Lecture Notes on Ordered Type Checking

15-417/817: HOT Compilation
Frank Pfenning

Lecture 13
February 27, 2025

1 Introduction

In the last two lectures we introduced ordered types and derived some consequences of parametricity for ordered types. The latter used a logical predicate indexed by a monoid, and also showed how to modify it for linear types, which requires a commutative monoid.

We did not yet discuss how to perform type checking for the ordered case. So far it has been perfectly okay (and, in fact, led to simpler code) to keep the context in order even though the type system itself was linear or adjoint. Strangely, when we actually order the context that is no longer possible (or at least I don't know how to ...). In today's lecture we explain why and also propose an algorithm which, as of this writing, we haven't proved complete. It reuses the idea of the monoid from the last lecture for purposes of type checking.

2 The Difficulty

A natural attempt is to apply the additive approach. The subtractive approach has its own complications [Polakow, 2000, 2001]. The main judgment then is $\Gamma \vdash e \iff A \mid \Omega$ with the intent that the output context Ω contains all variables used in their proper order. That is: $\Omega \vdash e \iff A$.

We start with pairs, for which matters still go relatively well.

$$\frac{\Gamma \vdash e_1 \Longleftarrow A \mid \Omega_A \quad \Gamma \vdash e_2 \Longleftarrow B \mid \Omega_B}{\Gamma \vdash (e_1, e_2) \Longleftarrow A \bullet B \mid \Omega_A\, \Omega_B}\ \bullet I$$

Here, context concatenation has to make sure that there are no duplicate variable declarations in Ω_A and Ω_B.

The elimination rule is a bit trickier. After synthesizing a type $A \bullet B$ for e, we do not know how to insert x and y in the correct place in Γ. There may be many variables in Γ that are not used, and it could even be that e does not use any variables at all (for example, it might use a metavariable). So we keep the lexical input context Γ unordered and add x and y to it.

$$\frac{\Gamma \vdash e \Longrightarrow A \bullet B \mid \Omega_{AB} \quad \Gamma, x : A, y : B \vdash e' \Longleftarrow C \mid \Omega'}{\Gamma \vdash \mathbf{match}\ e\ ((x, y) \Rightarrow e') \Longleftarrow C \mid ?}\ \bullet E?$$

What does Ω' have to look like? It should be of the form $\Omega_L\,(x : A)\,(y : B)\,\Omega_R$. Since ordered variables are also linear, the occurrences of x and y must be unique, and we can split Ω' and then insert Ω_{AB} in the middle.

$$\frac{\Gamma \vdash e \Longrightarrow A \bullet B \;/\; \Omega_{AB} \quad \Gamma, x : A, y : B \vdash e' \Longleftarrow C \;/\; \Omega' \quad \Omega' = \Omega_L\,(x : A)\,(y : B)\,\Omega_R}{\Gamma \vdash \mathbf{match}\; e\; ((x, y) \Rightarrow e') \Longleftarrow C \;/\; \Omega_L\,\Omega_{AB}\,\Omega_R}\;\bullet E$$

If Ω' cannot be split in this way then type-checking has to fail.

Unfortunately, the usually harmless unit $\mathbf{1}$ throws a wrench into the works. The introduction rule is fine.

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

Let's propose the elimination rule as for pairs.

$$\frac{\Gamma \vdash e \Longrightarrow \mathbf{1} \;/\; \Omega_1 \quad \Gamma \vdash e' \Longleftarrow C \;/\; \Omega' \quad \Omega' = \Omega_L\,\Omega_R}{\Gamma \vdash \mathbf{match}\; e\; ((\,) \Rightarrow e') \Longleftarrow C \;/\; \Omega_L\,\Omega_1\,\Omega_R}\;\mathbf{1}E?$$

The problem here is that we do not have variables x and y to serve as guides where to split Ω'. In other words, this rule is nondeterministic.

The problem with nondeterministic rules is that in an implementation one might need to backtrack over the choices. In a backtracking type-checker it is generally quite difficult to give good error messages when type-checking fails because we may have many branches, all of which fail and a particular source of the error is difficult to identify.

Often, in lieu of backtracking (when this is even possible) we can collect *constraints* and try to solve them. This is where the monoid structure from the last lecture [Aberlé et al., 2025] comes to the rescue. It is also a reinstantiation of the idea of a constructive resource semantics [Reed and Pfenning, 2010].

3 Resource Tracking

One way to think about a resource semantics is by eliminating the restriction in the structural rules and use algebraic reasoning instead. We would rewrite (first in logical form)

$$A_1 \dots A_n \vdash C \quad \text{as} \quad A_1[\alpha_1], \dots, A_n[\alpha_n] \vdash C[\alpha_1 \cdots \alpha_n]$$

where α_i are taken from a resource algebra, and $\alpha_1 \cdots \alpha_n$ is an term in the resource algebra. It means that C must be proved using exactly the resources $\alpha_1 \cdots \alpha_n$, where each of the hypotheses is labeled with a resource. This would allow us to restore weakening, because

$$B[\beta], A_1[\alpha_1], \dots, A_n[\alpha_n] \vdash C[\alpha_1 \cdots \alpha_n]$$

the hypothesis $B[\beta]$ could simply not be used. Even contraction may be possible, because in

$$A[\alpha], A[\alpha] \vdash C[\alpha]$$

We could use the first or second copy of A to prove C, but not both because $\alpha \cdot \alpha \neq \alpha$.

We show what a few rules might look like, although there are certainly some options.

$$\frac{x : A[p] \in \Gamma}{\Gamma \vdash x \Longrightarrow A[p]}\ \mathsf{var}$$

$$\frac{\Gamma \vdash e_1 \Longleftarrow A[p] \quad \Gamma \vdash e_2 \Longleftarrow B[q]}{\Gamma \vdash (e_1, e_2) \Longleftarrow A \bullet B[p \cdot q]}\ \bullet I$$

$$\frac{\Gamma \vdash e \Longrightarrow A \bullet B[p] \quad \Gamma, x : A[\alpha], y : B[\beta] \vdash e' \Longleftarrow C[q_1 \cdot \alpha \cdot \beta \cdot q_2]}{\Gamma \vdash \mathbf{match}\ e\ ((x, y) \Rightarrow e') \Longleftarrow C[q_1 \cdot p \cdot q_2]}\ \bullet E^{\alpha,\beta}$$

In the $\bullet E^{\alpha,\beta}$ rule, the parameters α and β must be chosen fresh and distinct. We can see the relationship to the formulation with ordered output context because from

$$\Gamma \vdash e \Longleftrightarrow C[p]$$

we can construct Ω from Γ and p so that

$$\Gamma \vdash e \Longleftrightarrow C \mathbin{/} \Omega$$

The more creative we become with the resources, the more difficult it is to establish this relationship.

Forgetting about resources for the moment, the bidirectional typing derivation of an expression is determined by its shape because the rules are *syntax-directed*. The validity of typing for an *ordered* derivation then reduces to the problem of checking equations between resources. In the case of ordered types, these are equations in a monoid where the α_i are parameters, that is, they act like constants. Or one can think of them as being universally quantified. The laws here are just

$$\begin{array}{lcl} p \cdot (q \cdot r) & = & (p \cdot q) \cdot r \\ p \cdot \epsilon & = & p \\ \epsilon \cdot q & = & q \end{array}$$

where ϵ as the unit of the monoid represents the empty resource. If we are working in linear logic, we have to add commutativity to the equations, but otherwise the same construction applies.

How does this help? The idea is that we can also introduce *existentially quantified* variables ω. Instead of just solving ground equations, we now have to find valid instances. For example:

$$\frac{}{\Gamma \vdash (\,) \Longleftarrow \mathbf{1}[\epsilon]}\ \mathbf{1}I \qquad \frac{\Gamma \vdash e \Longrightarrow A \bullet B[p] \quad \Gamma \vdash e' \Longleftarrow C[q] \quad \exists \omega_L, \omega_R.\, q = \omega_L \cdot \omega_R}{\Gamma \vdash \mathbf{match}\ e\ ((\,) \Rightarrow e') \Longleftarrow C[\omega_L \cdot p \cdot \omega_R]}\ \mathbf{1}E$$

It is now a straightforward matter to rewrite the remaining bidirectional rules for ordered types. We can then collect all the equations and try to solve them. We show a few more.

$$\frac{x : A[p] \in \Gamma}{\Gamma \vdash x \Longrightarrow A[p]}\ \mathsf{var}$$

$$\frac{\Gamma, x : A[\alpha] \vdash e \Longleftarrow B[p] \quad \exists \omega.\, p = \omega \cdot \alpha}{\Gamma \vdash \lambda x.\, e \Longleftarrow A \twoheadrightarrow B[\omega]}\ \twoheadrightarrow I^{\alpha} \qquad \frac{\Gamma \vdash e_1 \Longrightarrow A \twoheadrightarrow B[p] \quad \Gamma \vdash e_2 \Longleftarrow A[q]}{\Gamma \vdash e_1\, e_2 \Longrightarrow B[p \cdot q]}\ \twoheadrightarrow E$$

In the rule $\twoheadrightarrow I$ there is a check that is swept under the rug: we need to know that α is fresh, that is, α does not occur in ω (after a solution for ω has been found). In the implementation we do not explicitly check this, because we make a first pass with the previously described type-checking algorithm that guarantees that the term is *linearly well-typed*. This is only possible if $\alpha \notin \omega$, so this test is omitted here. A better description would quantify over α, as in $\exists\omega.\, \forall\alpha.\, p = \omega \cdot \alpha$. But this would complicate the constraint simplification algorithm described in the next section.

4 Constraint Solving

We have implemented a simple constraint solving algorithm that assumes that the expression has already been linearly type-checked before. We do not know whether this algorithm is either sound or complete. Overall, the set of equations is a conjunction of equations $p = q$, for monoid terms p and q containing parameters α and variables ω.

Substitution. A constraint $\omega = p \wedge E$ where $\omega \notin p$ is rewritten as $[p/\omega]E$. This reduces the number of variables.

Reflexivity. A constraint $p = p \wedge E$ is rewritten as E. This reduces the number of equations.

Splitting. A constraint $p_1 \cdot \alpha \cdot p_2 = q_1 \cdot \alpha \cdot q_2 \wedge E$ is rewritten as $p_1 = q_1 \wedge p_2 = q_2 \wedge E$. This reduces the number of parameters α occurring in the equations. We can also say that we replace one equation by two smaller ones.

Emptiness. A constraint $\epsilon = p \cdot q \wedge E$ is simplified to $\epsilon = p \wedge \epsilon = q \wedge E$. This replaces one equation by two smaller equations.

These are applied in the given order (although we don't know whether this is significant). Also, splitting is applied nondeterministically, that is, there may be multiple candidates and one is picked arbitrarily.

Equations such as $\epsilon = \alpha \cdot p$ or $\epsilon = p \cdot \alpha$ or $\alpha \cdot p = \beta \cdot q$ or $p \cdot \alpha = q \cdot \beta$ for $\alpha \neq \beta$ have no solution due to the implicit universal quantification over the parameters α and β and are marked as inconsistent.

It is conceivable that we end up with an equation such as $\omega_1 \cdot \omega_2 = \omega_3 \cdot \omega_4$ which has solutions but cannot be simplified to the empty constraint. We currently just report an indefinite constraint (type-checking neither succeeds nor fails), but we haven't encountered one in our examples.

References

C. B. Aberlé, Chris Martens, and Frank Pfenning. Substructural parametricity. Submitted, February 2025. URL http://www.cs.cmu.edu/~fp/papers/ordered25.pdf.

Jeff Polakow. Linear logic programming with an ordered context. In M. Gabbrielli and F. Pfenning, editors, *Conference on Principles and Practice of Declarative Programming (PPDP 2000)*, pages 68–79, Montreal, Canada, September 2000. ACM.

Jeff Polakow. *Ordered Linear Logic and Applications*. PhD thesis, Department of Computer Science, Carnegie Mellon University, August 2001.

Jason C. Reed and Frank Pfenning. Focus-preserving embeddings of substructural logics in intuitionistic logic. Unpublished Manuscript, January 2010. URL http://www.cs.cmu.edu/~fp/papers/substruct10.pdf.