Substructural Proof/Type Theory

CAS CS 392: Rust, in Theory and in Practice

March 18, 2025 (Lecture 13)

Outline

Recap

Structural Rules

Linear Types

Intuitionistic Propositional Logic

Syntax:

$$V ::= p \mid q \mid r \ldots$$
$$T ::= V \mid \bot \mid T \to T \mid T \wedge T \mid T \vee T$$

Proof System:

modus ponens

$$\frac{}{\Gamma, \phi, \Delta \vdash \phi} \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash \phi} \qquad \boxed{\frac{\Gamma \vdash \phi \to \psi \qquad \Gamma \vdash \phi}{\Gamma \vdash \psi}}$$

$$\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \to \psi} \qquad \boxed{\frac{\Gamma \vdash \phi \qquad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi}} \qquad \frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \phi} \qquad \frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \phi}$$

$$\frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi} \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi} \qquad \frac{\Gamma \vdash \phi \vee \psi \qquad \Gamma, \phi \vdash \xi \qquad \Gamma, \psi \vdash \xi}{\Gamma \vdash \xi}$$

Untyped Lambda Calculus

Python:

lamba (x): M

f(M)

Syntax:

$$V ::= x \mid y \mid z \ldots$$
$$T ::= V \mid \lambda V.T \mid TT$$

Small-Step Semantics:

$$\frac{M \longrightarrow M'}{\lambda x.M \longrightarrow \lambda x.M'} \qquad \frac{M \longrightarrow M'}{MN \longrightarrow M'N} \qquad \frac{N \longrightarrow N'}{MN \longrightarrow MN'}$$

$$\frac{}{(\lambda x.M)N \longrightarrow M[N/x]}$$

$$(\lambda x.\,\boxed{x} + 1)\,\boxed{(4+5)} \longrightarrow (4+5) + 1$$

$$\Omega = (\lambda x.\,x\,x)(\lambda x.\,x\,x) \longrightarrow \Omega$$

Simply Typed Lambda Calculus (STLC)

Syntax:

$$\begin{aligned} V_{Ty} &::= a \mid b \mid c \dots \\ Ty &::= V_{Ty} \mid \bot \mid Ty \to Ty \\ V_T &::= x \mid y \mid z \dots \\ T &::= V_T \mid \lambda V.T \mid TT \end{aligned}$$

$$\cfrac{\cfrac{\cfrac{}{\cancel{x : A \vdash x : A \to B}} \qquad \cfrac{}{x : A \vdash x : A}}{x : A \vdash xx : B}}{\emptyset \vdash \lambda x.xx : A \to B}$$

Type System:

$$\frac{}{\Gamma, x : A, \Delta \vdash x : A} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}$$

Curry-Howard Isomorphism

STLC Type System:

Brouwer - Heyting - Kolmogrov Interp.

$$\frac{}{\Gamma, x : A, \Delta \vdash x : A} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}$$

IPL Proof System:

$$\frac{}{\Gamma, \phi, \Delta \vdash \phi} \qquad \frac{\Gamma \vdash \phi \to \psi \qquad \Gamma \vdash \phi}{\Gamma \vdash \psi}$$

$$\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \to \psi}$$

STLC+ (Syntax)

$T * T$ ('a, 'b) result

$$
\begin{aligned}
V_{Ty} &::= a \mid b \mid c \ldots \\
Ty &::= V_{Ty} \mid \bot \mid \top \mid Ty \to Ty \mid T \wedge T \mid T \vee T \\
V_T &::= x \mid y \mid z \ldots \\
T &::= V_T \mid \lambda V.T \mid TT \mid \langle T, T \rangle \mid \pi_1(T) \mid \pi_2(T) \\
&\quad \mid \iota_1(T) \mid \iota_2(T) \mid \text{case } T \text{ of } \iota_1(V) \to T; \iota_2(V) \to T \\
&\quad \mid \text{explode(M)} \mid \bullet
\end{aligned}
$$

STLC+ (Product Types)

$$Ty ::= Ty \wedge Ty$$

$$T ::= \langle T, T \rangle \mid \pi_0(T) \mid \pi_1(T)$$

Intro:

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \wedge B}$$

elimination?

$$\frac{\Gamma \vdash p : A \wedge B}{\Gamma \vdash \pi_0(p) : A} \qquad \frac{\Gamma \vdash p : A \wedge B}{\Gamma \vdash \pi_1(p) : B}$$

$$\pi_0(\langle M, N \rangle) \to M$$

$$\pi_1(\langle M, N \rangle) \to N$$

$$\frac{\Gamma \vdash \phi \qquad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi}$$

STLC+ (Product Types)

$$Ty ::= A \wedge B$$

$$T ::= \langle A, B \rangle \mid \text{match } T \text{ with}$$
$$\mid \langle v, v \rangle \to T$$

$$\frac{\Gamma \vdash p : A \wedge B \qquad \Gamma, x : A, y : A \vdash M : C}{\Gamma \vdash \text{match } p \text{ with } \mid (x, y) \to M : C}$$

STLC+ (Union Types)

$$Ty ::= A \vee B$$

Ok   Error

$$T ::= \iota_1(T) \mid \iota_2(T) \mid \text{match } T \text{ with } \mid \iota_1(V) \to T \mid \iota_2(V) \to T$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \iota_0(M) : A \vee B}$$

$$\frac{\Gamma \vdash N : B}{\Gamma \vdash \iota_1(N) : A \vee B}$$

$$\frac{\Gamma \vdash M : A \vee B \quad \Gamma, x : A \vdash N_1 : C \quad \Gamma, x : B \vdash N_2 : C}{\Gamma \vdash \text{match } M \text{ with} \mid \iota_0(x) \to N_1 \mid \iota_1(x) \to N_2}$$

$$\text{match } \iota_0(M) \text{ with } \mid \iota_0(x) \to N_1 \mid \iota_1(x) \to N_2$$

$$\downarrow$$

$$N_1[M/x]$$

STLC+ (Unit type and Empty Type)

Example

$$\overbrace{\boxed{f : A \to B, g : A \to C, x : A}}^{\Gamma} \vdash \langle fx, gx \rangle : B \wedge C$$

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash f : A \to B} \quad \dfrac{}{\Gamma \vdash x : A}}{\Gamma \vdash f\, x : B} \qquad \dfrac{\dfrac{}{\Gamma \vdash g : A \to C} \quad \dfrac{}{\Gamma \vdash x : A}}{\Gamma \vdash g\, x : C}}{f : A \to B,\ g : A \to C,\ x : A \vdash \langle f x, g x \rangle : B \wedge C}$$

Aside: Proof Reduction

Proofs can have unnecessary parts, e.g., building a pair only to immediately destruct it

This is also related to the notion of *cut-elimination*, an important topic in the area of proof theory

Proof reduction corresponds to *evaluation* in the CH isomorphism

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \overset{M}{\phi} \qquad \Gamma \vdash \overset{N}{\psi}}{\Gamma \vdash \overset{\langle M,N\rangle}{\phi \wedge \psi}}}{\overset{\pi_2(M,N)}{\Gamma \vdash \psi}} \qquad \vdots}{\vdots} \quad \Longrightarrow \quad \dfrac{\overset{N}{\Gamma \vdash \psi} \qquad \vdots}{\vdots}$$

$$\pi_1(\langle M, N \rangle) \longrightarrow N$$

Theme of the Day

A type system "draws a circle" around a class of programs with nice properties, which often manifest in the *semantics*

Theme of the Day

A type system "draws a circle" around a class of programs with nice properties, which often manifest in the *semantics*

Type systems open possibilities to better semantics

Theme of the Day

A type system "draws a circle" around a class of programs with nice properties, which often manifest in the *semantics*

Type systems open possibilities to better semantics

Rust, for example, can avoid using a garbage collector, not because you can write drastically different programs than in C, but because it restricts the kinds of C-like programs you can write

Outline

Recap

Structural Rules

Linear Types

Assumptions

$$\frac{}{\Gamma, x : A, \Delta \vdash x : A}$$

The assumption rule is actually doing quite a bit of heavy lifting. In our system, we *cannot* add variables to our context mid-proof

Assumptions

$$\overline{\Gamma, x : A, \Delta \vdash x : A}$$

The assumption rule is actually doing quite a bit of heavy lifting. In our system, we *cannot* add variables to our context mid-proof

This is not a huge problem, we can change our contexts in the *meta-theory*

Admissible Rules

A rule is **admissible** or **derivable** if adding the rule to the system does not change what judgments can be derived

Lemma. If $\Gamma \vdash M : B$ and $x \notin \Gamma$, then $\Gamma, x : A \vdash M : B$.

Proof. Induction on derivations (applied to $\Gamma \vdash M : B$)

$\boxed{\text{Func. App.}}$

$$\frac{\Gamma \vdash M' : C \to B \qquad \Gamma \vdash N' : C}{\Gamma \vdash \boxed{M'N'} : B} \qquad M = M'N'$$

By IH:

$$\frac{\Gamma, x : A \vdash M' : C \to B \qquad \Gamma, x : A \vdash N' : C}{\Gamma, x : A \vdash M'N' : B}$$

by rule:

$\Gamma, y : C, x : A \vdash M' : D$

$\boxed{\text{Abs.}}$

$$\frac{\Gamma, y : C \vdash M' : D}{\Gamma \vdash \lambda y . M' : C \to D} \qquad \begin{array}{l} M = \lambda x . M' \\ B = C \to D \end{array}$$

Structural Rules

Structural rules allow us to change the state of our context mid-proof

All structural rules are admissible in STLC (and IPL)

$\boxed{\text{Wkn}}$

$$\frac{\Gamma \vdash M : A}{\Gamma, x : B \vdash M : A}$$

$\boxed{\text{Exchange}}$

$$\frac{\Gamma, x : A, y : B, \Delta \vdash M : C}{\Gamma, y : B, x : A, \Delta \vdash M : C}$$

$\boxed{\text{Contraction}}$

$$\frac{\Gamma, x : A, y : A \vdash M : B}{\Gamma, x : A \vdash M[x/y] : B}$$

Alternative System

We can rewrite the type system to include structural rules instead of the assumptions rule

$$\frac{}{x : A \vdash x : A}$$

$$\frac{\Gamma \vdash M : A}{\Gamma, x : B \vdash M : A}$$

$$\frac{\Gamma, x : A, y : B, \Delta \vdash M : A}{\Gamma, y : B, x : B, \Delta \vdash M : A}$$

$$\frac{\Gamma, x : A, y : A \vdash M : B}{\Gamma, x : A \vdash M[y/x] : B}$$

other rules

Substructural Logics

Once we write are system to have structural rules, we have a degree of freedom to define *new systems*

Substructural logics/type systems disallow certain structural rules

System	Weakening	Contraction	Variable use
Unrestricted	yes	yes	any number of times
Affine	yes	no	at most once
Relevant	no	yes	at least once
Linear	no	no	exactly once

Outline

Recap

Structural Rules

Linear Types

Linearity in Rust

We cannot use a variable more than once (without references):

```
// This does not compile
fn dup<T>(t: T) -> (T, T) {
  (t, t)
}
```

: Copy

clone

Linearity in Rust

We cannot use a variable more than once (without references):

```
// This does not compile
fn dup<T>(t: T) -> (T, T) {
  (t, t)
}
```

Rust without references is *linear*

Linearity in Rust

We can't implement the example from before:

```rust
// This does not compile
fn example<T, U, V, F, G>(f: F, g: G, x: T) -> (U, V)
where
    F : Fn(T) -> U,
    G : Fn(T) -> V,
{
    (f(x), g(x))
}
```

Question. How can we fix this?

Linear Logic

"**Truth is free**. Having proved a theorem, you may use this proof as many times as you wish, at no extra cost. **Food, on the other hand, has a cost.** Having baked a cake, you may eat it only once. If traditional logic is about truth, then linear logic is about food." (Walder)

Jean-Yves Girard introduced linear logic in the 80s as a *resource-sensitive* logic which made explicit certain dualities between classical and intuitionsitic logic

It is now most commonly used in PL and Quantum

Linear Typed λ-Calculus (LTLC)

Syntax:

$$
\begin{aligned}
V_{Ty} &::= a \mid b \mid c \ldots \\
Ty &::= V_{Ty} \mid \bot \mid Ty \multimap Ty \\
V_T &::= x \mid y \mid z \ldots \\
T &::= V_T \mid \lambda V.T \mid TT
\end{aligned}
$$

Type system:

start

exchange

structural

$$\boxed{\frac{}{x : A \vdash x : A}} \qquad \boxed{\frac{\Gamma \vdash M : A}{\pi(\Gamma) \vdash M : A}}$$

$$\frac{\Gamma \vdash M : A \multimap B \qquad \Delta \vdash N : A}{\Gamma, \Delta \vdash MN : B} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \multimap B}$$

Example

$$\vdash \lambda f.\lambda x.fx : (A \multimap B) \multimap A \multimap B$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f : A \multimap B \; : \; f : A \multimap B} \qquad \dfrac{}{x : A \vdash x : A}}{f : A \multimap B,\; x : A \vdash f\,x : B}}{\vdots}}{\vdash \lambda f.\lambda x.\,f\,x : (A \multimap B) \multimap A \multimap B}$$

Non-Example

$$\vdash \lambda x.\lambda y.x : A \multimap B \multimap A$$

$$\dfrac{\dfrac{\begin{array}{c}\cancel{\vdots} \\ x : A, y : B \vdash x : A\end{array}}{\vdots}}{\vdash \lambda x.\lambda y.x : A \multimap B \multimap A}$$

The Key Lemma

Lemma. If $\Gamma \vdash M : A$ then

- x is free in M if and only if x appears in Γ
- each free variables appears exactly *once* in M

This is what allows us to develop semantics which allow for a unique pointer to the heap (more on that next week)

LTLC+

It is natural to want more data types in LTLC

Furthermore, we might also want to *combine* linearity and nonlinerity (as is done in Rust)

In the reading, Wadler introduces Girard's *Logic of Unity* as a way of combining these ideas

LTLC+ (Intuitionistic Assumptions)

LTLC+ (Unlimited Resources)

LTLC+ (Sum Types)

LTLC+ (Product Types)

Linearity in Rust

If Rust was *really* linear this would not be possible:

```
fn proj<S, T>(p: (S, T)) -> S {
    p.0
}
```

Linearity in Rust

If Rust was *really* linear this would not be possible:

```
fn proj<S, T>(p: (S, T)) -> S {
    p.0
}
```

This code is morally equivalent to:

```
fn proj<S, T>(p: (S, T)) -> S {
    let out = p.0;
    drop(p.1);
    out
}
```

`drop` is also non-linear. Is `drop` as implicit? Is `drop` a language construct? (more on that in this week's assignment)

Closing Remarks: Array Updates

Closing Remarks: Revisiting Intuitionistic Assumptions