Lecture Notes on Linear Natural Deduction

15-417/817: HOT Compilation
Sophia Roshal

Lecture 3
January 21, 2025

1 Introduction

So far, we have seen type checking for Sax [DeYoung et al., 2020], an intermediate language, and had a bit of experience writing code in it (hopefully enough experience that you are on board with not wanting to hand-write a lot of Sax code). The goal of today's lecture is to introduce our front-end functional programming language (or at least part of it). We will for now stick to just the positive types and remain purely linear. This is a fragment of adjoint natural deduction [Jang et al., 2024].

2 Terms and Base Typing Rules

We first review the types that we have so far:

$$A ::= 1 \mid A \otimes B \mid \oplus_{\ell \in L} \{\ell : A_\ell\}$$

We must now find terms that correspond to each type. Unlike in Sax, where all the terms were reads and writes, we are now in a functional programming language, and so our terms will no longer have direct correlation to memory. We go through the types one-by-one, assigning terms and constructing their typing rules. Because we are now in a natural deduction setting and not in sequent calculus, our rules will be a bit different. Rather than left and right rules (or, in the case of Sax, left rules and axioms), we now have introduction and elimination rules. We think of introduction rules as introducing or constructing a term and of elimination rules as eliminating or destructing a term.

First, let's define the terms for 1. We write the value of type 1 as $()$, and it has the following introduction rule:

$$\frac{}{\cdot \vdash () : 1}\ \mathbf{1}I$$

Note that the context that we check the unit in must be empty in a linear system. To destruct the term, as with all positive types, we use a **match** as follows:

$$\frac{\Gamma \vdash e : 1 \quad \Delta \vdash e' : C}{\Gamma, \Delta \vdash \mathbf{match}\ e\ \mathbf{with}\ () \Rightarrow e' : C}\ \mathbf{1}E$$

Γ and Δ must be disjoint to ensure that no variable in the context is used more than once. Next, we define the terms for $\otimes$. Again, as is usual, this corresponds to a pair of terms. The rules are as follows:

$$\frac{\Gamma_1 \vdash e_1 : A_1 \quad \Gamma_2 \vdash e_2 : A_2}{\Gamma_1, \Gamma_2 \vdash (e_1, e_2) : A_1 \otimes A_2} \otimes I \qquad \frac{\Gamma \vdash e : A_1 \otimes A_2 \quad \Delta, x_1 : A_1, x_2 : A_2 \vdash e' : C}{\Gamma, \Delta \vdash \mathbf{match}\ e\ \mathbf{with}\ (x_1, x_2) \Rightarrow e' : C} \otimes E$$

Again, to ensure linearity, contexts must be disjoint.

For sums, we have the following rules:

$$\frac{k \in L \quad \Gamma \vdash e : A_k}{\Gamma \vdash k\ e : \oplus\{\ell : A_\ell\}_{\ell \in L}} \oplus I \qquad \frac{\Gamma \vdash e : \oplus\{\ell : A_\ell\}_{\ell \in L} \quad (\Delta, x : A_\ell \vdash e_\ell : C) \quad (\forall \ell \in L)}{\Gamma, \Delta \vdash \mathbf{match}\ e\ \mathbf{with}\ \{\ell(x) \Rightarrow e_\ell\}_{\ell \in L} : C} \oplus E$$

Lastly, we need a rule for checking variables:

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

Note that the context here must only contain x and no other variables, again, this is to ensure linearity. If other variables were present, then we would be able to pretend they had been used.

3 Typechecking

While the above rules are all correct, they do not yet give us a way to actually implement a type-checking algorithm. There are two parts missing here. Part 1: How do we actually know what types we should be checking terms at? Part 2: As with Sax type checking, the context splits here are nondeterministic, so how do we change that? We start with addressing the first of these concerns using bidirectional type checking [Dunfield and Krishnaswami, 2022].

3.1 Bidirectional Type Checking

The main change from the previous rules to bidirectional rules is that we split the judgment $e : A$ into two as follows:

$$e \Longleftarrow A$$

$$s \Longrightarrow A$$

We read the first of these as e checks against A and the second as e synthesizes A. When thinking about this from the implementation standpoint, we think of both the term and the type being input in the first case, and we verify that the term e does in fact have the type A, and we think of only the term being input in the second case, and we try to figure out (synthesize) the type for the term. In general, in all our introduction rules, the principal term will be a checked, and in all our elimination rules, the principal term will synthesize. There is then some ambiguity about what happens to the other pieces. Let's walk through all of these. As before, we begin with unit. The introduction rule is straight forward:

$$\frac{}{\cdot \vdash (\,) \Longleftarrow 1}\ \mathbf{1}I$$

For the elimination rule, we start with what we know:

$$\frac{\Gamma \vdash s \Longrightarrow 1 \quad \Delta \vdash e' \ ?\ C}{\Gamma, \Delta \vdash \mathbf{match}\ s\ \mathbf{with}\ (\,) \Rightarrow e' \ ?\ C}\ \mathbf{1}E$$

There is now ambiguity about whether the other two judgments should be checking or synthesis. From a computational standpoint, either would make sense, but to simplify type checking, making these checking judgments helps a little bit. This is because when we run type checking, we want to first apply all the introduction rules we can, followed by the elimination rules. If we are at a boundary point where we have to start applying elimination rules, we are still in a checking direction, and so it would be helpful to not have to do something to move directions. Another reason to keep these as checking judgments is related to proof theory and the relationship between the sequent calculus and natural deduction. We make anything that would appear to the right of the turnstile in sequent calculus a checking judgment, while anything that would appear to the left of the turnstile a synthesizing one. This has the consequence that bidirectional typing checks exactly the *normal forms*, that is, terms that cannot be directly reduced. Therefore the top-level definitions of metavariables are the initial source of computation.

The upshot is that we have the following rule:

$$\frac{\Gamma \vdash s \Longrightarrow \mathbf{1} \quad \Delta \vdash e' \Longleftarrow C}{\Gamma, \Delta \vdash \mathbf{match}\ s\ \mathbf{with}\ (\,) \Rightarrow e' \Longleftarrow C}\ \mathbf{1}E$$

Using similar reasoning, we derive (almost) the rest of the bidirectional rules:

$$\frac{\Gamma_1 \vdash e_1 \Longleftarrow A_1 \quad \Gamma_2 \vdash e_2 \Longleftarrow A_2}{\Gamma_1, \Gamma_2 \vdash (e_1, e_2) \Longleftarrow A_1 \otimes A_2}\otimes I \qquad \frac{\Gamma \vdash s \Longrightarrow A_1 \otimes A_2 \quad \Delta, x_1 : A_1, x_2 : A_2 \vdash e' \Longleftarrow C}{\Gamma, \Delta \vdash \mathbf{match}\ s\ \mathbf{with}\ (x_1, x_2) \Rightarrow e' \Longleftarrow C}\otimes E$$

$$\frac{(k \in L) \quad \Gamma \vdash e \Longleftarrow A_k}{\Gamma \vdash k(e) \Longleftarrow \oplus\{\ell : A_\ell\}_{\ell \in L}}\oplus I \qquad \frac{\Gamma \vdash s \Longrightarrow \oplus\{\ell : A_\ell\}_{\ell \in L} \quad (\Delta, x_\ell : A_\ell \vdash e_\ell \Longleftarrow C) \quad (\forall \ell \in L)}{\Gamma, \Delta \vdash \mathbf{match}\ s\ \mathbf{with}\ \{\ell(x_\ell) \Rightarrow e_\ell\}_{\ell \in L} \Longleftarrow C}\oplus E$$

We still need to find the correct direction for the variable rule, and it turns out we need one more rule as well. First, we start with the variable rule. Since we maintain the types of variables at all times variables synthesize their types as follows:

$$\frac{}{x : A \vdash x \Longrightarrow A}\ \mathsf{var}$$

Now lets try and prove a very simple type derivation (assuming we are a magical type checker that knows how to split contexts correctly):

$$\frac{x_1 : A \vdash x_1 \overset{?}{\Longleftarrow} A \quad x_2 : B \vdash x_2 \overset{?}{\Longleftarrow} B}{x_1 : A, x_2 : B \vdash (x_1, x_2) \Longleftarrow A \otimes B}\otimes I$$

We hit a problem here. We are in the checking phase, but have variables, so it seems like we are stuck! We need one more rule:

$$\frac{\Gamma \vdash e \Longrightarrow A}{\Gamma \vdash e \Longleftarrow A}\ \Rightarrow/\Leftarrow$$

Although this rule is correct, to make it more algorithmic, we think about how we would go about implementing this. Remember that synthesis takes only the term as input, not the type, so we don't know for sure that we will synthesize A. So we actually should write

$$\frac{\Gamma \vdash e \Longrightarrow A' \quad A' = A}{\Gamma \vdash e \Longleftarrow A}\ \Rightarrow/\Leftarrow$$

This rule can also easily be modified to take into account subtyping. With this rule we can now finish the partial proof from above by applying this change of direction rule on both sides and then the variable rule.

3.2 Additive Typing

We now address the second issue, building on top of the bidirectional rules. The construction here is very similar to that from Sax. We propagate one *input* context Γ, calculating an output context Ξ which specifies the variables that were used in that derivation. All context operations on these output contexts will remain the same as for SAX.

$$\frac{}{\Gamma \vdash () \Longleftarrow \mathbf{1} \;/\; \cdot}\;\mathbf{1}I \qquad \frac{\Gamma \vdash s \Longrightarrow \mathbf{1} \;/\; \Xi_1 \quad \Gamma \vdash e' \;\Longleftarrow\; C \;/\; \Xi_2}{\Gamma \vdash \mathbf{match}\; s\; \mathbf{with}\; () \Rightarrow e' \;\Longleftarrow\; C \;/\; \Xi_1 \,;\, \Xi_2}\;\mathbf{1}E$$

$$\frac{\Gamma \vdash e_1 \Longleftarrow A_1 \;/\; \Xi_1 \quad \Gamma \vdash e_2 \Longleftarrow A_2 \;/\; \Xi_2}{\Gamma \vdash (e_1, e_2) \Longleftarrow A_1 \otimes A_2 \;/\; \Xi_1 \,;\, \Xi_2}\otimes I$$

$$\frac{\Gamma \vdash s \Longrightarrow A_1 \otimes A_2 \;/\; \Xi_1 \quad \Gamma, x_1 : A_1, x_2 : A_2 \vdash e' \Longleftarrow C \;/\; \Xi_2}{\Gamma \vdash \mathbf{match}\; s\; \mathbf{with}\; (x_1, x_2) \Rightarrow e' \Longleftarrow C \;/\; \Xi_1 \,;\, ((\Xi_2 \setminus x_1) \setminus x_2)}\otimes E$$

$$\frac{(k \in L) \quad \Gamma \vdash e \Longleftarrow A_k \;/\; \Xi}{\Gamma \vdash k\; e \Longleftarrow \oplus\{\ell : A_\ell\}_{\ell \in L} \;/\; \Xi}\oplus I$$

$$\frac{\Gamma \vdash s \Longrightarrow \oplus\{\ell : A_\ell\}_{\ell \in L} \;/\; \Xi_1 \quad (\Gamma, x_\ell : A_\ell \vdash e_\ell \Longleftarrow C \;/\; \Xi_\ell \quad \Xi_\ell \setminus x_\ell = \Xi_2) \quad (\forall \ell \in L)}{\Gamma \vdash \mathbf{match}\; s\; \mathbf{with}\; \{\ell(x_\ell) \Rightarrow e_\ell\}_{\ell \in L} \Longleftarrow C \;/\; \Xi_1 \,;\, \Xi_2}\oplus E$$

$$\frac{}{\Gamma \vdash x \Longrightarrow A \;/\; x : A}\;\mathsf{var} \qquad \frac{\Gamma \vdash e \Longrightarrow A' \;/\; \Xi \quad A' = A}{\Gamma \vdash e \Longleftarrow A \;/\; \Xi}\;\Rightarrow/\Leftarrow$$

Lastly, we need to consider the top level definition rules. Similarly to Sax, we will have a top level signature Σ of definitions that can be mutually recursive. We write

$$\Sigma ::= (\cdot) \mid f[\Delta] : A = e, \Sigma$$

where Δ is a context of variables and their types and represents the inputs to the definition named f. A is the return type of the definition. To make a call to such a definition we need the following type checking rule, which just checks that each of the inputs to the call are the correct type.

$$\frac{f[\Delta] : A = e \in \Sigma \quad \Gamma \vdash \theta \Longleftarrow \Delta \;/\; \Xi}{\Gamma \vdash \mathbf{call}\; f\; \theta \Longrightarrow A \;/\; \Xi}\;\mathsf{call}$$

There are a few new things in this rule. First, θ is a list of expressions whose values will be substituted for variables in Δ during execution. Second, the second premise is not yet fully defined. What does it mean to check a list of expressions against a context? As should be expected, this is just a point-wise definition:

$$\frac{\Gamma \vdash \theta \Longleftarrow \Delta \;/\; \Xi_1 \quad \Gamma \vdash e \Longleftarrow A \;/\; \Xi_2}{\Gamma \vdash (\theta, e) \Longleftarrow (\Delta, x : A) \;/\; \Xi_1 \,;\, \Xi_2} \qquad \frac{}{\Gamma \vdash (\cdot) \Longleftarrow (\cdot) \;/\; (\cdot)}$$

Now, to check the full list of definitions that we have, we need to check each definition one-by-one. To do so, we iterate through all of them with a new judgment sig which just indicates that signature is well typed. The rules are as follows:

$$\frac{\vdash_\Sigma \Sigma'\;\mathsf{sig} \quad \Delta \vdash e \Longleftarrow A \;/\; \Xi \quad \Xi = \Delta}{\vdash_\Sigma \Sigma', f[\Delta] : A = e\;\mathsf{sig}} \qquad \frac{}{\vdash_\Sigma (\cdot)\;\mathsf{sig}}$$

The last premise of the first rule here ensures that all the inputs (Δ) were in fact used. The code that was written during lecture can be found on the website.

References

Henry DeYoung, Frank Pfenning, and Klaas Pruiksma. Semi-axiomatic sequent calculus. In Z. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, pages 29:1–29:22, Paris, France, June 2020. LIPIcs 167.

Jana Dunfield and Neel Krishnaswami. Bidirectional typing. *ACM Computing Surveys*, 54(5):98:1–98:38, 2022.

Junyoung Jang, Sophia Roshal, Frank Pfenning, and Brigitte Pientka. Adjoint natural deduction. In Jakob Rehof, editor, *9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024)*, pages 15:1–15:23, Tallinn, Estonia, July 2024. LIPIcs 299. Extended version available as https://arxiv.org/abs/2402.01428.