Formalizing a Secure Foreign Function Interface – Extended Version

Adriaan Larmuseau and Dave Clarke

Uppsala University, Sweden,
first.last@it.uu.se

Abstract. Many high-level functional programming languages provide programmers with the ability to interoperate with untyped and low-level languages such as C and assembly. Research into such interoperation has generally focused on a closed world scenario, one where both the high-level and low-level code are defined and analyzed statically. In practice, however, components are sometimes linked in at run-time through malicious means. In this paper we formalise an operational semantics that securely combines MiniML, a light-weight ML, with a model of a low-level attacker, without relying on any static checks. We prove that the operational semantics are secure by establishing that they preserve and reflect the equivalences of MiniML. To that end a notion of bisimulation for the interaction between the attacker and MiniML is developed.

Keywords: Language Interoperation, Full Abstraction, Bisimulation

1 Introduction

Modern software systems consist of numerous interoperating components written in different source languages. Such language interoperation is usually achieved through a foreign function interface (FFI) that details how data is exchanged and functions are called across the language boundary between the source language and the foreign language. A FFI, however, introduces an explicit security risk: if the abstractions of the source language are not preserved in the foreign language, programs in the foreign language may be able to obtain confidential information or break the integrity of the program in the source langauge [1].

Preserving language abstractions is commonly formalised through a notion of *full abstraction*: Two source langauge terms t_1 and t_2 are indistinguishable to all source language contexts C (partial programs with a hole $[\cdot]$) if and only if they are indistinguishable to all foreign langauge contexts that they interact with through the FFI. Full abstraction thus preserves the abstractions of the source language in its interactions with the foreign language, ensuring that the programmer can safely limit himself to reasoning about the abstractions of the source langauge. Full abstraction does not, however, protect a programmer from writing insecure programs in the source language. That concern must be addressed by the source language through, for example, the typing system.

This paper introduces MiniML$^+$ a fully abstract formal model for an FFI between a source language MiniML, a light-weight ML featuring higher-order functions, references, tuples and recursion and a machine-level language such as assembly or C. The foreign language is not explicitated in the model. It is instead simplified to an attacker model that captures all the threats to full abstraction that a machine-level language may pose. To establish that the FFI is indeed fully abstract we develop bisimulations over MiniML and MiniML$^+$ and systematically relate the states of both bisimulations.

This paper is an extension of previous work on a formal model for a low-level memory protection mechanism [10]. In contrast to the basic λ-calculus addressed in that work, this paper considers a more complex source language, featuring references and divergence, and provides a more complete formal model that considers in detail the exchange of data structures and memory locations. The introduced formal model differs from previous formalisms such as Matthews' and Findler's multi-language semantics [11] in that it is less abstract. In contrast to multi-language semantics where the concrete details of function calls and data exchange are left to the implementation, our model MiniML$^+$ provides concrete insight into how to implement these mechanisms in a secure fashion.

The remainder of this paper is organized as follows. Firstly the paper provides an overview of the attacker model and the design of the secure FFI (Section 2). The paper describes the formal model MiniML$^+$ (Section 3) and provides a proof of full abstraction (Section 4). Finally the paper presents related work (Section 5) and concludes (Section 6).

2 The Interoperating Languages

The secure foreign function interface combines MiniML (Section 2.1) and an attacker model MiniML^a for a low-level language such as assembly or C (Section 2.2). The terms, types and contexts of MiniML are typeset in a `black font`, in contrast the attacker model MiniML^a are typeset in a **bold red font** .

2.1 The Source Language MiniML

The source language is MiniML: an extension of the typed λ-calculus featuring constants, references, tuples and recursion. The syntax is as follows.

$$
\begin{array}{lll}
t & ::= & v \mid x \mid (t_1\ t_2) \mid \langle t_i^{i\in 1..n}\rangle \mid \texttt{op}\ t_1\ t_2 \mid t.i \mid \texttt{if}\ t_1\ t_2\ t_3 \mid \texttt{ref}\ t \\
 & & \mid \texttt{let}\ x = t_1\ \texttt{in}\ t_2 \mid t_1 := t_2 \mid t_1; t_2 \mid \texttt{fix}\ t \mid !t \mid \texttt{hash}\ t \\
 & & \mid \texttt{letrec}\ x : \tau = t_1\ \texttt{in}\ t_2 \\
\texttt{op} & ::= & + \mid - \mid * \mid < \mid > \mid == \\
v & ::= & \langle v_i^{i\in 1..n}\rangle \mid \texttt{unit} \mid l_i \mid \overline{n} \mid (\lambda x : \tau.t) \mid \texttt{true} \mid \texttt{false} \\
\tau & ::= & \texttt{Bool} \mid \texttt{Int} \mid \texttt{Unit} \mid \tau_1 \to \tau_2 \mid \texttt{Ref}\ \tau \mid \langle \tau_i^{i\in 1..n}\rangle \\
E & ::= & [\cdot] \mid E\ t \mid v\ E \mid \texttt{op}\ E\ t \mid \texttt{op}\ v\ E \mid E.i \mid \texttt{fix}\ E \\
 & & \mid \langle v_i^{i\in 1..j}, E, t_k^{(k\in j+1..n)}\rangle \mid \texttt{if}\ E\ t_1\ t_2 \mid \texttt{let}\ x = E\ \texttt{in}\ t \mid !E \mid \texttt{ref}\ E \\
 & & \mid E := t \mid v := E \mid E;\ t \mid \texttt{hash}\ E
\end{array}
$$

Where $\overline{n}$ indicates the syntactic term representing the number n, `letrec` operator is syntactic sugar for a `let`-term combined with a `fix` operator and `E` is a Felleisen-and-Hieb-style evaluation context with a hole $[\cdot]$ that lift the basic reduction steps to a standard left-to-right call-by-value semantics [2].

The locations l_i are an artefact of the dynamic semantics that do not appear in the syntax used by programmers. The locations are tracked at run-time in a location store $\mu ::= \emptyset \mid \mu, l_i = v$, that is assumed to be an ideal store: it never runs out of space, and are allocated deterministically $l_1, l_2, .., l_n$. The term `hash` t implements Java's *.hashCode* method that converts references to integers, by mapping a location to its index: $l_i \mapsto \bar{i}$.

The reduction rules of MiniML are as expected.

$$\frac{}{\mu \mid E[\texttt{if true}\ t_2\ t_3] \longrightarrow \mu \mid E[t_2]} \qquad \frac{}{\mu \mid E[\texttt{if true}\ t_2\ t_3] \longrightarrow \mu \mid E[t_3]}$$

$$\frac{}{\mu \mid E[((\lambda x : \tau.t)\ v)] \longrightarrow \mu \mid E[t[v/x]]}$$

$$\frac{}{\mu \mid E[\texttt{let}\ x = v\ in\ t] \longrightarrow \mu \mid E[e[v/x]]}$$

$$\frac{t' = \texttt{let}\ x = \texttt{fix}(\lambda x : \tau.t_1)\ \texttt{in}\ t_2}{\mu \mid E[\texttt{letrec}\ x : \tau = t_1\ \texttt{in}\ t_2] \longrightarrow \mu \mid E[t']}$$

$$\frac{}{\mu \mid E[l_i] \longrightarrow \mu \mid E[\bar{i}]} \qquad \frac{}{\mu \mid E[\langle v_i^{i \in 1..n} \rangle.j] \longrightarrow \mu \mid E[v_j]}$$

$$\frac{}{\mu \mid E[\texttt{op}\ \overline{n_1}\ \overline{n_2}] \longrightarrow \mu \mid E[\overline{max(rnd(n_1\ \texttt{op}\ n_2), 0)}]}$$

$$\frac{}{\mu \mid E[\texttt{cp}\ \overline{n_1}\ \overline{n_2}] \longrightarrow \mu \mid E[b]} \qquad \frac{\mu' = \mu, [l_i \mapsto v]}{\mu \mid E[\texttt{ref}\ v] \longrightarrow \mu' \mid E[l_i]}$$

$$\frac{\mu(l_i) = v}{\mu \mid E[!l_i] \longrightarrow \mu \mid E[v]} \qquad \frac{}{\mu \mid E[\texttt{unit}; t_2] \longrightarrow \mu \mid E[t_2]}$$

$$\frac{}{\mu \mid E[\texttt{fix}(\lambda x : \tau.t)] \longrightarrow \mu \mid E[t[(\lambda x : \tau.t)/x]]} \qquad \frac{\mu' = [l \mapsto v]\mu}{\mu \mid E[l := v] \longrightarrow \mu' \mid E[\texttt{unit}]}$$

The typing rules of MiniML are entirely standard as well.

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x\ :\ \tau} \qquad \frac{}{\Gamma \vdash b\ :\ \texttt{Bool}} \qquad \frac{}{\Gamma \vdash \overline{n} : \texttt{Int}}$$

$$\frac{\Gamma \vdash t : \langle \tau_i^{i \in 1..n} \rangle}{\Gamma \vdash t.j\ :\ \tau_j} \qquad \frac{\Gamma \vdash t : \texttt{ref}\ \tau}{\Gamma \vdash \texttt{hash}\ t :\ \texttt{Int}} \qquad \frac{}{\Gamma \vdash \texttt{unit} : \texttt{Unit}}$$

$$\frac{\Gamma \vdash t_1 : \texttt{Int} \quad \Gamma \vdash t_2 : \texttt{Int}}{\Gamma \vdash (\texttt{op}\ t_1\ t_2)\ :\ \texttt{Int}} \qquad \frac{\Gamma \vdash t_1 : \texttt{Int} \quad \Gamma \vdash t_2 : \texttt{Int}}{\Gamma \vdash (\texttt{cp}\ t_1\ t_2)\ :\ \texttt{Bool}} \qquad \frac{\Gamma \vdash e_1 : \tau \to \tau}{\Gamma \vdash \texttt{fix}\ e_1 : \tau}$$

$$\frac{\Gamma(l) : \tau}{\Gamma \vdash l_i : \texttt{Ref}\ \tau} \qquad \frac{\Gamma \vdash t_1 : \texttt{Ref}\ \tau}{\Gamma \vdash !t_1 : \tau} \qquad \frac{\Gamma \vdash t_1 : \tau}{\Gamma \vdash \texttt{ref}\ t_1 : \texttt{Ref}\ \tau}$$

$$\frac{\Gamma \vdash t_1 : \texttt{Unit} \quad \Gamma \vdash t_2 : \tau}{\Gamma \vdash t_1;\ t_2 : \tau} \qquad \frac{\Gamma \vdash t_1 : \texttt{Ref}\ \tau \quad \Gamma \vdash t_2 : \tau}{\Gamma \vdash t_1 := t_2 : \texttt{Unit}}$$

$$\frac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash (\lambda x : \tau_1 = t)\ :\ \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash t_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash t_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x = t_1\ \texttt{in}\ t_2\ :\ \tau_2}$$

$$\frac{\Gamma, x : \tau_1 \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash t_2 : \tau_2}{\Gamma \vdash \texttt{letrec}\ x : \tau_1 = t_1\ \texttt{in}\ t_2\ :\ \tau_2} \qquad \frac{\Gamma \vdash t_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash t_2 : \tau_1}{\Gamma \vdash t_1\ t_2\ :\ \tau_2}$$

$$\frac{\forall\ i\ \Gamma \vdash t_i : \tau_i}{\Gamma \vdash \langle t_i^{i \in 1..n} \rangle\ :\ \langle \tau_i^{i \in 1..n} \rangle} \qquad \frac{\Gamma \vdash t_1 : \texttt{Bool} \quad \Gamma \vdash t_2 : \tau \quad \Gamma \vdash t_3 : \tau}{\Gamma \vdash \texttt{if}\ t_1\ t_2\ t_3\ :\ \tau}$$

Where the typing environment Γ is defined formally as follows.

$$\Gamma\ ::=\ \emptyset\ \mid\ \Gamma, l_i : \tau\ \mid\ \Gamma, x : \tau$$

Contextual Equivalence The secure FFI aims to preserve the contextual equivalences of MiniML. A MiniML context C is a MiniML term with a single hole $[\cdot]$, two MiniML terms are contextually equivalent if and only if there is no context C that can distinguish them. Contextual equivalence is formalised as follows.

Definition 1. Contextual equivalence ($\simeq$) *is defined as:*

$$t_1\ \simeq\ t_2\ \stackrel{def}{=}\ \forall C.\ C[t_1]\Uparrow\ \Longleftrightarrow\ C[t_2]\Uparrow$$

where $\Uparrow$ denotes divergence, t_1 and t_2 are closed terms and neither the terms and the contexts feature explicit locations l_i as they are not part of the static semantics. Note that two contextually equivalent MiniML terms t_1 and t_2 have the same type τ as a context C observes the same typing rules as the terms.

Two MiniML terms $\overline{1}$ and $\overline{2}$ are, for example, not contextually equivalent as a context $C = (\texttt{if}\ ([\cdot] == \overline{1})\ \Omega\ \texttt{true})$, where Ω is a diverging term, can distinguish them. MiniML's λ-terms, in contrast, introduce many equivalences. There is no context C, for example, that can distinguish the following terms.

$$(\lambda x : \texttt{Int}.(+\ x\ x)) \qquad (\lambda x : \texttt{Int}.(*\ 2\ x)) \tag{Ex-1}$$

The equivalences over the locations of MiniML are a little more complex. Due to the deterministic allocation order and the inclusion of the `hash` operation, a context can observe the number of locations as well as their indices. The following two terms, for example, are not contextually equivalent.

$$\begin{array}{ll} \texttt{let}\ x =\ \texttt{ref true in} & \texttt{let}\ x =\ \texttt{ref true in} \\ \texttt{let}\ y\ =\ \texttt{ref true in}\ x & \texttt{let}\ y\ =\ \texttt{ref true in}\ y \end{array} \tag{Ex-2}$$

As the context $C = (\texttt{if}\ (\texttt{hash}[\cdot] == \overline{1})\ \Omega\ \texttt{true})$ can distinguish both terms. Locations when kept secret, however, can still produce equivalences as a context C cannot contain a location l_i unless it is shared at run-time. The following two terms, for example, are thus contextually equivalent.

$$\texttt{let}\ x = \texttt{ref false in}\ \overline{1} \qquad \texttt{let}\ x = \texttt{ref true in}\ \overline{1} \tag{Ex-3}$$

2.2 The Attacker Model MiniML^a

A malicous machine-level attacker can break the abstractions of a MiniML program that it interoperates with in the following two ways:

Inspection An attacker can break the abstractions of MiniML by inspecting and manipulating the internal state of a MiniML program. An attacker can achieve this by either reading and writing to references shared through the FFI (an existing vulnerability in the Java Virtual Machine [14]) or by abusing low-level privileges to inspect the memory directly.

Breaking Type Safety When interoperating a MiniML program will not only share language constructs over the FFI, but also receive language constructs. The attacker can take advantage of this by passing language constructs that do not adhere to the typing rules of MiniML.

Our attacker model incorporates both these threats to full abstraction. The attacker model is formalised as a language MiniML^a that is derived from MiniML by removing typing safety and incorporating reflection. Removing type safety is achieved by both removing the types and adding a new term **wr** that captures non reducible expressions.

Reflection is added to MiniML^a by means of a syntactical equality testing operator modulo α-equivalence $\equiv_\alpha$. Given two terms $\mathbf{t_1}$ and $\mathbf{t_2}$, the term $\mathbf{t_1} \equiv_\alpha \mathbf{t_2}$ will thus only reduce to **true** if $\mathbf{t_1}$ and $\mathbf{t_2}$ are syntactically equal except for the names assigned to the variables.

The syntax of MiniML^a is as follows.

$$\begin{array}{lll}
\mathbf{t} & ::= & \mathbf{v} \mid \mathbf{x} \mid (\mathbf{t_1}\ \mathbf{t_2}) \mid \langle \mathbf{t_i}^{\mathbf{i \in 1..n}} \rangle \mid \mathbf{op}\ \mathbf{t_1}\ \mathbf{t_2} \mid \mathbf{t.i} \mid \mathbf{if}\ \mathbf{t_1}\ \mathbf{t_2}\ \mathbf{t_3} \\
 & & \mid \mathbf{let}\ \mathbf{x} = \mathbf{t_1}\ \mathbf{in}\ \mathbf{t_2} \mid !\mathbf{t} \mid \mathbf{ref}\ \mathbf{t} \mid \mathbf{letrec}\ \mathbf{x} = \mathbf{t_1}\ \mathbf{in}\ \mathbf{t_2} \mid \mathbf{t_1}; \mathbf{t_2} \mid \mathbf{fix}\ \mathbf{t} \\
 & & \mid \mathbf{wr} \mid \mathbf{t_1} := \mathbf{t_2}\ \mathbf{hash}\ \mathbf{t_1} \\
\mathbf{op} & ::= & + \mid - \mid * \mid < \mid > \mid == \\
\mathbf{v} & ::= & \langle \mathbf{v_i}^{\mathbf{i \in 1..n}} \rangle \mid \mathbf{unit} \mid \mathbf{l_i} \mid \overline{\mathbf{n}} \mid (\lambda \mathbf{x.t}) \mid \mathbf{true} \mid \mathbf{false} \\
\mathbf{E} & ::= & [\cdot] \mid \mathbf{E}\ \mathbf{t} \mid \mathbf{v}\ \mathbf{E} \mid \langle \mathbf{v_i}^{\mathbf{i \in 1..j}}, \mathbf{E}, \mathbf{t_k}^{(\mathbf{k \in j+1..n})} \rangle \mid \mathbf{E}.i \mid \mathbf{op}\ \mathbf{E}\ \mathbf{t} \\
 & & \mid \mathbf{op}\ \mathbf{v}\ \mathbf{E} \mid \mathbf{if}\ \mathbf{E}\ \mathbf{t_1}\ \mathbf{t_2} \mid \mathbf{let}\ \mathbf{x} = \mathbf{E}\ \mathbf{in}\ \mathbf{t} \mid !\mathbf{E} \mid \mathbf{ref}\ \mathbf{E} \mid \mathbf{E} := \mathbf{t} \mid \mathbf{E};\ \mathbf{t} \\
 & & \mid \mathbf{v} := \mathbf{E} \mid \mathbf{fix}\ \mathbf{E} \mid \mathbf{hash}\ \mathbf{E}
\end{array}$$

The reduction rules of MiniML^a are the reduction rules of MiniML extended with new reductions for the α-equivalence operator as well as reduction rules that capture stuck states.

$$\frac{}{\mu \mid \mathbf{E}[\mathbf{if\ true\ t_2\ t_3}] \longrightarrow \mu \mid \mathbf{E}[\mathbf{t_2}]} \qquad \frac{}{\mu \mid \mathbf{E}[\mathbf{if\ false\ t_2\ t_3}] \longrightarrow \mu \mid \mathbf{E}[\mathbf{t_3}]}$$

$$\frac{\mathbf{v} \neq \mathbf{b}}{\mu \mid \mathbf{E}[\mathbf{if\ v\ t_2\ t_3}] \longrightarrow \mu \mid \mathbf{wr}} \qquad \frac{\mathbf{v} \neq (\lambda \mathbf{x} : \tau.\mathbf{t})}{\mu \mid \mathbf{E}[(\mathbf{v\ v'})] \longrightarrow \mu \mid \mathbf{wr}}$$

$$\frac{}{\mu \mid \mathbf{E}[((\lambda \mathbf{x} : \tau.\mathbf{t})\ \mathbf{v})] \longrightarrow \mu \mid \mathbf{E}[\mathbf{t}[\mathbf{v}/\mathbf{x}]]}$$

$$\frac{}{\mu \mid \mathbf{E}[\mathbf{let\ x = v\ in\ t}] \longrightarrow \mu \mid \mathbf{E}[\mathbf{e}[\mathbf{v}/\mathbf{x}]]}$$

$$\frac{\mathbf{t'} = \mathbf{let\ x = fix}(\lambda \mathbf{x} : \tau.\mathbf{t_1})\ \mathbf{in\ t_2}}{\mu \mid \mathbf{E}[\mathbf{letrec\ x} : \tau = \mathbf{t_1\ in\ t_2}] \longrightarrow \mu \mid \mathbf{E}[\mathbf{t'}]}$$

$$\frac{}{\mu \mid E[\langle v_i^{i \in 1..n} \rangle.j] \longrightarrow \mu \mid E[v_j]} \qquad \frac{\mathbf{v} \neq \langle v_i^{i \in 1..n} \rangle}{\mu \mid E[v.j] \longrightarrow \mu \mid \mathbf{E}[\mathbf{wr}]}$$

$$\frac{\mathbf{v_1} \neq \overline{\mathbf{n}} \qquad \mathbf{v_2} \neq \overline{\mathbf{n}}}{\mu \mid \mathbf{E}[\mathsf{cp}\ \mathbf{v_1}\ \mathbf{v_2}] \longrightarrow \mu \mid \mathbf{E}[\mathbf{wr}]} \qquad \frac{\mathbf{v_1} \neq \overline{\mathbf{n}} \qquad \mathbf{v_2} \neq \overline{\mathbf{n}}}{\mu \mid \mathbf{E}[\mathsf{op}\ \mathbf{v_1}\ \mathbf{v_2}] \longrightarrow \mu \mid \mathbf{E}[\mathbf{wr}]}$$

$$\frac{}{\mu \mid \mathbf{E}[\mathsf{op}\ \overline{\mathbf{n_1}}\ \overline{\mathbf{n_2}}] \longrightarrow \mu \mid \mathbf{E}[\overline{\mathbf{max}(\mathbf{rnd}(\mathbf{n_1}\ \mathsf{op}\ \mathbf{n_2}), \mathbf{0})}]}$$

$$\frac{}{\mu \mid \mathbf{E}[\mathsf{cp}\ \overline{\mathbf{n_1}}\ \overline{\mathbf{n_2}}] \longrightarrow \mu \mid \mathbf{E}[\mathbf{b}]} \qquad \frac{\mu' = \mu, [\mathbf{l_i} \mapsto \mathbf{v}]}{\mu \mid \mathbf{E}[\mathbf{ref\ v}] \longrightarrow \mu' \mid \mathbf{E}[\mathbf{l_i}]}$$

$$\frac{\mu(\mathbf{l_i}) = \mathbf{v}}{\mu \mid \mathbf{E}[!\mathbf{l_i}] \longrightarrow \mu \mid \mathbf{E}[\mathbf{v}]} \qquad \frac{}{\mu \mid \mathbf{E}[\mathbf{unit}; \mathbf{t_2}] \longrightarrow \mu \mid \mathbf{E}[\mathbf{t_2}]}$$

$$\frac{\mathbf{t_1}\&\mathbf{t_2}\ \textit{are}\ \alpha - \textit{equiv}}{\mu \mid \mathbf{E}[\mathbf{t_1} \equiv_\alpha \mathbf{t_2}] \longrightarrow \mu \mid \mathbf{E}[\mathbf{true}]} \qquad \frac{\mathbf{t_1}\&\mathbf{t_2}\ \textit{are not}\ \alpha - \textit{equiv}}{\mu \mid \mathbf{E}[\mathbf{t_1} \equiv_\alpha \mathbf{t_2}] \longrightarrow \mu \mid \mathbf{E}[\mathbf{false}]}$$

$$\frac{\mathbf{l_i} \notin \mu}{\mu \mid \mathbf{E}[!\mathbf{l_i}] \longrightarrow \mu \mid \mathbf{E}[\mathbf{wr}]} \qquad \frac{\mathbf{v} \neq \mathbf{unit}}{\mu \mid \mathbf{E}[\mathbf{v}; \mathbf{t_2}] \longrightarrow \mu \mid \mathbf{E}[\mathbf{wr}]}$$

$$\frac{}{\mu \mid \mathbf{E}[\mathbf{fix}(\lambda \mathbf{x} : \tau.\mathbf{t})] \longrightarrow \mu \mid \mathbf{E}[\mathbf{t}[(\lambda \mathbf{x} : \tau.\mathbf{t})/\mathbf{x}]]}$$

$$\frac{\mu' = [\mathbf{l_i} \mapsto \mathbf{v}]\mu}{\mu \mid \mathbf{E}[\mathbf{l_i} := \mathbf{v}] \longrightarrow \mu' \mid \mathbf{E}[\mathbf{unit}]} \qquad \frac{}{\mu \mid \mathbf{E}[\mathbf{hash\ l_i}] \longrightarrow \mathbf{E}[\bar{\mathbf{i}}]}$$

Contextual Equivalence or Lack Thereof The addition of reflection in MiniML^a through the α-equivalence testing operator, renders the abstractions and source level restrictions of MiniML obsolete [16]. Consider, for example, the equivalent

λ-terms of Ex-1 in Section 2.1. The following MiniML^a context:

$$\mathbf{C} = (\mathbf{if}\ ((\lambda \mathbf{y}.(* \ \mathbf{2}\ \mathbf{y})) \equiv_\alpha [\cdot])\ \mathbf{\Omega}\ ..)$$

distinguishes them due to the $\equiv_\alpha$ operator's ability to inspect the syntax of MiniML terms. A similar context $\mathbf{C}$ can thus be built for the contextually equivalent terms of Ex-3 and for every other pair of contextually equivalent terms.

3 The MiniML$^+$-Calculus: a Secure FFI

The MiniML$^+$-calculus securely interoperates between MiniML and MiniML^a in a manner that secures the MiniML program from the MiniML^a attacker (Section 3.1). The MiniML$^+$-calculus introduces new syntax (Section 3.2), new operational semantics (Section 3.3), additional typing rules (Section 3.4) and a modified notion of type soundness (Section 3.5).

3.1 Overview

To formalise a secure interoperation between the attacker and the source language the MiniML$^+$-calculus applies the following three insights.

Separated Program States Preserving full abstraction when faced with a machine level attacker has been achieved by employing memory isolation mechanisms that prevent the attacker from directly accessing the memory of the program being secured [12]. To that end the program state P is split into two sub- states: the attacker state $\mathbf{A}$ and the secured program state $\mathtt{M}$ that incorporates the MiniML program. Formally a program P is defined as: $P = \mathbf{A} \mid\mid \mathtt{M}$.

Call Stacks To ensure that the program state is separable, the combined language must encode the interaction between both languages. To do so each state is extended with a call stack. The secure state $\mathtt{M}$ encodes this call stack as a type annotated stack of evaluation contexts $\Sigma ::= \overline{\mathtt{E}} : \overline{\tau \rightarrow \tau'} \ \mid\ \varepsilon$, where $\overline{\mathtt{E}}$ denotes a sequence of evaluation contexts $\mathtt{E}$ that represent the continuation of computation when a call to the attacker returns and are thus only to be filled in by input originating from the attacker. The stack of evaluation contexts is type annotated. In MiniML$^+$ these annotated types are incorporated into dynamic type checks to ensure that the input from the attacker does not break the type safety of the original MiniML program.

In contrast the attacker encodes the call stack through a sequence of contexts $\overline{\mathbf{C}}$ not a sequence of evaluation contexts $\mathbf{E}$. An evaluation context $\mathbf{E}$ is derived from call-by-value semantics, which limits the hole $[\cdot]$ to certain sub-terms. The evaluation context $\mathbf{E}$ is thus a less powerful threat to full-abstraction than the context $\mathbf{C}$, where the hole can be anywhere. More specifically, for each possible pair of terms $\mathbf{t_1}$ and $\mathbf{t_2}$ received from the MiniML program there exists a context $\mathbf{C}$ of the form: $(\mathbf{if}\ (\mathbf{t_1} \equiv_\alpha [\cdot])\ \mathbf{\Omega}\ \mathbf{true})$ that can distinguish them.

Reference Objects To ensure that the state of the MiniML program is isolated from any kind of inspection by the attacker, the terms of MiniML programs that introduce equivalences/abstractions: namely λ-terms and locations, should not be shared directly with attacker. Instead, those terms are shared by providing the attacker with reference objects, objects that refer to the original terms of the program in MiniML. These reference objects, have two purposes. Firstly they mask the contents of the original term and secondly they enable MiniML$^+$ to keep track of which locations or λ-terms have been shared with the attacker. The MiniML$^+$-calculus models reference objects for λ-terms and locations through names $\mathbf{n_i^f}$ and $\mathbf{n_i^l}$ respectively. Both names are tracked in the secure state through a map $\mathtt{N}$ that records not only the associated term but also the associated type, thus enabling MiniML$^+$ to perform run-time type checks on the attackers interactions with these names. Formally $\mathtt{N}$ is defined as.

$$\mathtt{N} ::= \star \mid \mathtt{N}, \mathbf{n_i^f} \mapsto (t, \tau) \mid \mathtt{N}, \mathbf{n_i^l} \mapsto (t, \tau)$$

A name $\mathbf{n_i^f}$ is created deterministically every time a λ-term is shared between the secure state and the attacker. The name $\mathbf{n_1^f}$ refers to the first shared λ-term, the name $\mathbf{n_2^f}$ refers to the second shared λ-term (even if it is the exact same λ-term as the first one) and so forth. In contrast the index i of the name $\mathbf{n_i^l}$ will correspond to the index of the location it refers to ($\mathbf{n_i^l} \mapsto l_i$). This is because the `hash` operation in MiniML allows a MiniML context/attacker to observe the index of the location, as illustrated in Ex-2 of Section 2.1. This observational power should thus not be taken away from the attacker in MiniML$^+$.

Note that the names $\mathbf{n_i^f}$ and $\mathbf{n_i^l}$ are terms of the MiniML^a-calculus but not of MiniML. Also note that we don't compile or translate the λ-terms and locations into these names.

3.2 Syntax

While basic values such as numbers and booleans can simply be converted to the correct representation when exchanged, no such conversion is possible for λ-terms and locations l_i. As detailed in Section 3.1, in MiniML$^+$ the MiniML^a attacker is restricted to reference objects formalized as names $\mathbf{n_i^f}$ and $\mathbf{n_i^l}$ that refer to λ-terms and locations shared by the MiniML program respectively. A MiniML^a attacker can compare these names through its α-equivalence testing operator $\equiv_\alpha$ and can also apply, read and write them in MiniML$^+$ using the newly added terms: $\mathbf{call\ n_i^f\ v}$, $\mathbf{deref\ n_i^l}$ and $\mathbf{set\ n_i^l\ v}$ respectively. The attacker can also create new names $\mathbf{n_i^l}$, that point to freshly allocated locations l_i in the MiniML program, through a term $\mathbf{fref^\tau\ v}$. Where τ represent the MiniML type that the attacker promises the value $\mathbf{v}$ conforms to. This promise is checked at run-time by MiniML$^+$. The syntax of MiniML^a is thus extended as follows.

$$\begin{aligned}
t &::= \ldots \mid \mathbf{call\ t_1\ t_2} \mid \mathbf{set\ t_1\ t_2} \mid \mathbf{deref\ t} \mid \mathbf{fref^\tau\ t} \\
v &::= \ldots \mid \mathbf{n_i^f} \mid \mathbf{n_i^l} \\
E &::= \ldots \mid \mathbf{call\ E\ t} \mid \mathbf{call\ v\ E} \mid \mathbf{set\ E\ t} \mid \mathbf{set\ v\ E} \mid \mathbf{deref\ E} \mid \mathbf{fref^\tau\ E}
\end{aligned}$$

In contrast, the terms of MiniML are only extended with one new value: ${}^{\tau}\mathtt{F}(\lambda \mathbf{x}.\mathbf{t})$ that embeds a MiniML^a λ-term in MiniML, modelling an attacker function that the MiniML program can use. The type τ is included with the value to enable MiniML$^+$ to type check the use of this attacker function at run-time. The MiniML-calculus is not extended with a term to embed locations of MiniML^a, as manipulating the attacker memory harms the full abstraction result. This does not harm the interoperation, as the attacker can simply create an MiniML location through the $\mathbf{fref}^{\tau}\ t$ term instead of sharing its own.

The marshalling process of MiniML$^+$ transitions between terms of MiniML and MiniML^a within the secure state $\mathtt{M}$. The marshalling terms m are as follows.

$$m \ ::= \ v \ | \ \mathbf{v} \ | \ \langle m_i^{i \in 1..n} \rangle$$

The marshalling converts MiniML values to MiniML^a values and vice versa. Marshalling a tuple of size n is not immediate but takes n steps. To capture the intermediate state where some members are converted and others are not a tuple of terms m is included.

3.3 Operational Semantics

The reduction rules of the MiniML-calculus are denoted as $P \twoheadrightarrow P'$. As described in Section 3.1 a program $P = \mathtt{M} \mid\mid \mathbf{A}$ composes two states $\mathtt{M}$ and $\mathbf{A}$. The secure state $\mathtt{M}$ is either (1) executing a term t of type τ, (2) marshalling out values, (3) marshalling in input from the attacker that is expected to be of type τ or (4) waiting on input from the attacker.

$$(1)\ \mathtt{N}; \mu \Vdash \Sigma \circ t : \tau \quad (2)\ \mathtt{N}; \mu \Vdash \Sigma \rhd m : \tau \quad (3)\ \mathtt{N}; \mu \Vdash \Sigma \lhd m : \tau \quad (4)\ \mathtt{N}; \mu \Vdash \Sigma$$

The attacker state takes two forms: (1) it executes a term $\mathbf{t}$ or (2) is suspended waiting on input from the MiniML program.

$$(1)\ \mathbf{A} = \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{t} \qquad (2)\ \mathbf{A} = \mu \Vdash \overline{\mathbf{C}}$$

The states never execute at the same time. For every possible program state P the attacker or the secure state will be suspended. We divide the reduction rules over the program state P into four categories: internal computations, marshalling values in, marshalling value out and cross-boundary commands.

Internal Computations Internal computations are reduction rules that only affect the terms of one of the two languages. These are thus simply the reduction rules of MiniML and MiniML^a set within the program state of MiniML$^+$.

$$\mu \Vdash \overline{\mathbf{C}} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \circ t : \tau \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \mid\mid \mathtt{N}; \mu' \Vdash \Sigma \circ t' : \tau \qquad \text{(Internal MiniML)}$$

$$\mu \Vdash \overline{\mathbf{C}} \circ \mathbf{t} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \twoheadrightarrow \mu' \Vdash \overline{\mathbf{C}}' \circ \mathbf{t}' \mid\mid \mathtt{N}; \mu \Vdash \Sigma \qquad \text{(Internal MiniML}^a\text{)}$$

Note that unlike the internal computations of MiniML the internal computations of MiniML$_a$ can modify its call stack. This is because as previously mentioned the attacker reduces to wrong when dealing with errors.

$$\mu \Vdash \overline{\mathbf{C}} \circ \mathbf{E}[\mathbf{wr}] \mid\mid \mathtt{N}; \mu \Vdash \Sigma \twoheadrightarrow \mu \Vdash \mathbf{wr} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \qquad \text{(A-Wr)}$$

In practice, a low-level attacker can recover from errors. In MiniML$^+$ however, whenever something goes wrong the attacker is to blame, similar to how Wadler's and Findler's blame calculus [15] blames the less-precise portion of a program, and the program is terminated by clearing out the ML-configuration. As such the actions of an attacker after something has gone wrong are not relevant to the overall security result.

Marshalling Values Out Whenever the embedded MiniML program reduces to a value v, that value needs to be converted to the appropriate representation before it is shared with the head of the attacker's call stack $\overline{\mathbf{C}}$. If the value is a location or a λ-term then it must be masked with a name $\mathbf{n_i^l}$ or $\mathbf{n_i^f}$, and the association between the name, the term and the term's type recorded in the map $\mathtt{N}$. Otherwise, the value is simply converted to the corresponding MiniML^a value. This conversion happens in a designated marshalling state as follows.

$$\mu \Vdash \overline{\mathbf{C}} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \circ v : \tau \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \rhd v : \tau \qquad \text{(Setup)}$$

To save space in the following marshalling rules, we have compressed the state $\mu \Vdash \overline{\mathbf{C}} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \rhd m$ into a wrapper $\llbracket m \rrbracket_\tau^{\mathtt{N}}$ that denotes the only two constructs relevant to the marshalling process: the expected type τ and the map of shared names $\mathtt{N}$. Note the tuple conversion rule: it converts every member individually, ensuring that the embedded λ-terms and locations become names.

$$\frac{}{\llbracket b \rrbracket_{\mathtt{Bool}}^{\mathtt{N}} \twoheadrightarrow \llbracket \mathbf{b} \rrbracket_{\mathtt{Bool}}^{\mathtt{N}}} \qquad \frac{}{\llbracket \mathtt{unit} \rrbracket_{\mathtt{Unit}}^{\mathtt{N}} \twoheadrightarrow \llbracket \mathbf{unit} \rrbracket_{\mathtt{Unit}}^{\mathtt{N}}} \qquad \frac{}{\llbracket \overline{n} \rrbracket_{\mathtt{Int}}^{\mathtt{N}} \twoheadrightarrow \llbracket \overline{\mathbf{n}} \rrbracket_{\mathtt{Int}}^{\mathtt{N}}}$$

$$\frac{j = |\mathtt{N}| + 1 \quad \mathtt{N}' = (\mathtt{N}, \mathbf{n_j} \mapsto ((\lambda x : \tau.t), \tau \to \tau'))}{\llbracket (\lambda x : \tau.t) \rrbracket_{\tau \to \tau'}^{\mathtt{N}} \twoheadrightarrow \llbracket \mathbf{n_j^f} \rrbracket_{\tau \to \tau'}^{\mathtt{N}'}} \qquad \frac{\mathtt{N}' = (\mathtt{N}, \mathbf{n_i^l} \mapsto (l_i, \mathtt{Ref}\tau))}{\llbracket l_i \rrbracket_{\mathtt{Ref}\tau}^{\mathtt{N}} \twoheadrightarrow \llbracket \mathbf{n_j^l} \rrbracket_{\mathtt{Ref}\tau}^{\mathtt{N}'}}$$

$$\frac{\forall i \in 1..n.\ \llbracket v_i \rrbracket_\tau^{\mathtt{N}_{i-1}} \twoheadrightarrow \llbracket \mathbf{v_i} \rrbracket_\tau^{\mathtt{N}_i}}{\llbracket \langle v_i^{i \in 1..n} \rangle \rrbracket_{\tau \times \tau'}^{\mathtt{N}} \twoheadrightarrow \llbracket \langle \mathbf{v_i^{i \in 1..n}} \rangle \rrbracket_{\tau \times \tau'}^{\mathtt{N}_n}} \qquad \frac{\exists i \in 1..n.\ \llbracket v_i \rrbracket_\tau^{\mathtt{N}} \twoheadrightarrow \llbracket \mathbf{wr} \rrbracket_\tau^{\mathtt{N}'}}{\llbracket \langle v_i^{i \in 1..n} \rangle \rrbracket_{\tau \times \tau'}^{\mathtt{N}} \twoheadrightarrow \llbracket \mathbf{wr} \rrbracket_{\tau \times \tau'}^{\mathtt{N}'}}$$

$$\frac{\tau = \tau_1 \to \tau_2}{\llbracket {}^\tau \mathtt{F}(\lambda \mathbf{x.t}) \rrbracket_\tau^{\mathtt{N}} \twoheadrightarrow \llbracket (\lambda \mathbf{x.t}) \rrbracket_\tau^{\mathtt{N}}} \qquad \frac{\emptyset \vdash t : \tau' \qquad \tau \neq \tau'}{\llbracket t \rrbracket_\tau^{\mathtt{N}} \twoheadrightarrow \llbracket \mathbf{wr} \rrbracket_\tau^{\mathtt{N}}}$$

If the marshalling succeeds (there is no type error) the result is shared, otherwise the secure state is cleared and the attacker is updated with wrong: $\mathbf{wr}$.

$$\mu \Vdash \overline{\mathbf{C}}, \mathbf{C} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \rhd \mathbf{v} : \tau \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{C}[\mathbf{v}] \mid\mid \mathtt{N}; \mu \Vdash \Sigma \qquad \text{(Share)}$$

$$\mu \Vdash \overline{\mathbf{C}}, \mathbf{C} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \rhd \mathbf{wr} : \tau \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}}, \mathbf{C}[\mathbf{wr}] \mid\mid \star; \emptyset \Vdash \varepsilon \qquad \text{(Type Error)}$$

Marshalling Values In Whenever the attacker reduces to a value and the secure state's call stack Σ is not empty the value is input into the secure state.

$$\mu \Vdash \overline{\mathbf{C}} \circ \mathbf{v} \mid\mid \mathtt{N}; \mu \Vdash \Sigma, \mathtt{E} : \tau \to \tau' \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \lhd \mathbf{v} : \tau \qquad \text{(Input)}$$

The input value must be marshalled to the correct representation before it is plugged into the head of the stack of evaluation contexts Σ. Note that as denoted

in the reduction rule Input the marshalling rules will verify that the input value matches the argument type τ of the to be plugged evaluation context. The marshalling reduction rules are analogous to the previously detailed marshalling out reductions in that they perform the reverse operation: they convert the input into the appropriate MiniML representation, fetching names from the map $\mathtt{N}$ instead of introducing names.

$$\frac{}{\llbracket \mathbf{b} \rrbracket^{\mathtt{N}}_{\mathtt{Bool}} \twoheadrightarrow \llbracket b \rrbracket^{\mathtt{N}}_{\mathtt{Bool}}} \qquad \frac{\mathbf{t} \neq \mathbf{b}}{\llbracket \mathbf{t} \rrbracket^{\mathtt{N}}_{\mathtt{Bool}} \twoheadrightarrow \llbracket \mathbf{wr} \rrbracket^{\mathtt{N}}_{\mathtt{Bool}}} \qquad \frac{}{\llbracket \overline{\mathbf{n}} \rrbracket^{\mathtt{N}}_{\mathtt{Int}} \twoheadrightarrow \llbracket \overline{n} \rrbracket^{\mathtt{N}}_{\mathtt{Int}}}$$

$$\frac{\mathbf{t} \neq \overline{\mathbf{n}}}{\llbracket \mathbf{t} \rrbracket^{\mathtt{N}}_{\mathtt{Int}} \twoheadrightarrow \llbracket \mathbf{wr} \rrbracket^{\mathtt{N}}_{\mathtt{Int}}} \qquad \frac{}{\llbracket \mathbf{unit} \rrbracket^{\mathtt{N}}_{\mathtt{Unit}} \twoheadrightarrow \llbracket \mathtt{unit} \rrbracket^{\mathtt{N}}_{\mathtt{Unit}}} \qquad \frac{\mathbf{t} \neq \mathbf{unit}}{\llbracket \mathbf{unit} \rrbracket^{\mathtt{N}}_{\mathtt{Unit}} \twoheadrightarrow \llbracket \mathbf{wr} \rrbracket^{\mathtt{N}}_{\mathtt{Unit}}}$$

$$\frac{\mathbf{t} \neq \mathbf{l}}{\llbracket \mathbf{l} \rrbracket^{\mathtt{N}}_{\mathtt{Ref}\tau} \twoheadrightarrow \llbracket \mathbf{wr} \rrbracket^{\mathtt{N}}_{\mathtt{Ref}\tau}} \qquad \frac{\mathbf{t} \neq (\lambda.\mathbf{x} : \tau.\mathbf{t}')}{\llbracket \mathbf{t} \rrbracket^{\mathtt{N}}_{\tau \to \tau'} \twoheadrightarrow \llbracket \mathbf{wr} \rrbracket^{\mathtt{N}}_{\tau \to \tau'}} \qquad \frac{\mathtt{N}(\mathbf{n}^{\mathbf{l}}_{\mathbf{j}}) = (l, \mathtt{Ref}\tau)}{\llbracket \mathbf{n}^{\mathbf{l}}_{\mathbf{j}} \rrbracket^{\mathtt{N}'}_{\mathtt{Ref}\tau} \twoheadrightarrow \llbracket n^{l}_{j} \rrbracket^{\mathtt{N}}_{\mathtt{Ref}\tau}}$$

$$\frac{\mathtt{N}(\mathbf{n}^{\mathbf{f}}_{\mathbf{j}}) = ((\lambda x : \tau.t), \tau \to \tau')}{\llbracket \mathbf{n}^{\mathbf{f}}_{\mathbf{j}} \rrbracket^{\mathtt{N}'}_{\tau \to \tau'} \twoheadrightarrow \llbracket (\lambda x : \tau.t) \rrbracket^{\mathtt{N}}_{\tau \to \tau'}} \qquad \frac{\exists i \in 1..n.\ \llbracket v_i \rrbracket^{\mathtt{N}_{i-1}}_{\tau} \twoheadrightarrow \llbracket \mathbf{wr} \rrbracket^{\mathtt{N}_i}_{\tau}}{\llbracket \langle \mathbf{v}^{\mathbf{i} \in \mathbf{1..n}}_{\mathbf{i}} \rangle \rrbracket^{\mathtt{N}}_{\tau \times \tau'} \twoheadrightarrow \llbracket \mathbf{wr} \rrbracket^{\mathtt{N}_i}_{\tau \times \tau'}}$$

$$\frac{\forall i \in 1..n.\ \llbracket v_i \rrbracket^{\mathtt{N}_{i-1}}_{\tau} \twoheadrightarrow \llbracket \mathbf{v_i} \rrbracket^{\mathtt{N}_i}_{\tau}}{\llbracket \langle \mathbf{v}^{\mathbf{i} \in \mathbf{1..n}}_{\mathbf{i}} \rangle \rrbracket^{\mathtt{N}}_{\tau \times \tau'} \twoheadrightarrow \llbracket \langle v_i^{i \in 1..n} \rangle \rrbracket^{\mathtt{N}_n}_{\tau \times \tau'}} \qquad \frac{}{\llbracket (\lambda \mathbf{x}.\mathbf{t})^{\mathtt{N}}_{\tau \to \tau'} \rrbracket \twoheadrightarrow \llbracket {}^{\tau \to \tau'}\mathtt{F}(\lambda \mathbf{x}.\mathbf{t}) \rrbracket^{\mathtt{N}}_{\tau \to \tau'}}$$

If the input doesn't conform with the type annotated to the evaluation context the state $\mathtt{M}$ is cleared and the attacker terminates to $\mathbf{wr}$, otherwise the marshalled value is used to plug the evaluation context.

$$\mu \Vdash \overline{\mathbf{C}}, \mathbf{C} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \lhd \mathbf{wr} : \tau \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}}, \mathbf{C}[\mathbf{wr}] \mid\mid \star; \emptyset \Vdash \varepsilon \qquad \text{(Type-Error-In)}$$

$$\mu \Vdash \overline{\mathbf{C}} \mid\mid \mathtt{N}; \mu \Vdash \Sigma, \mathtt{E} : \tau \to \tau' \lhd v : \tau \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \mid\mid \mathtt{N}', \mu \Vdash \Sigma \circ \mathtt{E}[v] : \tau' \qquad \text{(Plug)}$$

Cross Boundary Commands The cross boundary commands enable the MiniML program to manipulate shared λ-terms and locations as follows.

$$\mu \Vdash \overline{\mathbf{C}}, \mathbf{C} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \circ \mathtt{E}[({}^{\tau_1 \to \tau_2}\mathtt{F}(\lambda \mathbf{x}.\mathbf{t})\ v)] : \tau \twoheadrightarrow \qquad \text{(M-Call)}$$
$$\mu \Vdash \overline{\mathbf{C}}, \mathbf{C}[(\mathbf{t}\ [\cdot])] \mid\mid \mathtt{N}; \mu \Vdash \Sigma, \mathtt{E} : \tau_2 \to \tau \rhd v : \tau_1$$

As listed above a MiniML program is able to apply a MiniML^a λ-term (M-Call). The application is done in two steps as it consists of two components: the shared λ-term and an argument v. In the first step an evaluation context that consists of an application of the shared λ-term to a hole $[\cdot]$ is placed inside the context $\mathbf{C}$ while the secure state is setup for marshalling. In a second step the argument v is then marshalled out as described previously and plugged into the newly constructed evaluation context after which control is reverted to the attacker.

Note that this cross boundary function application serves as an input to the attacker as it is plugged into the top context/attack $\mathbf{C}$ of the attacker's call stack $\overline{\mathbf{C}}$. This is because the attacker must be able to inspect this function call

as accurately as the machine-level attacker who is able to observe which of its functions are called using which arguments.

The cross boundary commands also enable the attacker to manipulate shared MiniML λ-terms and locations as follows.

$$\mu \Vdash \overline{\mathbf{C}} \circ \mathbf{E}[\mathbf{call}\ \mathbf{n_i^f}\ \mathbf{v}] \mid\mid \mathtt{N}; \mu \Vdash \Sigma \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{v} \mid\mid \mathtt{N}; \mu \Vdash \Sigma, (t\ [\cdot]) : \tau \to \tau' \quad \text{(A-Call)}$$
$$\textit{where } \mathtt{N}(\mathbf{n_i^f}) = (t, \tau \to \tau')$$

$$\mu \Vdash \overline{\mathbf{C}} \circ \mathbf{E}[\mathbf{set}\ \mathbf{n_i^l}\ \mathbf{v}] \mid\mid \mathtt{N}; \mu \Vdash \Sigma \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{v} \mid\mid \mathtt{N}; \mu \Vdash \Sigma, (t := [\cdot]) : \tau \to \mathtt{Unit}$$
$$\textit{where } \mathtt{N}(\mathbf{n_i^l}) = (t, \mathtt{Ref}\ \tau) \quad \text{(A-Set)}$$

$$\mu \Vdash \overline{\mathbf{C}} \circ \mathbf{E}[\mathbf{deref}\ \mathbf{n_i^l}] \mid\mid \mathtt{N}; \mu \Vdash \Sigma \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \circ !l : \tau \quad \text{(A-Der)}$$
$$\textit{where } \mathtt{N}(\mathbf{n_i^l}) = (l_i, \mathtt{Ref}\ \tau)$$

$$\mu \Vdash \overline{\mathbf{C}} \circ \mathbf{E}[\mathbf{fref}^{\tau}\ \mathbf{v}] \mid\mid \mathtt{N}; \mu \Vdash \Sigma \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \circ v \mid\mid \mathtt{N}; \mu \Vdash \Sigma, (\mathtt{ref}\ [\cdot]) : \tau \to \mathtt{Ref}\ \tau \quad \text{(A-Ref)}$$

$$\mu \Vdash \overline{\mathbf{C}}, \mathbf{C} \circ \mathbf{deref}\ \mathbf{n_i^l} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{C}[\mathbf{wr}] \mid\mid \star; \emptyset \Vdash \varepsilon \quad \text{(A-WrD)}$$
$$\textit{where } \mathbf{n_i^l} \notin \mathit{dom}(\mathtt{N}) \textit{ or } \mathtt{N}(\mathbf{n_i}) = (t, \tau \to \tau')$$

$$\mu \Vdash \overline{\mathbf{C}}, \mathbf{C} \circ \mathbf{call}\ \mathbf{n_i^f}\ \mathbf{v} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{C}[\mathbf{wr}] \mid\mid \star; \emptyset \Vdash \varepsilon \quad \text{(A-WrC)}$$
$$\textit{where } \mathbf{n_i^f} \notin \mathit{dom}(\mathtt{N}) \textit{ or } \mathtt{N}(\mathbf{n_i}) = (t, \mathtt{Ref}\ \tau')$$

$$\mu \Vdash \overline{\mathbf{C}}, \mathbf{C} \circ \mathbf{set}\ \mathbf{n_i^l}\ \mathbf{v} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{C}[\mathbf{wr}] \mid\mid \star; \emptyset \Vdash \varepsilon \quad \text{(A-WrS)}$$
$$\textit{where } \mathbf{n_i^l} \notin \mathit{dom}(\mathtt{N}) \textit{ or } \mathtt{N}(\mathbf{n_i^l}) = (t, \tau \to \tau')$$

A command from the attacker is not an input to the MiniML program, but rather a task it must carry out, and is as such not plugged into the head of the stack of evaluation contexts Σ, but is instead executed on top the stack. As was the case for the function application by a MiniML program, applying a λ-term (A-Call), writing to a shared location (A-Set) or referencing a new location (A-Ref) requires two steps. In the first step a new evaluation context is constructed. In the second the argument is marshalled out as described previously. Every time the command does not confirm to the typing rules of MiniML$^+$ the attacker is updated with **wr** (A-WrD,A-WrC,A-WrS). Dereferencing a shared MiniML location (A-Der) requires but one step as it involves only the shared name $\mathbf{n_i^l}$ and thus does not need to marshall out a value.

Note that in each of these rules the current evaluation context of the attacker (**E**) is discarded. While discarding this evaluation context changes the way MiniML^a operates within the FFI, that does not affect its usefulness as an attacker model. On the contrary, we remove it to strengthen the attacker model. As detailed in Section 3.1, the contexts **C** of the attackers call stack $\overline{\mathbf{C}}$ pose a real threat to the abstractions MiniML, whereas an evaluation context **E** doesn't.

3.4 MiniML$^+$ Typing rules

A MiniML$^+$ run-time program state P is type checked by type checking each individual evaluation context of the secure state's evaluation stack Σ as well as each association in the state's map $\mathtt{N}$ and each location in the secure store μ.

$$\frac{\Gamma \vdash \mathtt{N}; \mu \Vdash \Sigma \quad \Gamma, \mathtt{x} : \tau_1 \vdash \mathtt{N} \Vdash \mathtt{E}[x] : \tau_2}{\Gamma \vdash \mathtt{N}; \mu \Vdash \Sigma, \mathtt{E} : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash \mathtt{N} \quad \Gamma \vdash \mu}{\Gamma \vdash \mathtt{N}; \mu \Vdash \varepsilon}$$

$$\frac{\Gamma \vdash \mathtt{N}; \mu \Vdash \Sigma \qquad \Gamma \vdash t : \tau}{\Gamma \vdash \mathtt{N}; \mu \Vdash \Sigma \circ t : \tau} \qquad \frac{\Gamma \vdash \mathtt{N}; \mu \Vdash \Sigma}{\Gamma \vdash \mathtt{N}; \mu \Vdash \Sigma \rhd m : \tau} \qquad \frac{\Gamma \vdash \mathtt{N}; \mu \Vdash \Sigma}{\Gamma \vdash \mathtt{N}; \mu \Vdash \Sigma \lhd m : \tau}$$

$$\frac{\Gamma \vdash \mathtt{N} \qquad \Gamma \vdash t : \tau}{\Gamma \vdash \mathtt{N}; [\mathbf{n_i} \mapsto (t, \tau)]} \qquad \frac{}{\Gamma \vdash \star} \qquad \frac{}{\Gamma \vdash \emptyset} \qquad \frac{\Gamma \vdash \mu \qquad \Gamma \vdash v : \tau}{\Gamma \vdash \mu, l_i = v}$$

Typing the terms of the secure state is done through the regular MiniML typing rules extended with one additional rule for type checking the additional value ${}^{\tau}\mathtt{F}\ (\lambda \mathbf{x}.\mathbf{t})$ that embeds a MiniML^a λ-term.

$$\frac{}{\Gamma \vdash\ {}^{\tau}\mathtt{F}(\lambda \mathbf{x}.\mathbf{t}) : \tau}$$

3.5 Type Soundness

Only the secure state $\mathtt{M}$ of a program P is type checked. As such we cannot rely on a traditional notion of type soundness. Instead, similar to Wadler's and Findler's blame calculus [15], we establish that whenever a program gets stuck or reduces to the error **wr** the attacker is at fault. As usual type soundness is split into theorems of progress and preservation.

Theorem 1 (Type Preservation). *Given* $\Gamma \vdash P$ *and* $P \twoheadrightarrow P'$ *we have* $\Gamma \vdash P'$.

Proof. . By induction on a derivation of $\Gamma \vdash P$, focussing only on the secure state $\mathtt{M}$ of P as that is the only state to be type checked.

– $\Gamma \vdash \mathbf{A} \,||\, \mathtt{N}; \mu \Vdash \Sigma \circ t : \tau$: in this case there are two sub cases.

- Internal reductions:

$$\Gamma \vdash \mathbf{A} \,||\, \mathtt{N}; \mu \Vdash \Sigma \circ t : \tau \twoheadrightarrow \mathbf{A} \,||\, \mathtt{N}; \mu' \Vdash \Sigma \circ t' : \tau$$

  By the fact that the internal reductions of MiniML$^+$ preserve the semantics of MiniML, we conclude the thesis.

$$\Gamma \vdash \mathbf{A} \,||\, \mathtt{N}; \mu' \Vdash \Sigma \circ \mathtt{t}' : \tau$$

- Setup:

$$\Gamma \vdash \mathbf{A} \,||\, \mathtt{N}; \mu \Vdash \Sigma \circ v : \tau \twoheadrightarrow \mathbf{A} \,||\, \mathtt{N}; \mu \Vdash \Sigma \rhd v : \tau$$

  In this transition no terms and types change, we conclude the thesis.

$$\Gamma \vdash \mathbf{A} \,||\, \mathtt{N}; \mu \Vdash \Sigma \rhd v : \tau$$

– $\Gamma \vdash \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \rhd m : \tau$: in this case there are three sub cases.

- Marshalling reductions:

$$\Gamma \vdash \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \rhd m : \tau \twoheadrightarrow \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \rhd m' : \tau$$

  The marshalling rules preform the type checks dynamically: enforcing the thesis.

$$\Gamma \vdash \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \rhd m' : \tau$$

- Share:

$$\mu \Vdash \overline{\mathbf{C}}, \mathbf{C} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \rhd \mathbf{v} : \tau \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{C}[\mathbf{v}] \mid\mid \mathtt{N}; \mu \Vdash \Sigma$$

  It follows from the type rules that $\mathtt{N}$, μ and Σ are type sound and thus we conclude that the thesis holds.

$$\Gamma \vdash \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{C}[\mathbf{v}] \mid\mid \mathtt{N}; \mu \Vdash \Sigma$$

- Type Error:

$$\mu \Vdash \overline{\mathbf{C}}, \mathbf{C} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \rhd \mathbf{wr} : \tau \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}}, \mathbf{C}[\mathbf{wr}] \mid\mid \star; \emptyset \Vdash \varepsilon$$

  The type rules type check only the secure state and type checks the empty state. We thus conclude that the thesis holds.

$$\Gamma \vdash \mu \Vdash \overline{\mathbf{C}}, \mathbf{C}[\mathbf{wr}] \mid\mid \star; \emptyset \Vdash \varepsilon$$

– $\Gamma \vdash \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \lhd m : \tau$: in this case there are three sub cases.

- Marshalling reductions:

$$\Gamma \vdash \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \lhd m : \tau \twoheadrightarrow \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \lhd m' : \tau$$

  The marshalling rules preform the type checks dynamically: enforcing the thesis.

$$\Gamma \vdash \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \lhd m' : \tau$$

- Plug:

$$\mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma, \mathtt{E} : \tau \to \tau' \lhd v : \tau \twoheadrightarrow \mathbf{A} \mid\mid \mathtt{N}', \mu \Vdash \Sigma \circ \mathtt{E}[v] : \tau'$$

  The open evaluation context $\mathtt{E}$ is type checked similar to how a λ-term is type checked. As is the case for λ-term application, context plugging thus holds.

$$\Gamma \vdash \mathbf{A} \mid\mid \mathtt{N}', \mu \Vdash \Sigma \circ \mathtt{E}[v] : \tau'$$

- Type Error:

$$\mu \Vdash \overline{\mathbf{C}}, \mathbf{C} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \lhd \mathbf{wr} : \tau \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}}, \mathbf{C}[\mathbf{wr}] \mid\mid \star; \emptyset \Vdash \varepsilon$$

The type rules type check only the secure state and type checks the empty state. We thus conclude that the thesis holds.

$$\Gamma \vdash \mu \Vdash \overline{\mathbf{C}}, \mathbf{C}[\mathbf{wr}] \mid\mid \star; \emptyset \Vdash \varepsilon$$

– $\Gamma \vdash \mu \Vdash \overline{\mathbf{C}}, \mathbf{C} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \circ \mathtt{E}[({}^{\tau_1 \to \tau_2}\mathtt{F}(\lambda \mathbf{x}.\mathbf{t})\ v)] : \tau$: In this case only one transition is possible:

$$\mu \Vdash \overline{\mathbf{C}}, \mathbf{C} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \circ \mathtt{E}[({}^{\tau_1 \to \tau_2}\mathtt{F}(\lambda \mathbf{x}.\mathbf{t})\ v)] : \tau \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}}, \mathbf{C}[(\mathbf{t}\ [\cdot])] \mid\mid \mathtt{N}; \mu \Vdash \Sigma, \mathtt{E} : \tau_2 \to \tau \rhd v : \tau_1$$

It follows from the subsumed type rules of MiniML that given an application $(\lambda x : \tau_1.t)\ v)$ where the argument type of the λ-term is a type τ_1 then the argument will be of type τ_1 as well. Similarly it follows from the type rules of MiniML that if the return type of the λ-term is τ_2 then the result of the application is of type τ_2. The open context $\mathtt{E}$ thus type checks for an input of type τ_2. We conclude that the thesis holds.

$$\Gamma \vdash \mu \Vdash \overline{\mathbf{C}}, \mathbf{C}[(\mathbf{t}\ [\cdot])] \mid\mid \mathtt{N}; \mu \Vdash \Sigma, \mathtt{E} : \tau_2 \to \tau \rhd v : \tau_1$$

– $\Gamma \vdash \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma$: There are 6 sub cases.

- Input:

$$\mu \Vdash \overline{\mathbf{C}} \circ \mathbf{v} \mid\mid \mathtt{N}; \mu \Vdash \Sigma, \mathtt{E} : \tau \to \tau' \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \lhd \mathbf{v} : \tau$$

The type rules do not restrict the input value $\mathbf{v}$, that is left to the run-time type checks. We conclude that the thesis holds.

$$\Gamma \vdash \mu \Vdash \overline{\mathbf{C}} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \lhd \mathbf{v} : \tau$$

- A-Call:

$$\mu \Vdash \overline{\mathbf{C}} \circ \mathbf{E}[\mathbf{call}\ \mathbf{n_i^f}\ \mathbf{v}] \mid\mid \mathtt{N}; \mu \Vdash \Sigma \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{v} \mid\mid \mathtt{N}; \mu \Vdash \Sigma, (t\ [\cdot]) : \tau \to \tau'$$
$$\textit{where}\ \mathtt{N}(\mathbf{n_i^f}) = (t, \tau \to \tau')$$

It follows from the typing rule for the map $\mathtt{N}$that $\Gamma \vdash t : \tau \to \tau'$, where t is a λ-term, as in MiniML. The newly constructed context applies the λ-term to a hole. The hole must thus be filled in by an argument of type τ. The result will be of type: τ'. We conclude that the thesis holds.

$$\Gamma \vdash \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{v} \mid\mid \mathtt{N}; \mu \Vdash \Sigma, (t\ [\cdot]) : \tau \to \tau'$$

- A-Set:

$$\mu \Vdash \overline{\mathbf{C}} \circ \mathbf{E}[\mathbf{set}\ \mathbf{n_i^l}\ \mathbf{v}] \mid\mid \mathtt{N}; \mu \Vdash \Sigma \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{v} \mid\mid \mathtt{N}; \mu \Vdash \Sigma, (t := [\cdot]) : \tau \to \mathtt{Unit}$$
$$\textit{where}\ \mathtt{N}(\mathbf{n_i^l}) = (t, \mathtt{Ref}\ \tau)$$

It follows from the typing rule for the map Nthat $\Gamma \vdash t : \mathtt{Ref}\ \tau$ as in MiniML. The newly constructed context assigns the location to a hole. The hole must thus be filled in by an argument of type τas per the typing rules of MiniML. An assignment reduces to `unit` and is thus of type `Unit`. We conclude that the thesis holds.

$$\Gamma \vdash \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{v} \mid\mid \mathtt{N}; \mu \Vdash \Sigma, (t := [\cdot]) : \tau \to \mathtt{Unit}$$

- A-Ref:

$$\mu \Vdash \overline{\mathbf{C}} \circ \mathbf{E}[\mathbf{fref}^{\tau}\ \mathbf{v}] \mid\mid \mathtt{N}; \mu \Vdash \Sigma \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \circ v \mid\mid \mathtt{N}; \mu \Vdash \Sigma, (\mathtt{ref}\ [\cdot]) : \tau \to \mathtt{Ref}\ \tau$$

In this case the attacker promises an argument of type τ to the reference operation. This promise is checked by the marshalling rules. It follows from the MiniML typing rules that a term $\Gamma \vdash \mathtt{ref}\ t$ types to $\mathtt{Ref}\ \tau'$ where τ' is the type of t. We conclude that the thesis holds.

$$\mu \Vdash \overline{\mathbf{C}} \circ v \mid\mid \mathtt{N}; \mu \Vdash \Sigma, (\mathtt{ref}\ [\cdot]) : \tau \to \mathtt{Ref}\ \tau$$

- A-Der:

$$\mu \Vdash \overline{\mathbf{C}} \circ \mathbf{E}[\mathbf{deref}\ \mathbf{n_i^l}] \mid\mid \mathtt{N}; \mu \Vdash \Sigma \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \circ !l : \tau$$
$$\textit{where}\ \mathtt{N}(\mathbf{n_i^l}) = (l_i, \mathtt{Ref}\ \tau)$$

It follows from the typing rule for the map Nthat $\Gamma \vdash l_i : \mathtt{Ref}\ \tau$ as in MiniML. It also follows from the typing rules of MiniML that $\Gamma \vdash !l_1 : \tau$. We conclude that the thesis holds.

$$\Gamma \vdash \mu \Vdash \overline{\mathbf{C}} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \circ !l : \tau$$

- A-Wr*:

$$\mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \twoheadrightarrow \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{C}[\mathbf{wr}] \mid\mid \star; \emptyset \Vdash \varepsilon$$

As was the case for the error rules of the marshalling, we note that the type rules type check only the secure state and type checks the empty state, and thus conclude that the thesis holds.

$$\Gamma \vdash \mu \Vdash \overline{\mathbf{C}}, \mathbf{C}[\mathbf{wr}] \mid\mid \star; \emptyset \Vdash \varepsilon$$

□

Theorem 2 (Type Progress). *Given* $\Gamma \vdash P$ *then if* $P \twoheadrightarrow \mu \Vdash \mathbf{wr} \mid\mid \star, \emptyset \Vdash \varepsilon$ *or* $P \not\twoheadrightarrow P'$ *then the attacker is the cause.*

Proof. By induction on a derivation of $\Gamma \vdash P$, focussing only on the secure state $\mathtt{M}$ of P as that is the only state to be type checked.

- $\Gamma \vdash \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \circ t : \tau$: It follows from the typing rules of MiniML that the term t is either a value v in which case Setup applies, or else there exists a t' such that:

$$\Gamma \vdash \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \circ t : \tau \twoheadrightarrow \mathbf{A} \mid\mid \mathtt{N}; \mu' \Vdash \Sigma \circ t' : \tau$$

- $\Gamma \vdash \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \rhd m : \tau$: There are three cases.

  - $m = \mathbf{v}$: In this case the reduction rule Setup applies.
  - $m = \mathbf{wr}$: A type error is only possible in the case where ${}^{\tau_1 \to \tau_2}\mathtt{F}(\lambda \mathbf{x}.\mathbf{t})$ does not embed a λ-term.
  - Otherwise by the completeness of the marshalling rules, we have that there exists a m' such that:

$$\Gamma \vdash \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \rhd m : \tau \twoheadrightarrow \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \rhd m' : \tau$$

- $\Gamma \vdash \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \lhd m : \tau$: There are three cases.

  - $m = \mathbf{v}$: In this case the reduction rule Setup applies.
  - $m = \mathbf{wr}$: A type error is only possible if the input from the attacker does not conform to the type τ.
  - Otherwise by the completeness of the marshalling rules, we have that there exists a m' such that:

$$\Gamma \vdash \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \rhd m : \tau \twoheadrightarrow \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \rhd m' : \tau$$

- $\mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma \circ \mathtt{E}[({}^{\tau_1 \to \tau_2}\mathtt{F}(\lambda \mathbf{x}.\mathbf{t})\ v)] : \tau$: There are two sub cases depending on the form of the attacker $\mathbf{A}$.

  - $\mathbf{A} = \mu \Vdash \overline{\mathbf{C}}, \mathbf{C}$: In this case the reduction rule M-Call applies.
  - $\mathbf{A} = \mu \Vdash \varepsilon$: In this case no reduction is possible, because the attacker failed to supply sufficient contexts.
- $\Gamma \vdash \mathbf{A} \mid\mid \mathtt{N}; \mu \Vdash \Sigma, \mathtt{E} : \tau \to \tau'$: In this case there are 5 sub cases depending on the form of the attacker $\mathbf{A}$.

  - $\mathbf{A} = \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{v}$: if $\mathtt{M}$ is of the form: $\mathtt{N}; \mu \Vdash \Sigma, \mathtt{E} : \tau \to \tau'$: the reduction rule Input applies. Otherwise the program is stuck due to an incorrectly formed context by the attacker.
  - $\mathbf{A} = \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{E}[\mathbf{call}\ \mathbf{n_i^f}\ \mathbf{v}]$: if $\mathbf{n_i^f}$ is in $\mathtt{N}$ then the rule A-Call applies, otherwise by A-WrC we have that the attacker will reduce to error due to an inappropriate use of the name $\mathbf{n_i^f}$.

- $\mathbf{A} = \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{E}[\mathbf{set}\ \mathbf{n_i^l}\ \mathbf{v}]$: if $\mathbf{n_i^l}$ is in $\mathtt{N}$ then the rule A-Set applies, otherwise by A-WrS we have that the attacker will reduce to error due to an inappropriate use of the name $\mathbf{n_i^l}$.
- $\mathbf{A} = \mu \Vdash \overline{\mathbf{C}} \circ \mathbf{E}[\mathbf{deref}\ \mathbf{n_i^l}]$: if $\mathbf{n_i^l}$ is in $\mathtt{N}$ then the rule A-Der applies, otherwise by A-WrD we have that the attacker will reduce to error due to an inappropriate use of the name $\mathbf{n_i^l}$.
- $\mu \Vdash \overline{\mathbf{C}} \circ \mathbf{E}[\mathbf{fref}^{\tau}\ \mathbf{v}]$: in this case the MiniML$^+$ reduction rule A-Ref applies.

□

3.6 Contextual Equivalence

The MiniML$^+$-calculus program state P combines the secure state $\mathtt{M}$ and the attacker state $\mathbf{A}$. However, our definition of contextual equivalence will only relate the secure states $\mathtt{M}$ that embed the MiniML program as preserving the security properties of MiniML in MiniML$^+$ is the goal of this paper. The attacker state $\mathbf{A}$ thus serves as the context in which the secure state $\mathtt{M}$ operates.

Definition 2. Contextual equivalence over MiniML$^+$ ($\simeq^+$) *is defined as:*

$$\mathtt{M}_1 \simeq \mathtt{M}_2 \stackrel{def}{=} \forall \mathbf{A}.(\mathbf{A} \mid\mid \mathtt{M}_1)\Uparrow \iff (\mathbf{A} \mid\mid \mathtt{M}_2)\Uparrow$$

Consider, for example, the equivalent λ-terms of Ex-1 in Section 2.1. These λ-terms remain equivalent when placed within two secure states as follows.

$$\star; \emptyset \Vdash \varepsilon \circ (\lambda x : \mathtt{Int}.(+\ x\ x)) \qquad \star; \emptyset \Vdash \varepsilon \circ (\lambda x : \mathtt{Int}.(*\ 2\ x))$$

There exists no attacker $\mathbf{A}$ that can distinguish these two secure states. The marshalling out rule for λ-terms will convert both λ-terms to the name $\mathbf{n_1}$ as they are the first λ-term to be shared with the attacker. An attacker $\mathbf{A}$ will observe that name, but cannot observe that the names refer to λ-terms that are not α-equivalent as, due to the dynamic type checking rules of MiniML$^+$,it can only apply the name $\mathbf{n_1}$ to numbers $\overline{n}$ as in MiniML.

Alternatively, the following two secure states are not contextually equivalent.

$$\star, \mathbf{n_1^f} \mapsto (\lambda x : \mathtt{Ref\ Int}.\overline{1}); \emptyset \Vdash \varepsilon \qquad \star, \mathbf{n_1^f} \mapsto (\lambda x : \mathtt{Ref\ Int}.!x); \emptyset \Vdash \varepsilon$$

As an attacker $\mathbf{A} = (\emptyset \Vdash (\mathbf{if}(\overline{\mathbf{1}} == [\cdot])\ \Omega\ \mathbf{true}), \mathbf{call}\ \mathbf{n_1^f}\ [\cdot] \circ \mathbf{fref}^{\tau}\ \overline{\mathbf{2}})$ can distinguish them. Reducing $\mathbf{fref}^{\tau}\ \overline{\mathbf{2}}$ will result in both secure states returning a name $\mathbf{n_1^l}$ associated with a location l_1 where: $l_1 \mapsto \overline{2}$. This name $\mathbf{n_1^l}$ serves as input to the second context where the name $\mathbf{n_1^f}$ is applied to it. The name $\mathbf{n_i^f}$ refers to terms that are not equivalent in MiniML, a fact that is subsequently observed.

4 Full Abstraction

To establish that the MiniML$^+$-calculus is a secure FFI, we show that the FFI preserves the equivalences beteen MiniML terms despite the presence of the attacker. Direct proofs over contextual equivalence are, however, difficult as one

needs to reason about every reduction in every context. To that end we develop notions of bisimulation that coincide with contextual equivalence for MiniML (Section 4.1) and for MiniML$^+$ (Section 4.2). Proving that the FFI is fully abstract is done by relating these bisimulations (Section 4.3).

4.1 A Congruent Bisimulation for MiniML

We define a bisimulation relation $\mathcal{S}$ over the programs of MiniML that is congruent: it coincides with the contextual equivalence relation $\simeq$. There have been multiple different bisimulations and trace semantics over typed λ-calculi with references. In this paper we use an applicative bismulation that is a combination of Jeffrey's and Rathke's applicative bisimulation for the $v$ref-calculus [8] and the fully abstract trace semantics for the $\lambda_\mu$hashref-calculus by Jagadeesan [7] Applicative bisimulation is defined through an LTS. The LTS models the interactions between a MiniML context C and a MiniML program. The LTS is formally defined as a triple $(\zeta, \alpha, \xrightarrow{\alpha})$. The state $\zeta = K; \mu \mid t$ is the MiniML run-time state extended with a sequence K that records the locations l_i that the opponent has knowledge of. This is needed to capture fact that locations are not part of the static semantics and thus do not appear in contexts unless made available at run-time. The labels α of the LTS are defined as follows.

$$\alpha ::= \gamma \mid \tau \qquad \gamma ::= @v \mid .i \mid l_i := v \mid \texttt{ref}\ v \mid !l_i \mid l_i \mid b \mid \overline{n} \mid \texttt{unit}$$

The most relevant labelled reductions are as follows.

$$K; \mu \mid t \xrightarrow{\tau} K; \mu' \mid t' \quad (\mathsf{Sil}) \qquad K; \mu \mid \overline{n} \xrightarrow{\overline{n}} K; \mu' \mid \overline{n} \quad (\mathsf{O\text{-}N})$$

$$K; \mu \mid (\lambda x : \tau.t) \xrightarrow{@v} K; \mu \mid ((\lambda x : \tau.t)\ v) \quad \textit{where} \vdash v : \tau \quad (\mathsf{I\text{-}App})$$

$$K; \mu \mid v \xrightarrow{l_i := v'} K; \mu \mid l_i := v' \quad \textit{where } l \in K \textit{ and } \vdash l_i : \tau \textit{ and } \vdash v : \tau \quad (\mathsf{I\text{-}S})$$

$$K; \mu \mid v \xrightarrow{!l_i} K; \mu \mid !l \quad \textit{if } l_i \in K \quad (\mathsf{I\text{-}D}) \qquad K; \mu \mid l_i \xrightarrow{l_i} K, l_i; \mu \mid v_i \quad (\mathsf{O\text{-}L})$$

$$K; \mu \mid v \xrightarrow{\texttt{ref}\ v'} K; \mu \mid \texttt{ref}\ v' \quad (\mathsf{I\text{-}Ref}) \quad K; \mu \mid \langle v_i^{i \in 1..n} \rangle \xrightarrow{.i} K; \mu \mid v_i \quad (\mathsf{I\text{-}Proj})$$

$$K; \mu \mid \texttt{unit} \xrightarrow{\texttt{unit}} K; \mu \mid \texttt{unit} \quad (\mathsf{O\text{-}U})$$

$$K; \mu \mid \texttt{true} \xrightarrow{\texttt{true}} K; \mu \mid \texttt{true} \quad (\mathsf{O\text{-}T})$$

$$K; \mu \mid \texttt{false} \xrightarrow{\texttt{false}} K; \mu \mid \texttt{false} \quad (\mathsf{O\text{-}F})$$

Reduction steps between terms cannot be observed by a context and are thus labelled as silent through the label τ (Sil). Whenever a MiniML program reduces to a value that is not a λ-term or tuple (as it may contain a λ-term), the context can observe that value (O-N,O-L,O-T,O-F,O-U). Observing a label (O-L) is a special case as it adds a new location l_i to K : the list of observed locations. A context interacts with a λ-term by applying it to values (I-App), likewise a context queries members of a tuple instead of observing it directly (I-Proj). A context can also dereference observed locations l_i (I-D), create new ones (I-Ref) and assign them values (I-S).

We define a weak bisimulation over this LTS. In contrast to a strong bisimulation, such a bisimulation does not use the silent transitions between two states. Define the transition relation $\zeta \overset{\gamma}{\Rightarrow} \zeta'$ as $\zeta \xrightarrow{\tau}^{*} \xrightarrow{\gamma} \zeta'$ where $\xrightarrow{\tau}^{*}$ is the reflexive transitive closure of the silent transitions $\xrightarrow{\tau}$. A Bisimulation over this LTS is now formally defined as follows.

Definition 3. *The relation $\mathcal{S}$ is a* ***bisimulation*** *iff $\zeta_1 \; \mathcal{S} \; \zeta_2$ implies:*
(1) Given $\zeta_1 \overset{\gamma}{\Rightarrow} \zeta_1'$ there is a ζ_2' such that: $\zeta_2 \overset{\gamma}{\Rightarrow} \zeta_2'$ and $\zeta_1' \; \mathcal{S} \; \zeta_2'$
(2) Given $\zeta_2 \overset{\gamma}{\Rightarrow} \zeta_2'$ there is a ζ_1' such that: $\zeta_1 \overset{\gamma}{\Rightarrow} \zeta_1'$ and $\zeta_1' \; \mathcal{S} \; \zeta_2'$

We denote bisimilarity, the largest bisimulation, as $\approx$. We now establish that the bisimilarity $\approx$ coincides with contextual equivalence $\simeq$.

Theorem 3 (Congruence of the Bisimilarity). $t_1 \simeq t_2 \;\Leftrightarrow\; t_1 \approx t_2$

A proof of this theorem is an adaption of existing results [8,7]. The proof splits the theorem into two sublemma: contextual equivalence implies bisimilarity (Completeness) and bisimilarity implies contextual equivalence (Soundness).

Theorem 4 (Completeness). $t_1 \simeq t_2 \Rightarrow t_1 \approx t_2$

Proof. Proving that MiniML contextual equivalence implies bisimilarity is done by showing that the contextual equivalence relation is itself a bisimulation. Assume that: $t_1 \simeq t_2$. Because bisimilarity is symmetrical, we divide the proof into two parts:

1. Assume that: $\mu \mid t_1 \overset{\gamma}{\Rightarrow} \mu' \mid t_1'$. We now establish that there exists a $\mu_2' \mid t_2'$ such that:
   (a) $\mu_2 \mid t_2 \overset{\gamma}{\Rightarrow} \mu_2' \mid t_2'$
   (b) $t_1' \simeq t_2'$
   The proof proceeds by case analysis on the label γ. For every label that decodes an action of an attacker we establish the label can be encoded as a MiniML context `C`. Note that we simplify the proof by only establishing that there exists a context that can distinguish between t_1 and t_2, to comply with the definition of contextual equivalence the context must also diverge in one case.
   - `true`: From the LTS transition O-T it follows that t_1 reduces to `true`. It follows from $t_1 \simeq t_2$ that t_2 also reduces to true (and thus produces a label `true`) as otherwise the context: ($[\cdot]$ == `true`) can distinguish between t_1 and t_2. It also follows from $t_1 \simeq t_2$ that μ and μ_2 or equivalent before and after reduction in that the locations point to contextually equivalent terms, otherwise a context C: ($[\cdot]; !l_i ...$) could distinguish them. It follows from O-T that $t_1' = t_2' =$ `true`. It also follows from O-T that the stores are unchanged by the labelled transition. Both theses thus hold.
   - `false`: Similar to the `true` case.
   - `unit`: Similar to the `true` case.

- $\overline{n}$: Similar to the `true` case.
- l_i: From the LTS transition O-L it follows that t_1 reduces to l_i. It follows from $t_1 \simeq t_2$ that t_2 also reduces to a location l_i (and thus produces a label l_i) as otherwise the contexts: ($\texttt{hash}[\cdot] == \bar{i}$) can distinguish between t_1 and t_2. The stores are equal as detailed in the `true` case. It follows from O-L that $t_1' = t_2' = l_i$. It also follows from O-L that the stores are unchanged by the labelled transition. Both theses thus hold.
- $@v$: From the contextual equivalence between t_1 and t_2 it follows that t_1 and t_2 are both function types and thus both reduce to λ-terms v_1 and v_2. Given that $\gamma = @v$ it now follows from the LTS that $t_1' = (v_1\ v)$ and $t_2' = (v_2\ v)$. The label $@v$ can thus be encoded as the context $C = ([\cdot]\ v)$, because contextual equivalence is closed under contexts we can now conclude that: $(v_1\ v) \simeq (v_2\ v)$.
- $.i$: From the contextual equivalence between t_1 and t_2 it follows that both t_1 and t_2 are both tuple types of the same length n and thus both reduce to tuples of length n: $\langle v_i^{i \in 1..n} \rangle$ and $\langle v_{i'}^{i' \in 1..n} \rangle$, where $\forall i \in 1..n. v_i \simeq v_i'$. Otherwise depending on the type of a member v_i the following context could distinguish them:
  - $v_i = \overline{n} \Rightarrow C = ([\cdot].i == \overline{n})$
  - $v_i = b \Rightarrow C = ([\cdot].i == b)$
  - $v_i = l_j \Rightarrow C = (hash([\cdot].i) == \overline{j})$
  - $v_i = (\lambda x : \tau.t) \Rightarrow C = (([\cdot].i)@v)$
  - $v_i = \langle v_j^{j \in 1..n} \rangle \Rightarrow C = (([\cdot].i).j...)$

  The label $.i$ can be encoded as the context $C = ([\cdot].i)$, because contextual equivalence is closed under contexts we can now conclude that: $\langle v_i^{i \in 1..n} \rangle.i \simeq \langle v_{i'}^{i' \in 1..n} \rangle.i$.
- $!l_i$: It follows from I-D that $t_1' = !l$ and $t_2' = !l$ and that $l_i \in K_1$. By $t_1 \simeq t_2$ we have that $K_1 = K_2$. The label $!\texttt{l}_\texttt{i}$ can thus be encoded as the context $C = ([\cdot]; !l_i)$, because contextual equivalence is closed under contexts we can now conclude that: $(t_1; !l_i) \simeq (t_2; !l_i)$.
- $l_i := v$: Similar to the $!l$ case.
- $\texttt{ref}v$: Similar to the $!l$ case.

2. As in case 1, *mutatis mutandis*.

□

Theorem 5 (Soundness). $t_1 \approx t_2 \Rightarrow t_1 \simeq t_2$

Proof. The thesis $t_1 \simeq t_2$ becomes $\forall C.\ C[t_1]\Uparrow \iff C[t_2]\Uparrow$ The proof is divided into two cases, one case for each side of the co-implication.

1. $\Rightarrow$ In this case the thesis is $\forall C.\ C[t_1]\Uparrow \Rightarrow C[t_2]\Uparrow$. The thesis is redefined as:

$$\forall C.\ \forall k \in \mathbb{N}.\ C[t_1] \rightarrow^k C[t_1]' \Rightarrow \forall m \in \mathbb{N}.\ C[t_2] \rightarrow^m C[t_2]'$$

The proof proceeds by induction on m.
Base case: $m = 0$. Straightforward: $C[t_2] \rightarrow^{0} C[t_2]$.
Inductive case: $m = h + 1$. The thesis is: $C[t_2] \rightarrow^{h+1} C[t_2]'$. The inductive hypotheses (IH) is:

$$\forall C.\ \forall k \in \mathbb{N}.\ C[t_1] \rightarrow^{k} C[t_1]' \Rightarrow C[t_2] \rightarrow^{h} C[t_2]^{h}$$

We know from this IH that:

$$\exists C.\ C[t_1] \rightarrow^{h} C[t_1]^{h} \rightarrow^{k-h} C[t_1]'$$

We prove the thesis by reasoning about what the presence or absence of the last observable label γ tells us about the existence of a next reduction step $h + 1$. There are two cases: either the context C reduces or the term t reduces.

(a) The MiniML term t is executing and the context is passive. In this case there are two sub-cases:
   i. $\exists \gamma.\ \mu^{h} \mid t_1^{h} \overset{\gamma}{\Rightarrow} \mu^{h'} \mid t_1^{h'}$. By the assumption $t_1 \approx t_2$ we conclude that $\mu_2^{h} \mid t_2^{h} \overset{\gamma}{\Rightarrow} \mu_2^{h'} \mid t_2^{h'}$ and $t_1^{h'} \approx t_2^{h'}$. This, in conjunction with the IH, implies the thesis:

$$C[t_2] \rightarrow^{h+1} C[t_2]'$$

   ii. $\nexists \gamma.\ \mu^{h} \mid t_1^{h} \overset{\gamma}{\Rightarrow} \mu^{h'} \mid t_1^{h'}$. As per the definition of $\overset{\gamma}{\Rightarrow}$ we have that this is only possible if the term is diverging. By the assumption $t_1 \approx t_2$ we have that both t_1 and t_2 diverge after producing the same set of labels, which implies the thesis.

(b) The context C is executing and the term t is a passive value v. In this case there are two sub-cases as well:
   i. $\exists \gamma.\ \mu^{h} \mid v_1^{h} \overset{\gamma}{\Rightarrow} \mu^{h'} \mid t_1^{h'}$. Because the observable label γ is produced by the context, we must thus show that: $C_1^{h} = C_2^{h}$, where the existence of C_2^{h} derives from the induction hypothesis. We know by the assumption: $t_1 \approx t_2$ that both terms where modified by the same stream of observable labels if there are any such labels : $\exists k \in \mathbb{N}.k \leq h \wedge C[t_1] \rightarrow^{k} C[t_1]^{k} \wedge C[t_1]^{k} \rightarrow^{h-k} C[t_1]^{h}$ where $C[t_1] \overset{\overline{\gamma}}{\Rightarrow} C[t_1]_1^{k}$ and that $\exists k \in \mathbb{N}.k \leq h \wedge C[t_2] \rightarrow^{k} C[t_2]^{k} \wedge C[t_2]^{k} \rightarrow^{h-k} C[t_2]^{h}$ where $C \overset{\overline{\gamma}}{\Rightarrow} C[t-2]^{k}$.
   Combining the fact that the reduction rules of MiniML are deterministic and with the fact that the MiniML-contexts are updated in the same way by identical labels γ, we conclude that $C_1^{h} = C_2^{h}$ and that $C[t_2]^{h} \overset{\gamma}{\Rightarrow} C[t_2]^{h'}$. This implies the thesis.
   ii. $\nexists \gamma.\ \mu^{h} \mid v_1^{h} \overset{\gamma}{\Rightarrow} \mu^{h'} \mid t_1^{h'}$. If there exists no label γ the MiniML-context is diverging. In the previous case we established that $C_1^{h} = C_2^{h}$. As such both $C[t_1]$ and $C[t_2]$ divergence, which implies the thesis.

2. $\Leftarrow$ As in case 1, *mutatis mutandis*.

□

4.2 A Bisimulation for the MiniML$^+$-Calculus

We define a notion of bismulation ($\mathcal{S}^+$) that coincides with the contextual equivalence relation ($\simeq^+$). Again we rely on an applicative bisimulation defined through an LTS. The LTS is a triple $(\mathtt{M}, \alpha^+, \xrightarrow{\alpha^+})$ where the secure states $\mathtt{M}$ are the states, α^+ the set of labels and $\xrightarrow{\alpha^+}$ the labelled transitions between states. The labels α^+, which denote the observations of the attacker, are defined as follows.

$$\alpha^+ ::= \gamma^+ \mid \tau^+ \mid \surd$$

$$\gamma^+ ::= \mathbf{v}? \mid \mathbf{v}! \mid \mathbf{wr} \mid \gg (\lambda \mathbf{x}.\mathbf{t}) \mid \gg \mathbf{n_i^l} \mid \gg \mathbf{n_i^f} \mid \gg \mathbf{ref}^\tau \mid !\mathbf{n_i^l}$$

The labelled reductions of the LTS are of the form: $\mathtt{M} \xrightarrow{\gamma^+} \mathtt{M}'$. Although the attacker state $\mathbf{A}$ is not represented in these labelled reductions, the changes to the attacker state can be derived from the labels. The transitions are as follows.

$$\mathtt{N}; \mu \Vdash \Sigma \circ t : \tau \xrightarrow{\tau} \mathtt{N}; \mu' \Vdash \Sigma \circ t' : \tau \qquad \text{(S-Inner)}$$

$$\mathtt{N}; \mu \Vdash \Sigma \circ v : \tau \xrightarrow{\tau} \mathtt{N}; \mu \Vdash \Sigma \rhd v : \tau \qquad \text{(S-Setup)}$$

$$\mathtt{N}; \mu \Vdash \Sigma, \mathtt{E} \lhd v : \tau \xrightarrow{\tau} \mathtt{N}; \mu \Vdash \Sigma \circ \mathtt{E}[v] : \tau \qquad \text{(S-Plug)}$$

$$\mathtt{N}; \mu \Vdash \Sigma \lhd m : \tau \xrightarrow{\tau} \mathtt{N}; \mu \Vdash \Sigma \lhd m' : \tau \qquad \text{(S-MarshIN)}$$

$$\mathtt{N}; \mu \Vdash \Sigma \rhd m : \tau \xrightarrow{\tau} \mathtt{N}'; \mu \Vdash \Sigma \rhd m' : \tau \qquad \text{(S-MarshOut)}$$

$$\mathtt{N}; \mu \Vdash \Sigma, \mathtt{E} : \tau \to \tau' \xrightarrow{\mathbf{v}?} \mathtt{N} \Vdash \Sigma, \mathtt{E} : \tau \to \tau' \lhd \mathbf{v} : \tau \qquad \text{(A-V)}$$

$$\mathtt{N}; \mu \Vdash \Sigma \rhd \mathbf{v} : \tau \xrightarrow{\mathbf{v}!} \mathtt{N}; \mu \Vdash \Sigma \qquad \text{(M-V)}$$

$$\mathtt{N}; \mu \Vdash \Sigma \rhd \mathbf{wr} : \tau \xrightarrow{\mathbf{wr}} \star; \emptyset \Vdash \varepsilon \quad \text{(Wr-O)} \qquad \mathtt{N}; \mu \Vdash \Sigma \xrightarrow{\mathbf{wr}} \star; \emptyset \Vdash \varepsilon \quad \text{(Wr-C)}$$

$$\mathtt{N}; \mu \Vdash \Sigma \lhd \mathbf{wr} : \tau \xrightarrow{\mathbf{wr}} \star; \emptyset \Vdash \varepsilon \quad \text{(Wr-I)} \qquad \star; \mu \Vdash \varepsilon \xrightarrow{\surd} \star; \emptyset \Vdash \varepsilon \quad \text{(Done)}$$

$$\mathtt{N}; \mu \Vdash \Sigma \xrightarrow{\gg \mathbf{ref}^\tau} \mathtt{N}; \mu \Vdash \Sigma, (\mathtt{ref}\ [\cdot]) : \tau \to \mathtt{Ref}\ \tau \qquad \text{(A-R)}$$

$$\mathtt{N}; \mu \Vdash \Sigma \xrightarrow{!\mathbf{n_i^l}} \mathtt{N}; \mu \Vdash \Sigma \circ !l_i : \tau \quad \textit{where } \mathtt{N}(\mathbf{n_i^l}) = (l_i, \mathtt{Ref}\ \tau) \qquad \text{(D-N)}$$

$$\mathtt{N}; \mu \Vdash \Sigma \xrightarrow{\gg \mathbf{n_i^f}} \mathtt{N}; \mu \Vdash \Sigma, (t\ [\cdot]) : \tau \to \tau' \quad \textit{where } \mathtt{N}(\mathbf{n_i^f}) = (t, \tau \to \tau') \qquad \text{(C-N)}$$

$$\mathtt{N}; \mu \Vdash \Sigma \xrightarrow{\gg \mathbf{n_i^l}} \mathtt{N}; \mu \Vdash \Sigma, (l_i := [\cdot]) : \tau \to \mathtt{Unit} \quad \textit{if } \mathtt{N}(\mathbf{n_i^l}) = (l_i, \mathtt{Ref}\ \tau) \qquad \text{(S-N)}$$

$$\mathtt{N}; \mu \Vdash \Sigma \circ \mathtt{E}[({}^{\tau_1 \to \tau_2}\mathtt{F}(\lambda \mathbf{x}.\mathbf{t})\ v)] : \tau \xrightarrow{\gg (\lambda \mathbf{x}.\mathbf{t})} \mathtt{N}; \mu \Vdash \Sigma, \mathtt{E} : \tau_2 \to \tau \rhd v : \tau_1 \quad \text{(C-L)}$$

The internal reduction steps, the marshalling transitions as well as the rules that setup the marshalling and plug the stack Σ are labelled as silent through the label τ (S-*). The values $\mathbf{v}$ that the attacker returns or inputs are decorated with ? (A-V). Likewise the inputs or returned values of the secure state, converted to MiniML^a values $\mathbf{v}$ by the marshalling rules, are decorated with ! (M-V). Whenever the marshalling fails (Wr-O,Wr-I) or the attacker makes an inappropriate call (Wr-C), the transition is labelled as wrong $\mathbf{wr}$. Dereferencing shared names is a one step transition and is labelled accordingly (D-N).

Setting and creating shared locations (S-N,A-R) or applying shared λ-terms (C-N,C-L), are as detailed in Section 3.3 two step operations which are captured by two labels. In the first step, whose label is decorated with $\gg$, a new context is constructed that encodes the shared term and the operation to be performed on it. In the second step the argument is passed across the boundary as captured by the value sharing rules (A-V,M-V). Note that when the secure state applies a MiniML^a (C-L) the argument is marshalled first (S-MarshOut).

As in Section 4.1 we define weak bisimulation. Define the transition relation $\mathtt{M} \overset{\gamma^+}{\Longrightarrow} \mathtt{M}'$ as $\mathtt{M} \xrightarrow{\tau^+}^{*} \xrightarrow{\gamma^+} \mathtt{M}'$ where $\xrightarrow{\tau^+}^{*}$ is the reflexive transitive closure of the silent transitions $\xrightarrow{\tau^+}$. Bisimulation is now defined as follows.

Definition 4. *The relation $\mathcal{S}^+$ is a **bisimulation** iff $\mathtt{M}_1 \ \mathcal{S}^+ \ \mathtt{M}_2$ implies:*

(1) Given $\mathtt{M}_1 \overset{\gamma^+}{\Longrightarrow} \mathtt{M}'_1$ there is $\mathtt{M}'_2$ such that: $\mathtt{M}_2 \overset{\gamma^+}{\Longrightarrow} \mathtt{M}'_2$ and $\mathtt{M}'_1 \ \mathcal{S}^+ \ \mathtt{M}'_2$

(2) Given $\mathtt{M}_2 \overset{\gamma^+}{\Longrightarrow} \mathtt{M}'_2$ there is $\mathtt{M}'_1$ such that: $\mathtt{M}_1 \overset{\gamma^+}{\Longrightarrow} \mathtt{M}'_1$ and $\mathtt{M}'_1 \ \mathcal{S}^+ \ \mathtt{M}'_2$

Again, we denote bisimilarity as $\approx^+$ and prove that it is a congruence.

Theorem 6 (Congruence of the Bisimilarity). $\mathtt{M}_1 \simeq^+ \mathtt{M}_2 \iff \mathtt{M}_1 \approx^+ \mathtt{M}_2$

The proof splits the thesis into two sublemma: completeness and soundness.

Lemma 1. (Completeness) $\mathtt{M}_1 \simeq^+ \mathtt{M}_2 \Rightarrow \mathtt{M}^1 \approx^+ \mathtt{M}^2$

Proof. To prove that contextual equivalence implies bisimilarity we show that the contextual equivalence relation ($\simeq^+$) is itself a bisimulation ($\mathcal{S}^+$). Assume that: $\mathtt{M}_1 \simeq^+ \mathtt{M}_2$. Because bisimilarity is symmetrical, we divide the proof into two parts.

1. Assume that: $\mathtt{M}_1 \overset{\gamma^+}{\Longrightarrow} \mathtt{M}'_1$. We must show that there exists a $\mathtt{M}_2$ such that (1) $\mathtt{M}_2 \overset{\gamma^+}{\Longrightarrow} \mathtt{M}'_2$ and that (2) $\mathtt{M}'_1 \simeq^+ \mathtt{M}'_2$. The proof proceeds by case analysis on the labels γ^+. For the labels produced by the secure state $\mathtt{M}$ we rely on the fact that it follows from the assumption $\mathtt{M}_1 \simeq^+ \mathtt{M}_2$ that the MiniML terms t_1 and t_2 reduced by both states are contextually equivalent as well and thus reduce to the same value and produce the same label. For the labels produced by the attacker state $\mathbf{A}$ we simply show that every label produced by the attacker can be encoded as a context $\mathbf{C}$, because contextual equivalence is closed under contexts that suffices to imply the thesis.
   - $\gamma^+ = \surd$: It follows from the LTS rule Done that $\mathtt{M}_1 = \star; \mu \Vdash \varepsilon$ and that $\mathtt{M}'_1 = \star; \emptyset \Vdash \varepsilon$. From the assumption $\mathtt{M}_1 \simeq^+ \mathtt{M}_2$ we have that $\mathtt{M}_2 = \star; \mu \Vdash \varepsilon$ as otherwise there exists a context $\mathbf{A}$ that can distinguish $\mathtt{M}_1$ and $\mathtt{M}_2$ as follows.
     - $\mathtt{N} \neq \star$: In this case there is some name $\mathbf{n_i}$ that the attacker $\mathbf{A}$ can invoke as per the rules C-N and S-N.
     - $\Sigma \neq \varepsilon$: In this case there is some input $\mathbf{v}$ that the secure context $\mathtt{M}$ accepts from the attacker $\mathbf{A}$ as per the rule A-V.

- $\star; \mu \Vdash \varepsilon \circ t \;\mid\; \star; \mu \Vdash \varepsilon \rhd m \;\mid\; \star; \mu \Vdash \varepsilon \lhd m$: In this case $\mathtt{M}_2$ will either produce a value as per M-V which the attacker can observe, otherwise $\mathtt{M}_2$ diverges in which case the attacker distinguishes the states by default.

It follows from the LTS that $\mathtt{M}_2 = \star; \mu \Vdash \varepsilon \overset{\surd}{\Rightarrow} \mathtt{M}_2' = \star; \emptyset \Vdash \varepsilon$, we conclude that thesis (1) and (2) hold.

– $\gamma^+ = \mathbf{v}!$: It follows from the LTS rule M-V that $\mathtt{M}_1 = \mathtt{N}; \mu \Vdash \Sigma \rhd \mathbf{v}$ and that $\mathtt{M}_1' = \mathtt{N}; \mu \Vdash \Sigma$. From the assumption $\mathtt{M}_1 \simeq^+ \mathtt{M}_2$ we have that $\mathtt{M}_2 = \mathtt{N}'; \mu' \Vdash \Sigma' \rhd \mathbf{v}'$ where $\mathbf{v}' = \mathbf{v}$, $\forall \mathbf{n_i} \in Dom(\mathtt{N}).\mathtt{N}(\mathbf{n_i}) \simeq \mathtt{N}'(\mathbf{n_i})$ and $\forall \mathtt{E} \in \Sigma. \forall \mathtt{E}' \in \Sigma'. \mathtt{E} \simeq \mathtt{E}'$: as otherwise there exists a context $\mathbf{A}$ that can distinguish $\mathtt{M}_1$ and $\mathtt{M}_2$ as follows.
  - $\mathtt{M}_2 = \mathtt{N}'; \mu' \Vdash \Sigma' \rhd \mathbf{v}' \;\wedge\; \mathbf{v}' \neq \mathbf{v}$: The attacker $\mathbf{A} = ([\cdot] \equiv_\alpha \mathbf{v})$ can distinguish $\mathtt{M}_1$ and $\mathtt{M}_2$.
  - $\mathtt{M}_2 \Uparrow$: $\mathtt{M}_1$ does not diverge, the attacker does distinguishes them by default.
  - $\mathtt{N} \not\simeq \mathtt{N}'$: In this case there is some name $\mathbf{n_i}$ that the attacker $\mathbf{A}$ can invoke as per the rules C-N and S-N to distinguish $\mathtt{M}_1$ and $\mathtt{M}_2$.
  - $\Sigma \not\simeq \Sigma'$: In this case there is some input $\mathbf{v}$ that the secure context $\mathtt{M}$ accepts from the attacker $\mathbf{A}$ as per the rule A-V that can be used to distinguish $\mathtt{M}_1$ and $\mathtt{M}_2$.

It follows from the LTS that $\mathtt{M}_2 = \mathtt{N}'; \mu' \Vdash \Sigma' \rhd \mathbf{v} \overset{\mathbf{v}}{\Rightarrow} \mathtt{M}_2' = \mathtt{N}'; \mu' \Vdash \Sigma'$, we conclude that thesis (1) and (2) hold.

– $\gamma^+ = \mathbf{v}?$: It follows from the LTS rule A-V that $\mathtt{M}_1 = \mathtt{N}; \mu \Vdash \Sigma$ and that $\mathtt{M}_1' = \mathtt{N}; \mu \Vdash \Sigma$. From the assumption $\mathtt{M}_1 \simeq^+ \mathtt{M}_2$ we have that $\mathtt{M}_2 = \mathtt{N}'; \mu' \Vdash \Sigma'$ as otherwise there exists a context $\mathbf{A}$ that can distinguish $\mathtt{M}_1$ and $\mathtt{M}_2$. The label $\mathbf{v}?$ can be encoded as a context $\mathbf{A} = (\_ \circ \mathbf{v})$, as contextual equivalence is closed under contexts we have that for the resulting $\mathtt{M}_2' = \mathtt{N}'; \mu' \Vdash \Sigma' \lhd \mathbf{v}$ we can conclude that $\mathtt{M}_1' \simeq \mathtt{M}_2'$.

– $\gamma^+ = \mathbf{wr}$: There are three sub cases depending on which LTS rule produces the label:
  - Wr-C: This LTS transition captures the MiniML$^+$ reduction rules A-WrD,A-WrS and A-WrC, each of these rules are actions by the attacker that result in the failure state. If $\mathtt{M}_2$ doesn't produce the failure state for these actions the states $\mathtt{M}_1$ and $\mathtt{M}_2$ can be distinguished contradicting the assumption $\mathtt{M}_1 \simeq \mathtt{M}_2$.
  - Wr-O: If $\mathtt{M}_2$ does not reduce the program to the failure state the states $\mathtt{M}_1$ and $\mathtt{M}_2$ can be distinguished contradicting the assumption $\mathtt{M}_1 \simeq \mathtt{M}_2$.
  - Wr-I: similar to Wr-O.

In each of these cases as per the LTS $\mathtt{M}_1' = \star; \emptyset \Vdash \varepsilon$ and $\mathtt{M}_2' = \star; \emptyset \Vdash \varepsilon$, we conclude that $\mathtt{M}_1' \simeq \mathtt{M}_2'$.

– $\gamma^+ = !\mathbf{n_i}^!$: It follows from the LTS rule D-N that $\mathtt{M}_1 = \mathtt{N}; \mu \Vdash \Sigma$ and that $\mathtt{M}_1' = \mathtt{N}; \mu \Vdash \Sigma \circ !l_i$. From the assumption $\mathtt{M}_1 \simeq^+ \mathtt{M}_2$ we have that

$\mathtt{M}_2 = \mathtt{N}'; \mu' \Vdash \Sigma'$ as otherwise there exists a context $\mathbf{A}$ that can distinguish $\mathtt{M}_1$ and $\mathtt{M}_2$. The label $\mathbf{v}?$ can be encoded as a context $\mathbf{A} = (\_ \circ \mathbf{n_i})$, as contextual equivalence is closed under contexts we have that for the resulting $\mathtt{M}'_2 = \mathtt{N}'; \mu' \Vdash \Sigma' \circ !l_i$ we can conclude that $\mathtt{M}'_1 \simeq \mathtt{M}'_2$.

- $\gamma^+ = \gg \mathbf{n_i^f}$: It follows from the LTS rule C-N that $\mathtt{M}_1 = \mathtt{N}; \mu \Vdash \Sigma$ and that $\mathtt{M}'_1 = \mathtt{N}; \mu \Vdash \Sigma, (\lambda(x : \tau.t)\ [\cdot])$. From the assumption $\mathtt{M}_1 \simeq^+ \mathtt{M}_2$ we have that $\mathtt{M}_2 = \mathtt{N}'; \mu' \Vdash \Sigma'$ as otherwise there exists context that can distinguish $\mathbf{A}$ that can distinguish them. It follows from the MiniML$^+$ rule A-Call that the label is always followed by an LTS transition A-V. As such we can encode the label as a context $\mathbf{A} = \mathbf{call}\ \mathbf{n_i}\ \mathbf{v}$. As contextual equivalence is closed under contexts we have that for the resulting $\mathtt{M}'_2 = \mathtt{N}'; \mu' \Vdash \Sigma', (\lambda(x : \tau.t)\ [\cdot]) \lhd \mathbf{v}$ we can conclude that $\mathtt{M}'_1 \simeq \mathtt{M}'_2$.
- $\gamma^+ = \gg \mathbf{n_i^l}$: It follows from the LTS rule S-N that $\mathtt{M}_1 = \mathtt{N}; \mu \Vdash \Sigma$ and that $\mathtt{M}'_1 = \mathtt{N}; \mu \Vdash \Sigma, (l_i := [\cdot])$. From the assumption $\mathtt{M}_1 \simeq^+ \mathtt{M}_2$ we have that $\mathtt{M}_2 = \mathtt{N}'; \mu' \Vdash \Sigma'$ as otherwise there exists context that can distinguish $\mathbf{A}$ that can distinguish them. It follows from the MiniML$^+$ rule A-Set that the label is always followed by an LTS transition A-V. As such we can encode the label a context $\mathbf{A} = \mathbf{n_i} := \mathbf{v}$. As contextual equivalence is closed under contexts we have that for the resulting $\mathtt{M}'_2 = \mathtt{N}'; \mu' \Vdash \Sigma', (l_i := [\cdot]) \lhd \mathbf{v}$ we can conclude that $\mathtt{M}'_1 \simeq \mathtt{M}'_2$.
- $\gamma^+ = \gg \mathbf{ref}^t au$: similar to $\gg \mathbf{n_i^f}$ and $\gg \mathbf{n_i^l}$.
- $\gamma^+ = \gg (\lambda \mathbf{x.t})$: It follows from the LTS rule C-L that $\mathtt{M}_1 = \mathtt{N}; \mu \Vdash \Sigma \circ \mathtt{E}[({}^{\tau_1 \to \tau_2}\mathtt{F}(\lambda \mathbf{x.t})\ v)] : \tau$ and that $\mathtt{M}'_1 = \mathtt{N}; \mu \Vdash \Sigma, \mathtt{E} : \tau_2 \to \tau \rhd v : \tau_1$. From the assumption $\mathtt{M}_1 \simeq^+ \mathtt{M}_2$ we have that
  $\mathtt{M}_2 = \mathtt{N}'; \mu' \Vdash \Sigma' \circ \mathtt{E}[({}^{\tau_1 \to \tau_2}\mathtt{F}(\lambda \mathbf{x.t})\ v')] : \tau$ where $v \simeq v'$ as otherwise the following context:

$$\mathbf{A} = (\mu \Vdash \overline{\mathbf{C}}, (((\lambda \mathbf{x.t})\ \mathbf{v}) \equiv_\alpha [\cdot]))$$

  can distinguish between $\mathtt{M}_1$ and $\mathtt{M}_2$. We conclude that:
  $\mathtt{M}'_2 = \mathtt{N}'; \mu' \Vdash \Sigma', \mathtt{E}' : \tau_2 \to \tau \rhd v : \tau_1$. and thus that the thesis $\mathtt{M}'_1 \simeq \mathtt{M}'_2$ holds.

2. As in case 1, *mutatis mutandis*.

□

Lemma 2. (Soundness) $\mathtt{M}_1 \approx^+ \mathtt{M}_2 \Rightarrow \mathtt{M}_1 \simeq^+ \mathtt{M}_2$

Proof. As per Definition 2 we have that the thesis $\mathtt{M}_1 \simeq^+ \mathtt{M}_2$ becomes:

$$\forall \mathbf{A}.\ \mathbf{A} \mid\mid \mathtt{M}_1 \Uparrow \iff \mathbf{A} \mid\mid \mathtt{M}_2 \Uparrow$$

The proof is divided into two cases, one case for each side of the co-implication.

1. $\Rightarrow$ In this case the thesis is:

$$\forall \mathbf{A}.\ \mathbf{A} \mid\mid \mathtt{M}_1 \Uparrow \Rightarrow \mathbf{A} \mid\mid \mathtt{M}_2 \Uparrow .$$

The thesis can be redefined as:

$$\forall \mathbf{A}.\ \forall k \in \mathbb{N}.\ \mathbf{A} \mid\mid \mathtt{M}_1 \twoheadrightarrow^k \mathbf{A}_1' \mid\mid \mathtt{M}_1' \Rightarrow \forall m \in \mathbb{N}.\ \mathbf{A} \mid\mid \mathtt{M}_2 \twoheadrightarrow^m \mathbf{A}_2' \mid\mid \mathtt{M}_2'$$

The proof proceeds by induction on m.
Base case: $m = 0$. Straightforward: $\mathbf{A} \mid\mid \mathtt{M}_2 \twoheadrightarrow^0 \mathbf{A} \mid\mid \mathtt{M}_2$.
Inductive case: $m = h + 1$. The thesis is:

$$\mathbf{A} \mid\mid \mathtt{M}_2 \twoheadrightarrow^{h+1} \mathbf{A}_2' \mid\mid \mathtt{M}_2'.$$

The inductive hypotheses (IH) is:

$$\forall \mathbf{A}.\ \forall k \in \mathbb{N}.\ \mathbf{A} \mid\mid \mathtt{M}_1 \twoheadrightarrow^k \mathbf{A}_1' \mid\mid \mathtt{M}_1' \Rightarrow \mathbf{A} \mid\mid \mathtt{M}_2 \twoheadrightarrow^h \mathbf{A}_2^h \mid\mid \mathtt{M}_2^h$$

We know from this IH that:

$$\exists \mathbf{A}.\mathtt{M}_1.\ \mathbf{A} \mid\mid \mathtt{M}_1 \twoheadrightarrow^h \mathbf{A}_1^h \mid\mid \mathtt{M}_1^h \twoheadrightarrow^{k-h} \mathbf{A}_1' \mid\mid \mathtt{M}_1'$$

We prove the thesis by reasoning about what the presence or absence of the last observable label γ^+ tells us about the existence of a next reduction step $h + 1$. There are two cases: either the attacker in MiniML^a is passive or executing.

(a) The attacker is passive and the program in MiniML is executing. In this case there are two sub-cases:

i. $\exists \gamma^+.\ \mathtt{M}_1^h \overset{\gamma^+}{\Longrightarrow} \mathtt{M}_1^{h'}$.

By the assumption $\mathtt{M}_1 \approx^+ \mathtt{M}_2$ we conclude that $\mathtt{M}_2^h \overset{\gamma^+}{\Longrightarrow} \mathtt{M}_2^{h'}$ and $\mathtt{M}_1^{h'} \approx^+ \mathtt{M}_2^{h'}$. This, in conjunction with the IH, implies the thesis:

$$\mathbf{A} \mid\mid \mathtt{M}_2 \twoheadrightarrow^{h+1} \mathbf{A}' \mid\mid \mathtt{M}_2'.$$

ii. $\nexists \gamma^+.\ \mathtt{M}_{\mathtt{e}1}{}^h \overset{\gamma^+}{\Longrightarrow} \mathtt{M}_1^{h'}$.

Per the definition of bisimulation we have that this is only possible if $\mathtt{M}_1^h$ diverges, more particularly the MiniML term t_1^h that it executes diverges. It follows from the assumption that $\mathtt{M}_1 \approx^+ \mathtt{M}_2$ that $\mathtt{M}_1^h \approx^+ \mathtt{M}_2^h$ and thus that: $\nexists \gamma^+.\ \mathtt{M}_{\mathtt{e}1}{}^h \overset{\gamma^+}{\Longrightarrow} \mathtt{M}_1^{h'}$.

(b) The attacker is executing and the MiniML program is waiting for input from the attacker. In this case there are two sub-cases as well:

i. $\exists \gamma^+.\ \mathtt{N} \Vdash \Sigma_1^h \overset{\gamma^+}{\Longrightarrow} \mathtt{M}_1^{h'}$.

Because the observable label γ^+ is produced by the respective attackers, we must thus show that: $\mathbf{A}_1^h = \mathbf{A}_2^h$, where the existence of $\mathbf{A}_2^h$ derives from the induction hypothesis.

We know by the assumption: $\mathtt{M}_1 \approx^+ \mathtt{M}_2$ that both attacker states were modified by the same stream of observable labels if there are any such labels : $\exists k \in \mathbb{N}.k \leq h \wedge \mathbf{A} \mid\mid \mathtt{M}_1 \twoheadrightarrow^k \mathbf{A}_1^k \mid\mid \mathtt{M}_1^k \wedge \mathbf{A}_1^k \mid\mid \mathtt{M}_1^k \twoheadrightarrow^{h-k} \mathbf{A}_1^h \mid\mid \mathtt{M}_1^h$ where $\mathtt{M}_1 \overset{\overline{\gamma^+}}{\Longrightarrow} \mathtt{M}_1^k$ and that $\exists k \in \mathbb{N}.k \leq h \wedge \mathbf{A} \mid\mid \mathtt{M}_2 \twoheadrightarrow^k \mathbf{A}_2^k \mid\mid \mathtt{M}_2^k \wedge \mathbf{A}_2^k \mid\mid \mathtt{M}_2^k \twoheadrightarrow^{h-k} \mathbf{A}_2^h \mid\mid \mathtt{M}_2^h$ where $\mathtt{M}_2 \overset{\overline{\gamma^+}}{\Longrightarrow} \mathtt{M}_2^k$.

Combining the fact that the reduction rules of the MiniML$^+$-calculus are deterministic and with the fact that the MiniML^a contexts are updated in the same way by identical labels γ^+ we conclude that $\mathbf{A}_1^h = \mathbf{A}_2^h$ and that $\mathtt{N} \Vdash \Sigma_2^h \overset{\gamma^+}{\Longrightarrow} \mathtt{M}_2^{h'}$. This implies the thesis.

ii. $\nexists \gamma^+.\ \mathtt{N} \Vdash \Sigma_1^h \overset{\gamma^+}{\Longrightarrow} \mathtt{M}_1^{h'}$.
If there exists no label γ^+ the attacker is diverging. In the previous case we established that $\mathbf{A}_1^h = \mathbf{A}_2^h$. As such both $\mathbf{A} \mid\mid \mathtt{M}_1$ and $\mathbf{A} \mid\mid \mathtt{M}_2$ diverge, which implies the thesis.

2. $\Leftarrow$ As in case 1, *mutatis mutandis*.

4.3 The FFI Preserves and Reflects the Equivalences of MiniML

To prove that the FFI is secure we prove that injecting a MiniML term t into a secure state of MiniML$^+$ as follows:

$$\{\mu \mid t\}^{\uparrow} \overset{\text{def}}{=} \star; \mu \Vdash \boldsymbol{\varepsilon} \circ t \textit{ where } \Gamma \vdash t : \tau$$

will preserve the abstraction t irrespective of which attacker $\mathbf{A}$ it faces. Formally stated: bisimilar terms in MiniML remain bisimilar when injected to MiniML$^+$.

Theorem 7 (A Secure FFI). $\mu_1 \mid t_1 \approx \mu_2 \mid t_2 \iff \{\mu_1 \mid t_1\}^{\uparrow} \approx^+ \{\mu_2 \mid t_2\}^{\uparrow}$.

The proof splits the thesis into two sublemma: preservation and reflection.

Lemma 3. (Preservation) $t_1 \approx t_2 \Rightarrow t_1^{\uparrow} \approx^+ t_2^{\uparrow}$.

Proof. We must develop a relation $\mathcal{R}$ such that:

$$\mathtt{t_1}^{\uparrow}\ \mathcal{R}\ \mathtt{t_2}^{\uparrow} \tag{1}$$

and that for all $\mathtt{M}_1\ \mathcal{R}\ \mathtt{M}_2$ we have that:

$$\mathtt{M}_1 \overset{\gamma^+}{\Longrightarrow} \mathtt{M}_1' \textit{ and } \exists \mathtt{M}_2'.\mathtt{M}_2 \overset{\gamma^+}{\Longrightarrow} \mathtt{M}_2' \ \Rightarrow\ \mathtt{M}_1'\ \mathcal{R}\ \mathtt{M}_2' \tag{2}$$

$$\mathtt{M}_2 \overset{\gamma^+}{\Longrightarrow} \mathtt{M}_2' \textit{ and } \exists \mathtt{M}_1'.\mathtt{M}_1 \overset{\gamma^+}{\Longrightarrow} \mathtt{M}_1' \ \Rightarrow\ \mathtt{M}_1'\ \mathcal{R}\ \mathtt{M}_2' \tag{3}$$

We define $\mathcal{R}$ as $\mathcal{R} = \mathcal{R}_0 \cup \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3$:

$$\begin{aligned}
\mathcal{R}_0 =& \{(\mathtt{N}_1; \mu_1 \Vdash \Sigma_1, \mathtt{N}_2; \mu_2 \Vdash \Sigma_2) \mid \forall \mathtt{N}_1, \mathtt{N}_2, \Sigma_1, \Sigma_2 \textit{ such that} \\
& \quad \mathtt{N}_1 \approx_N \mathtt{N}_2 \textit{ and } Dom(\mu_1) = Dom(\mu_2) \textit{ and } \Sigma_1 \approx_\Sigma \Sigma_2\} \\
\mathcal{R}_1 =& \{(\mathtt{N}_1; \mu_1 \Vdash \Sigma_1 \circ t_1 : \tau_1, \mathtt{N}_2; \mu_2 \Vdash \Sigma_2 \circ t_2 : \tau_2) \mid \forall \mathtt{N}_1, \mathtt{N}_2, \Sigma_1, \Sigma_2, t_1, t_2 \\
& \quad \textit{such that } (\mathtt{N}_1; \mu_1 \Vdash \Sigma_1, \mathtt{N}_2; \mu_2 \Vdash \Sigma_2) \in \mathcal{R}_0 \textit{ and } t_1 \approx t_2 \textit{ and } \tau_1 = \tau_2\} \\
\mathcal{R}_2 =& \{(\mathtt{N}_1; \mu_1 \Vdash \Sigma_1 \rhd m_1 : \tau_1, \mathtt{N}_2; \mu_2 \Vdash \Sigma_2 \circ \ \rhd m_2 : \tau_2) \mid \forall \mathtt{N}_1, \mathtt{N}_2, \Sigma_1, \Sigma_2, m_1, m_2 \\
& \quad \textit{such that } (\mathtt{N}_1; \mu_1 \Vdash \Sigma_1, \mathtt{N}_2; \mu_2 \Vdash \Sigma_2) \in \mathcal{R}_0 \textit{ and } m_1 \approx_m m_2 \textit{ and } \tau_1 = \tau_2\} \\
\mathcal{R}_3 =& \{(\mathtt{N}_1; \mu_1 \Vdash \Sigma_1 \lhd m_1 : \tau_1, \mathtt{N}_2; \mu_2 \Vdash \Sigma_2 \circ \ \lhd m_2 : \tau_2) \mid \forall \mathtt{N}_1, \mathtt{N}_2, \Sigma_1, \Sigma_2, m_1, m_2 \\
& \quad \textit{such that } (\mathtt{N}_1; \mu_1 \Vdash \Sigma_1, \mathtt{N}_2; \mu_2 \Vdash \Sigma_2) \in \mathcal{R}_0 \textit{ and } m_1 \approx_m m_2 \textit{ and } \tau_1 = \tau_2\}
\end{aligned}$$

where $\approx_N$ is defined as:

$$\mathtt{N}_1 \approx_N \mathtt{N}_2 \iff dom(\mathtt{N}_1) = dom(\mathtt{N}_2) \textit{ and } \forall \mathbf{n_i} \in dom(\mathtt{N}_1).\ \mathtt{N}_1(\mathbf{n_i}) \approx \mathtt{N}_2(\mathbf{n_i})$$

where $\approx_\Sigma$ is defined as:

$$\begin{gathered}\Sigma_1 \approx_\Sigma \Sigma_2 \iff \\ \textit{For } (\mathtt{E}^1_1, ..., \mathtt{E}^1_\mathtt{n}) : (\tau^1_1, ..., \tau^1_\mathtt{n}) \textit{ in } \Sigma_1 \textit{ and} \\ (\mathtt{E}^2_1, ..., \mathtt{E}^2_{\mathtt{n}'}) : (\tau^2_1, ..., \tau^2_\mathtt{n}) \textit{ in } \Sigma_2 \\ \textit{we have } \mathtt{n} = \mathtt{n}' \textit{ and for every } 1 \leq \mathtt{i} \leq \mathtt{n} : \tau^1_\mathtt{i} = \tau^2_\mathtt{i} \textit{ and} \\ \forall \mathtt{t}', \mathtt{t}''.\Gamma \vdash \mathtt{t}' : \tau^1_\mathtt{i} \wedge \Gamma \vdash \mathtt{t}'' : \tau^1_\mathtt{i} \wedge \mathtt{t}' \approx \mathtt{t}'' \;\Rightarrow\; \mathtt{E}^1_\mathtt{i}[\mathtt{t}'] \approx \mathtt{E}^2_\mathtt{i}[\mathtt{t}'']\end{gathered}$$

and where $\approx_m$ is defined as:

$$\begin{gathered}m_1 \approx_m m_2 \iff \\ (m_1 = t_1 \wedge m_2 = t_2 \wedge t_1 \approx t_2) \\ \vee (m_1 = \mathbf{t_1} \wedge m_2 = \mathbf{t_2} \wedge \mathbf{t_1} \equiv_\alpha \mathbf{t_2}) \\ \vee (m_1 = \langle m_i^{i \in 1..n} \rangle \wedge m_2 = \langle m_i'^{i' \in 1..n} \rangle \wedge \forall i \in 1..n. m_i \approx_m m_i')\end{gathered}$$

We now proof the three cases.

- In case (1) we have that $\star; \emptyset \Vdash \varepsilon; \emptyset \circ t_1 \;\mathcal{R}\; \star; \emptyset \Vdash \varepsilon \circ t_2$ as we have that $t_1 \approx t_2$ from the assumption.
- In case (2) we assume $\mathtt{M}_1 \mathcal{R} \mathtt{M}_2$ and that $\mathtt{M}_1 \overset{\gamma^+}{\Longrightarrow} \mathtt{M}_1$. We proceed by case analysis on γ^+.
  - $\gamma^+ = \mathbf{v}!$: By the LTS we have that:

    $$(\mathtt{M}_1 =)\mathtt{N}_1; \mu_1 \Vdash \Sigma_1 \rhd \mathbf{v} : \tau \overset{\mathbf{v}!}{\Longrightarrow} (\mathtt{M}_1{}' =)\mathtt{N}_1; \mu_1 \Vdash \Sigma_1$$

    It follows from $\mathtt{M}_1 \mathcal{R} \mathtt{M}_2$ more specifically $\mathtt{M}_1 \mathcal{R}_2 \mathtt{M}_2$ that:

    $$\mathtt{M}_2 = \mathtt{N}_2; \mu_2 \Vdash \Sigma_2 \rhd \mathbf{v} : \tau$$

    By the LTS we have that:

    $$\mathtt{M}_2 \overset{\mathbf{v}!}{\Longrightarrow} (\mathtt{M}_2{}' =)\mathtt{N}_2; \mu_2 \Vdash \Sigma_2$$

    It follows from $\mathtt{M}_1 \mathcal{R}_0 \mathtt{M}_2$ that $\mathtt{N}_1 \approx_N \mathtt{N}_2$, $Dom(\mu_1) = Dom(\mu_2)$ and $\Sigma_1 \approx_\Sigma \Sigma_2$ and thus that $\mathtt{M}_1' \mathcal{R} \mathtt{M}_2'$.
  - $\gamma^+ = \mathbf{v}?$: By the LTS we have that:

    $$(\mathtt{M}_1 =)\mathtt{N}_1; \mu_1 \Vdash \Sigma_1, \mathtt{E}_1 : \tau_1 \to \tau_1' \overset{\mathbf{v}?}{\Longrightarrow} (\mathtt{M}_1{}' =)\mathtt{N}_1; \mu_1 \Vdash \Sigma_1, \mathtt{E}_1 : \tau_1 \to \tau_1' \lhd \mathbf{v} : \tau_1$$

    It follows from $\mathtt{M}_1 \mathcal{R} \mathtt{M}_2$ more specifically $\mathtt{M}_1 \mathcal{R}_0 \mathtt{M}_2$ that:

    $$\mathtt{M}_2 = \mathtt{N}_2; \mu_2 \Vdash \Sigma_2, \mathtt{E}_2 : \tau_2 \to \tau_2'$$

    Where $\mathtt{N}_1 \approx_N \mathtt{N}_2$, $Dom(\mu_1) = Dom(\mu_2)$, $\Sigma_1, \mathtt{E}_1 : \tau_1 \to \tau_1' \approx_\Sigma \Sigma_2, \mathtt{E}_2; \tau_2 \to \tau_2'$ and thus that $\tau_1 = \tau_2$ and $\tau_1' = \tau_2'$. By the LTS we have that:

    $$\mathtt{M}_2 \overset{\mathbf{v}?}{\Longrightarrow} (\mathtt{M}_2{}' =)\mathtt{N}_2; \mu_2 \Vdash \Sigma_2, \mathtt{E} : \tau_2 \to \tau_2' \lhd \mathbf{v} : \tau_2$$

    We conclude by $\mathtt{M}_1' \mathcal{R}_3 \mathtt{M}_2'$ that $\mathtt{M}_1' \mathcal{R} \mathtt{M}_2'$.

- $\gamma^+ = \surd$: By the LTS we have three cases:

$$(\mathtt{M}_1 =)\star; \mu_1 \Vdash \varepsilon \overset{\surd}{\Longrightarrow} (\mathtt{M}_1' =)\star; \emptyset \Vdash \varepsilon$$

It follows from $\mathtt{M}_1 \mathcal{R} \mathtt{M}_2$ more specifically $\mathtt{M}_1 \mathcal{R}_0 \mathtt{M}_2$ that:

$$\mathtt{M}_2 = \star; \mu_2 \Vdash \Sigma_2$$

By the LTS we have that:

$$\mathtt{M}_2 \overset{\surd}{\Rightarrow} \star; \emptyset \Vdash \varepsilon$$

Given that $\star \approx_N \star$, $Dom(\emptyset) = Dom(\emptyset)$ and $\varepsilon \approx_\Sigma \varepsilon$ we conclude that $\mathtt{M}_1' \mathcal{R} \mathtt{M}_2'$.

- $\gamma^+ = \mathbf{wr}$: By the LTS we have three cases:
  1. $(\mathtt{M}_1 =)\mathtt{N}_1; \mu_1 \Vdash \Sigma_1 \overset{\mathbf{wr}}{\Longrightarrow} (\mathtt{M}_1' =)\star; \emptyset \Vdash \varepsilon$ It follows from $\mathtt{M}_1 \mathcal{R} \mathtt{M}_2$ more specifically $\mathtt{M}_1 \mathcal{R}_0 \mathtt{M}_2$ that:

     $$\mathtt{M}_2 = \mathtt{N}_2; \mu_2 \Vdash \Sigma_2$$

     Where $\mathtt{N}_1 \approx_N \mathtt{N}_2$, $Dom(\mu_1) = Dom(\mu_2)$ and $\Sigma_1 \approx_\Sigma \Sigma_2$. The LTS rule refrul:Wr-C corresponds to the MiniML$^+$ reduction rules A-WrD, A-WrC and A-WrS. In each of these reduction rules the error state is produced by type check fails. It follows from $\mathtt{N}_1 \approx_N \mathtt{N}_2$ that the names of $\mathtt{M}_1$ and $\mathtt{M}_2$ share the same types and thus fails in the same situations.

     $$\mathtt{M}_2 \overset{\mathbf{wr}}{\Longrightarrow} \star; \emptyset \Vdash \varepsilon$$

     Given that $\star \approx_N \star$, $Dom(\emptyset) = Dom(\emptyset)$ and $\varepsilon \approx_\Sigma \varepsilon$ we conclude that $\mathtt{M}_1' \mathcal{R} \mathtt{M}_2'$.
  2. $(\mathtt{M}_1 =)\mathtt{N}_1; \mu_1 \Vdash \Sigma_1 \lhd \mathbf{wr} : \tau \overset{\mathbf{wr}}{\Longrightarrow} (\mathtt{M}_1' =)\star; \emptyset \Vdash \varepsilon$: It follows from $\mathtt{M}_1 \mathcal{R} \mathtt{M}_2$ more specifically $\mathtt{M}_1 \mathcal{R}_3 \mathtt{M}_2$ that:

     $$\mathtt{M}_2 = \mathtt{N}_2; \mu_2 \Vdash \Sigma_2 \lhd \mathbf{wr} : \tau$$

     It follows from the LTS that:

     $$\mathtt{M}_2 \overset{\mathbf{wr}}{\Longrightarrow} \star; \emptyset \Vdash \varepsilon$$

     Given that $\star \approx_N \star$, $Dom(\emptyset) = Dom(\emptyset)$ and $\varepsilon \approx_\Sigma \varepsilon$ we conclude that $\mathtt{M}_1' \mathcal{R} \mathtt{M}_2'$.
  3. $(\mathtt{M}_1 =)\mathtt{N}_1; \mu_1 \Vdash \Sigma_1 \rhd \mathbf{wr} \overset{\mathbf{wr}}{\Longrightarrow} (\mathtt{M}_1' =)\star; \emptyset \Vdash \varepsilon$: Analogous to the previous case.

- $\gamma^+ = !\mathbf{n_i}$: By the LTS we have that:

$$(\mathtt{M}_1 =)\mathtt{N}_1; \mu_1 \Vdash \Sigma_1 \xRightarrow{!\mathbf{n_i}} (\mathtt{M}'_1 =)\mathtt{N}_1; \mu_1 \Vdash \Sigma_1 \circ !l_i : \tau_1$$

Where $\mathtt{N}(\mathbf{n_i}) = (l_i, \mathtt{Ref}\ \tau)$. It follows from $\mathtt{M}_1 \mathcal{R} \mathtt{M}_2$ more specifically $\mathtt{M}_1 \mathcal{R}_0 \mathtt{M}_2$ that:

$$\mathtt{M}_2 = \mathtt{N}_2; \mu_2 \Vdash \Sigma_2$$

Where $\mathtt{N}_1 \approx_N \mathtt{N}_2$, $Dom(\mu_1) = Dom(\mu_2)$ and $\Sigma_1 \approx_\Sigma \Sigma_2$. By the LTS we thus have that:

$$\mathtt{M}_2 \xRightarrow{!\mathbf{n_i}} (\mathtt{M}'_2 =)\mathtt{N}_2; \mu_1 \Vdash \Sigma_2 \circ !l_i : \tau_2$$

It follows from $\mathtt{N}_1 \approx_N \mathtt{N}_2$ that $\mathtt{N}_1(\mathbf{n_i}) \approx \mathtt{N}_2(\mathbf{n_i})$ from which we conclude that $!l_i \approx !l_i$ and that $\tau_1 = \tau_2$. We conclude that $\mathtt{M}'_1 \mathcal{R} \mathtt{M}'_2$.

- $\gamma^+ = \gg ref^\tau$: By the LTS we have that:

$$(\mathtt{M}_1 =)\mathtt{N}_1; \mu_1 \Vdash \Sigma_1 \xRightarrow{\gg \mathbf{ref}^\tau} (\mathtt{M}'_1 =)\mathtt{N}_1; \mu_1 \Vdash \Sigma_1, (\mathtt{ref}\ [\cdot]) : \tau \to \mathtt{Ref}\ \tau$$

It follows from $\mathtt{M}_1 \mathcal{R} \mathtt{M}_2$ more specifically $\mathtt{M}_1 \mathcal{R}_0 \mathtt{M}_2$ that:

$$\mathtt{M}_2 = \mathtt{N}_2; \mu_2 \Vdash \Sigma_2$$

Where $\mathtt{N}_1 \approx_N \mathtt{N}_2$, $Dom(\mu_1) = Dom(\mu_2)$ and $\Sigma_1 \approx_\Sigma \Sigma_2$. By the LTS we thus have that:

$$\mathtt{M}_2 \xRightarrow{\gg \mathbf{ref}^\tau} (\mathtt{M}'_2 =)\mathtt{N}_2; \mu_2 \Vdash \Sigma_2 (\mathtt{ref}\ [\cdot]) : \tau \to \mathtt{Ref}\ \tau$$

By $(\mathtt{ref}\ [\cdot]) : \tau \to \mathtt{Ref}\ \tau \approx_\Sigma (\mathtt{ref}\ [\cdot]) : \tau \to \mathtt{Ref}\ \tau$ we conclude that $\mathtt{M}'_1 \mathcal{R}_0 \mathtt{M}'_2$ and thus that the thesis: $\mathtt{M}'_1 \mathcal{R} \mathtt{M}'_2$ holds.

- $\gamma^+ = \gg \mathbf{n_i^f}$: Similar to the $\gg ref^\tau$ case.
- $\gamma^+ = \gg \mathbf{n_i^l}$: Similar to the $\gg ref^\tau$ case.
- $\gamma^+ = \gg (\lambda \mathbf{x.t})$: By the LTS we have that:

$$\begin{aligned}&(\mathtt{M}_1 =)\mathtt{N}_1; \mu_1 \Vdash \Sigma_1 \circ \mathtt{E}_1[({}^{\tau_1 \to \tau_2}\mathtt{F}(\lambda \mathbf{x.t})\ v)] : \tau \xRightarrow{\gg (\lambda \mathbf{x.t})} \\ &(\mathtt{M}'_1 =)\mathtt{N}_1; \mu_1 \Vdash \Sigma_1, \mathtt{E}_1 : \tau_2 \to \tau \rhd v : \tau_1\end{aligned}$$

It follows from $\mathtt{M}_1 \mathcal{R} \mathtt{M}_2$ more specifically $\mathtt{M}_1 \mathcal{R}_1 \mathtt{M}_2$ that:

$$\mathtt{M}_2 = \mathtt{N}_2; \mu_2 \Vdash \Sigma_2 \circ t_2$$

Where $\mathtt{N}_1 \approx_N \mathtt{N}_2$, $Dom(\mu_1) = Dom(\mu_2)$, $\Sigma_1 \approx_\Sigma \Sigma_2$ and $t_1 \approx t_2$. It follows from the definition of $\{\mu \mid t\}^\uparrow$ that $({}^{\tau_1 \to \tau_2}\mathtt{F}(\lambda \mathbf{x.t})\ v)$ derives from an input by the attacker. From the assumption $\mathtt{M}_1 \mathcal{R} \mathtt{M}_2$ it follows that $\mathtt{M}_1$ and $\mathtt{M}_2$ received the same inputs from the attacker and we thus have that like $t_1 = \mathtt{E}_1[({}^{\tau_1 \to \tau_2}\mathtt{F}(\lambda \mathbf{x.t})\ v)]$, t_2 was produced by an input $(\lambda \mathbf{x.t})$ from the

attacker. We can thus rewrite t_1 as $((\lambda x : \tau_1 \to \tau_2.t_1')\ ^{\tau_1 \to \tau_2}\mathtt{F}(\lambda \mathbf{x.t}))$ and t_2 as $((\lambda x : \tau_1 \to \tau_2.t_2')\ ^{\tau_1 \to \tau_2}\mathtt{F}(\lambda \mathbf{x.t}))$. Where $(\lambda x : \tau_1 \to \tau_2.t_1') \approx (\lambda x : \tau_1 \to \tau_2.t_2')$. We must now show that both λ-terms call their arguments an equal number of times with contextually equivalent arguments. From Theorem 3 we have that: $(\lambda x : \tau_1 \to \tau_2.t_1') \approx (\lambda x : \tau_1 \to \tau_2.t_2') \iff (\lambda x : \tau_1 \to \tau_2.t_1') \simeq (\lambda x : \tau_1 \to \tau_2.t_2')$. It follows from the inclusion of references in MiniML that $(\lambda x : \tau_1 \to \tau_2.t_1')$ and $(\lambda x : \tau_1 \to \tau_2.t_2')$ as otherwise an argument of the form $(\lambda x : \tau.l_i := (+!l_i 1); x)$ can be used to distinguish both terms. We thus conclude that:

$$\mathtt{M}_2 \xRightarrow{\gg (\lambda \mathbf{x.t})} (\mathtt{M}_2' =)\mathtt{N}_2; \mu_2 \Vdash \Sigma_2, \mathtt{E}_2 : \tau_2 \to \tau \rhd v : \tau_1$$

and that $\mathtt{M}_1' \mathcal{R}_2 \mathtt{M}_2'$ and thus that the thesis: $\mathtt{M}_1' \mathcal{R} \mathtt{M}_2'$ holds.

– For case (3) we have that: mutatis mutandis.

□

Lemma 4. (Reflection) $t_1^{\uparrow} \approx^{+} t_2^{\uparrow} \Rightarrow t_1 \approx t_2$.

Proof. We prove the lemma by the contrapositive, the lemma is restated as:

$$t_1 \not\approx t_2 \Rightarrow \{t_1\}^{\uparrow} \not\approx^{+} \{t_2\}^{\uparrow}$$

The proof has two cases. In the first case the bisimulation fails immediately as t_1 and t_2 produce differently labelled transitions after silently reduction.

1. $\emptyset; \emptyset \mid t_1 \overset{\gamma}{\Rightarrow} \zeta_1' \wedge \nexists \zeta_2'.\emptyset; \emptyset \mid t_2 \overset{\gamma}{\Rightarrow} \zeta_2'$.
   We proceed by case analysis over the label γ:
   – `true`: We have that: $\emptyset; \emptyset \mid t_1 \to^{*} \emptyset; \mu_1 \mid \mathtt{true} \overset{\mathtt{true}}{\Longrightarrow} \emptyset; \mu_1 \mid t_1'$. By the assumption, $t_1 \not\approx \mathtt{t}_2$ and the LTS we have that: $\emptyset; \emptyset \mid t_2 \to^{*} \emptyset; \mu_2 \mid v_2$ where $v_2 \neq \mathtt{true}$. By the reduction rules (Internal MiniML) and (Setup) we have that:

   $$\forall \mathbf{A}.\mathbf{A} \mid\mid \{t_1\}^{\uparrow} \twoheadrightarrow^{*} \mathbf{A} \mid\mid \star; \mu_1 \Vdash \varepsilon \rhd \mathtt{true} : \mathtt{Bool}$$

   and that

   $$\forall \mathbf{A}.\mathbf{A} \mid\mid \{t_2\}^{\uparrow} \twoheadrightarrow^{*} \mathbf{A} \mid\mid \star; \mu_2 \Vdash \varepsilon \rhd v_2 : \tau_2$$

   Where $v_2 \neq \mathtt{true}$. We now conclude from the marshalling rules and the LTS rule (M-V) that $\{t_1\}^{\uparrow} \overset{\mathtt{true}}{\Longrightarrow} \mathtt{M}_1'$ and that $\{t_2\}^{\uparrow} \overset{\mathtt{true}}{\not\Longrightarrow} \mathtt{M}_2'$
   – `false`: Analogous to the `true` case.
   – `unit`: Analogous to the `true` case.
   – $\overline{n}$: Analogous to the `true` case.
   – l_i: Analogous to the `true` case.

- $.i$: We have that: $\emptyset;\emptyset \mid t_1 \rightarrow^* \emptyset;\mu_1 \mid \langle v_i^{i\in 1..n}\rangle \overset{.i}{\Rightarrow} \emptyset;\mu_1 \mid v_i$. By the assumption, $t_1 \not\approx \mathtt{t}_2$ and the LTS we have that: $\emptyset;\emptyset \mid t_2 \rightarrow^* \emptyset;\mu_2 \mid v_2$ where $v_2 \neq \langle v_i^{i\in 1..n}\rangle$. By the reduction rules (Internal MiniML) and (Setup) we have that:

$$\forall \mathbf{A}.\mathbf{A} \mid\mid \{t_1\}^{\uparrow} \twoheadrightarrow^* \mathbf{A} \mid\mid \star;\mu_1 \Vdash \varepsilon \rhd \langle v_i^{i\in 1..n}\rangle : \langle \tau_i^{i\in 1..n}\rangle$$

and that

$$\forall \mathbf{A}.\mathbf{A} \mid\mid \{t_2\}^{\uparrow} \twoheadrightarrow^* \mathbf{A} \mid\mid \star;\mu_2 \Vdash \varepsilon \rhd v_2 : \tau_2$$

Where $v_2 \neq \langle v_i^{i\in 1..n}\rangle$. We now conclude from the marshalling rules and the LTS rule (M-V) that $\{t_1\}^{\uparrow} \xRightarrow{\langle \mathbf{v_i^{i\in 1..n}}\rangle} \mathtt{M}_1'$ and that $\{t_2\}^{\uparrow} \not\xRightarrow{\langle \mathbf{v_i^{i\in 1..n}}\rangle} \mathtt{M}_2'$

- $@v$: We have that: $\emptyset;\emptyset \mid t_1 \rightarrow^* \emptyset;\mu_1 \mid (\lambda x : \tau.t_1') \overset{@v}{\Longrightarrow} \emptyset;\mu_1 \mid ((\lambda x : \tau.t_1')\ v)$. By the assumption, $t_1 \not\approx \mathtt{t}_2$ and the LTS we have that: $\emptyset;\emptyset \mid t_2 \rightarrow^* \emptyset;\mu_2 \mid v_2$ where $v_2 \neq (\lambda x : \tau'.t_2')$ or $v_2 = (\lambda x : \tau'.t_2')$ where $\Gamma \vdash (\lambda x : \tau.t_1') : \tau_1 \wedge \Gamma \vdash (\lambda x : \tau.t_2') : \tau_2 \wedge \tau_1 \neq \tau_2$. By the reduction rules (Internal MiniML) and (Setup) we have that:

$$\forall \mathbf{A}.\mathbf{A} \mid\mid \{t_1\}^{\uparrow} \twoheadrightarrow^* \mathbf{A} \mid\mid \star;\mu_1 \Vdash \varepsilon \rhd (\lambda x : \tau.t_1') : \tau_1$$

and that

$$\forall \mathbf{A}.\mathbf{A} \mid\mid \{t_2\}^{\uparrow} \twoheadrightarrow^* \mathbf{A} \mid\mid \star;\mu_2 \Vdash \varepsilon \rhd v_2 : \tau_2$$

Where $\tau_1 \neq \tau_2$. We now conclude from the marshalling rules and the LTS rule (M-V) that $\{t_1\}^{\uparrow} \overset{\mathbf{n_i^f}}{\Longrightarrow} \mathtt{M}_1'$ and that $\{t_2\}^{\uparrow} \not\overset{\mathbf{n_i^f}}{\Longrightarrow} \mathtt{M}_2'$

- `ref` v: Given that the that the store μ never runs out of space, we have that the label applies to any two terms t_1 and t_2, contradicting the assumption.

- $!l_i$: By the rules of the LTS we have that $\emptyset;\emptyset \mid t_1 \not\overset{l_i}{\Rightarrow} \zeta_1'$ as $!l_i$ is only applicable if $K_1 \neq \emptyset$. This case thus contradicts the assumption.

- $l_i := v$: Similar as the $!l_i$ case.

2. $\emptyset;\emptyset \mid t_2 \overset{\gamma}{\Rightarrow} \zeta_2' \wedge \nexists \zeta_1'.\ \emptyset;\emptyset \mid t_1 \overset{\gamma}{\Rightarrow} \zeta_1'$: Similar to case (1).

In the second case there is a sequence of MiniML context actions that result in two states where different LTS transitions apply. In this case we establish the thesis by showing that each MiniML context action can be replicated by an MiniML$^+$ attacker action. We proceed by case analysis over these actions of the MiniML context:

- $\gamma = @v$: By the LTS over $\approx$ we have that:

$$K;\mu \mid (\lambda x : \tau.t) \xrightarrow{@v} K;\mu \mid ((\lambda x : \tau.t)\ v) \quad \textit{where} \ \vdash v : \tau$$

The LTS over $\approx^+$ can replicate this action as follows:

$$\begin{aligned}
&\mathtt{N}; \mu \Vdash \Sigma \circ \ : (\lambda x : \tau.t)\tau : \tau_1 \to \tau_2 \xrightarrow{\tau}{}^{*} \\
&\quad \mathtt{N}, \mathbf{n_i^f} \mapsto ((\lambda x : \tau.t), \tau_1 \to \tau_2); \mu \Vdash \Sigma \rhd \mathbf{n_i^f} : \tau_1 \to \tau_2 \\
&\mathtt{N}, \mathbf{n_i^f} \mapsto ((\lambda x : \tau.t), \tau_1 \to \tau_2); \mu \Vdash \Sigma \rhd \mathbf{n_i^f} : \tau_1 \to \tau_2 \xrightarrow{\mathbf{n_i^f}!} \\
&\quad \mathtt{N}, \mathbf{n_i^f} \mapsto ((\lambda x : \tau.t), \tau_1 \to \tau_2); \mu \Vdash \Sigma \\
&\mathtt{N}, \mathbf{n_i^f} \mapsto ((\lambda x : \tau.t), \tau_1 \to \tau_2); \mu \Vdash \Sigma \xrightarrow{\gg \mathbf{n_i^f}} \\
&\quad \mathtt{N}, \mathbf{n_i^f} \mapsto ((\lambda x : \tau.t), \tau_1 \to \tau_2); \mu \Vdash \Sigma, (t\ [\cdot]) : \tau_1 \to \tau_2 \\
&\mathtt{N}, \mathbf{n_i^f} \mapsto ((\lambda x : \tau.t), \tau_1 \to \tau_2); \mu \Vdash \Sigma, (t\ [\cdot]) : \tau_1 \to \tau_2 \xrightarrow{\mathbf{v}?} \\
&\quad \mathtt{N}, \mathbf{n_i^f} \mapsto ((\lambda x : \tau.t), \tau_1 \to \tau_2) \Vdash \Sigma, (t\ [\cdot]) \ : \tau_1 \to \tau_2 \lhd \mathbf{v} : \tau_1 \\
&\mathtt{N}, \mathbf{n_i^f} \mapsto ((\lambda x : \tau.t), \tau_1 \to \tau_2) \Vdash \Sigma, (t\ [\cdot]) : \tau_1 \to \tau_2 \lhd v : \tau_1 \xrightarrow{\tau} \\
&\quad \mathtt{N}, \mathbf{n_i^f} \mapsto ((\lambda x : \tau.t), \tau_1 \to \tau_2) \Vdash \Sigma, (t\ v) : \tau_2
\end{aligned}$$

– $\gamma = !l_i$: By the LTS over $\approx$ we have that:

$$K; \mu \mid v \xrightarrow{!l_i} K; \mu \mid !l \quad \textit{where } l_i \in K$$

Every time a location is shared in MiniML the LTS over $\approx$ adds it to the context knowledge base K. Similarly every shared location in MiniML$^+$ is added to $\mathtt{N}$. It follows from the fact hat MiniML$^+$ preserves the semantics of MiniML that every location shared by the MiniML term will be shared and stored to $\mathtt{N}$ for $\{t\}^{\uparrow}$. The LTS over $\approx^+$ can thus replicate this action as follows:

$$\mathtt{N}, \mathbf{n_i^l} \mapsto (l_i, \mathtt{Ref}\ \tau); \mu \Vdash \Sigma \xrightarrow{!\mathbf{n_i^l}} \mathtt{N}; \mu \Vdash \Sigma \circ !l_i : \tau$$

– $\gamma = \mathtt{ref}\ v$: By the LTS over $\approx$ we have that:

$$K; \mu \mid v \xrightarrow{\mathtt{ref}\ v'} K; \mu \mid \mathtt{ref}\ v'$$

The LTS over $\approx^+$ can replicate this action as follows:

$$\begin{aligned}
&\mathtt{N}; \mu \Vdash \Sigma \xrightarrow{\gg \mathbf{ref}^{\tau}} \mathtt{N}; \mu \Vdash \Sigma, (\mathtt{ref}\ [\cdot]) : \tau \to \mathtt{Ref}\ \tau \\
&\mathtt{N}; \mu \Vdash \Sigma, (\mathtt{ref}\ [\cdot]) : \tau \to \mathtt{Ref}\ \tau \xrightarrow{\mathbf{v}?} \mathtt{N}; \mu \Vdash \Sigma, (\mathtt{ref}\ v) : \tau \to \mathtt{Ref}\ \tau
\end{aligned}$$

Where $\Gamma \vdash v : \tau$.

– $\gamma = l_i := v$: Similar to the $!l_i$ case.

□

5 Related Work

This paper further extended and refined a secure interoperation semantics for the λ-calculus introduced in previous work [10] with references, divergence, accurate marshalling and stream-lined bisimulations. Formalisations that capture foreign function interface implementations have been developped before. Matthews' and Findler's multi-language semantics [11] provide a technique for specifying operational semantics that allows two languages to interoperate in a way that preserves termination and type safety. In their work however, they aim to abstract away low-level details and instead focus on semantic properties. Our formalism in contrast, focusses on lifting low-level interoperation details into the formalism to study the effects on security. Furr and Foster investigate a sound FFI between OCaml and C, by developing a multi-language type system that embeds OCaml types in C and vice-versa [4]. They, however, assume that the C code is not an attacker and will thus not circumvent their typing system. Tan et al. proprose a framework that adds type safety to the default Java FFI [14]. Their system however, requires both static and dynamic checks on the C code. Our formalism in contrast, details an FFI that does not enforce any static checks. The marshalling rules are reminiscent of Finne et. al's H/Direct FFI [3]. Our marshalling rules, however, mask security relevant values.

The notions of applicative bisimulations for MiniML and MiniML$^+$ are based on the applicative bisimulation for the νref-calculus by Jeffrey and Rathke [8], the fully abstract trace semantics for the $\lambda_\mu$hashref-calculus by Jagadeesan [7] and the trace semantics for general references by Laird [9]. The labels of our bisimulation differ from the labels used in the latter as they do not explicitly stated the shared location store in the trace labels. The proof of congruence for the bisimulations over MiniML$^+$ relies on Gordon's proof of congruence for FPC [5]. That proof is itself an adaption of Howe's proof method for bisimulation [6]. A possible alternative to the applicative bisimulation for MiniML$^+$ are the environmental bisimulations of Sangiorgi et al. [13]. Our definition of bisimulation is however much simpler than their respective definitions, as the names used in the formalism of MiniML$^+$ are not local but global and enumerable.

6 Conclusions

This paper introduced a formal model for a foreign function interface between a light-weight programming language like ML and a low-level attacker. The foreign function interface is secure in that it preserves the abstractions of MiniML in its interactions with the low-level attacker. This security property was proven by establishing that contextually equivalent terms in MiniML remain contextually equivalent when interoperating with the low-level attacker through the FFI.

There are several directions for future work. One is to investigate a source language with more advanced typing techniques. Certain typing techniques, however, do complicate the proof technique. Polymorphic types, for example, cannot be captured through an applicative bisimulation. Another possibility is to investigate concurrent source languages. The formal model already removes the direct

term embedding found in most existing multi-language formalisms and is thus an ideal starting point for removing execution order dependencies as well.

References

1. M. Abadi. Protection in programming-language translations. In *Secure Internet Programming*, volume 1603 of *LNCS*, 1999.
2. M. Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theor. Comput. Sci.*, 103(2):235–271, Sept. 1992.
3. S. Finne, D. Leijen, E. Meijer, and S. L. P. Jones. H/direct: A binary foreign language interface for haskell. In *ICFP '98*, pages 153–162. ACM.
4. M. Furr and J. S. Foster. Checking type safety of foreign function calls. *TOPLAS*, 30(4), July 2008.
5. A. Gordon. *Bisimilarity as a Theory of Functional Programming: Mini-course*. BRICS notes series. Computer Science Department, 1995.
6. D. J. Howe. Equality in lazy computation systems. In *LICS*, pages 198–203. IEEE Computer Society Press, June 1989.
7. R. Jagadeesan, C. Pitcher, J. Rathke, and J. Riely. Local memory via layout randomization. In *CSF '11*. IEEE, 2011.
8. A. Jeffrey and J. Rathke. Towards a theory of bisimulation for local names. Computer Science Report 02-2000, University of Sussex, 2000.
9. J. Laird. A fully abstract trace semantics for general references. In *Automata, Languages and Programming*, volume 4596 of *LNCS*, pages 667–679. Springer, 2007.
10. A. Larmuseau, M. Patrignani, and D. Clarke. Operational semantics for secure interoperation. In *PLAS Workshop '14*. ACM, 2014.
11. J. Matthews and R. B. Findler. Operational semantics for multi-language programs. *TOPLAS*, 31(3), Apr. 2009.
12. M. Patrignani, P. Agten, R. Strackx, B. Jacobs, D. Clarke, and F. Piessens. Secure compilation to protected module architectures. *ACM TOPLAS*, 2015.
13. D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. *ACM TOPLAS*, 33(1):5:1–5:69, Jan. 2011.
14. G. Tan, S. Chakradhar, R. Srivaths, and R. D. Wang. Safe Java native interface. In *ESSoS*, pages 97–106, March 2006.
15. P. Wadler and R. B. Findler. Well-typed programs can't be blamed. In *ESOP '09*, pages 1–16. Springer-Verlag, 2009.
16. M. Wand. The theory of fexprs is trivial. *Lisp and Symbolic Computation*, 10(3):189–199, 1998.