Gradual Typing for Fluent languages

On the path to Gradual Typing for Generic Type-and-Effect Systems

Felipe Bañados

University of Chile

May 27, 2013

What is an Effect System?

- A type system that also verifies and guarantees side-effect invariants over programs.
- Allows static reasoning about a program's execution.
- The process of verifying these invariants is called effect checking.

Motivation for our work

- What if we want to add effect checking / handling to an existent codebase?

Example (Exceptions)

```
class Stack{
  private Object[] array = new Object[20];
  private int head=-1;

  public void push(Object element){
      int location=head++;
      array[location]=element;
      head=location;
  }
  ...
}
```

Example (Exceptions)

```
  public void push(Object element)
    throws FullStackException{
      int location=head++;
      if (location == array.length)
        throw new FullStackException();
      array[location]=element;
      head=location;
  }
}
```

Motivation for our work

We want to explore the relation between Gradual Typing and Effect Systems.

Examples of Effect Systems

- Exception Handling
- Memory updates
- I/O

Lambda Calculus + References[1]

This extension to the STLC adds assignment, through the following language constructs:

- Memory Allocation. (`ref` v).
- Access the contents of a memory location. ($!e$)
- Change the contents of a memory location. ($l \leftarrow v$)

Type checking

Typechecking requires, besides a type context, a store typing Σ which maps locations to the type of their content.

$$\text{T-Var}\ \frac{\Gamma(x) = \tau}{\Gamma; \Sigma \vdash x : \tau} \qquad \text{T-Abs}\ \frac{\Gamma, x : \tau_1; \Sigma \vdash e_2 : \tau_2}{\Gamma; \Sigma \vdash \lambda x : \tau_1 \,.\, e_2 : \tau_1 \longrightarrow \tau_2}$$

$$\text{T-App}\ \frac{\Gamma; \Sigma \vdash e_1 : \tau_1 \longrightarrow \tau_2 \quad \Gamma; \Sigma \vdash e_2 : \tau_1}{\Gamma; \Sigma \vdash e_1\ e_2 : \tau_2} \qquad \text{T-Unit}\ \frac{}{\Gamma; \Sigma \vdash \mathrm{unit} : \mathrm{Unit}}$$

Type checking

Typechecking requires, besides a type context, a store typing Σ which maps locations to the type of their content.

$$\text{T-Var}\ \frac{\Gamma(x) = \tau}{\Gamma; \Sigma \vdash x : \tau} \qquad \text{T-Abs}\ \frac{\Gamma, x : \tau_1; \Sigma \vdash e_2 : \tau_2}{\Gamma; \Sigma \vdash \lambda x : \tau_1 \,.\, e_2 : \tau_1 \longrightarrow \tau_2}$$

$$\text{T-App}\ \frac{\Gamma; \Sigma \vdash e_1 : \tau_1 \longrightarrow \tau_2 \quad \Gamma; \Sigma \vdash e_2 : \tau_1}{\Gamma; \Sigma \vdash e_1\ e_2 : \tau_2} \qquad \text{T-Unit}\ \frac{}{\Gamma; \Sigma \vdash \mathrm{unit} : \mathrm{Unit}}$$

Type checking

$$\text{T-Loc}\ \frac{\Sigma(l) = \tau}{\Gamma; \Sigma \vdash l : \texttt{Ref}\ \tau} \qquad \text{T-Ref}\ \frac{\Gamma; \Sigma \vdash e : \tau}{\Gamma; \Sigma \vdash \texttt{ref}\ e : \texttt{Ref}\ \tau}$$

$$\text{T-Deref}\ \frac{\Gamma; \Sigma \vdash e : \texttt{Ref}\ \tau}{\Gamma; \Sigma \vdash !e : \tau} \qquad \text{T-Assign}\ \frac{\Gamma; \Sigma \vdash e_1 : \texttt{Ref}\ \tau \quad \Gamma; \Sigma \vdash e_2 : \tau}{\Gamma; \Sigma \vdash e_1 \leftarrow e_2 : \text{Unit}}$$

Fluent Languages

- First Effect System (proposed in 1986 by Gifford and Lucassen)
- Side-Effect: State.
- Goal: To identify opportunities for concurrent execution and memoization.

Effect classes

Programs can be split into 4 categories called "effect classes". They represent the subset of the language required by the program (related to state).

- Pure: $(\lambda x\colon \texttt{Nat} \,.\, (+\ 1\ x))$
- Function: $(\lambda x\colon \texttt{Ref Nat} \,.\, x)(\texttt{ref}\ 1)$
- Observer: $(\lambda x\colon \texttt{Ref Nat} \,.\, !x)(\texttt{ref}\ 5)$
- Procedure:$(\lambda x\colon \texttt{Ref Nat} \,.\, x \leftarrow 2)(\texttt{ref}\ 5)$

Effect Classes as a set of privileges

A program's effect class shows the privileges granted for it to use:

- Pure $= \emptyset$.
- Function $= \{\text{alloc}\}$
- Observer $= \{\text{alloc}, \text{read}\}$
- Procedure $= \{\text{alloc}, \text{read}, \text{write}\}$

We will use the privilege sets for our language definitions.

Effect Checking in the Fluent Language

The key feature of the Fluent Language is that it allows to enforce restrictions over the privileges available for an expression to use.

$$(\texttt{with-effects}\ \emptyset\ \text{unit}) \tag{1}$$

$$(\texttt{with-effects}\ \emptyset\ (\texttt{ref}\ \text{unit})) \tag{2}$$

$$\text{T-Loc}\ \frac{\Sigma(l) = \tau}{\Gamma; \Sigma \vdash l : \texttt{Ref}\ \tau} \qquad \text{T-Ref}\ \frac{\Gamma; \Sigma \vdash e : \tau}{\Gamma; \Sigma \vdash \texttt{ref}\ e : \texttt{Ref}\ \tau}$$

$$\text{T-Deref}\ \frac{\Gamma; \Sigma \vdash e : \texttt{Ref}\ \tau}{\Gamma; \Sigma \vdash !e : \tau} \qquad \text{T-Assign}\ \frac{\Gamma; \Sigma \vdash e_1 : \texttt{Ref}\ \tau \quad \Gamma; \Sigma \vdash e_2 : \tau}{\Gamma; \Sigma \vdash e_1 \leftarrow e_2 : \text{Unit}}$$

$$\text{T-Ref}\ \frac{\Gamma; \Sigma; \Phi \vdash e : \tau}{\Gamma; \Sigma; \Phi \vdash \text{ref}\ e : \text{Ref}\ \tau} \qquad \text{T-Deref}\ \frac{\Gamma; \Sigma; \Phi \vdash e : \text{Ref}\ \tau}{\Gamma; \Sigma; \Phi \vdash \text{deref}\ e : \tau}$$

$$\text{T-Assign}\ \frac{\begin{array}{c}\Gamma; \Sigma; \Phi \vdash e_1 : \text{Ref}\ \tau_1 \\ \Gamma; \Sigma; \Phi \vdash e_2 : \tau_2 \\ \tau_2 <: \tau_1\end{array}}{\Gamma; \Sigma; \Phi \vdash \text{assign}\ e_1\ e_2 : \text{Unit}}$$

$$\text{T-With-Effects}\ \frac{\Gamma; \Sigma; \Phi_1 \vdash e : \tau}{\Gamma; \Sigma; \Phi \vdash \texttt{with-effects}\ \ \Phi_1\ e : \tau}$$

$$\text{T-Ref}\ \frac{\Gamma; \Sigma; \Phi \vdash e : \tau \qquad \{\text{alloc}\} \subseteq \Phi}{\Gamma; \Sigma; \Phi \vdash \text{ref}\ e : \text{Ref}\ \tau} \qquad \text{T-Deref}\ \frac{\Gamma; \Sigma; \Phi \vdash e : \text{Ref}\ \tau \qquad \{\text{read}\} \subseteq \Phi}{\Gamma; \Sigma; \Phi \vdash \text{deref}\ e : \tau}$$

$$\text{T-Assign}\ \frac{\Gamma; \Sigma; \Phi \vdash e_1 : \text{Ref}\ \tau_1 \qquad \Gamma; \Sigma; \Phi \vdash e_2 : \tau_2 \qquad \{\text{write}\} \subseteq \Phi \qquad \tau_2 <: \tau_1}{\Gamma; \Sigma; \Phi \vdash \text{assign}\ e_1\ e_2 : \text{Unit}}$$

$$\text{T-With-Effects}\ \frac{\Gamma; \Sigma; \Phi_1 \vdash e : \tau \qquad \Phi_1 \subseteq \Phi}{\Gamma; \Sigma; \Phi \vdash \texttt{with-effects}\ \ \Phi_1\ e : \tau}$$

Effects and Types

Should the following program be statically accepted or rejected on its own?

$$(\lambda f : \text{Unit} \longrightarrow \text{Unit} \;.\; (\texttt{with-effects}\ \emptyset\ (f\ \text{unit}))) \qquad (3)$$

Effects and Types

We can annotate the function type with the privilege set required by the function body

$$(\lambda f : \text{Unit} \xrightarrow{\emptyset} \text{Unit} \,.\, (\texttt{with-effects}\ \ \emptyset\ (f\ \text{unit})))$$

Effects and Types

$$\text{T-Fn}\ \frac{\Gamma, x : \tau_1; \Sigma; \Phi_1 \vdash e : \tau_2}{\Gamma; \Sigma; \Phi \vdash (\lambda x : \tau_1\ .\ e) : \tau_1 \xrightarrow{\Phi_1} \tau_2}$$

$$\text{T-App}\ \frac{\begin{array}{c}\Gamma; \Sigma; \Phi \vdash e_1 : (\tau_1 \xrightarrow{\Phi_1} \tau_3) \\ \Gamma; \Sigma; \Phi \vdash e_2 : \tau_2 \\ \tau_2 <: \tau_1 \qquad \Phi_1 \subseteq \Phi\end{array}}{\Gamma; \Sigma; \Phi \vdash e_1\ e_2 : \tau_3}$$

Subtyping

$$\text{ST-Unit}\ \frac{}{\text{Unit} <: \text{Unit}} \qquad \text{ST-Ref}\ \frac{\tau <: \sigma \quad \sigma <: \tau}{\texttt{Ref}\ \sigma <: \texttt{Ref}\ \tau}$$

$$\text{ST-Fun}\ \frac{\tau_1 <: \sigma_1 \quad \sigma_2 <: \tau_2 \quad \Phi_1 \subseteq \Phi_2}{\sigma_1 \xrightarrow{\Phi_1} \sigma_2 <: \tau_1 \xrightarrow{\Phi_2} \tau_2}$$

Gradual Effect Annotations

The fluent language requires mandatory privilege set annotations for function types. We want to loosen this restriction, so we can accept programs without annotations, by adding some runtime checks.

$$(\lambda f : \text{Unit} \longrightarrow \text{Unit} \,.\, (\texttt{with-effects}\ \emptyset\ (f\ \text{unit}))) \qquad (4)$$

Runtime checks

$$\text{E-Enforce}\ \frac{\Phi' \vdash e \mid \mu \Downarrow v \mid \mu' \qquad \Phi' \subseteq \Phi}{\Phi \vdash \text{enforce}\ \Phi'\ e \mid \mu \Downarrow v \mid \mu'}$$

$$\text{E-Check}\ \frac{\Phi \vdash e \mid \mu \Downarrow v \mid \mu' \qquad \Phi' \subseteq \Phi}{\Phi \vdash \text{check}\ \Phi'\ e \mid \mu \Downarrow e \mid \mu'}$$

Runtime checks

$$\text{E-Enforce}\ \frac{\begin{array}{c}\Phi' \vdash e \mid \mu \Downarrow v \mid \mu' \\ \Phi' \subseteq \Phi\end{array}}{\Phi \vdash \text{enforce}\ \Phi'\ e \mid \mu \Downarrow v \mid \mu'}$$

$$\text{E-Check}\ \frac{\begin{array}{c}\Phi \vdash e \mid \mu \Downarrow v \mid \mu' \\ \Phi' \subseteq \Phi\end{array}}{\Phi \vdash \text{check}\ \Phi'\ e \mid \mu \Downarrow e \mid \mu'}$$

Unknown privileges

We will consider that there are unknown privileges in the set. Thus we can keep information about the known part of the set and reason about it too.

Unknown privileges

Do we want to statically accept or reject the following programs?

$$(\lambda f : \text{Unit} \xrightarrow{\{¿\}} \text{Unit} \,.\, (\texttt{with-effects}\ \emptyset\ (f\ \text{unit}))) \quad (5)$$

$$(\lambda f : \text{Unit} \xrightarrow{\{\text{read},¿\}} \text{Unit} \,.\, (\texttt{with-effects}\ \emptyset\ (f\ \text{unit}))) \quad (6)$$

$$(\lambda f : \text{Unit} \xrightarrow{\{¿\}} \text{Unit} \,.\, \text{with-effects}\ \emptyset\ (f(\text{deref}\ (\text{ref}\ \text{unit})))) \quad (7)$$

Unknown privileges

What happens then with our type system containment verifications?

$$\text{T-With-Effects}\ \frac{\Gamma; \Sigma; \Phi_1 \vdash e : \tau \qquad \Phi_1 \subseteq \Phi}{\Gamma; \Sigma; \Phi \vdash \texttt{with-effects}\ \ \Phi_1\ e : \tau} \tag{8}$$

Unknown privileges

What happens then with our type system containment verifications?

$$\text{T-With-Effects}\ \frac{\Gamma; \Sigma; \Phi_1 \vdash e : \tau \qquad \Phi_1 \mathrel{\tilde{\sqsubseteq}} \Phi}{\Gamma; \Sigma; \Phi \vdash \texttt{with-effects}\ \ \Phi_1\ e : \tau} \tag{8}$$

Defining $a \underset{\sim}{\sqsubseteq} b$

- We want to keep the semantics of $a \subseteq b$, whenever ¿ $\notin a$ and ¿ $\notin b$
- $\{¿\} \underset{\sim}{\sqsubseteq} \{\text{read}, \text{write}, \text{alloc}\}$
- $\{\text{read}, \text{write}, \text{alloc}, ¿\} \underset{\sim}{\sqsubseteq} \{\text{read}, \text{write}, \text{alloc}\}$
- $\{¿\} \underset{\sim}{\sqsubseteq} \emptyset$
- $\{¿, \text{read}\} \not\underset{\sim}{\sqsubseteq} \emptyset$
- $\{\text{read}, \text{write}, \text{alloc}\} \underset{\sim}{\sqsubseteq} \{¿\}$

Defining $a \underset{\sim}{\sqsubseteq} b$

$$a \underset{\sim}{\sqsubseteq} b \iff a \setminus \{¿\} \subseteq b \vee ¿ \in b \tag{9}$$

Either:

1. The known part of a is contained in b (¿ might be $\emptyset$), or
2. b has an unknown part (which at runtime might be the universe of privileges)

The final type system

$$\text{T-Unit}\ \frac{}{\Gamma; \Sigma; \Phi \vdash \text{unit} : \text{Unit}}$$

$$\text{T-Fn}\ \frac{\Gamma, x : \tau_1; \Sigma; \Phi_1 \vdash e : \tau_2}{\Gamma; \Sigma; \Phi \vdash (\lambda x : \tau_1 \ . \ e) : \tau_1 \xrightarrow{\Phi_1} \tau_2}$$

$$\text{T-Loc}\ \frac{\Sigma(l) = \tau}{\Gamma; \Sigma; \Phi \vdash l : \text{Ref}\ \tau}$$

$$\text{T-Var}\ \frac{\Gamma(x) = \tau}{\Gamma; \Sigma; \Phi \vdash x : \tau}$$

$$\text{T-Ref}\ \frac{\Gamma; \Sigma; \Phi \vdash e : \tau \quad \{\text{alloc}\} \sqsubseteq \Phi}{\Gamma; \Sigma; \Phi \vdash \text{ref}\ e : \text{Ref}\ \tau}$$

$$\text{T-Deref}\ \frac{\Gamma; \Sigma; \Phi \vdash e : \text{Ref}\ \tau \quad \{\text{read}\} \sqsubseteq \Phi}{\Gamma; \Sigma; \Phi \vdash \text{deref}\ e : \tau}$$

The final type system

$$\text{T-Assign}\ \frac{\Gamma;\Sigma;\Phi \vdash e_1 : \text{Ref}\ \tau_1 \quad \Gamma;\Sigma;\Phi \vdash e_2 : \tau_2 \quad \{\text{write}\} \sqsubseteq_{\sim} \Phi \quad \tau_2 <: \tau_1}{\Gamma;\Sigma;\Phi \vdash \text{assign}\ e_1\ e_2 : \text{Unit}}$$

$$\text{T-App}\ \frac{\Gamma;\Sigma;\Phi \vdash e_1 : (\tau_1 \xrightarrow{\Phi_1} \tau_3) \quad \Gamma;\Sigma;\Phi \vdash e_2 : \tau_2 \quad \tau_2 <: \tau_1 \quad \Phi_1 \sqsubseteq_{\sim} \Phi}{\Gamma;\Sigma;\Phi \vdash e_1\ e_2 : \tau_3}$$

$$\text{T-With-Effects}\ \frac{\Gamma;\Sigma;\Phi_1 \vdash e : \tau \quad \Phi_1 \sqsubseteq_{\sim} \Phi}{\Gamma;\Sigma;\Phi \vdash \text{with-effects}\ \Phi_1\ e : \tau}$$

Intermediate Language

Programs in the Gradual Effect Fluent Language are translated into an intermediate language where the privilege restrictions that could not be statically verified are enforced, and privileges which were uncertain are checked.

Intermediate Language

$$\text{C-Ref}\ \frac{\Gamma; \Sigma; \Phi \vdash e \Rightarrow e' : \tau \qquad \{\text{alloc}\} \sqsubseteq_{\sim} \Phi \qquad \{\text{alloc}\} \subseteq \Phi}{\Gamma; \Sigma; \Phi \vdash \text{ref}\ e \Rightarrow \text{ref}\ e' : \text{Ref}\ \tau}$$

$$\text{C-Ref-¿}\ \frac{\Gamma; \Sigma; \Phi \vdash e \Rightarrow e' : \tau \qquad \{\text{alloc}\} \sqsubseteq_{\sim} \Phi \qquad \{\text{alloc}\} \not\subseteq \Phi}{\Gamma; \Sigma; \Phi \vdash \text{ref}\ e \Rightarrow \text{check}\ \{\text{alloc}\}\ \text{ref}\ e' : \text{Ref}\ \tau}$$

$$\mathit{effect\text{-}check?}\ \Phi_1\ \Phi_2\ e = \begin{cases} e & \text{if } \Phi_1 \subseteq \Phi_2 \\ \texttt{check}\ \ \Phi_1\ e & \text{if } \Phi_1 \not\subseteq \Phi_2 \end{cases}$$

$$\text{C-Ref}\ \frac{\Gamma; \Sigma; \Phi \vdash e \Rightarrow e' : \tau \qquad \{\mathsf{alloc}\} \sqsubseteq \Phi}{\Gamma; \Sigma; \Phi \vdash \texttt{ref}\ \ e \Rightarrow \mathit{effect\text{-}check?}\ \ \{\mathsf{alloc}\}\ \Phi\ \texttt{ref}\ \ e' : \mathsf{Ref}\ \tau}$$

$$\text{C-Deref}\ \frac{\Gamma;\Sigma;\Phi \vdash e \Rightarrow e' : \mathsf{Ref}\ \tau \qquad \{\mathsf{read}\} \sqsubseteq_{\sim} \Phi}{\Gamma;\Sigma;\Phi \vdash \texttt{deref}\ \ e \Rightarrow \textit{effect-check?}\ \ \{\mathsf{read}\}\ \Phi\ \texttt{deref}\ \ e' : \tau}$$

$$\text{C-Assign}\ \frac{\Gamma;\Sigma;\Phi \vdash e_1 \Rightarrow e_1' : \mathsf{Ref}\ \tau_1 \qquad \Gamma;\Sigma;\Phi \vdash e_2 \Rightarrow e_2' : \tau_2 \qquad \{\mathsf{write}\} \sqsubseteq_{\sim} \Phi \qquad \tau_2 <: \tau_1}{\begin{array}{l}\Gamma;\Sigma;\Phi \vdash \texttt{assign}\ \ e_1\ e_2 \Rightarrow \\ \textit{effect-check?}\ \ \{\mathsf{write}\}\ \Phi\ \texttt{assign}\ \ e_1'\ e_2' : \mathsf{Unit}\end{array}}$$

$$\textit{effect-enforce?}\ \Phi_1\ \Phi_2\ e = \begin{cases} e & \text{if } \Phi_1 \subseteq \Phi_2 \\ \texttt{enforce}\ \ \Phi_1\ e & \text{if } \Phi_1 \not\subseteq \Phi_2 \end{cases}$$

$$\text{C-With-Effects}\ \frac{\Gamma; \Sigma; \Phi_1 \vdash e \Rightarrow e' : \tau \qquad \Phi_1 \sqsubseteq_{\sim} \Phi}{\Gamma; \Sigma; \Phi \vdash \quad \texttt{with-effects}\ \ \Phi_1\ e \Rightarrow \textit{effect-enforce?}\ \ \Phi_1\ \Phi\ e' : \tau}$$

Metatheory

- The intermediate language is Type Safe (Progress and Preservation)
- Programs from the Fluent language get the same type in Gradual Effect Fluent
- Programs translated from the Fluent Language into the intermediate language have neither “check” nor “enforce” constructs.

Accepting programs without annotations

We would also like to accept programs of the form

$$\lambda f \,.\, \texttt{with-effects}\ \emptyset\ (f\ \texttt{unit}) \tag{10}$$

We can achieve this by incorporating Gradual Typing.

Gradual Typing

- Unannotated elements receive the type ?.
- There is a notion of type consistency which defines how type equality can be relaxed.
- There is an intermediate language to which programs are translated to incorporate runtime verifications (casts).

Type Consistency

In Siek/Taha's Gradual Typing, Type Consistency is defined as:

$$\text{C-Refl}\ \frac{}{\tau \sim \tau} \qquad \text{C-Fun}\ \frac{\sigma_1 \sim \tau_1 \quad \sigma_2 \sim \tau_2}{\sigma_1 \to \sigma_2 \sim \tau_1 \to \tau_2}$$

$$\text{C-UnR}\ \frac{}{\tau \sim ?} \qquad \text{C-UnL}\ \frac{}{? \sim \tau}$$

Type Consistency

To define type consistency, we need a notion of consistent equivalence of privilege sets:

$$a \simeq b \iff a \sqsubseteq_{\sim} b \wedge b \sqsubseteq_{\sim} a$$

$$\text{C-Fun}\ \frac{\sigma_1 \sim \tau_1 \qquad \sigma_2 \sim \tau_2 \qquad \Phi_\sigma \simeq \Phi_\tau}{\sigma_1 \xrightarrow{\Phi_\sigma} \sigma_2 \sim \tau_1 \xrightarrow{\Phi_\tau} \tau_2}$$

Typing rules

$$\text{G-Deref}\ \frac{\Gamma;\Sigma;\Phi \vdash e : \sigma \qquad \sigma \sim \mathsf{Ref}\ \tau \qquad \{\mathsf{read}\} \sqsubseteq_{\sim} \Phi}{\Gamma;\Sigma;\Phi \vdash \mathtt{deref}\ \ e : \tau}$$

$$\text{G-Assign}\ \frac{\Gamma;\Sigma;\Phi \vdash e_1 : \sigma \qquad \sigma \sim \mathsf{Ref}\ \tau_1 \qquad \Gamma;\Sigma;\Phi \vdash e_2 : \tau_2 \qquad \tau_2 \lesssim \tau_1 \qquad \{\mathsf{write}\} \sqsubseteq_{\sim} \Phi}{\Gamma;\Sigma;\Phi \vdash \mathtt{assign}\ \ e_1\ e_2 : \mathsf{Unit}}$$

$$\text{G-App}\ \frac{\Gamma;\Sigma;\Phi \vdash e_1 : \sigma \qquad \sigma \sim (\tau_1 \xrightarrow{\Phi_1} \tau_3) \qquad \Gamma;\Sigma;\Phi \vdash e_2 : \tau_2 \qquad \tau_2 \lesssim \tau_1 \qquad \Phi_1 \sqsubseteq_{\sim} \Phi}{\Gamma;\Sigma;\Phi \vdash e_1\ e_2 : \tau_3}$$

Consistent Subtyping

As in Siek/Taha's "Gradual Typing for Objects" paper, we require to consider subtyping.

$$\sigma \lesssim \tau \equiv \sigma|_\tau <: \tau|_\sigma$$

Consistent Subtyping

Consistent Subtyping is defined in terms of Standard Subtyping plus a "forget" operator "$|$":

$$\sigma \lesssim \tau \equiv \sigma|_\tau <: \tau|_\sigma$$

$$\sigma|_\tau = \begin{cases} ? & \text{if } \tau = \ ? \\ \sigma_1|_{\tau_1} \xrightarrow{\Phi_1} \sigma_2|_{\tau_2} & \text{if } \sigma = \sigma_1 \xrightarrow{\Phi_1} \sigma_2 \wedge \tau = \tau_1 \xrightarrow{\Phi_2} \tau_2 \\ \sigma & \text{otherwise} \end{cases}$$

Typing rules

$$\text{G-Deref}\ \frac{\Gamma;\Sigma;\Phi \vdash e : \sigma \qquad \sigma \sim \mathsf{Ref}\ \tau \qquad \{\mathsf{read}\} \sqsubseteq_{\sim} \Phi}{\Gamma;\Sigma;\Phi \vdash \mathtt{deref}\ \ e : \tau}$$

$$\text{G-Assign}\ \frac{\Gamma;\Sigma;\Phi \vdash e_1 : \sigma \qquad \sigma \sim \mathsf{Ref}\ \tau_1 \qquad \Gamma;\Sigma;\Phi \vdash e_2 : \tau_2 \qquad \tau_2 \lesssim \tau_1 \qquad \{\mathsf{write}\} \sqsubseteq_{\sim} \Phi}{\Gamma;\Sigma;\Phi \vdash \mathtt{assign}\ \ e_1\ e_2 : \mathsf{Unit}}$$

$$\text{G-App}\ \frac{\Gamma;\Sigma;\Phi \vdash e_1 : \sigma \qquad \sigma \sim (\tau_1 \xrightarrow{\Phi_1} \tau_3) \qquad \Gamma;\Sigma;\Phi \vdash e_2 : \tau_2 \qquad \tau_2 \lesssim \tau_1 \qquad \Phi_1 \sqsubseteq_{\sim} \Phi}{\Gamma;\Sigma;\Phi \vdash e_1\ e_2 : \tau_3}$$

Typing rules

$$\text{G-Deref}\ \frac{\Gamma;\Sigma;\Phi \vdash e : \sigma \qquad \sigma \sim \mathsf{Ref}\ \tau \qquad \{\mathsf{read}\} \sqsubseteq_{\sim} \Phi}{\Gamma;\Sigma;\Phi \vdash \mathtt{deref}\ \ e : \tau}$$

$$\text{G-Assign}\ \frac{\Gamma;\Sigma;\Phi \vdash e_1 : \sigma \qquad \sigma \sim \mathsf{Ref}\ \tau_1 \qquad \Gamma;\Sigma;\Phi \vdash e_2 : \tau_2 \qquad \tau_2 \lesssim \tau_1 \qquad \{\mathsf{write}\} \sqsubseteq_{\sim} \Phi}{\Gamma;\Sigma;\Phi \vdash \mathtt{assign}\ \ e_1\ e_2 : \mathsf{Unit}}$$

$$\text{G-App}\ \frac{\Gamma;\Sigma;\Phi \vdash e_1 : \sigma \qquad \sigma \sim (\tau_1 \xrightarrow{\Phi_1} \tau_3) \qquad \Gamma;\Sigma;\Phi \vdash e_2 : \tau_2 \qquad \tau_2 \lesssim \tau_1 \qquad \Phi_1 \sqsubseteq_{\sim} \Phi}{\Gamma;\Sigma;\Phi \vdash e_1\ e_2 : \tau_3}$$

Intermediate Language

The intermediate language for Gradual Fluent only needs one extra cast, besides those proposed by Siek/Taha. No extra privilege checks or enforcements.

$$\text{C-App}\ \frac{\Gamma;\Sigma;\Phi \vdash e_1 \Rightarrow e_1' : ? \qquad \Gamma;\Sigma;\Phi \vdash e_2 \Rightarrow e_2' : \sigma}{\Gamma;\Sigma;\Phi \vdash e_1\ e_2 \Rightarrow \left(\langle \sigma \xrightarrow{\Phi} \tau \Leftarrow ? \rangle\ e_1'\right) e_2' : \tau}$$

Future Work

- We finally want to add Gradual Typing and Gradual Effect Checking to Generic Type-and-Effect Systems.
  - The Fluent Language we worked on is an instance of the Generic Framework proposed by Marino and Millstein 2009.

Questions.