forge): Given:

$$\mathsf{M} \vdash \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright \mathsf{free(a)} \xrightarrow{\mathsf{ffree\ a}} \mathsf{P}, \mathsf{K}, \mathsf{H}', \mathsf{A}' \triangleright 0$$

Let $\alpha = \mathsf{ffree\ a}$ and $\alpha = \mathbf{trap}$. From (Rt-Free-Forge), we have

$$\mathsf{P}, \Gamma \vdash \mathsf{free(a)} : \mathsf{int} \sim_\delta \langle \mathbf{segfree}; \mathbf{u32.const\ 0}, \mathbf{v} \rangle$$

with premises:

$$\mathsf{P}, \Gamma \vdash \mathsf{a} : \mathsf{ptr\,w} \sim_\delta \langle [], \mathbf{v} \rangle$$

and inturn from (C-data-Ptr-corrupt), we have

$$\mathsf{a} \sim_\delta \langle \mathbf{b}, \_, \_, \mathbf{0}, \mathbf{n_{id}} \rangle$$

This implies $\mathbf{v} = \langle \mathbf{b}, \_, \_, \mathbf{0}, \mathbf{n_{id}} \rangle$. This satisfies the premise of (H-Free-trap) and thus

$$\mathbf{\Phi}^* \vdash \langle \mathbf{\Sigma}, \theta, \mathbf{segfree} : \mathbf{i}^*, \mathbf{v} : \mathbf{v}^* \rangle \xrightarrow{\mathbf{trap}} \langle \mathbf{\Sigma}, \theta, \epsilon, \epsilon \rangle$$

Case (Src-Free-Nop): This handles the case when an authentic pointer is freed in an incorrect way—either the pointer is in the middle of an allocated slot or has already been freed. Given:

$$\mathsf{M} \vdash \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright \mathsf{free}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}) \xrightarrow{\mathsf{free}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})})} \mathsf{P}, \mathsf{K}, \mathsf{H}, \mathsf{A} \triangleright 0$$

From $\Omega \sim_\delta \mathbf{\Omega}$ and (Rt-Free), we have

$$\mathsf{P}, \Gamma \vdash \mathsf{free}(\mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})}) : \mathsf{int} \sim_\delta \langle \mathbf{segfree; u32.const\ 0}, \mathbf{v} \rangle$$

From the premises of (Rt-Free), we have

$$\mathsf{P}, \Gamma \vdash \mathsf{a}^{(\mathsf{b},\ell,\mathsf{w},\mathsf{n_{id}})} : \mathsf{ptr}\,\mathsf{w} \sim_\delta \langle [], \mathbf{v} \rangle$$

We have two cases: $\mathsf{a} \neq \mathsf{b}$ and $\mathsf{a} = \mathsf{b}$.

Case $\mathsf{a} \neq \mathsf{b}$: From (C-data-Valid-Ptr-Array) and (C-data-Valid-Ptr-Single), we have

$$\mathsf{a}^{(\mathsf{b},\ell,\tau,\mathsf{n_{id}})} \sim_\delta \langle \mathbf{b}, \mathbf{n_o}, \mathbf{n_l}, \mathbf{1}, \mathbf{n_{id}} \rangle$$

From the premises of (C-data-Valid-Ptr-Array), we have $(\mathsf{a} - \mathsf{b})$ * $\mathbf{sz}(\mathsf{w}) = \mathbf{n_o}$. Since $\mathsf{a} \neq \mathsf{b}$, we have $\mathbf{n_o} \neq \mathbf{0}$. Thus, target takes the step (H-Free-trap).

From the premises of (C-data-Valid-Ptr-Single), we have $(\mathsf{a} - \mathsf{b}) = \mathbf{n_o} = 0$. Thus $\mathsf{a} = \mathsf{b}$—a contradiction. Thus (C-data-Valid-Ptr-Single) does not hold.

Case $\mathsf{a} = \mathsf{b}$: From the premises of (C-data-Valid-Ptr-Array) and (C-data-Valid-Ptr-Single), we have $\mathbf{n_o} = \mathbf{0}$. Assume (H-Free). Then,

$$\langle \mathbf{T}, \mathbf{A} \rangle \xhookrightarrow{\mathbf{sfree(a,n_{id})}} \langle \mathbf{T}', \mathbf{A}' \rangle$$

From the premises of (Seg-Free),

$$\mathbf{s_A^*} = \mathbf{s_1^*} \cdot (\mathbf{b}, \_, \mathbf{n_{id}}) \cdot \mathbf{s_2^*}$$

Given $\Omega \sim_\delta \mathbf{\Omega}$. This implies, from (Alloc-Eq), we have that $\mathsf{s_A}^* \sim_\delta \mathbf{s_A}^*$. Thus, $(\mathbf{b}, \_, \mathbf{n_{id}}) \in \mathbf{s_A^*}$ implies $(\mathsf{b}, \ell, \mathsf{w}, \mathsf{n_{id}}) \in \mathsf{s_A^*}$. However, from (Src-Free-Nop), we have that $(\mathsf{b}, \ell, \mathsf{w}, \mathsf{n_{id}}) \notin \mathsf{s_A^*}$—a contradiction. This is because we assumed that the allocator state changes in (H-Free). Thus,

$$\langle \mathbf{T}, \mathbf{A} \rangle \xhookrightarrow{\mathbf{sfree(a,n_{id})}} \langle \mathbf{T}, \mathbf{A} \rangle$$

This implies (H-Free-trap). Hence proved.

□