Semantic Typing and Separation Logic

A Tutorial

Andrew Wagner

February 21, 2024

Northeastern University

A Short History of Type Soundness

A Denotational Semantic Approach to Type Soundess [Mil78]

1. Specify the syntax and type system.

A Denotational Semantic Approach to Type Soundess [Mil78]

1. Specify the syntax and type system.
2. Assign each expression e a mathematical object $[\![e]\!]$ 🔮 , using a distinguished object WRONG 😈 when the expression is nonsense.

A Denotational Semantic Approach to Type Soundess [Mil78]

1. Specify the syntax and type system.
2. Assign each expression e a mathematical object $[\![e]\!]$ 🔮 , using a distinguished object WRONG 😈 when the expression is nonsense.
3. Assign each type T a set of mathematical objects $[\![T]\!]$🔮 .

A Denotational Semantic Approach to Type Soundess [Mil78]

1. Specify the syntax and type system.
2. Assign each expression e a mathematical object $[\![e]\!]$ 🔮, using a distinguished object WRONG 😈 when the expression is nonsense.
3. Assign each type T a set of mathematical objects $[\![T]\!]$🔮.
4. Ensure WRONG 😈 is not in any $[\![T]\!]$.

A Denotational Semantic Approach to Type Soundess [Mil78]

1. Specify the syntax and type system.
2. Assign each expression e a mathematical object $[\![e]\!]$ 🔮, using a distinguished object WRONG 😈 when the expression is nonsense.
3. Assign each type T a set of mathematical objects $[\![T]\!]$ 🔮.
4. Ensure WRONG 😈 is not in any $[\![T]\!]$.
5. Show that if $e : T$ then $[\![e]\!] \in [\![T]\!]$.

A Denotational Semantic Approach to Type Soundess [Mil78]

1. Specify the syntax and type system.
2. Assign each expression e a mathematical object $\llbracket e \rrbracket$ 🔮 , using a distinguished object WRONG 😈 when the expression is nonsense.
3. Assign each type T a set of mathematical objects $\llbracket T \rrbracket$🔮 .
4. Ensure WRONG 😈 is not in any $\llbracket T \rrbracket$.
5. Show that if $e : T$ then $\llbracket e \rrbracket \in \llbracket T \rrbracket$.
6. **Theorem (Type Soundness):** If $e : T$ then $\llbracket e \rrbracket$ is not WRONG.

A Denotational Semantic Approach to Type Soundess [Mil78]

1. Specify the syntax and type system.
2. Assign each expression e a mathematical object $[\![e]\!]$ 🔮, using a distinguished object WRONG 😈 when the expression is nonsense.
3. Assign each type T a set of mathematical objects $[\![T]\!]$🔮.
4. Ensure WRONG 😈 is not in any $[\![T]\!]$.
5. Show that if $e : T$ then $[\![e]\!] \in [\![T]\!]$.
6. **Theorem (Type Soundness):** If $e : T$ then $[\![e]\!]$ is not WRONG.

- Statically ill-typed programs can be proved safe

A Denotational Semantic Approach to Type Soundess [Mil78]

1. Specify the syntax and type system.
2. Assign each expression e a mathematical object $[\![e]\!]$ 🔮, using a distinguished object WRONG 😈 when the expression is nonsense.
3. Assign each type T a set of mathematical objects $[\![T]\!]$ 🔮.
4. Ensure WRONG 😈 is not in any $[\![T]\!]$.
5. Show that if $e : T$ then $[\![e]\!] \in [\![T]\!]$.
6. **Theorem (Type Soundness):** If $e : T$ then $[\![e]\!]$ is not WRONG.

- Statically ill-typed programs can be proved safe
- Denotational semantics for real languages is hard 😮‍💨

A Syntactic Approach to Type Soundness [WF94]

1. Specify the syntax and type system.

A Syntactic Approach to Type Soundness [WF94]

1. Specify the syntax and type system.
2. Specify the (small-step) operational semantics.

A Syntactic Approach to Type Soundness [WF94]

1. Specify the syntax and type system.
2. Specify the (small-step) operational semantics.
3. **Lemma (Progress):** If $e : T$ then either e is a value or e can take a step.

A Syntactic Approach to Type Soundness [WF94]

1. Specify the syntax and type system.
2. Specify the (small-step) operational semantics.
3. **Lemma (Progress):** If $e : T$ then either e is a value or e can take a step.
4. **Lemma (Preservation):** If $e : T$ and $e \to e'$ then $e' : T$.

A Syntactic Approach to Type Soundness [WF94]

1. Specify the syntax and type system.
2. Specify the (small-step) operational semantics.
3. **Lemma (Progress):** If $e : T$ then either e is a value or e can take a step.
4. **Lemma (Preservation):** If $e : T$ and $e \to e'$ then $e' : T$.
5. **Theorem (Type Soundness):** If $e : T$ then e does not go wrong (e.g., get stuck or error).

A Syntactic Approach to Type Soundness [WF94]

1. Specify the syntax and type system.
2. Specify the (small-step) operational semantics.
3. **Lemma (Progress):** If $e : T$ then either e is a value or e can take a step.
4. **Lemma (Preservation):** If $e : T$ and $e \to e'$ then $e' : T$.
5. **Theorem (Type Soundness):** If $e : T$ then e does not go wrong (e.g., get stuck or error).

- Just induction 🔥 Scales extremely well

A Syntactic Approach to Type Soundness [WF94]

1. Specify the syntax and type system.
2. Specify the (small-step) operational semantics.
3. **Lemma (Progress):** If $e : T$ then either e is a value or e can take a step.
4. **Lemma (Preservation):** If $e : T$ and $e \to e'$ then $e' : T$.
5. **Theorem (Type Soundness):** If $e : T$ then e does not go wrong (e.g., get stuck or error).

- Just induction 🔥 Scales extremely well
- Only applies to statically well-typed programs

An Operational Semantic Approach to Type Soundness [AF00; AM01; AAV02]

1. Specify the syntax and type system.

An Operational Semantic Approach to Type Soundness [AF00; AM01; AAV02]

1. Specify the syntax and type system.
2. Specify the (small-step) operational semantics.

An Operational Semantic Approach to Type Soundness [AF00; AM01; AAV02]

1. Specify the syntax and type system.
2. Specify the (small-step) operational semantics.
3. Assign each type T a set of expressions $[\![T]\!]$.

An Operational Semantic Approach to Type Soundness [AF00; AM01; AAV02]

1. Specify the syntax and type system.
2. Specify the (small-step) operational semantics.
3. Assign each type T a set of expressions $[\![T]\!]$.
4. **Lemma (Adequacy):** If $e \in [\![T]\!]$ then e is safe to run.

An Operational Semantic Approach to Type Soundness [AF00; AM01; AAV02]

1. Specify the syntax and type system.
2. Specify the (small-step) operational semantics.
3. Assign each type T a set of expressions $[\![T]\!]$.
4. **Lemma (Adequacy):** If $e \in [\![T]\!]$ then e is safe to run.
5. **Lemma (Fundamental Property):** If $e : T$ then $e \in [\![T]\!]$.

An Operational Semantic Approach to Type Soundness [AF00; AM01; AAV02]

1. Specify the syntax and type system.
2. Specify the (small-step) operational semantics.
3. Assign each type T a set of expressions $[\![T]\!]$.
4. **Lemma (Adequacy):** If $e \in [\![T]\!]$ then e is safe to run.
5. **Lemma (Fundamental Property):** If $e : T$ then $e \in [\![T]\!]$.
6. **Theorem (Type Soundness):** If $e : T$ then e is safe to run.

Simple Semantic Typing

Syntax and Statics

$\boxed{\Gamma \vdash e : T}$ Expr. e has type T under context Γ

$$\textsc{Id}\ \frac{\Gamma \ni x : T}{\Gamma \vdash x : T} \qquad \text{Unit-I}\ \ \Gamma \vdash () : \mathrm{Unit} \qquad \to\text{-I}\ \frac{\Gamma, x : T_1 \vdash e : T_2}{\Gamma \vdash \lambda x.e : T_1 \to T_2} \qquad \to\text{-E}\ \frac{\Gamma \vdash e_1 : T_1 \qquad \Gamma \vdash e_2 : T_1 \to T_2}{\Gamma \vdash e_2\ e_1 : T_2}$$

$\boxed{e \to e'}$ Expr. e reduces to expr. e'

$$\to\text{-LEFT}\ \frac{e_1 \to e_1'}{e_1\ e_2 \to e_1'\ e_2} \qquad \to\text{-RIGHT}\ \frac{e \to e'}{v\ e \to v\ e'} \qquad \to\text{-RED}\ \ (\lambda x.e)v \to e[x \mapsto v]$$

Semantic Types

Semantic Types

$$v \quad \in \quad \mathcal{V}\,[\![\mathrm{Unit}]\!] \qquad \text{iff} \quad v = ()$$

Semantic Types

$$\begin{array}{lcll} v & \in & \mathcal{V}[\![\text{Unit}]\!] & \text{iff} \quad v = () \\ v_2 & \in & \mathcal{V}[\![T_1 \to T_2]\!] & \text{iff} \quad v_1 \in \mathcal{V}[\![T_1]\!] \ \text{implies} \ (v_2\ v_1) \in \mathcal{E}[\![T_2]\!] \end{array}$$

Semantic Types

$$
\begin{array}{lcll}
v & \in & \mathcal{V}\,[\![\text{Unit}]\!] & \text{iff} \quad v = () \\
v_2 & \in & \mathcal{V}\,[\![T_1 \to T_2]\!] & \text{iff} \quad v_1 \in \mathcal{V}\,[\![T_1]\!] \;\text{ implies } (v_2\; v_1) \in \mathcal{E}\,[\![T_2]\!] \\
e & \in & \mathcal{E}\,[\![T]\!] & \text{iff} \quad e \to^* v \text{ for some } v \in \mathcal{V}\,[\![T]\!]
\end{array}
$$

Semantic Types

$$
\begin{array}{llll}
v & \in & \mathcal{V}[\![\text{Unit}]\!] & \text{iff} \quad v = () \\
v_2 & \in & \mathcal{V}[\![T_1 \to T_2]\!] & \text{iff} \quad v_1 \in \mathcal{V}[\![T_1]\!] \text{ implies } (v_2\ v_1) \in \mathcal{E}[\![T_2]\!] \\
e & \in & \mathcal{E}[\![T]\!] & \text{iff} \quad e \to^* v \text{ for some } v \in \mathcal{V}[\![T]\!] \\
\sigma & \in & \mathcal{S}[\![\Gamma]\!] & \text{iff} \quad x : T \in \Gamma \text{ implies } \sigma(x) \in \mathcal{V}[\![T]\!]
\end{array}
$$

Semantic Types

$$\begin{array}{lcll}
v & \in & \mathcal{V}[\![\text{Unit}]\!] & \text{iff} \quad v = () \\
v_2 & \in & \mathcal{V}[\![T_1 \rightarrow T_2]\!] & \text{iff} \quad v_1 \in \mathcal{V}[\![T_1]\!] \text{ implies } (v_2\ v_1) \in \mathcal{E}[\![T_2]\!] \\
e & \in & \mathcal{E}[\![T]\!] & \text{iff} \quad e \rightarrow^* v \text{ for some } v \in \mathcal{V}[\![T]\!] \\
\sigma & \in & \mathcal{S}[\![\Gamma]\!] & \text{iff} \quad x : T \in \Gamma \text{ implies } \sigma(x) \in \mathcal{V}[\![T]\!] \\
 & & \Gamma \vDash e : T & \text{iff} \quad \sigma \in \mathcal{S}[\![\Gamma]\!] \text{ implies } e[\sigma] \in \mathcal{E}[\![T]\!]
\end{array}$$

Funamental Property

Lemma (Fundamental Property)

If $\Gamma \vdash e : T$ *then* $\Gamma \vDash e : T$.

Funamental Property

Lemma (Fundamental Property)

If $\Gamma \vdash e : T$ *then* $\Gamma \vDash e : T$.

$$\frac{\Gamma \ni x : T}{\Gamma \vDash x : T}\ \textsc{id}$$

$$\Gamma \vDash () : \text{Unit}\ \ \text{Unit-I-}\textsc{compat}$$

$$\frac{\Gamma, x : T_1 \vDash e : T_2}{\Gamma \vDash \lambda x.e : T_1 \to T_2}\ \to\text{-I-}\textsc{compat}$$

$$\frac{\Gamma \vDash e_1 : T_1 \qquad \Gamma \vDash e_2 : T_1 \to T_2}{\Gamma \vDash e_2\ e_1 : T_2}\ \to\text{-E-}\textsc{compat}$$

Decisions, Decisions

$$e \in \mathcal{E}[\![T]\!] \quad \text{iff} \quad e \to^* v \text{ for some } v \in \mathcal{V}[\![T]\!]$$

It *will* terminate at a value

Decisions, Decisions

$e \in \mathcal{E}[\![T]\!]$ iff $e \rightarrow^* v$ for some $v \in \mathcal{V}[\![T]\!]$

It *will* terminate at a value

vs.

$e \in \mathcal{E}[\![T]\!]$ iff $e \rightarrow^* e' \not\rightarrow$ implies e' is a value and $e' \in \mathcal{V}[\![T]\!]$

If it stops running, it will be at a value

Typing "Unsafe" Code

$$\text{loop} \triangleq \underbrace{(\lambda x.x\ x)(\lambda x.x\ x)}_{\text{Not statically typable in STLC!}}$$

Lemma (loop is Loopy)

- $\text{loop} \to \text{loop}$
- $\text{loop} \to^* e$ *implies* $e \to \text{loop}$

Typing "Unsafe" Code

$$\text{loop} \triangleq \underbrace{(\lambda x.x\ x)(\lambda x.x\ x)}_{\text{Not statically typable in STLC!}}$$

Lemma (loop is Loopy)

- $\text{loop} \longrightarrow \text{loop}$
- $\text{loop} \longrightarrow^* e$ *implies* $e \longrightarrow \text{loop}$

Lemma (loop Considered Safe 😱)

$\vDash \text{loop} : T$ *for any* T

Typing fix

$$\begin{aligned} \text{fix} &\triangleq \text{fix}' \ \text{fix}' \\ \text{fix}' &\triangleq \lambda s.\lambda f.f \ (\lambda d.s \ s \ f \ d) \end{aligned}$$

Not statically typable in STLC

Lemma (fix Unrolls)

$\text{fix}\ f \rightarrow^{+} f\ (\lambda d.\text{fix}\ f\ d)$

Typing fix

$$\text{fix} \triangleq \text{fix}'\ \text{fix}'$$
$$\text{fix}' \triangleq \lambda s.\lambda f.f\ (\lambda d.s\ s\ f\ d)$$

Not statically typable in STLC

Lemma (fix Unrolls)

$\text{fix}\ f \to^{+} f\ (\lambda d.\text{fix}\ f\ d)$

Lemma (fix is Semantically Well-Typed 😱)

$\vDash \text{fix} : ((\text{Unit} \to T) \to \text{Unit} \to T) \to \text{Unit} \to T$

A Proof Attempt

$$\text{fix} \in \mathcal{V}[\![((\text{Unit} \to T) \to \text{Unit} \to T) \to \text{Unit} \to T]\!]$$

A Proof Attempt

$$
\begin{array}{ll}
 & \mathsf{fix} \in \mathcal{V}\,[\![((\mathsf{Unit} \to T) \to \mathsf{Unit} \to T) \to \mathsf{Unit} \to T]\!] \\
\text{if} & \mathsf{fix}\,f \in \mathcal{E}\,[\![\mathsf{Unit} \to T]\!] \text{ for all } f \in \mathcal{V}\,[\![(\mathsf{Unit} \to T) \to \mathsf{Unit} \to T]\!]
\end{array}
$$

A Proof Attempt

$$
\begin{array}{ll}
 & \mathsf{fix} \in \mathcal{V}[\![((\mathsf{Unit} \to T) \to \mathsf{Unit} \to T) \to \mathsf{Unit} \to T]\!] \\
\text{if} & \mathsf{fix}\, f \in \mathcal{E}[\![\mathsf{Unit} \to T]\!] \text{ for all } f \in \mathcal{V}[\![(\mathsf{Unit} \to T) \to \mathsf{Unit} \to T]\!] \\
\text{if} & f\ (\lambda d.\mathsf{fix}\, f\ d) \in \mathcal{E}[\![\mathsf{Unit} \to T]\!] \ \text{by reduction}
\end{array}
$$

A Proof Attempt

$$
\begin{array}{ll}
 & \text{fix} \in \mathcal{V}[\![((\text{Unit} \to T) \to \text{Unit} \to T) \to \text{Unit} \to T]\!] \\
\text{if} & \text{fix}\, f \in \mathcal{E}[\![\text{Unit} \to T]\!] \text{ for all } f \in \mathcal{V}[\![(\text{Unit} \to T) \to \text{Unit} \to T]\!] \\
\text{if} & f\ (\lambda d.\text{fix}\, f\ d) \in \mathcal{E}[\![\text{Unit} \to T]\!] \quad \text{by reduction} \\
\text{if} & \lambda d.\text{fix}\, f\ d \in \mathcal{V}[\![\text{Unit} \to T]\!] \quad \text{by } f\text{'s semantic type}
\end{array}
$$

A Proof Attempt

$$
\begin{array}{ll}
 & \text{fix} \in \mathcal{V}\,[\![((\text{Unit} \to T) \to \text{Unit} \to T) \to \text{Unit} \to T]\!] \\
\text{if} & \text{fix}\, f \in \mathcal{E}\,[\![\text{Unit} \to T]\!] \text{ for all } f \in \mathcal{V}\,[\![(\text{Unit} \to T) \to \text{Unit} \to T]\!] \\
\text{if} & f\ (\lambda d.\text{fix}\, f\ d) \in \mathcal{E}\,[\![\text{Unit} \to T]\!] \ \text{ by reduction} \\
\text{if} & \lambda d.\text{fix}\, f\ d \in \mathcal{V}\,[\![\text{Unit} \to T]\!] \ \text{ by } f\text{'s semantic type} \\
\text{if} & \text{fix}\, f\ d \in \mathcal{E}\,[\![T]\!] \text{ for all } d \in \mathcal{V}\,[\![\text{Unit}]\!]
\end{array}
$$

A Proof Attempt

$$
\begin{array}{ll}
& \text{fix} \in \mathcal{V}[\![((\text{Unit} \to T) \to \text{Unit} \to T) \to \text{Unit} \to T]\!] \\
\text{if} & \text{fix}\, f \in \mathcal{E}[\![\text{Unit} \to T]\!] \text{ for all } f \in \mathcal{V}[\![(\text{Unit} \to T) \to \text{Unit} \to T]\!] \\
\text{if} & f\ (\lambda d.\text{fix}\, f\ d) \in \mathcal{E}[\![\text{Unit} \to T]\!] \ \text{ by reduction} \\
\text{if} & \lambda d.\text{fix}\, f\ d \in \mathcal{V}[\![\text{Unit} \to T]\!] \ \text{ by } f\text{'s semantic type} \\
\text{if} & \text{fix}\, f\ d \in \mathcal{E}[\![T]\!] \text{ for all } d \in \mathcal{V}[\![\text{Unit}]\!]
\end{array}
$$

We know $f \in \mathcal{V}[\![(\text{Unit} \to T) \to \text{Unit} \to T]\!]$ and $d \in \mathcal{V}[\![\text{Unit}]\!]$. We could finish if we knew $\text{fix} \in \mathcal{V}[\![((\text{Unit} \to T) \to \text{Unit} \to T) \to \text{Unit} \to T]\!]$, but that's what we're trying to prove. Induction, where are you? 😩

Step-Indexing [AM01; AAV02; Ahm04]

$$
\begin{array}{lclll}
(k, v) & \in & \mathcal{V}[\![\text{Unit}]\!] & \text{iff} & v = () \\
(k, v_2) & \in & \mathcal{V}[\![T_1 \to T_2]\!] & \text{iff} & j \leq k \text{ and } (j, v_1) \in \mathcal{V}[\![T_1]\!] \text{ implies } (j, v_2\ v_1) \in \mathcal{E}[\![T_2]\!] \\
(k, e) & \in & \mathcal{E}[\![T]\!] & \text{iff} & j < k \text{ and } e \to^{j} e' \not\to \text{ implies } (k - j, e') \in \mathcal{V}[\![T]\!] \\
 & & \Gamma \vDash e : T & \text{iff} & \text{for all } k, (k, \sigma) \in \mathcal{S}[\![\Gamma]\!] \text{ implies } (k, e[\sigma]) \in \mathcal{E}[\![T]\!]
\end{array}
$$

Step-Indexing [AM01; AAV02; Ahm04]

$$
\begin{array}{llll}
(k, v) & \in \mathcal{V}[\![\text{Unit}]\!] & \text{iff} & v = () \\
(k, v_2) & \in \mathcal{V}[\![T_1 \to T_2]\!] & \text{iff} & j \leq k \text{ and } (j, v_1) \in \mathcal{V}[\![T_1]\!] \text{ implies } (j, v_2\ v_1) \in \mathcal{E}[\![T_2]\!] \\
(k, e) & \in \mathcal{E}[\![T]\!] & \text{iff} & j < k \text{ and } e \to^{j} e' \not\to \text{ implies } (k - j, e') \in \mathcal{V}[\![T]\!] \\
 & \Gamma \vDash e : T & \text{iff} & \text{for all } k, (k, \sigma) \in \mathcal{S}[\![\Gamma]\!] \text{ implies } (k, e[\sigma]) \in \mathcal{E}[\![T]\!]
\end{array}
$$

Lemma (fix is Semantically Well-Typed!)

$$\vDash \text{fix} : ((\text{Unit} \to T) \to \text{Unit} \to T) \to \text{Unit} \to T$$

Proof: *As before, but using induction on the step index* 🎉

Semantic Typing for Resources

The Linear Lambda Calculus

ID

$$x : T \vdash x : T$$

Unit-I

$$\vdash () : \mathsf{Unit}$$

Unit-E

$$\frac{\Gamma_1 \vdash e_1 : \mathsf{Unit} \qquad \Gamma_2 \vdash e_2 : T}{\Gamma_1, \Gamma_2 \vdash \mathsf{let}\,() = e_1 \,\mathsf{in}\, e_2 : T}$$

$\rightarrow$-I

$$\frac{\Gamma, x : T_1 \vdash e : T_2}{\Gamma \vdash \lambda x.e : T_1 \rightarrow T_2}$$

$\rightarrow$-E

$$\frac{\Gamma_1 \vdash e_1 : T_1 \qquad \Gamma_2 \vdash e_2 : T_1 \rightarrow T_2}{\Gamma_1, \Gamma_2 \vdash e_2\ e_1 : T_2}$$

$\times$-I

$$\frac{\Gamma_1 \vdash e_1 : T_1 \qquad \Gamma_2 \vdash e_2 : T_2}{\Gamma_1, \Gamma_2 \vdash (e_1, e_2) : T_1 \times T_2}$$

$\times$-E

$$\frac{\Gamma_1 \vdash e_1 : T_1^x \times T_1^y \qquad \Gamma_2, x : T_1^x, y : T_1^y \vdash e_2 : T_2}{\Gamma_1, \Gamma_2 \vdash \mathsf{let}\,(x, y) = e_1 \,\mathsf{in}\, e_2 : T_2}$$

An API for Unique References $\mathsf{Unq}\,T$

$$
\begin{array}{lcll}
\mathsf{new} & : & T \to \mathsf{Unq}\,T & \\
(\underbrace{\mu}_{\text{memory}}, \mathsf{new}\,v) & \to & (\mu[\ell \mapsto v], \ell) & (\ell \notin \mathrm{dom}(\mu)) \\
\mathsf{swap} & : & \mathsf{Unq}\,T_1 \times T_2 \to T_1 \times \mathsf{Unq}\,T_2 & \\
(\mu[\ell \mapsto v_1], \mathsf{swap}\,\ell\,v_2) & \to & \big(\mu[\ell \mapsto v_2], (v_1, \ell)\big) & \\
\mathsf{free} & : & \mathsf{Unq}\,T \to T & \\
(\mu[\ell \mapsto v], \mathsf{free}\,\ell) & \to & (\mu, v) &
\end{array}
$$

Semantic Types for LLC + Unq

$$(\mu, v) \quad \in \quad \mathcal{V}[\![\text{Unit}]\!] \qquad \text{iff} \quad \mu = \varnothing \text{ and } v = ()$$

Semantic Types for LLC + Unq

$$
\begin{array}{llll}
(\mu, v) & \in & \mathcal{V}[\![\text{Unit}]\!] & \text{iff} \quad \mu = \varnothing \text{ and } v = () \\
(\mu, v) & \in & \mathcal{V}[\![T_1 \times T_2]\!] & \text{iff} \quad \mu = \mu_1 \uplus \mu_2 \text{ and } v = (v_1, v_2) \\
 & & & \qquad \text{and } (\mu_1, v_1) \in \mathcal{V}[\![T_1]\!] \text{ and } (\mu_2, v_2) \in \mathcal{V}[\![T_2]\!]
\end{array}
$$

Semantic Types for LLC + Unq

$$
\begin{array}{llll}
(\mu, v) & \in & \mathcal{V}[\![\text{Unit}]\!] & \text{iff} \quad \mu = \varnothing \text{ and } v = () \\
(\mu, v) & \in & \mathcal{V}[\![T_1 \times T_2]\!] & \text{iff} \quad \mu = \mu_1 \uplus \mu_2 \text{ and } v = (v_1, v_2) \\
 & & & \qquad \text{and } (\mu_1, v_1) \in \mathcal{V}[\![T_1]\!] \text{ and } (\mu_2, v_2) \in \mathcal{V}[\![T_2]\!] \\
(\mu_2, v_2) & \in & \mathcal{V}[\![T_1 \rightarrow T_2]\!] & \text{iff} \quad (\mu_1, v_1) \in \mathcal{V}[\![T_1]\!] \text{ and } \mu_1 \text{ disjoint from } \mu_2 \\
 & & & \qquad \text{implies } (\mu_1 \uplus \mu_2, v_2\ v_1) \in \mathcal{E}[\![T_2]\!]
\end{array}
$$

Semantic Types for LLC + Unq

$$
\begin{array}{llll}
(\mu, v) & \in & \mathcal{V}[\![\mathsf{Unit}]\!] & \text{iff} \quad \mu = \varnothing \text{ and } v = () \\
(\mu, v) & \in & \mathcal{V}[\![T_1 \times T_2]\!] & \text{iff} \quad \mu = \mu_1 \uplus \mu_2 \text{ and } v = (v_1, v_2) \\
 & & & \phantom{\text{iff}} \quad \text{and } (\mu_1, v_1) \in \mathcal{V}[\![T_1]\!] \text{ and } (\mu_2, v_2) \in \mathcal{V}[\![T_2]\!] \\
(\mu_2, v_2) & \in & \mathcal{V}[\![T_1 \to T_2]\!] & \text{iff} \quad (\mu_1, v_1) \in \mathcal{V}[\![T_1]\!] \text{ and } \mu_1 \text{ disjoint from } \mu_2 \\
 & & & \phantom{\text{iff}} \quad \text{implies } (\mu_1 \uplus \mu_2, v_2\ v_1) \in \mathcal{E}[\![T_2]\!] \\
(\mu, v) & \in & \mathcal{V}[\![\mathsf{Unq}\ T]\!] & \text{iff} \quad v = \ell \text{ and } \mu = \mu_\ell \uplus [\ell \mapsto v_\ell] \text{ and } (\mu_\ell, v_\ell) \in \mathcal{V}[\![T]\!]
\end{array}
$$

Semantic Types for LLC + Unq

$$
\begin{array}{lcll}
(\mu, v) & \in & \mathcal{V}[\![\text{Unit}]\!] & \text{iff} \quad \mu = \varnothing \text{ and } v = () \\
(\mu, v) & \in & \mathcal{V}[\![T_1 \times T_2]\!] & \text{iff} \quad \mu = \mu_1 \uplus \mu_2 \text{ and } v = (v_1, v_2) \\
 & & & \qquad \text{and } (\mu_1, v_1) \in \mathcal{V}[\![T_1]\!] \text{ and } (\mu_2, v_2) \in \mathcal{V}[\![T_2]\!] \\
(\mu_2, v_2) & \in & \mathcal{V}[\![T_1 \to T_2]\!] & \text{iff} \quad (\mu_1, v_1) \in \mathcal{V}[\![T_1]\!] \text{ and } \mu_1 \text{ disjoint from } \mu_2 \\
 & & & \qquad \text{implies } (\mu_1 \uplus \mu_2, v_2\ v_1) \in \mathcal{E}[\![T_2]\!] \\
(\mu, v) & \in & \mathcal{V}[\![\text{Unq}\ T]\!] & \text{iff} \quad v = \ell \text{ and } \mu = \mu_\ell \uplus [\ell \mapsto v_\ell] \text{ and } (\mu_\ell, v_\ell) \in \mathcal{V}[\![T]\!] \\
(\mu, e) & \in & \mathcal{E}[\![T]\!] & \text{iff} \quad \mu_f \text{ disjoint from } \mu \text{ and } (\mu \uplus \mu_f, e) \to^* (\mu', e') \not\to \\
 & & & \qquad \text{implies } \mu' = \mu_v \uplus \mu_f \text{ and } e' = v \text{ and } (\mu_v, v) \in \mathcal{V}[\![T]\!]
\end{array}
$$

A Logical Approach to Type Soundness [DAB11; Tim+22]

- We want to use semantic types as a specification for how a program of a given type should behave.
- Specifications should be comprehensible!
- We should design good abstractions for the specification language to make it easier to understand, and easier to reason about.
- We can use a domain-specific logic for specifying types and proving properties about programs.
- Examples: separation logics, step-indexed logics

Separation Logic

$$\mu \quad \in \quad P \land Q \qquad \text{iff} \quad \mu \in P \text{ and } \mu \in Q$$

$$\vdots$$

Separation Logic

$$
\begin{array}{llll}
\mu & \in & P \wedge Q & \text{iff} \quad \mu \in P \text{ and } \mu \in Q \\
\vdots & & & \\
\mu & \in & P \star Q & \text{iff} \quad \mu = \mu_p \uplus \mu_q \text{ and } \mu_p \in P \text{ and } \mu_q \in Q
\end{array}
$$

Separation Logic

$$
\begin{array}{llll}
\mu & \in & P \wedge Q & \text{iff} \quad \mu \in P \text{ and } \mu \in Q \\
\vdots & & & \\
\mu & \in & P \star Q & \text{iff} \quad \mu = \mu_p \uplus \mu_q \text{ and } \mu_p \in P \text{ and } \mu_q \in Q \\
\mu_{q-p} & \in & P \mathrel{-\!\!\star} Q & \text{iff} \quad \mu_p \in P \text{ and } \mu_p \text{ disjoint from } \mu_{q-p} \\
& & & \qquad \text{implies } \mu_p \uplus \mu_{q-p} \in Q
\end{array}
$$

Separation Logic

$$
\begin{array}{llll}
\mu & \in & P \wedge Q & \text{iff} \quad \mu \in P \text{ and } \mu \in Q \\
\vdots & & & \\
\mu & \in & P \star Q & \text{iff} \quad \mu = \mu_p \uplus \mu_q \text{ and } \mu_p \in P \text{ and } \mu_q \in Q \\
\mu_{q-p} & \in & P \mathrel{-\!\!\star} Q & \text{iff} \quad \mu_p \in P \text{ and } \mu_p \text{ disjoint from } \mu_{q-p} \\
& & & \qquad \text{implies } \mu_p \uplus \mu_{q-p} \in Q \\
\mu & \in & \ell \mapsto v & \text{iff} \quad \mu = \ell \mapsto v
\end{array}
$$

Separation Logic

$$
\begin{array}{llll}
\mu & \in & P \wedge Q & \text{iff} \quad \mu \in P \text{ and } \mu \in Q \\
\vdots & & & \\
\mu & \in & P \star Q & \text{iff} \quad \mu = \mu_p \uplus \mu_q \text{ and } \mu_p \in P \text{ and } \mu_q \in Q \\
\mu_{q-p} & \in & P \mathrel{-\!\!\star} Q & \text{iff} \quad \mu_p \in P \text{ and } \mu_p \text{ disjoint from } \mu_{q-p} \\
 & & & \qquad \text{implies } \mu_p \uplus \mu_{q-p} \in Q \\
\mu & \in & \ell \mapsto v & \text{iff} \quad \mu = \ell \mapsto v \\
\mu & \in & \ulcorner \textbf{math} \urcorner & \text{iff} \quad \mu = \varnothing \text{ and the } \textbf{math} \text{ is true}
\end{array}
$$

Separation Logic

$$
\begin{array}{llll}
\mu & \in & P \wedge Q & \text{iff} \quad \mu \in P \text{ and } \mu \in Q \\
\vdots & & & \\
\mu & \in & P \star Q & \text{iff} \quad \mu = \mu_p \uplus \mu_q \text{ and } \mu_p \in P \text{ and } \mu_q \in Q \\
\mu_{q-p} & \in & P \mathrel{-\!\!\star} Q & \text{iff} \quad \mu_p \in P \text{ and } \mu_p \text{ disjoint from } \mu_{q-p} \\
 & & & \qquad \text{implies } \mu_p \uplus \mu_{q-p} \in Q \\
\mu & \in & \ell \mapsto v & \text{iff} \quad \mu = \ell \mapsto v \\
\mu & \in & \ulcorner \textbf{math} \urcorner & \text{iff} \quad \mu = \varnothing \text{ and the } \textbf{math} \text{ is true} \\
\mu & \in & \text{wp}(e)\{Q\} & \text{iff} \quad \mu_f \text{ disjoint from } \mu \text{ and } (\mu \uplus \mu_f, e) \rightarrow^* (\mu', e') \not\rightarrow \\
 & & & \qquad \text{implies } \mu' = \mu_v \uplus \mu_f \text{ and } e' = v \text{ and } \mu_v \in Q(v)
\end{array}
$$

Semantic Types for LLC + Unq, Revisited

$$
\begin{aligned}
v &\in \mathcal{V}[\![\mathrm{Unit}]\!] &&\iff \ulcorner v = () \urcorner \\
v &\in \mathcal{V}[\![T_1 \times T_2]\!] &&\iff \exists v_1, v_2.\ \ulcorner v = (v_1, v_2) \urcorner \star v_1 \in \mathcal{V}[\![T_1]\!] \star v_2 \in \mathcal{V}[\![T_2]\!] \\
v_2 &\in \mathcal{V}[\![T_1 \to T_2]\!] &&\iff \forall v_1.\ v_1 \in \mathcal{V}[\![T_1]\!] -\!\!\star\ v_2\ v_1 \in \mathcal{V}[\![T_2]\!] \\
v &\in \mathcal{V}[\![\mathrm{Unq}\ T]\!] &&\iff \exists \ell, v_\ell.\ \ulcorner v = \ell \urcorner \star \ell \mapsto v_\ell \star v_\ell \in \mathcal{V}[\![T]\!] \\
e &\in \mathcal{E}[\![T]\!] &&\iff \mathrm{wp}(e)\{v.\ v \in \mathcal{V}[\![T]\!]\}
\end{aligned}
$$

An API for Shareable Resources

$$
\begin{array}{lcl}
\text{Shareable Type } S & ::= & \text{Unit} \mid S_1 \times S_2 \mid \text{Shr}\, S \\
\text{dup} & : & S \rightarrow S \times S \\
\text{drop} & : & S \rightarrow \text{Unit} \\
\text{share} & : & \text{Unq}\, S \rightarrow \text{Shr}\, S \\
\text{load} & : & \text{Shr}\, S \rightarrow S
\end{array}
$$

Fictional Separation

Even when resources aren't physically disjoint, they might be logically compatible.

Fictional Separation

Even when resources aren't physically disjoint, they might be logically compatible.

$$\underbrace{\widehat{\mu}}_{\text{logical memory}} \quad : \quad \textsc{Loc} \rightharpoonup \underbrace{\{\mathsf{shr}, \mathsf{unq}\}}_{\text{logical flag}} \times \textsc{Val}$$

Fictional Separation

Even when resources aren't physically disjoint, they might be logically compatible.

$$\underbrace{\widehat{\mu}}_{\text{logical memory}} \quad : \quad \textsc{Loc} \rightharpoonup \underbrace{\{\mathsf{shr}, \mathsf{unq}\}}_{\text{logical flag}} \times \textsc{Val}$$

$$\lfloor \widehat{\mu} \rfloor \quad \triangleq \quad [\ell \mapsto v \mid \widehat{\mu}(\ell) = (-, v)]$$

Fictional Separation

Even when resources aren't physically disjoint, they might be logically compatible.

$$\underbrace{\widehat{\mu}}_{\text{logical memory}} \quad : \quad \textsc{Loc} \rightharpoonup \underbrace{\{\mathsf{shr}, \mathsf{unq}\}}_{\text{logical flag}} \times \textsc{Val}$$

$$\lfloor \widehat{\mu} \rfloor \quad \triangleq \quad [\ell \mapsto v \mid \widehat{\mu}(\ell) = (-, v)]$$

$$\widehat{\mu}_1 \text{ compatible with } \widehat{\mu}_2 \quad \text{iff} \quad \ell \in \mathrm{dom}(\widehat{\mu}_1) \cap \mathrm{dom}(\widehat{\mu}_2) \text{ implies } \widehat{\mu}_1(\ell) = \widehat{\mu}_2(\ell) = (\mathsf{shr}, v)$$

Fictional Separation

Even when resources aren't physically disjoint, they might be logically compatible.

$$\underbrace{\widehat{\mu}}_{\text{logical memory}} \quad : \quad \text{Loc} \rightharpoonup \underbrace{\{\text{shr}, \text{unq}\}}_{\text{logical flag}} \times \text{Val}$$

$$\lfloor \widehat{\mu} \rfloor \triangleq [\ell \mapsto v \mid \widehat{\mu}(\ell) = (-, v)]$$

$$\widehat{\mu}_1 \text{ compatible with } \widehat{\mu}_2 \text{ iff } \ell \in \text{dom}(\widehat{\mu}_1) \cap \text{dom}(\widehat{\mu}_2) \text{ implies } \widehat{\mu}_1(\ell) = \widehat{\mu}_2(\ell) = (\text{shr}, v)$$

$$\widehat{\mu}_1 \bullet \widehat{\mu}_2 \triangleq \widehat{\mu}_1 \cup \widehat{\mu}_2 \text{ if } \widehat{\mu}_1 \text{ compatible with } \widehat{\mu}_2 \text{ else undefined}$$

Semantic Types for LLC + Unq + Shr

$$
\begin{array}{lllll}
v & \in & \mathcal{V}[\![\mathsf{Shr}\, T]\!] & \iff & \exists\, \ell, v_\ell.\ \ulcorner v = \ell \urcorner \star \ell \overset{\mathsf{shr}}{\mapsto} v_\ell \star !(v_\ell \in \mathcal{V}[\![T]\!]) \\
\widehat{\mu} & \in & !P & \text{iff} & \widehat{\mu} \in P \text{ and } \ell \overset{m}{\mapsto} v \in \widehat{\mu} \text{ implies } m = \mathsf{shr}
\end{array}
$$

Semantic Types for Low-Level Code

Realizability

1. Specify the source syntax and type system.
2. Specify the target syntax and operational semantics.
3. Assign each type T a set of target programs $[\![T]\!]$.
4. **Lemma (Adequacy):** If $e \in [\![T]\!]$ then e is safe to run.
5. **Lemma (Fundamental Property):** If $e : T$ and e compiles to e, then $e \in [\![T]\!]$.
6. **Theorem (Type Soundness):** If $e : T$ and e compiles to e, then e is safe to run.

A Baby Boolean "ABI"

$$\begin{array}{lcllll}
\text{true} & : & \text{Bool} & \rightsquigarrow & \lambda x.\lambda y.x \\
\text{false} & : & \text{Bool} & \rightsquigarrow & \lambda x.\lambda y.y \\
\text{and} & : & \text{Bool} \to \text{Bool} \to \text{Bool} & \rightsquigarrow & \lambda x.\lambda y.x\ y\ \text{false} \\
v & \in & \mathcal{V}[\![\text{Bool}]\!] & \text{iff} & v = \lambda x.\lambda y.x \text{ or } v = \lambda x.\lambda y.y
\end{array}$$

Lemma (and Compatible)

$$\text{and} \in \mathcal{V}[\![\text{Bool} \to \text{Bool} \to \text{Bool}]\!]$$

Example

$\vDash$ false and () : Bool

Thanks for listening! 😊