A Syntactic Account of Type Abstraction

Steve Zdancewic

with

Dan Grossman and Greg Morrisett

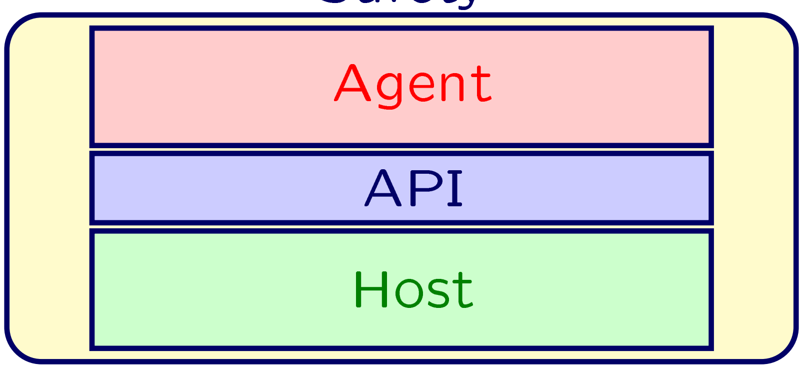
Type Safety

"Type structure is a *syntactic* discipline for enforcing levels of abstraction." — Reynolds 1983

Extensibility

- Web Browsers (Java applets)
- Operating Systems (Exokernel)
- Databases (Predator)
- Network Routers (PLAN)

Safety



Safety Properties I

Host Provides

(* file handle implemented as int *)
abstype fh
open : $string \rightarrow$ fh
read : fh $\rightarrow int$

Safety Property 1

Agent can't create file handles:
Must call open to obtain one.

Safety Properties II

Safety Property 2

File handles are abstract:

$$(\lambda h\!:\!\mathsf{fh}.agent) \quad handle$$

never steps to:

$$E[handle + 3]$$

Even if fh is implemented as int.

Safety Property 3

The read function is applied only to host-provided file handles.

How Do we Prove Them?

Use parametricity results.

System-F Encoding

$$\Lambda \mathsf{t}_{host}.\lambda x_{host}{:}\tau(\mathsf{t}_{host}).agent$$

Example

$$\Lambda \mathsf{fh}.\lambda open{:}string \rightarrow \mathsf{fh}.\ (open\,\text{``}myfile\text{''})$$

A Problem of Scale

Program Features

- Parametric Polymorphism
- Recursive Types
- References
- Continuations
- Threads
- Objects
- ...

Type Soundness

Syntactic versions of type soundness arguments based on Preservation and Progress do scale to languages with these features.

Safety Property 4 (Corollary of Type Soundness)

Well typed programs don't violate the *built in* abstractions of the language.

An Observation

Operational semantics don't preserve user defined type abstraction.

"Linked" Agent Code

$$\Lambda \mathsf{fh}.\lambda open : string \rightarrow \mathsf{fh}.open\ \text{``}myfile\text{''}\ int\ \ ho$$

$$\mapsto$$

$$(\lambda open : string \rightarrow int.open\ \text{``}myfile\text{''})\ \ ho$$

A Solution

Make type abstraction explicit in the syntax by:

- Coloring agent code red.
- Coloring host code green.
- Typechecking with different rules.

1. Host knows fn $= int$
2. Agent does not

- Tracking the colors of code throughout evaluation.

Syntax

Types

$$\tau \;::=\; \mathsf{fh} \mid b \mid \tau_1 \to \tau_2$$

Agent Terms & Values

$$\begin{aligned} a \;&::=\; x \mid c \mid \lambda x\!:\!\tau.a \mid a\ a' \mid \lceil h \rceil^{\tau} \\ \hat{a} \;&::=\; c \mid \lambda x\!:\!\tau.a \mid \lceil \hat{h} \rceil^{\mathsf{fh}} \end{aligned}$$

Host Terms & Values

$$\begin{aligned} h \;&::=\; x \mid c \mid \lambda x\!:\!\tau.h \mid h\ h' \mid \lceil a \rceil^{\tau} \\ \hat{h} \;&::=\; c \mid \lambda x\!:\!\tau.h \end{aligned}$$

Operational Semantics

Monochrome Steps

$$
\begin{array}{rcll}
e_1\ e_2 & \longmapsto & e_1'\ e_2 & \text{if } e_1 \longmapsto e_1' \\
(\lambda x\!:\!\tau.e)e' & \longmapsto & \{e'/x\}e &
\end{array}
$$

Agent Steps

$$
\begin{array}{rcll}
\llcorner h \lrcorner^{\tau} & \longmapsto & \llcorner h' \lrcorner^{\tau} & \text{if } h \longmapsto h' \\
\llcorner c \lrcorner^{b} & \longmapsto & c & \\
\llcorner \lambda x\!:\!\tau.h \lrcorner^{\tau_1 \to \tau_2} & \longmapsto & \lambda x\!:\!\tau_1.\llcorner \{\llcorner x \lrcorner^{\tau}/x\}h \lrcorner^{\tau_2} &
\end{array}
$$

Host Steps

$$
\begin{array}{rcll}
\llcorner a \lrcorner^{\tau} & \longmapsto & \llcorner a' \lrcorner^{\tau} & \text{if } a \longmapsto a' \\
\llcorner c \lrcorner^{b} & \longmapsto & c & \\
\llcorner \lambda x\!:\!\tau.a \lrcorner^{\tau_1 \to \tau_2} & \longmapsto & \lambda x\!:\!\{int/\mathsf{fh}\}\tau_1.\llcorner \{\llcorner x \lrcorner^{\tau}/x\}a \lrcorner^{\tau_2} & \\
\llcorner \llcorner \hat{h} \lrcorner^{\mathsf{fh}} \lrcorner^{int} & \longmapsto & \hat{h} &
\end{array}
$$

Example I

$$\lambda open : string \to \mathsf{fh}.open\ \ \text{“}myfile\text{”}\ \ [\lambda s : string.ho(s)]^{string \to \mathsf{fh}}$$

$$\longmapsto$$

$$[\lambda s : string.ho(s)]^{string \to \mathsf{fh}}\ \ \text{“}myfile\text{”}$$

$$\longmapsto$$

$$\lambda s : string.[ho([s]^{string})]^{\mathsf{fh}}\ \ \text{“}myfile\text{”}$$

$$\longmapsto$$

$$[ho([\text{“}myfile\text{”}]^{string})]^{\mathsf{fh}}$$

$$\longmapsto^{*}$$

$$[3]^{\mathsf{fh}}$$

Example II

$$[\lambda handle : int. hr(handle)]^{\mathsf{fh} \rightarrow int} \quad [3]^{\mathsf{fh}}$$

$$\longmapsto$$

$$\lambda handle : \mathsf{fn}. [hr([handle]^{int})]^{int} \quad [3]^{\mathsf{fh}}$$

$$\longmapsto$$

$$[hr([[3]^{\mathsf{fh}}]^{int})]^{int}$$

$$\longmapsto$$

$$[hr(3)]^{int}$$

$$\longmapsto^{*}$$

$$[611]^{int} \longmapsto 611$$

Static Semantics

Monochrome Rules

$$(\mathbf{var}) \qquad \Gamma \vdash x : \Gamma(x)$$

$$(\mathbf{const}) \qquad \Gamma \vdash c : b$$

$$(\mathbf{app}) \qquad \frac{\Gamma \vdash e_1 : \tau' \to \tau \quad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash e_1\ e_2 : \tau}$$

Agent Rules

$$(\mathbf{ah}) \quad \frac{\Gamma \vdash h : \{int/\mathsf{fh}\}\tau}{\Gamma \vdash \llcorner h \lrcorner^{\tau} : \tau} \qquad (\mathbf{afn}) \quad \frac{\Gamma[x : \tau'] \vdash a : \tau}{\Gamma \vdash \lambda x{:}\tau'.a : \tau' \to \tau}$$

Host Rules ($\mathsf{fh} = int$)

$$(\mathbf{ha}) \quad \frac{\Gamma \vdash a : \tau}{\Gamma \vdash \llcorner a \lrcorner^{\tau} : \{int/\mathsf{fh}\}\tau} \quad (\mathbf{hfn}) \quad \frac{\Gamma[x : \tau'] \vdash h : \tau}{\Gamma \vdash \lambda x{:}\tau'.h : \tau' \to \tau}(\mathsf{fh} \notin \tau')$$

Erasure

Ignoring the distinction between colors and erasing boxes leaves terms untouched.
In particular:

$$\lceil \lambda x : \tau . h \rceil^{\tau_1 \to \tau_2} \longmapsto \lambda x : \tau_1 . \lceil \{ \lceil x \rceil^{\tau} / x \} h \rceil^{\tau_2}$$

erases to:

$$\lambda x . h \longmapsto \lambda x . \{ x / x \} h$$

Theorems

Erasure

$$e \longmapsto^* \hat{e} \quad \text{iff} \quad erase(e) \longmapsto^* erase(\hat{e})$$

Preservation

If $\vdash e : \tau$ and $e \longmapsto e'$ then $\vdash e' : \tau$.

Progress

If $\vdash e : \tau$ then either $e = \hat{e}$ or $e \longmapsto e'$ for some e'.

Properties I

Corollary to Soundness

If $\vdash \lambda h\!:\!\mathsf{fh}.agent : \tau$ and $\vdash handle : \mathsf{fh}$ then:

$$(\lambda h\!:\!\mathsf{fh}.agent) \quad handle$$

never steps to:

$$E[handle + 3]$$

Even though fh is implemented as int.

Lemma

If $[x : \mathsf{fh}] \vdash a : \tau$ and a is host-free then:

$$\{\llcorner c \lrcorner^{\mathsf{fh}}/x\}a \longmapsto a_1 = \{\llcorner c \lrcorner^{\mathsf{fh}}/x\}a''$$

iff

$$\{\llcorner c' \lrcorner^{\mathsf{fh}}/x\}a \longmapsto a_1 = \{\llcorner c' \lrcorner^{\mathsf{fh}}/x\}a''$$

furthermore, $[x : \mathsf{fh}] \vdash a'' : \tau$ and a'' is host-free.

Type Abstraction

If $\vdash \lambda h\!:\!\mathsf{fh}.agent : \mathsf{fh} \to int$, $\vdash \ulcorner\hat{h}\urcorner^{\mathsf{fh}} : \mathsf{fh}$, $\vdash \ulcorner\hat{h}'\urcorner^{\mathsf{fh}} : \mathsf{fh}$, and $agent$ is host-free then:

$$(\lambda h\!:\!\mathsf{fh}.agent)\ \ \ulcorner\hat{h}\urcorner^{\mathsf{fh}} \longmapsto^{*} c$$

iff

$$(\lambda h\!:\!\mathsf{fh}.agent)\ \ \ulcorner\hat{h}'\urcorner^{\mathsf{fh}} \longmapsto^{*} c$$

Agent Must Call Open

Suppose $\lambda open : string \rightarrow \mathsf{fh}.agent$ is well-typed and $agent$ is host-free. If $(\lambda open\!:\!string \rightarrow \mathsf{fh}.agent)\ \ulcorner \lambda s\!:\!string.ho(s) \urcorner^{string \rightarrow \mathsf{fh}}$ steps to some term $agent'$ containing $\ulcorner \widehat{h} \urcorner^{\mathsf{fh}}$ as a sub-term, then there exists a string s such that

$$ho(s) \longmapsto^{*} \widehat{h}$$

Scaling Up

- Add fix, products, sums, etc.
- Add multiple abstract types to the host.
- Allow agent to have abstract types.

Multiple Agents

Allow arbitrary number of agents, with type sharing.

- Each agent has different type abstractions
- Allows for linking multiple libraries
- Polymorphism: Instantiating type variable spawns new agent:
  $(\Lambda^{\tau'}\alpha.e)\ \tau \longmapsto [e]^{\{\tau/\alpha\}\tau'}$

Related Work

- Types, Abstraction, and Parametric Polymorphism, Reynolds '83
- A Syntactic Approach to Type Soundness, Wright & Felleisen '92
- Security Properties of Typed Applets, Leroy & Rouaix '98
- The SLam Calculus: Programming with Secrecy and Integrity, Heintze & Riecke '98

Conclusion

Standard Operational Semantics

- No notion of principals (agents vs. hosts)
- Substituting types for variables loses abstraction

Our language

- Tracks pricipals during evaluation by "coloring" code
- Maintains type abstractions explicitly

- The hope is that this will scale to languages with many features.