An Adjoint Separation Logic for the Wasm Call Stack

Work in Progress!

Andrew Wagner   Zachary Eisbach   Amal Ahmed

Northeastern University

June 11, 2025 @ Dagstuhl 25241

Context: Formally Specifying ABIs

Application Binary Interface (ABI)

`The run-time contract for using a particular API`

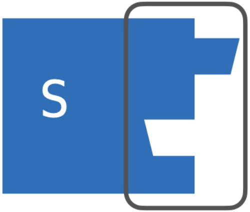
– Swift

Context: Formally Specifying ABIs

Application Binary Interface (ABI)

`The run-time contract for using a particular API`

– Swift



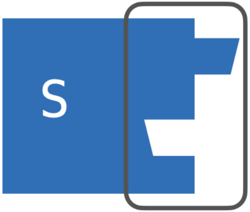
This Type τ

Context: Formally Specifying ABIs

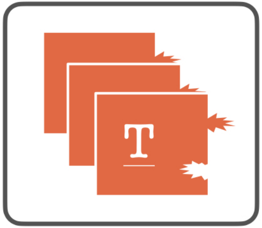
Application Binary Interface (ABI)

The run-time contract for using a particular API

– Swift



This Type τ



Is *Realistically Realized [Benton06]*

By These Target Programs

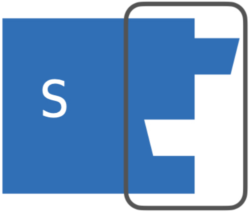
$$[\![\tau]\!] = \{ \underline{e} \mid \ldots \}$$

Context: Formally Specifying ABIs

Application Binary Interface (ABI)

The run-time contract for using a particular API

– Swift

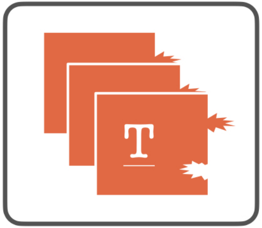


This Type τ

Our Approach

$\underline{e}$ is **ABI compliant** with τ if

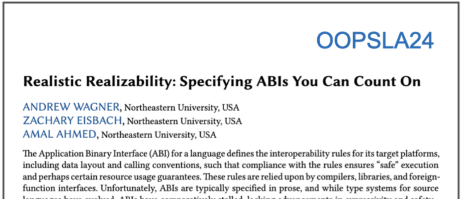
$$\underline{e} \in \llbracket \tau \rrbracket$$



Is *Realistically Realized [Benton06]*

By These Target Programs

$\llbracket \tau \rrbracket = \{ \underline{e} \mid \dots \}$



OOPSLA24

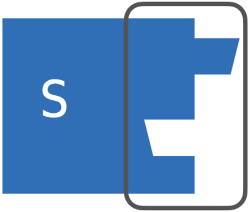
Realistic Realizability: Specifying ABIs You Can Count On

ANDREW WAGNER, Northeastern University, USA
ZACHARY EISBACH, Northeastern University, USA
AMAL AHMED, Northeastern University, USA

The Application Binary Interface (ABI) for a language defines the interoperability rules for its target platforms, including data layout and calling conventions, such that compliance with the rules ensures "safe" execution and perhaps certain resource usage guarantees. These rules are relied upon by compilers, libraries, and foreign-function interfaces. Unfortunately, ABIs are typically specified in prose, and while type systems for source

Context: Formally Specifying ABIs

Application Binary Interface (ABI)
`The run-time contract for using a particular API`
– Swift

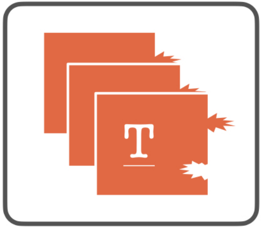


This Type τ

Our Approach

e is **ABI compliant** with τ if

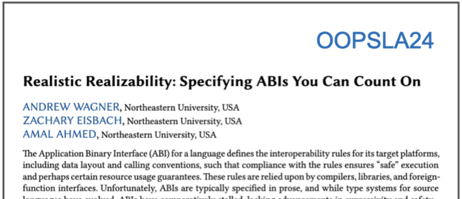
$$e \in \llbracket \tau \rrbracket$$



Is *Realistically Realized [Benton06]*
By These Target Programs

$\llbracket \tau \rrbracket = \{\, e \mid \dots \,\}$

Target-level separation logic predicate



OOPSLA24

Realistic Realizability: Specifying ABIs You Can Count On

ANDREW WAGNER, Northeastern University, USA
ZACHARY EISBACH, Northeastern University, USA
AMAL AHMED, Northeastern University, USA

The Application Binary Interface (ABI) for a language defines the interoperability rules for its target platforms, including data layout and calling conventions, such that compliance with the rules ensures "safe" execution and perhaps certain resource usage guarantees. These rules are relied upon by compilers, libraries, and foreign-function interfaces. Unfortunately, ABIs are typically specified in prose, and while type systems for source

Toward a Rust ABI for Wasm

We need:

Toward a Rust ABI for Wasm

We need:

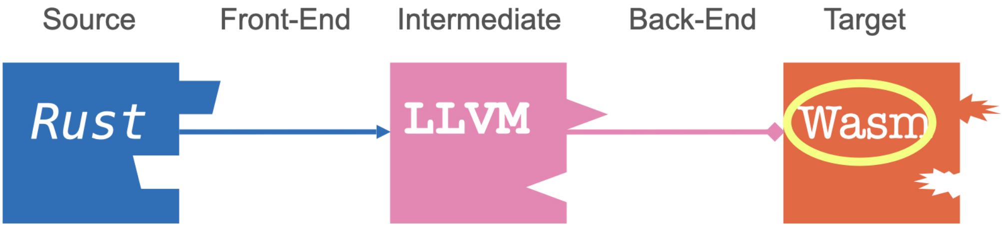
1. A `semantics` for borrowing

Toward a Rust ABI for Wasm

We need:

1. A `semantics` for borrowing
2. To support *independent updates* to the `front-end` and `back-end`

Toward a Rust ABI for Wasm

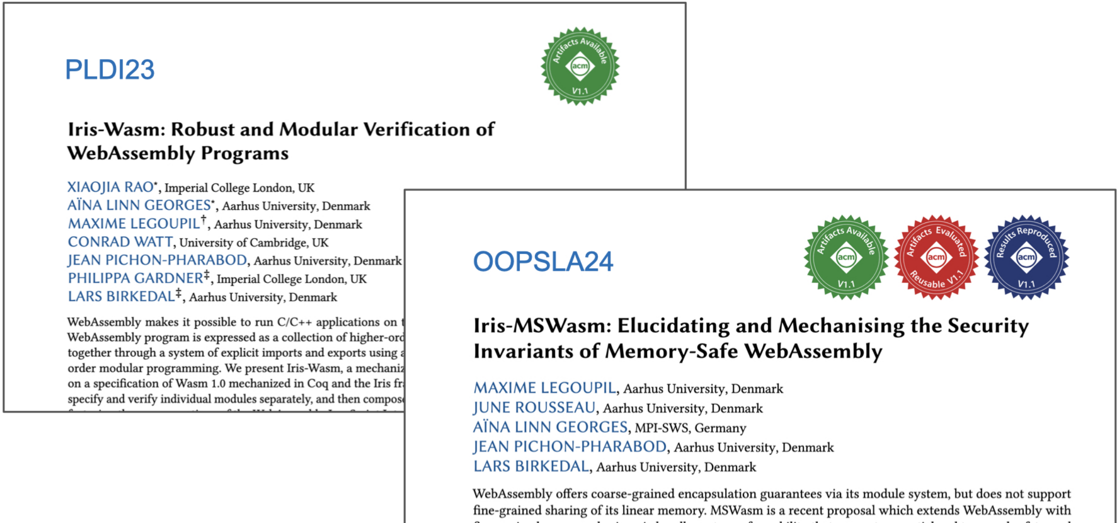


$$\Gamma \vdash e : T \cdots\cdots\blacktriangleright \{[\![\Gamma]\!]\}\ e\ \{[\![T]\!]\} \cdots\cdots\blacklozenge \{[\![\Gamma]\!]\}\ e\ \{[\![T]\!]\}$$

We need:

1. A `semantics` for borrowing
2. To support *independent updates* to the `front-end` and `back-end`
3. A separation logic for Wasm

State of the Art: IrisWasm



PLDI23

Iris-Wasm: Robust and Modular Verification of WebAssembly Programs

XIAOJIA RAO*, Imperial College London, UK
AÏNA LINN GEORGES*, Aarhus University, Denmark
MAXIME LEGOUPIL†, Aarhus University, Denmark
CONRAD WATT, University of Cambridge, UK
JEAN PICHON-PHARABOD, Aarhus University, Denmark
PHILIPPA GARDNER‡, Imperial College London, UK
LARS BIRKEDAL‡, Aarhus University, Denmark

WebAssembly makes it possible to run C/C++ applications on t
WebAssembly program is expressed as a collection of higher-ord
together through a system of explicit imports and exports using a
order modular programming. We present Iris-Wasm, a mechaniz
on a specification of Wasm 1.0 mechanized in Coq and the Iris fr
specify and verify individual modules separately, and then compose

OOPSLA24

Iris-MSWasm: Elucidating and Mechanising the Security Invariants of Memory-Safe WebAssembly

MAXIME LEGOUPIL, Aarhus University, Denmark
JUNE ROUSSEAU, Aarhus University, Denmark
AÏNA LINN GEORGES, MPI-SWS, Germany
JEAN PICHON-PHARABOD, Aarhus University, Denmark
LARS BIRKEDAL, Aarhus University, Denmark

WebAssembly offers coarse-grained encapsulation guarantees via its module system, but does not support fine-grained sharing of its linear memory. MSWasm is a recent proposal which extends WebAssembly with

IrisWasm: Sample wp Rules

`wp_binop`

$$\frac{[\![t.\mathit{binop}]\!](c_1, c_2) = c \ast \triangleright \Phi(\mathbf{immV}\ [t.\mathbf{const}\ c]) \ast \xhookrightarrow{\text{FR}} F}{\text{wp}\ [t.\mathbf{const}\ c_1; t.\mathbf{const}\ c_2; t.\mathbf{binop}\ \mathit{binop}]\ \left\{w, \Phi(w) \ast \xhookrightarrow{\text{FR}} F\right\}}$$

`wp_call`

$$\frac{(F.\text{inst}.\text{funcs}[i] = \mathit{addri}) \ast \xhookrightarrow{\text{FR}} F \ast \triangleright\left(\xhookrightarrow{\text{FR}} F \mathrel{-\!\!\ast} \text{wp}\ [\mathbf{invoke}\ \mathit{addri}]\ \{w, \Phi(w)\}\right)}{\text{wp}\ [\mathbf{call}\ i]\ \{w, \Phi(w)\}}$$

`wp_invoke_native`

$$\frac{\begin{array}{c}|\mathit{vs}| = |\mathit{ts}_1| \ast \mathit{cl} = \{(\mathit{inst}; \mathit{ts}); \mathit{es}\}^{\text{NativeCl}}_{(\mathit{ts}_1 \to \mathit{ts}_2)} \ast F' = \{\text{locs} := \mathit{vs} +\!\!+\ \mathbf{zeros}(\mathit{ts}); \text{inst} := \mathit{inst}\} \ast \\ i \xmapsto{\text{wf}} \mathit{cl} \ast \xhookrightarrow{\text{FR}} F \ast \triangleright \left[\begin{array}{c}(i \xmapsto{\text{wf}} \mathit{cl} \ast \xhookrightarrow{\text{FR}} F) \mathrel{-\!\!\ast} \\ \text{wp}\ \left[\mathbf{local}_{|\mathit{ts}_2|}\{F'\}\ (\mathbf{block}\ ([] \to \mathit{ts}_2)\ \mathit{es})\ \mathbf{end}\right]\ \{w, \Phi(w)\}\end{array}\right]\end{array}}{\text{wp}\ (\mathit{vs} +\!\!+\ \mathbf{invoke}\ i)\ \{w, \Phi(w)\}}$$

`wp_local_bind`

$$\frac{\xhookrightarrow{\text{FR}} F \ast \left(\xhookrightarrow{\text{FR}} F_1 \mathrel{-\!\!\ast} \text{wp}\ \mathit{es} \left\{w, \exists F_1', \xhookrightarrow{\text{FR}} F_1' \ast \left(\xhookrightarrow{\text{FR}} F \mathrel{-\!\!\ast} \text{wp}\ [\mathbf{local}_n\{F_1'\}\ w\ \mathbf{end}]\ \{w', \Phi(w')\}\right)\right\}\right)}{\text{wp}\ [\mathbf{local}_n\{F_1\}\ \mathit{es}\ \mathbf{end}]\ \{w', \Phi(w')\}}$$

What do all of these rules have in common?

IrisWasm: Sample wp Rules

`wp_binop`

$$\frac{[\![t.\mathit{binop}]\!](c_1, c_2) = c \;*\; \triangleright\Phi(\mathbf{immV}\ [t.\mathbf{const}\ c]) \;*\; \boxed{\xhookrightarrow{\textsc{Fr}} F}}{\mathrm{wp}\ [t.\mathbf{const}\ c_1; t.\mathbf{const}\ c_2; t.\mathbf{binop}\ \mathit{binop}]\ \left\{w, \Phi(w) * \boxed{\xhookrightarrow{\textsc{Fr}} F}\right\}}$$

`wp_call`

$$\frac{(F.\mathrm{inst.funcs}[i] = \mathit{addri}) * \boxed{\xhookrightarrow{\textsc{Fr}} F} * \triangleright\left(\boxed{\xhookrightarrow{\textsc{Fr}} F} \mathrel{-\!\!*} \mathrm{wp}\ [\mathbf{invoke}\ \mathit{addri}]\ \{w, \Phi(w)\}\right)}{\mathrm{wp}\ [\mathbf{call}\ i]\ \{w, \Phi(w)\}}$$

`wp_invoke_native`

$$\frac{\begin{array}{c}|\mathit{vs}| = |\mathit{ts}_1| * \mathit{cl} = \{(\mathit{inst}; \mathit{ts}); \mathit{es}\}^{\mathsf{NativeCl}}_{(\mathit{ts}_1 \to \mathit{ts}_2)} * F' = \{\mathrm{locs} := \mathit{vs} \mathbin{+\!\!+} \mathbf{zeros}(\mathit{ts}); \mathrm{inst} := \mathit{inst}\} * \\ i \xmapsto{\mathrm{wf}} \mathit{cl} * \boxed{\xhookrightarrow{\textsc{Fr}} F} * \triangleright\left[\begin{array}{c}(i \xmapsto{\mathrm{wf}} \mathit{cl} * \boxed{\xhookrightarrow{\textsc{Fr}} F}) \mathrel{-\!\!*} \\ \mathrm{wp}\ [\mathbf{local}_{|\mathit{ts}_2|}\{F'\}\ (\mathbf{block}\ ([] \to \mathit{ts}_2)\ \mathit{es})\ \mathbf{end}]\ \{w, \Phi(w)\}\end{array}\right]\end{array}}{\mathrm{wp}\ (\mathit{vs} \mathbin{+\!\!+} \mathbf{invoke}\ i)\ \{w, \Phi(w)\}}$$

`wp_local_bind`

$$\frac{\boxed{\xhookrightarrow{\textsc{Fr}} F} * \left(\boxed{\xhookrightarrow{\textsc{Fr}} F_1} \mathrel{-\!\!*} \mathrm{wp}\ \mathit{es}\left\{w, \exists F_1', \boxed{\xhookrightarrow{\textsc{Fr}} F_1'} * \left(\boxed{\xhookrightarrow{\textsc{Fr}} F} \mathrel{-\!\!*} \mathrm{wp}\ [\mathbf{local}_n\{F_1'\}\ w\ \mathbf{end}]\ \{w', \Phi(w')\}\right)\right\}\right)}{\mathrm{wp}\ [\mathbf{local}_n\{F_1\}\ \mathit{es}\ \mathbf{end}]\ \{w', \Phi(w')\}}$$

What do all of these rules have in common?

Explicit threading of the monolithic "frame resource" $\xhookrightarrow{\textsc{Fr}} F$

Context Splitting

$$\Gamma \vdash \mathsf{e} : \mathsf{T}$$

$$\rightsquigarrow$$

$$\{\llbracket \Gamma \rrbracket\}\ \mathsf{e}^{*}\ \{\llbracket \mathsf{T} \rrbracket\}$$

Context Splitting

$$\begin{tikzcd}
\Gamma \vdash e : T \arrow[d, rightsquigarrow] & (\Gamma = \overline{x : T_x}) \\
\{F \star \cdots\}\ e^*\ \{[\![T]\!]\} & (F = \overline{\$x \mapsto v_x})
\end{tikzcd}$$

Context Splitting

$$\otimes\mathrm{I}\;\frac{\Gamma_1 \vdash e_1 : T_1 \qquad \Gamma_2 \vdash e_2 : T_2}{\Gamma_1, \Gamma_2 \vdash (e_1, e_2) : T_1 \otimes T_2}$$

$$\rightsquigarrow$$

$$\{F_1 \star F_2 \star \cdots\}\; e_1^*; e_2^*\; \{[\![T_1]\!] \star [\![T_2]\!]\}$$

Separating the Frame

Local points-to: $\$x \xmapsto{\mathsf{loc}} v$

Separating the Frame

Local points-to: $\$x \xmapsto{\mathsf{loc}} v$

Locals are exclusive: $\$x \xmapsto{\mathsf{loc}} v_1 \star \$x \xmapsto{\mathsf{loc}} v_2 \vdash \bot$

Separating the Frame

Local points-to: $\$x \xrightarrow{\mathsf{loc}} v$

Locals are exclusive: $\$x \xrightarrow{\mathsf{loc}} v_1 \star \$x \xrightarrow{\mathsf{loc}} v_2 \vdash \bot$

Rules are small footprint:

WP-PLUS

$$\frac{\Phi(\mathsf{i32.const}\,(n_1 + n_2))}{\mathsf{wp}\ \ (\mathsf{i32.const}\, n_1; \mathsf{i32.const}\, n_2; \mathsf{i32.add})\ \{\Phi\}}$$

WP-LOC-GET

$$\frac{\$x \xrightarrow{\mathsf{loc}} v \star \Phi(v)}{\mathsf{wp}\ \ (\mathsf{local.get}\, \$x)\ \{\Phi\}}$$

WP-LOC-SET

$$\frac{\$x \xrightarrow{\mathsf{loc}} v' \star (\$x \xrightarrow{\mathsf{loc}} v \mathrel{-\!\!\star} \Phi(\epsilon))}{\mathsf{wp}\ \ (v; \mathsf{local.get}\, \$x)\ \{\Phi\}}$$

Separating the Frame

Local points-to: $\$x \xmapsto{\textsf{loc}} v$

Locals are exclusive: $\$x \xmapsto{\textsf{loc}} v_1 \star \$x \xmapsto{\textsf{loc}} v_2 \vdash \bot$

Rules are small footprint:

WP-PLUS

$$\frac{\Phi(\textsf{i32.const}\,(n_1 + n_2))}{\textsf{wp}\ (\textsf{i32.const}\,n_1; \textsf{i32.const}\,n_2; \textsf{i32.add})\ \{\Phi\}}$$

WP-LOC-GET

$$\frac{\$x \xmapsto{\textsf{loc}} v \star \Phi(v)}{\textsf{wp}\ (\textsf{local.get}\,\$x)\ \{\Phi\}}$$

WP-LOC-SET

$$\frac{\$x \xmapsto{\textsf{loc}} v' \star (\$x \xmapsto{\textsf{loc}} v \mathrel{-\!\!\star} \Phi(\epsilon))}{\textsf{wp}\ (v; \textsf{local.get}\,\$x)\ \{\Phi\}}$$

WP-FRAME-BIND†

$$\frac{F \mathrel{-\!\!\star} \textsf{wp}\ (e^*)\ \{v^n.\ \Phi(v^n)\}}{\textsf{wp}\ (\textsf{frame}_n\,F\,e^*)\ \{\Phi\}}$$

Problem 1: Popped Frames Not Encapsulated

$$\frac{}{\mathsf{wp}\ \left(\mathsf{frame}_0\,(\mathsf{locals}\,(\$x\ 42))\,\epsilon\right)\ \{\$x \xmapsto{\mathsf{loc}} 42\}}$$

Local $\$x$ should not be returnable from the frame. This *should* be false.

Problem 1: Popped Frames Not Encapsulated

$$\frac{\overline{\$x \overset{\mathsf{loc}}{\longmapsto} 42 \mathrel{-\!\!*} \mathsf{wp}\ (\epsilon)\ \{\$x \overset{\mathsf{loc}}{\longmapsto} 42\}}}{\mathsf{wp}\ (\mathsf{frame}_0\ (\mathsf{locals}\,(\$x\ 42))\,\epsilon)\ \{\$x \overset{\mathsf{loc}}{\longmapsto} 42\}}\ \text{WP-FRAME-BIND}\dagger$$

Local $\$x$ should not be returnable from the frame. This *should* be false.

Problem 1: Popped Frames Not Encapsulated

$$\dfrac{\dfrac{\$x \overset{\text{loc}}{\longmapsto} 42 \mathrel{-\!\!*} \$x \overset{\text{loc}}{\longmapsto} 42}{\$x \overset{\text{loc}}{\longmapsto} 42 \mathrel{-\!\!*} \text{wp}\ (\epsilon)\ \{\$x \overset{\text{loc}}{\longmapsto} 42\}}\ \text{WP-VALS}}{\text{wp}\ (\text{frame}_0\,(\text{locals}\,(\$x\ 42))\,\epsilon)\ \{\$x \overset{\text{loc}}{\longmapsto} 42\}}\ \text{WP-FRAME-BIND}\dagger$$

Local $\$x$ should not be returnable from the frame. This *should* be false.

Encapsulating Frames with the Pop Modality

The Pop Modality, $\downarrow P$

$\downarrow P$ holds whenever P would hold after popping the top frame.

Encapsulating Frames with the Pop Modality

The Pop Modality, $\downarrow P$

$\downarrow P$ holds whenever P would hold after popping the top frame.

Global facts always hold after a pop: $n \xmapsto{\text{mem}} v \vdash \downarrow (n \xmapsto{\text{mem}} v)$

Encapsulating Frames with the Pop Modality

The Pop Modality, $\downarrow P$

$\downarrow P$ holds whenever P would hold after popping the top frame.

Global facts always hold after a pop: $n \xmapsto{\text{mem}} v \vdash \downarrow (n \xmapsto{\text{mem}} v)$

Frame-local facts never hold after a pop: $\$x \xmapsto{\text{loc}} v \nvdash \downarrow (\$x \xmapsto{\text{loc}} v)$

Encapsulating Frames with the Pop Modality

The Pop Modality, $\downarrow P$

$\downarrow P$ holds whenever P would hold after popping the top frame.

Global facts always hold after a pop: $n \xmapsto{\text{mem}} v \vdash \downarrow (n \xmapsto{\text{mem}} v)$

Frame-local facts never hold after a pop: $\$x \xmapsto{\text{loc}} v \nvdash \downarrow (\$x \xmapsto{\text{loc}} v)$

Binding under a frame moves the postcondition under a pop:

WP-FRAME-BIND

$$\frac{F \mathrel{-\!\!*} \mathsf{wp}\ (e^*)\ \{v^n.\ \downarrow \Phi(v^n)\}}{\mathsf{wp}\ (\mathsf{frame}_n\ F\ e^*)\ \{\Phi\}}$$

Problem 2: Suspended Frames Not Encapsulated

$$\frac{}{\$x \overset{\mathsf{loc}}{\longmapsto} 42 \vdash \mathsf{wp}\ \ (\mathsf{frame}_1\,(\mathsf{locals})\,\mathsf{local.get}\,\$x)\ \{v.\ v = \mathsf{i32.const}\,42\}}$$

Local $\$x$ should not be accessible within the frame. This *should* be false.

Problem 2: Suspended Frames Not Encapsulated

$$\cfrac{\cfrac{}{\$x \xmapsto{\text{loc}} 42 \vdash \mathsf{wp}\ (\mathsf{local.get}\,\$x)\ \{v.\ \downarrow(v = \mathsf{i32.const}\,42)\}}}{\$x \xmapsto{\text{loc}} 42 \vdash \mathsf{wp}\ (\mathsf{frame}_1\ (\mathsf{locals})\ \mathsf{local.get}\,\$x)\ \{v.\ v = \mathsf{i32.const}\,42\}}\ \text{WP-FRAME-BIND}\dagger\dagger$$

Local $\$x$ should not be accessible within the frame. This *should* be false.

Problem 2: Suspended Frames Not Encapsulated

$$\dfrac{\dfrac{\dfrac{}{\$x \overset{\mathsf{loc}}{\longmapsto} 42 \vdash \downarrow(\mathsf{i32.const}\,42 = \mathsf{i32.const}\,42)}\ \downarrow\text{-PURE, REFL}}{\$x \overset{\mathsf{loc}}{\longmapsto} 42 \vdash \mathsf{wp}\ (\mathsf{local.get}\,\$x)\ \{v.\,\downarrow(v = \mathsf{i32.const}\,42)\}}\ \text{WP-LOCAL-GET}}{\$x \overset{\mathsf{loc}}{\longmapsto} 42 \vdash \mathsf{wp}\ (\mathsf{frame}_1\ (\mathsf{locals})\ \mathsf{local.get}\,\$x)\ \{v.\ v = \mathsf{i32.const}\,42\}}\ \text{WP-FRAME-BIND††}$$

Local $\$x$ should not be accessible within the frame. This *should* be false.

Encapsulating Frames with the Push Modality

The Push Modality, $\uparrow P$

$\uparrow P$ holds whenever P would hold after pushing a new frame.

Encapsulating Frames with the Push Modality

The Push Modality, $\uparrow P$

$\uparrow P$ holds whenever P would hold after pushing a new frame.

Global facts always hold after a push: $n \xmapsto{\text{mem}} v \vdash \uparrow (n \xmapsto{\text{mem}} v)$

Encapsulating Frames with the Push Modality

The Push Modality, $\uparrow P$

$\uparrow P$ holds whenever P would hold after pushing a new frame.

Global facts always hold after a push: $n \xmapsto{\textsf{mem}} v \vdash \uparrow (n \xmapsto{\textsf{mem}} v)$

Frame-local facts never hold after a push: $\$x \xmapsto{\textsf{loc}} v \nvdash \uparrow (\$x \xmapsto{\textsf{loc}} v)$

Encapsulating Frames with the Push Modality

The Push Modality, $\uparrow P$

$\uparrow P$ holds whenever P would hold after pushing a new frame.

Global facts always hold after a push: $n \xmapsto{\textsf{mem}} v \vdash \uparrow (n \xmapsto{\textsf{mem}} v)$

Frame-local facts never hold after a push: $\$x \xmapsto{\textsf{loc}} v \nvdash \uparrow (\$x \xmapsto{\textsf{loc}} v)$

Binding under a frame moves the continuation under a push:

WP-FRAME-BIND

$$\frac{\uparrow (F \mathrel{-\!\!*} \mathsf{wp}\ (e^*)\ \{v^n. \downarrow \Phi(v^n)\})}{\mathsf{wp}\ (\mathsf{frame}_n\ F\ e^*)\ \{\Phi\}}$$

Adjoint Logic

Benton and Wadler (1996), Reed (2009)

Adjoint Logic

Benton and Wadler (1996), Reed (2009)

Push and pop are both modalities:

$$\frac{P \vdash Q}{\uparrow P \vdash \uparrow Q}\ \uparrow\text{-}\textsc{Mono} \qquad \frac{P \vdash Q}{\downarrow P \vdash \downarrow Q}\ \downarrow\text{-}\textsc{Mono}$$

Adjoint Logic

Benton and Wadler (1996), Reed (2009)

Push and pop are both modalities:

$$\frac{P \vdash Q}{\uparrow P \vdash \uparrow Q}\ \uparrow\text{-}\textsc{Mono} \qquad \frac{P \vdash Q}{\downarrow P \vdash \downarrow Q}\ \downarrow\text{-}\textsc{Mono}$$

They form an adjunction:

$$\frac{\downarrow P \vdash Q}{\overline{P \vdash \uparrow Q}}\ \updownarrow\text{-}\textsc{Adj}$$

Adjoint Logic

Benton and Wadler (1996), Reed (2009)

Push and pop are both modalities:

$$\uparrow\text{-}\textsc{Mono}\quad \frac{P \vdash Q}{\uparrow P \vdash \uparrow Q} \qquad\qquad \downarrow\text{-}\textsc{Mono}\quad \frac{P \vdash Q}{\downarrow P \vdash \downarrow Q}$$

They form an adjunction:

$$\updownarrow\text{-}\textsc{Adj}\quad \begin{array}{c} \downarrow P \vdash Q \\ \hline\hline P \vdash \uparrow Q \end{array} \qquad\qquad \uparrow\downarrow\text{-}\textsc{Unit}\quad P \vdash \uparrow\downarrow P \qquad\qquad \downarrow\uparrow\text{-}\textsc{Counit}\quad \downarrow\uparrow P \vdash P$$

Example: Suspending and Resuming Frames

$\$x \xmapsto{\mathsf{loc}} 20 \vdash \mathsf{wp}\ ((\mathsf{frame}_1\ (\mathsf{locals})\ \mathsf{i32.const}\, 25)\, ; \mathsf{local.get}\, \$x ; \mathsf{i32.add})\ \{v.\ v = 45\}$

Example: Suspending and Resuming Frames

$$\text{(WP-CTX-BIND)}\ \frac{\$x \overset{\text{loc}}{\longmapsto} 20 \vdash \text{wp}\ (\text{frame}_1\ (\text{locals})\ \text{i32.const}\ 25)\ \{v'.\ \text{wp}\ (v'; \text{local.get}\ \$x; \text{i32.add})\ \{v.v = 45\}\}}{\$x \overset{\text{loc}}{\longmapsto} 20 \vdash \text{wp}\ ((\text{frame}_1\ (\text{locals})\ \text{i32.const}\ 25)\ ; \text{local.get}\ \$x; \text{i32.add})\ \{v.\ v = 45\}}$$

Example: Suspending and Resuming Frames

$$\dfrac{\dfrac{\$x \overset{\text{loc}}{\longmapsto} 20 \vdash {\uparrow}\,\mathsf{wp}\ (\mathsf{i32.const}\ 25)\ \{v'.\ {\downarrow}\,\mathsf{wp}\ (v'; \mathsf{local.get}\ \$x; \mathsf{i32.add})\ \{v.\ v = 45\}\}}{\$x \overset{\text{loc}}{\longmapsto} 20 \vdash \mathsf{wp}\ (\mathsf{frame}_1\ (\mathsf{locals})\ \mathsf{i32.const}\ 25)\ \{v'.\ \mathsf{wp}\ (v'; \mathsf{local.get}\ \$x; \mathsf{i32.add})\ \{v.v = 45\}\}}\ \text{(WP-FRAME-BIND)}}{\$x \overset{\text{loc}}{\longmapsto} 20 \vdash \mathsf{wp}\ ((\mathsf{frame}_1\ (\mathsf{locals})\ \mathsf{i32.const}\ 25)\ ; \mathsf{local.get}\ \$x; \mathsf{i32.add})\ \{v.\ v = 45\}}\ \text{(WP-CTX-BIND)}$$

Example: Suspending and Resuming Frames

$$\dfrac{\dfrac{\dfrac{\downarrow(\$x \overset{\text{loc}}{\longmapsto} 20) \vdash \text{wp }(\text{i32.const}\,25)\ \{v'.\ \downarrow\text{wp }(v';\text{local.get}\,\$x;\text{i32.add})\ \{v.\ v = 45\}\}}{\$x \overset{\text{loc}}{\longmapsto} 20 \vdash \uparrow\text{wp }(\text{i32.const}\,25)\ \{v'.\ \downarrow\text{wp }(v';\text{local.get}\,\$x;\text{i32.add})\ \{v.\ v = 45\}\}}\,(\updownarrow\text{-ADJ})}{\$x \overset{\text{loc}}{\longmapsto} 20 \vdash \text{wp }(\text{frame}_1\ (\text{locals})\ \text{i32.const}\,25)\ \{v'.\ \text{wp }(v';\text{local.get}\,\$x;\text{i32.add})\ \{v.v = 45\}\}}\,(\text{WP-FRAME-BIND})}{\$x \overset{\text{loc}}{\longmapsto} 20 \vdash \text{wp }((\text{frame}_1\ (\text{locals})\ \text{i32.const}\,25)\,;\text{local.get}\,\$x;\text{i32.add})\ \{v.\ v = 45\}}\,(\text{WP-CTX-BIND})$$

Example: Suspending and Resuming Frames

$$\downarrow(\$x \xmapsto{\mathsf{loc}} 20) \vdash \downarrow \mathsf{wp}\ (\mathsf{i32.const}\ 25; \mathsf{local.get}\ \$x; \mathsf{i32.add})\ \{v.\ v = 45\}$$

(WP-VAL)

$$\downarrow(\$x \xmapsto{\mathsf{loc}} 20) \vdash \mathsf{wp}\ (\mathsf{i32.const}\ 25)\ \{v'.\ \downarrow \mathsf{wp}\ (v'; \mathsf{local.get}\ \$x; \mathsf{i32.add})\ \{v.\ v = 45\}\}$$

($\updownarrow$ -ADJ)

$$\$x \xmapsto{\mathsf{loc}} 20 \vdash \uparrow \mathsf{wp}\ (\mathsf{i32.const}\ 25)\ \{v'.\ \downarrow \mathsf{wp}\ (v'; \mathsf{local.get}\ \$x; \mathsf{i32.add})\ \{v.\ v = 45\}\}$$

(WP-FRAME-BIND)

$$\$x \xmapsto{\mathsf{loc}} 20 \vdash \mathsf{wp}\ (\mathsf{frame}_1\ (\mathsf{locals})\ \mathsf{i32.const}\ 25)\ \{v'.\ \mathsf{wp}\ (v'; \mathsf{local.get}\ \$x; \mathsf{i32.add})\ \{v.v = 45\}\}$$

(WP-CTX-BIND)

$$\$x \xmapsto{\mathsf{loc}} 20 \vdash \mathsf{wp}\ ((\mathsf{frame}_1\ (\mathsf{locals})\ \mathsf{i32.const}\ 25)\ ; \mathsf{local.get}\ \$x; \mathsf{i32.add})\ \{v.\ v = 45\}$$

Example: Suspending and Resuming Frames

$$\$x \overset{\mathsf{loc}}{\longmapsto} 20 \vdash \mathsf{wp}\ (\mathsf{i32.const}\ 25; \mathsf{local.get}\ \$x; \mathsf{i32.add})\ \{v.\ v = 45\}$$

(↓ -Mono)

$$\downarrow (\$x \overset{\mathsf{loc}}{\longmapsto} 20) \vdash {\downarrow} \mathsf{wp}\ (\mathsf{i32.const}\ 25; \mathsf{local.get}\ \$x; \mathsf{i32.add})\ \{v.\ v = 45\}$$

(WP-VAL)

$$\downarrow (\$x \overset{\mathsf{loc}}{\longmapsto} 20) \vdash \mathsf{wp}\ (\mathsf{i32.const}\ 25)\ \{v'.\ {\downarrow} \mathsf{wp}\ (v'; \mathsf{local.get}\ \$x; \mathsf{i32.add})\ \{v.\ v = 45\}\}$$

(↕ -Adj)

$$\$x \overset{\mathsf{loc}}{\longmapsto} 20 \vdash {\uparrow} \mathsf{wp}\ (\mathsf{i32.const}\ 25)\ \{v'.\ {\downarrow} \mathsf{wp}\ (v'; \mathsf{local.get}\ \$x; \mathsf{i32.add})\ \{v.\ v = 45\}\}$$

(WP-FRAME-BIND)

$$\$x \overset{\mathsf{loc}}{\longmapsto} 20 \vdash \mathsf{wp}\ (\mathsf{frame}_1\ (\mathsf{locals})\ \mathsf{i32.const}\ 25)\ \{v'.\ \mathsf{wp}\ (v'; \mathsf{local.get}\ \$x; \mathsf{i32.add})\ \{v.v = 45\}\}$$

(WP-CTX-BIND)

$$\$x \overset{\mathsf{loc}}{\longmapsto} 20 \vdash \mathsf{wp}\ ((\mathsf{frame}_1\ (\mathsf{locals})\ \mathsf{i32.const}\ 25)\ ; \mathsf{local.get}\ \$x; \mathsf{i32.add})\ \{v.\ v = 45\}$$

Example: Suspending and Resuming Frames

(WP-CTX-BIND, WP-LOCAL-GET, WP-BINOP, WP-VAL)

$$\$x \overset{\text{loc}}{\longmapsto} 20 \vdash \text{wp}\ (\text{i32.const}\, 25; \text{local.get}\, \$x; \text{i32.add})\ \{v.\ v = 45\}$$

(↓ -Mono)

$$\downarrow (\$x \overset{\text{loc}}{\longmapsto} 20) \vdash \downarrow \text{wp}\ (\text{i32.const}\, 25; \text{local.get}\, \$x; \text{i32.add})\ \{v.\ v = 45\}$$

(WP-VAL)

$$\downarrow (\$x \overset{\text{loc}}{\longmapsto} 20) \vdash \text{wp}\ (\text{i32.const}\, 25)\ \{v'.\ \downarrow \text{wp}\ (v'; \text{local.get}\, \$x; \text{i32.add})\ \{v.\ v = 45\}\}$$

(↕ -Adj)

$$\$x \overset{\text{loc}}{\longmapsto} 20 \vdash \uparrow \text{wp}\ (\text{i32.const}\, 25)\ \{v'.\ \downarrow \text{wp}\ (v'; \text{local.get}\, \$x; \text{i32.add})\ \{v.\ v = 45\}\}$$

(WP-FRAME-BIND)

$$\$x \overset{\text{loc}}{\longmapsto} 20 \vdash \text{wp}\ (\text{frame}_1\ (\text{locals})\ \text{i32.const}\, 25)\ \{v'.\ \text{wp}\ (v'; \text{local.get}\, \$x; \text{i32.add})\ \{v.v = 45\}\}$$

(WP-CTX-BIND)

$$\$x \overset{\text{loc}}{\longmapsto} 20 \vdash \text{wp}\ ((\text{frame}_1\ (\text{locals})\ \text{i32.const}\, 25)\, ; \text{local.get}\, \$x; \text{i32.add})\ \{v.\ v = 45\}$$

Other Stacks

Other Stacks

A shadow stack

WP-SPUSH

$$\frac{\uparrow^{s} \Phi(\epsilon)}{\mathsf{wp}\ (\$\mathsf{spush})\ \{\Phi\}}$$

WP-SPOP

$$\frac{\downarrow_{s} \Phi(\epsilon)}{\mathsf{wp}\ (\$\mathsf{spop})\ \{\Phi\}}$$

WP-SALLOC

$$\frac{\forall n', b^{n}.\ n' \xmapsto{\mathsf{stk}} b^{n} \mathrel{-\!\!\star} \Phi(\mathsf{i32.const}\, n')}{\mathsf{wp}\ (\$\mathsf{salloc}\, n)\ \{\Phi\}}$$

Other Stacks

A shadow stack

$$\textsc{wp-spush}\ \frac{\uparrow^{s} \Phi(\epsilon)}{\mathsf{wp}\ (\$\mathsf{spush})\ \{\Phi\}} \qquad \textsc{wp-spop}\ \frac{\downarrow_{s} \Phi(\epsilon)}{\mathsf{wp}\ (\$\mathsf{spop})\ \{\Phi\}} \qquad \textsc{wp-salloc}\ \frac{\forall n', b^{n}.\ n' \xmapsto{\mathsf{stk}} b^{n} \mathrel{-\!\!\star} \Phi(\mathsf{i32.const}\, n')}{\mathsf{wp}\ (\$\mathsf{salloc}\, n)\ \{\Phi\}}$$

The operand stack?

$$\mathsf{top}(20) \star \downarrow_{o} \mathsf{top}(25) \star \downarrow_{o} \downarrow_{o} \mathsf{top}(10) \approx \mathsf{i32.const}\ 10; v; \mathsf{i32.const}\ 25; \mathsf{i32.const}\ 20$$

Other Stacks

A shadow stack

WP-SPUSH

$$\frac{\uparrow^{s} \Phi(\epsilon)}{\mathsf{wp}\ (\$\mathsf{spush})\ \{\Phi\}}$$

WP-SPOP

$$\frac{\downarrow_{s} \Phi(\epsilon)}{\mathsf{wp}\ (\$\mathsf{spop})\ \{\Phi\}}$$

WP-SALLOC

$$\frac{\forall n', b^{n}.\ n' \xmapsto{\mathsf{stk}} b^{n} \mathrel{-\!\!\star} \Phi(\mathsf{i32.const}\, n')}{\mathsf{wp}\ (\$\mathsf{salloc}\, n)\ \{\Phi\}}$$

The operand stack?

$$\mathsf{top}(20) \star {\downarrow_o} \mathsf{top}(25) \star {\downarrow_o} {\downarrow_o} \mathsf{top}(10) \approx \mathsf{i32.const}\ 10; v; \mathsf{i32.const}\ 25; \mathsf{i32.const}\ 20$$

Stack switching?

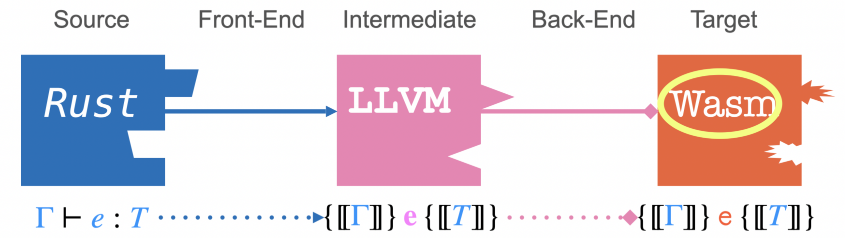
Summary

WP-FRAME-BIND

$$\frac{\uparrow (F \mathrel{-\!\!\star} \mathsf{wp}\ (e^*)\ \{v^n.\ \downarrow \Phi(v^n)\})}{\mathsf{wp}\ (\mathsf{frame}_n\ F\ e^*)\ \{\Phi\}}$$

$\updownarrow$-ADJ

$$\begin{array}{c} \downarrow P \vdash Q \\ \hline\hline P \vdash \uparrow Q \end{array}$$



Source

Front-End

Intermediate

Back-End

Target

Rust

LLVM

Wasm

$\Gamma \vdash e : T$

$\{[\![\Gamma]\!]\}\ e\ \{[\![T]\!]\}$

$\{[\![\Gamma]\!]\}\ e\ \{[\![T]\!]\}$