Ochre

CHARLIE LIDBURY, Imperial College London, United Kingdom

Abstract goes here

CCS Concepts: • **Theory of computation** → **Programming logic**; **Logic and verification**.

Additional Key Words and Phrases: Rust, verification, functional translation

ACM Reference Format:
Charlie Lidbury. 2024. Ochre. 1, 1 (May 2024), 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Hello There

2 ALL FIGURE

REFERENCES

Author's address: Charlie Lidbury, Imperial College London, United Kingdom, cal120@ic.ac.uk.

2024. XXXX-XXXX/2024/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

τ ::=	type
$\mathsf{bool} \mid \mathsf{uint32} \mid \mathsf{int32} \mid \ldots$	base types
$\&^{\rho}\,\mathsf{mut}\,\tau$	mutable borrow
$\&^{\rho}\,\tau$	immutable (shared) borrow
$T\ \vec{\tau}$	type application
$\alpha, \beta, \ldots$	type variables
$(\vec{\tau})$	tuple ($\mathsf{len}(\vec{\tau}) > 1$) or unit ($\mathsf{len}(\vec{\tau}) = 0$)
T ::=	type constructorapplication
t	user-defined data type
Box	boxed type
s ::=	statement
$\emptyset$	empty statement (nil)
$s;\ s$	sequence (cons)
$p := rv$	assignment
$p := f\ \vec{op}$	function call
$\mathsf{if}\ op\ \mathsf{then}\ s\ \mathsf{else}\ s$	conditional
$\mathsf{match}\ p\ \mathsf{with}\ \overrightarrow{C \to s}$	data typecase analysis
return	function exit
panic	unrecoverable error
$\ldots$	loops, etc.
rv ::=	assignable “r” values
op	operand
$\&\mathsf{mut}\ p$	mutable borrow
$\&\ p$	immutable (shared) borrow
$!op \mid op + op \mid op - op \mid \ldots$	operators
op ::=	operand
$\mathsf{move}\ p$	ownership transfer
$\mathsf{copy}\ p$	scalar copy
$\mathsf{true} \mid \mathsf{false} \mid n_{\mathsf{i32}} \mid n_{\mathsf{u32}} \mid \ldots$	constants
$C[\vec{f} = \vec{op}]$	data type constructor
$(\vec{op})$	tuple ($\mathsf{len}(\vec{op}) > 1$)or unit ($\mathsf{len}(\vec{op}) = 0$)
x	variable
p ::= $P[x]$	place
P ::=	path
$[.]$	base case
$*^{s} P$	deref shared borrow
$*^{m} P$	deref mutable borrow
$*^{b} P$	deref box
$P.f$	field selection
$P.n$	field selection (tuple)
D ::=	top-level declaration
$\mathsf{fn}\ f\ \langle\vec{\rho}\rangle\ (\vec{x}_{\mathsf{arg}} : \vec{\tau})\ (\vec{x}_{\mathsf{local}} : \vec{\tau})\ (x_{\mathsf{ret}} : \tau) = s$	function declaration
$\mathsf{type}\ t\ \vec{\alpha} = C[\vec{f} : \vec{\tau}] \mid \ldots$	data type declaration

Fig. 1. The Low-Level Borrow Calculus: Syntax

v	$::=$		value
		$\mathsf{true} \mid \mathsf{false} \mid n_{\mathsf{i32}} \mid n_{\mathsf{u32}} \mid \ldots$	booleans and integers
		$\mathsf{borrow}^m\ \ell\ v$	mutable borrow
		$\mathsf{borrow}^s\ \ell$	shared borrow
		$\mathsf{borrow}^r\ \ell$	reserved borrow
		$\mathsf{loan}^m\ \ell$	mutable loan
		$\mathsf{loan}^s\ \{\vec{\ell}\}\ v$	shared loan
		$\bot$	inaccessible value
		$C\,[\vec{f} = \vec{v}]$	data type constructor
		$(\vec{v})$	tuple constructor ($\mathsf{len}(\vec{v}) > 1$) or unit ($\mathsf{len}(\vec{v}) = 0$)
r	$::=$		result
		panic	unrecoverable error state
		$\mathsf{return}\,v$	successful, possibly-early exit
		$\ldots$	loop states, etc.
ℓ			loan identifier
Ω	$::=$		evaluation context
		$\emptyset$	empty
		$x \mapsto v, \Omega$	new mapping

Fig. 2. The Low-Level Borrow Calculus: Reduction Environments, Values

$$\textsc{Read}\ \frac{p = P[x] \qquad x \mapsto v_x \in \Omega \qquad \Omega \vdash P(v_x) \Rightarrow v}{\Omega(p) \Rightarrow v}
\qquad
\textsc{R-Box}\ \frac{\Omega \vdash P(v_P) \Rightarrow v}{\Omega \vdash (*^b P)(\mathsf{Box}\,v_P) \Rightarrow v}$$

$$\textsc{R-Mut-Borrow}\ \frac{\Omega \vdash P(v_P) \Rightarrow v}{\Omega \vdash (*^m P)(\mathsf{borrow}^m\ \ell\ v_P) \Rightarrow v}
\qquad
\textsc{R-Shared-Borrow}\ \frac{\mathsf{loan}^s\ \{\ell \cup _\}\ v_P \in \Omega \qquad \Omega \vdash P(v_P) \Rightarrow v}{\Omega \vdash (*^s P)(\mathsf{borrow}^s\ \ell) \Rightarrow v}$$

$$\textsc{R-Field}\ \frac{\Omega \vdash P(v_P) \Rightarrow v}{\Omega \vdash (P.f)(C\,[\overrightarrow{f = v_P}]) \Rightarrow v}
\qquad
\textsc{R-Base}\ \frac{}{\Omega \vdash [.](v) \Rightarrow v}
\qquad
\textsc{R-Shared-Loan}\ \frac{P \neq [.] \qquad \Omega \vdash P(v_P) \Rightarrow v}{\Omega \vdash P(\mathsf{loan}^s\ \{_\}\ v_P) \Rightarrow v}$$

$$\textsc{Write}\ \frac{\begin{matrix} x \mapsto v_x \in \Omega \qquad p = P[x] \\ \Omega \vdash P(v_x) \leftarrow v \Rightarrow v'_x \qquad \Omega' = \Omega[x \mapsto v'_x] \end{matrix}}{\Omega(p) \leftarrow v \Rightarrow \Omega'}
\qquad
\textsc{W-Box}\ \frac{\Omega \vdash P(v_P) \leftarrow v \Rightarrow v'_P}{\Omega \vdash (*^b P)(\mathsf{Box}\,v_P) \leftarrow v \Rightarrow \mathsf{Box}\,v'_P}$$

$$\textsc{W-Mut-Borrow}\ \frac{\Omega \vdash P(v_P) \leftarrow v \Rightarrow v'_P}{\Omega \vdash (*^m P)(\mathsf{borrow}^m\ \ell\ v_P) \leftarrow v \Rightarrow \mathsf{borrow}^m\ \ell\ v'_P}
\qquad
\textsc{W-Base}\ \frac{}{\Omega \vdash [.](v_P) \leftarrow v \Rightarrow v}$$

$$\textsc{W-Field}\ \frac{f = f_i \qquad \Omega \vdash P(v_{P,i}) \leftarrow v \Rightarrow v'_{P,i} \qquad v'_{P,j} = v_{P,j} \text{ for } j \neq i}{\Omega \vdash (P.f)(C[\overrightarrow{f = v_P}]) \leftarrow v \Rightarrow C[\overrightarrow{f = v'_P}]}$$

Fig. 3. Reading From and Writing To Our Structured Memory Model

$$\textsc{E-Mut-Borrow}\quad \frac{\Omega(p) \Rightarrow v \qquad \{\bot, \text{loan}, \text{borrow}^r\} \notin v \\ *^s \notin p \qquad \ell \text{ fresh} \qquad \Omega(p) \leftarrow \text{loan}^m\, \ell \Rightarrow \Omega'}{\Omega \vdash \&\text{mut}\, p \rightsquigarrow \text{borrow}^m\, \ell\, v \dashv \Omega'}$$

$$\textsc{E-Shared-Or-Reserved-Borrow}\quad \frac{\Omega(p) \Rightarrow v \qquad \{\bot, \text{loan}^m, \text{borrow}^r\} \notin v \\ \ell \text{ fresh} \qquad \Omega' = \Omega[p \mapsto v'] \\ v' = \begin{cases} \text{loan}^s\, \{\vec{\ell} \cup \ell\}\, v'' & \text{if } v = \text{loan}^s\, \{\vec{\ell}\}\, v'' \\ \text{loan}^s\, \{\ell\}\, v & \text{otherwise} \end{cases}}{\Omega \vdash \&\, p \rightsquigarrow \text{borrow}^{r,s}\, \ell \dashv \Omega'}$$

$$\textsc{E-Move}\quad \frac{\Omega(p) \Rightarrow v \qquad \{\bot, \text{loan}, \text{borrow}^r\} \notin v \\ \{*^m, *^s\} \notin p \qquad \Omega(p) \leftarrow \bot \Rightarrow \Omega'}{\Omega \vdash \text{move}\, p \rightsquigarrow v \dashv \Omega'}$$

$$\textsc{E-Copy}\quad \frac{\Omega(p) \Rightarrow v \qquad \{\bot, \text{loan}^m, \text{borrow}^{m,r}\} \notin v \\ \Omega \vdash \text{copy}\, v \Rightarrow v' \dashv \Omega'}{\Omega \vdash \text{copy}\, p \rightsquigarrow v' \dashv \Omega'}$$

$$\textsc{E-Constructor}\quad \frac{\Omega_i \vdash op_i \rightsquigarrow v_i \dashv \Omega_{i+1}}{\Omega_0 \vdash C[\overrightarrow{f = op}] \rightsquigarrow C[\overrightarrow{f = v}] \dashv \Omega_n}$$

$$\textsc{E-Return}\quad \frac{\Omega_i \vdash x_{\text{local},i} := \bot \rightsquigarrow () \dashv \Omega_{i+1} \\ \Omega_n(x_{\text{ret}}) \Rightarrow v \qquad \{\bot, \text{loan}, \text{borrow}^r\} \notin v}{\Omega_0 \vdash \text{return} \rightsquigarrow \text{return}\, v \dashv \Omega_n}$$

$$\textsc{E-IfThenElse-T}\quad \frac{\Omega \vdash op \rightsquigarrow \text{true} \dashv \Omega' \\ \Omega' \vdash s_1 \rightsquigarrow r \dashv \Omega''}{\Omega \vdash \text{if}\, op\, \text{then}\, s_1\, \text{else}\, s_2 \rightsquigarrow r \dashv \Omega''}$$

$$\textsc{E-Assign}\quad \frac{\Omega \vdash rv \rightsquigarrow v \dashv \Omega' \qquad \Omega'(p) \Rightarrow v_p \\ v_p \text{ has no outer loans} \qquad x_{\text{old}} \text{ fresh} \\ \Omega'(p) \leftarrow v \Rightarrow \Omega'' \qquad \Omega''' = \Omega''[x_{\text{old}} \mapsto v_p]}{\Omega \vdash p := rv \rightsquigarrow () \dashv \Omega'''}$$

$$\textsc{E-Free}\quad \frac{\Omega(p) \Rightarrow \text{Box}\, v \qquad \Omega \vdash p := \bot \rightsquigarrow () \dashv \Omega'}{\Omega \vdash \text{free}\, p \rightsquigarrow () \dashv \Omega'}$$

$$\textsc{E-Match}\quad \frac{\Omega \vdash p \overset{s}{\Rightarrow} C[\overrightarrow{f = v}] \qquad \Omega \vdash s \rightsquigarrow r \dashv \Omega'}{\Omega \vdash \text{match}\, p\, \text{with}\ \ldots \mid C \to s \mid \ldots \rightsquigarrow r \dashv \Omega'}$$

Fig. 4. Selected Reduction Rules for LLBC. We omit: tuples (similar to constructor), sequences (trivial). We also omit the handling of results – these prevent further execution and simply get carried through. Boxes behave like regular ADT constructors, except for the `free` Rust function, which receives primitive treatment, above.

$$\textsc{R-Not-Shared}\quad \frac{\Omega(p) \Rightarrow v \qquad v \neq \text{loan}^s\, \{\vec{l}\}\, v'}{\Omega(p) \overset{s}{\Rightarrow} v}$$

$$\textsc{R-Shared}\quad \frac{\Omega(p) \Rightarrow \text{loan}^s\, \{\vec{l}\}\, v}{\Omega(p) \overset{s}{\Rightarrow} v}$$

Fig. 5. Auxiliary Judgment: Reading a Possibly Immutably-Shared Value. Rust allows matching on a value for which there are oustanding *shared* borrows; the auxiliary $\overset{s}{\Rightarrow}$ read allows reading underneath a loan^s, if applicable.

$$\frac{\ell' \text{ fresh} \qquad \mathsf{loan}^s\,\{\ell \cup \vec{\ell}\}\,v \in \Omega \qquad \Omega' = \left[\mathsf{loan}^s\,\{\ell \cup \ell' \cup \vec{\ell}\}\,v / \mathsf{loan}^s\,\{\ell \cup \vec{\ell}\}\,v\right]\Omega}{\Omega \vdash \mathsf{copy}\,\mathsf{borrow}^s\,\ell \Rightarrow \mathsf{borrow}^s\,\ell' \dashv \Omega'}\;\textsc{C-Shared-Borrow}$$

$$\frac{\Omega \vdash \mathsf{copy}\,v \Rightarrow v' \dashv \Omega'}{\Omega \vdash \mathsf{copy}\,\mathsf{loan}^s\,\{\vec{\ell}\}\,v \Rightarrow v' \dashv \Omega'}\;\textsc{C-Shared-Loan}$$

$$\frac{v = \mathsf{true} \text{ or } \mathsf{false} \text{ or } n_{\mathsf{i32}} \text{ or } n_{\mathsf{u32}} \text{ or } \ldots}{\Omega \vdash \mathsf{copy}\,v \Rightarrow v \dashv \Omega}\;\textsc{C-Scalar}$$

$$\frac{}{\Omega \vdash \mathsf{copy}\,\mathsf{None} \Rightarrow \mathsf{None} \dashv \Omega}\;\textsc{C-None}$$

$$\frac{\Omega \vdash \mathsf{copy}\,v \Rightarrow v' \dashv \Omega'}{\Omega \vdash \mathsf{copy}\,\mathsf{Some}\,v \Rightarrow \mathsf{Some}\,v' \dashv \Omega'}\;\textsc{C-Some}$$

$$\frac{\Omega_i \vdash \mathsf{copy}\,v_i \Rightarrow v_i' \dashv \Omega_{i+1}}{\Omega_0 \vdash \mathsf{copy}\,(\vec{v}) \Rightarrow (\vec{v}') \dashv \Omega_n}\;\textsc{C-Tuple}$$

Fig. 6. Auxiliary Judgment: Copying. We mimic MIR and see the copy of options and tuples as primitive operations. The judgment is undefined for any other construct as Rust's MIR only permits copying primitive data.

$$\frac{\ominus\, v_p}{v_p \text{ has no outer loans}}\;\textsc{A-Shorthand} \qquad \frac{v = \mathsf{true} \text{ or } \mathsf{false} \text{ or } n_{\mathsf{i32}} \text{ or } n_{\mathsf{u32}} \text{ or } \ldots}{\ominus\, v}\;\textsc{A-Scalar}$$

$$\frac{\ominus\, v_i}{\ominus\, (\vec{v})}\;\textsc{A-Tuple} \qquad \frac{\ominus\, v_i}{\ominus\, C[\vec{f} = \vec{v}]}\;\textsc{A-Constructor}$$

$$\frac{}{\ominus\, \mathsf{borrow}^m\,\ell\,v}\;\textsc{A-Borrow-M} \qquad \frac{}{\ominus\, \mathsf{borrow}^{r,s}\ell}\;\textsc{A-Borrow-R-S} \qquad \frac{}{\ominus\, \bot}\;\textsc{A-Bot}$$

Fig. 7. Auxiliary Judgment: Absence of Outer Loans. We use shorthand notation $\ominus$ for this figure. Enforcing this criterion ensures that, at assignment-time, the memory we are about to write does not contained loaned-out data, as this would be unsound. This judgement is defined by omission, and is never valid for values of the form loan _. Such values, however, may appear underneath borrows, as the A-Borrow-* rules enforce no preconditions.

$$\frac{p = P[x] \qquad x \mapsto v_x \in \Omega \qquad \Omega \vdash p(v_x) \leftarrow v \overset{g}{\Rightarrow} v_x' \dashv \Omega' \qquad \Omega'' = \Omega'[x \mapsto v_x']}{\Omega[p \mapsto v] = \Omega''}\;\textsc{Write-G}$$

$$\frac{\mathsf{loan}^s\,\{\ell \cup \_\}\,v_p \in \Omega \qquad \Omega \vdash p(v_p) \leftarrow v \overset{g}{\Rightarrow} v_p' \dashv \Omega' \qquad \Omega'' = \left[\mathsf{loan}^s\,\{\ell \cup \_\}\,v_p' / \mathsf{loan}^s\,\{\ell \cup \_\}\,v_p\right]\Omega'}{\Omega \vdash (*^s p)(\mathsf{borrow}^s\,\ell) \leftarrow v \overset{g}{\Rightarrow} \mathsf{borrow}^s\,\ell \dashv \Omega''}\;\textsc{W-G-Shared-Borrow}$$

Fig. 8. Auxiliary Judgment: Ghost Write. This judgment inherits all of the rules of the form W-*.

$$\textsc{Not-Borrowed}\quad \frac{\nexists V', V''.\ V[\cdot] = V'[\mathsf{borrow}^m\ \_\ (V''[\cdot])] \qquad \nexists V', V''.\ V[\cdot] = V'[\mathsf{loan}^s\ \{\_\}\ (V''[\cdot])]}{\mathsf{not\_borrowed\_value}\ V}$$

$$\textsc{End-Shared-Or-Reserved-1}\quad \frac{\mathsf{not\_borrowed\_value}\ V}{\begin{array}{l}\Omega[x_1 \mapsto V[\mathsf{borrow}^{r,s}\ell], x_2 \mapsto V'[\mathsf{loan}^s\ \{\ell\}\ v]] \hookrightarrow \\ \Omega[x_1 \mapsto V[\bot], \qquad\qquad x_2 \mapsto V'[v]]\end{array}}$$

$$\textsc{Not-Shared}\quad \frac{\nexists V', V''.\ V[\cdot] = V'[\mathsf{loan}^s\ \{\_\}\ (V''[\cdot])]}{\mathsf{not\_shared\_value}\ V}$$

$$\textsc{End-Shared-Or-Reserved-2}\quad \frac{\mathsf{not\_borrowed\_value}\ V \qquad \ell \notin \vec{\ell}}{\begin{array}{l}\Omega[x_1 \mapsto V[\mathsf{borrow}^{r,s}\ell], x_2 \mapsto V'[\mathsf{loan}^s\ \{\ell \cup \vec{\ell}\}\ v]] \hookrightarrow \\ \Omega[x_1 \mapsto V[\bot], \qquad\qquad x_2 \mapsto V'[\mathsf{loan}^s\ \{\vec{\ell}\}\ v]]\end{array}}$$

$$\textsc{End-Mut}\quad \frac{\{\mathsf{loan}, \mathsf{borrow}^r\} \notin v \qquad \mathsf{not\_borrowed\_value}\ V}{\begin{array}{l}\Omega[x_1 \mapsto V[\mathsf{borrow}^m\ \ell\ v], x_2 \mapsto V'[\mathsf{loan}^m\ \ell]] \hookrightarrow \\ \Omega[x_1 \mapsto V[\bot], \qquad\qquad x_2 \mapsto V'[v]]\end{array}}$$

$$\textsc{Activate-Reserved}\quad \frac{\{\mathsf{loan}, \mathsf{borrow}^r\} \notin v \qquad \mathsf{not\_shared\_value}\ V'}{\begin{array}{l}\Omega[x_1 \mapsto V[\mathsf{borrow}^r\ \ell], \quad x_2 \mapsto V'[\mathsf{loan}^s\ \{\ell\}\ v]] \hookrightarrow \\ \Omega[x_1 \mapsto V[\mathsf{borrow}^m\ \ell\ v], x_2 \mapsto V'[\mathsf{loan}^m\ \ell]]\end{array}}$$

Fig. 9. Reorganizing Environments

v	$::=$		
		$\ldots$	(as before)
		$(\sigma : \tau)$	symbolic value
		$\mathsf{proj}_{\mathsf{in}}\ v$	input borrow projector
		$\mathsf{proj}_{\mathsf{l}}\ v$	loan projector
		$\mathsf{proj}_{\mathsf{out}}\ v$	output borrow projector
Ω	$::=$		
		$\ldots$	(as before)
		$A(\rho)\{\vec{v}\}, \Omega$	new region abstraction for region ρ
ρ	$::=$		region identifier

Fig. 10. Abstract Semantics: Environments, Values

Decompose-Tuple

$$\frac{\sigma_l, \sigma_r \text{ fresh}}{\Omega \underset{(\sigma_l, \sigma_r)}{\overset{\sigma}{\hookrightarrow}} \Big[((\sigma_l, \sigma_r) : (\tau_1, \tau_2)) \Big/ (\sigma : (\tau_1, \tau_2))\Big]\, \Omega}$$

Proj-Tuple

$$\frac{}{\begin{array}{l}\Omega[A(\rho) \mapsto \mathsf{proj}_{\mathsf{in,l,out}}\ (\sigma_l, \sigma_r)] \hookrightarrow \\ \Omega[A(\rho) \mapsto (\mathsf{proj}_{\mathsf{in,l,out}}\ \sigma_l, \mathsf{proj}_{\mathsf{in,l,out}}\ \sigma_r)]\end{array}}$$

Proj-I-Mut-Match

$$\frac{}{\begin{array}{l}\Omega[A(\rho) \mapsto \mathsf{proj}_{\mathsf{in}}\ (\mathsf{borrow}^m\ \ell\ \sigma : \&^\rho \mathsf{mut}\ \tau)] \hookrightarrow \\ \Omega[A(\rho) \mapsto \mathsf{borrow}^m\ \ell\ (\sigma : \tau)]\end{array}}$$

Proj-I-Mut-No-Match

$$\frac{}{\begin{array}{l}\Omega[A(\rho) \mapsto \mathsf{proj}_{\mathsf{in}}\ (\mathsf{borrow}^m\ \ell\ \_ : \&^\mu \mathsf{mut}\ \tau)] \hookrightarrow \\ \Omega[A(\rho) \mapsto \_]\end{array}}$$

Proj-I-Shared-Match

$$\frac{}{\begin{array}{l}\Omega[A(\rho) \mapsto \mathsf{proj}_{\mathsf{in}}\ (\mathsf{borrow}^s\ \ell : \&^\rho\ \tau)] \hookrightarrow \\ \Omega[A(\rho) \mapsto \mathsf{borrow}^s\ \ell]\end{array}}$$

Proj-I-Shared-No-Match

$$\frac{}{\begin{array}{l}\Omega[A(\rho) \mapsto \mathsf{proj}_{\mathsf{in}}\ (\mathsf{borrow}^s\ \ell : \&^\mu\ \tau)] \hookrightarrow \\ \Omega[A(\rho) \mapsto \_]\end{array}}$$

Proj-I-Symb

$$\frac{\& \notin \tau}{\Omega[A(\rho) \mapsto \mathsf{proj}_{\mathsf{in}}\ (\sigma : \tau)] \hookrightarrow \Omega[A(\rho) \mapsto \sigma]}$$

Proj-Unfold-Mut-Match

$$\frac{\sigma', \ell \text{ fresh}}{\begin{array}{ll}\Omega[p \mapsto \mathsf{proj}_{\mathsf{out}}\ (\sigma : \&^\rho \mathsf{mut}\ \tau), & A(\rho) \mapsto \mathsf{proj}_{\mathsf{l}}\ \sigma] \hookrightarrow \\ \Omega[p \mapsto \mathsf{borrow}^m\ \ell\ \sigma', & A(\rho) \mapsto \mathsf{loan}^m\ \ell]\end{array}}$$

Proj-Unfold-Shared-Match

$$\frac{\sigma', \ell \text{ fresh}}{\begin{array}{ll}\Omega[p \mapsto \mathsf{proj}_{\mathsf{out}}\ (\sigma : \&^\rho\ \tau), & A(\rho) \mapsto \mathsf{proj}_{\mathsf{l}}\ \sigma] \hookrightarrow \\ \Omega[p \mapsto \mathsf{borrow}^s\ \ell, & A(\rho) \mapsto \mathsf{loan}^s\ \{\ell\}\ \sigma']\end{array}}$$

Proj-L-No-Match

$$\frac{\& \notin \tau}{\begin{array}{l}\Omega[A(\rho) \mapsto \mathsf{proj}_{\mathsf{l}}\ (\sigma : \tau)] \hookrightarrow \\ \Omega[A(\rho) \mapsto \_]\end{array}}$$

End-Abstract-Mut

$$\frac{\mathsf{not\_borrowed\_value}\ V}{\begin{array}{ll}\Omega[x \mapsto V[\mathsf{borrow}^m\ \ell\ v], & A(\rho) \mapsto \mathsf{loan}^m\ \ell] \hookrightarrow \\ \Omega[x \mapsto V[\bot], & A(\rho) \mapsto v]\end{array}}$$

End-Abstraction

$$\frac{\begin{array}{c}\mathsf{borrows}^m(A(\rho)) = \overrightarrow{\mathsf{borrow}^m\ \ell\ \_} \\ \mathsf{borrows}^s(A(\rho)) = \overrightarrow{\mathsf{borrow}^s\ \ell'} \\ \mathsf{loan}, \mathsf{proj}_{\mathsf{l}} \notin A(\rho).v_r \qquad \vec{\sigma}' \text{ fresh} \qquad \vec{x}_g, \vec{y}_g \text{ fresh}\end{array}}{A(\rho), \Omega \hookrightarrow \Omega, \overrightarrow{x_g \mapsto \mathsf{borrow}^m\ \ell\ \sigma'}, \overrightarrow{y_g \mapsto \mathsf{borrow}^s\ \ell'}}$$

Fig. 11. Reorganizing Environments with Abstract Values and Projectors

$$\begin{array}{lcll}\mathsf{sym}(\alpha, (\vec{\sigma}), (\vec{v})) & = & (\overrightarrow{\mathsf{sym}(\alpha, \sigma, v)}) & \\ \mathsf{sym}(\alpha, \sigma, \mathsf{borrow}^m\ \ell\ v : \&^\alpha \mathsf{mut}\ \tau) & = & \mathsf{borrow}^m\ \ell\ \sigma & \\ \mathsf{sym}(\alpha, \_, \mathsf{borrow}^m\ \ell\ v : \&^\beta \mathsf{mut}\ \tau) & = & \bot & \alpha \neq \beta\end{array}$$

Fig. 12. The sym function, used for the generation of backward functions. Specifically, $\mathsf{sym}(\alpha, \sigma, v)$ models the caller invoking the backward function for α with the value originally returned by the forward function to said caller. We abstract away the concrete view of the return value (v) into a symbolic view (modeled by σ) – essentially saying that the caller may have arbitrarily mutated the return value while it owned it.

$$\textsc{Pure-Mut-Borrow}\ \frac{\Omega \vdash v \updownarrow e}{\Omega \vdash \mathsf{borrow}^m\ \ell\ v \updownarrow e} \qquad \textsc{Pure-Const}\ \frac{}{\Omega \vdash n_{\mathsf{i32}} \updownarrow n_{\mathsf{i32}}} \qquad \textsc{Pure-Tuple}\ \frac{\Omega \vdash \vec{v} \updownarrow \vec{e}}{\Omega \vdash (\vec{v}) \updownarrow (\vec{e})} \qquad \textsc{Pure-Symb}\ \frac{}{\Omega \vdash \sigma \updownarrow \sigma} \qquad \textsc{Pure-Box}\ \frac{\Omega \vdash v \updownarrow e}{\Omega \vdash \mathsf{Box}\, v \updownarrow e}$$

$$\textsc{Pure-Shared-Borrow}\ \frac{\begin{array}{c}\mathsf{loan}^s\ \{\vec{\ell}\}\ v \in \Omega \qquad \ell \in \vec{\ell} \\ \Omega \vdash v \updownarrow e\end{array}}{\Omega \vdash \mathsf{borrow}^s\ \ell \updownarrow e}$$

$$\textsc{T-Destruct}\ \frac{\vec{\sigma}'\ \text{fresh} \qquad M, \Omega \xhookrightarrow[(\vec{\sigma}')]{\sigma} M', \Omega'}{M, \Omega \vdash \emptyset \updownarrow \mathsf{let}\ (\vec{\sigma}') = \sigma\ \mathsf{in}\ [\cdot] \dashv M', \Omega''}$$

$$\textsc{T-Return-Forward}\ \frac{\begin{array}{c}\Omega_i \vdash x_{\mathsf{local},i} := \bot \rightsquigarrow () \dashv \Omega_{i+1} \qquad \Omega_n(x_{\mathsf{ret}}) \Rightarrow v \\ \Omega_n \vdash v \downarrow e \qquad \{\mathsf{loan}, \bot, \mathsf{borrow}^r\} \notin v\end{array}}{M, \Omega_0 \vdash \mathsf{return} \downarrow e}$$

$$\textsc{T-IfThenElse}\ \frac{\begin{array}{c}M, \Omega \vdash op \updownarrow \sigma \dashv M', \Omega' \qquad \Omega' \xhookrightarrow[\mathsf{true}]{\sigma} \Omega_1 \qquad \Omega' \xhookrightarrow[\mathsf{false}]{\sigma} \Omega_2 \\ M', \Omega_1 \vdash s_1 \updownarrow e_1 \qquad M', \Omega_2 \vdash s_2 \updownarrow e_2\end{array}}{M, \Omega \vdash \mathsf{if}\ op\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2 \updownarrow \mathsf{if}\ \sigma\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2}$$

$$\textsc{T-Seq}\ \frac{\begin{array}{c}M, \Omega \vdash s_1 \updownarrow E[\cdot] \dashv M', \Omega' \\ M', \Omega' \vdash s_2 \updownarrow e \dashv M'', \Omega''\end{array}}{M, \Omega \vdash s_1; s_2 \updownarrow E[e] \dashv M'', \Omega''}$$

$$\textsc{T-Call-Forward}\ \frac{\begin{array}{c}A(\rho_i) = \left\{\overrightarrow{\mathsf{proj_{in}}\ v, \mathsf{proj_l}\ \sigma_{\mathsf{ret}}}\right\} \qquad (\sigma : \tau) \in \vec{v}\ \text{implies}\ \& \notin \tau \qquad \vec{\rho}\ \text{fresh} \\ \sigma_{\mathsf{ret}}\ \text{fresh} \qquad \Omega \vdash v_i \updownarrow e_i \qquad \Omega, \overrightarrow{A(\rho)} \vdash p := \mathsf{proj_{out}}\ \sigma_{\mathsf{ret}} \rightsquigarrow () \dashv \Omega'\end{array}}{M, \Omega \vdash p := f\langle\vec{\rho}\rangle\, \vec{v} \updownarrow \sigma_{\mathsf{ret}} \leftharpoonup f_{\mathsf{fwd}}\, \vec{e};\ [\cdot] \dashv (f\langle\vec{\rho}\rangle\, \vec{e}), M; \Omega'}$$

$$\textsc{T-Reorg-Anytime}\ \frac{\begin{array}{c}M, \Omega \vdash \emptyset \updownarrow E[\cdot] \dashv M', \Omega' \\ M', \Omega' \vdash s \updownarrow E'[\cdot] \dashv M'', \Omega''\end{array}}{M, \Omega \vdash s \updownarrow E[E'[\cdot]] \dashv M'', \Omega''}$$

$$\textsc{T-Call-Backward}\ \frac{\begin{array}{c}\Omega = A(\rho), \Omega' \qquad M = f\langle \ldots, \rho, \ldots \rangle\, \vec{v}, M' \qquad \Omega \vdash A(\rho).v_{\mathsf{ret}} \updownarrow e_{\mathsf{ret}} \\ A(\rho), \Omega' \hookrightarrow \Omega'', \overrightarrow{x_g \mapsto \mathsf{borrow}^m\ \ell\ \sigma'}, \overrightarrow{y_g \mapsto \ldots}\ \text{via}\ \textsc{End-Abstraction}\end{array}}{M, \Omega \vdash \emptyset \updownarrow (\vec{\sigma}') \leftharpoonup f_{\mathsf{back}(\rho)}\, \vec{e}\, e_{\mathsf{ret}}; [\cdot] \dashv M', \Omega''}$$

$$\textsc{T-Return-Backward}\ \frac{\begin{array}{c}\Omega_i \vdash x_{\mathsf{local},i} := \bot \rightsquigarrow () \dashv \Omega_{i+1} \\ \Omega_n(x_{\mathsf{ret}}) \Rightarrow v_{\mathsf{ret}} \qquad \{\mathsf{loan}, \bot, \mathsf{borrow}^r\} \notin v_{\mathsf{ret}} \\ \Omega' = \Omega_n[x_{\mathsf{ret}} \mapsto \mathsf{sym}(\rho, \sigma_{\mathsf{ret}}, v_{\mathsf{ret}})] \\ \Omega' \vdash \emptyset \uparrow^\rho E[\cdot] \dashv \Omega'' \qquad \Omega''[A(\rho)] = \{\vec{v}\} \\ \{\mathsf{loan}, \mathsf{proj}\} \notin \vec{v} \qquad \Omega \vdash \vec{v} \uparrow \vec{e}\end{array}}{\Omega_0 \vdash \mathsf{return} \uparrow^\rho E[\mathsf{ret}\ (\vec{e})]}$$

$$\textsc{T-Fun-Backward}\ \frac{\begin{array}{c}A(\rho_i) = \left\{\overrightarrow{\mathsf{proj_l}\ (\sigma : \tau)}\right\} \qquad \vec{\sigma}\ \text{fresh} \qquad \emptyset, \Omega \vdash s \uparrow^\rho e \\ \Omega = \overrightarrow{A(\rho)}, \overrightarrow{x \mapsto \mathsf{proj_{out}}\ \sigma}, \vec{x}_{\mathsf{local}} \mapsto \vec{\bot}, x_{\mathsf{ret}} \mapsto \bot\end{array}}{\begin{array}{l}\mathsf{fn}\ f\ \langle\vec{\rho}\rangle\ (\vec{x} : \vec{\tau})\ (\vec{x}_{\mathsf{local}} : \vec{\tau}_{\mathsf{local}})\ (x_{\mathsf{ret}} : \tau_{\mathsf{ret}}) = s \uparrow^\rho \\ \mathsf{def}\ f_{\mathsf{back}(\rho)} = \lambda(\vec{\sigma} : \vec{\tau})(\sigma_{\mathsf{ret}} : \mathsf{proj}_\rho\ \tau_{\mathsf{ret}}).e\end{array}}$$

$$\textsc{T-Fun-Forward}\ \frac{\begin{array}{c}A(\rho_i) = \left\{\overrightarrow{\mathsf{proj_l}\ (\sigma : \tau)}\right\} \qquad \vec{\sigma}\ \text{fresh} \qquad \emptyset, \Omega \vdash s \downarrow e \\ \Omega = \overrightarrow{A(\rho)}, \overrightarrow{x \mapsto \mathsf{proj_{out}}\ \sigma}, \vec{x}_{\mathsf{local}} \mapsto \vec{\bot}, x_{\mathsf{ret}} \mapsto \bot\end{array}}{\begin{array}{l}\mathsf{fn}\ f\ \langle\vec{\rho}\rangle\ (\vec{x} : \vec{\tau})\ (\vec{x}_{\mathsf{local}} : \vec{\tau}_{\mathsf{local}})\ (x_{\mathsf{ret}} : \tau_{\mathsf{ret}}) = s \downarrow \\ \mathsf{def}\ f_{\mathsf{fwd}} = \lambda(\vec{\sigma} : \vec{\tau}).e\end{array}}$$

$$\textsc{T-Match-Concrete}\ \frac{\begin{array}{c}\Omega(p) \overset{s}{\Rightarrow} C[\overrightarrow{f = v}] \\ M, \Omega \vdash s \updownarrow e\end{array}}{M, \Omega \vdash \mathsf{match}\ p\ \mathsf{with}\ \ldots \mid C \to s \mid \ldots \updownarrow e}$$

$$\textsc{T-Match-Symbolic}\ \frac{\begin{array}{c}\Omega(p) \overset{s}{\Rightarrow} (\sigma : t\ \vec{\tau}) \qquad \mathsf{type}\ t\ \vec{\alpha} = \overrightarrow{C[\vec{f} : \vec{\tau}_f]} \qquad \vec{\sigma}_i\ \text{fresh} \\ M, \left[(C_i[\vec{f}_i : \vec{\sigma}_i] : t\ \vec{\tau}) \middle/ (\sigma : t\ \vec{\tau})\right] \Omega \vdash s_i \updownarrow e_i\end{array}}{M, \Omega \vdash \mathsf{match}\ p\ \mathsf{with}\ \overrightarrow{C \to s} \updownarrow \mathsf{match}\ \sigma\ \mathsf{with}\ \overrightarrow{C\ \vec{\sigma} \to e}}$$

Fig. 13. Functional Translation via our Symbolic Semantics