Lecture Notes on Substructural Type Systems

Jan Hoffmann

Lecture 6
Thursday, September 11, 2025

1 Introduction

In the type systems we have studied so far, we have taken so-called *structural properties* for granted. For example, consider the typing rule for variables.

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}$$

In the rule, we view the context $\Gamma' = \Gamma, x : \tau$ as a function from variables to types, and are not interested in the position at which x appears in Γ' if we view it as a list. So we just assume that it appears in the right-most position in the rule. Moreover, we are not interested in the domain of Γ. Of course, the context sggΓ' would be malformed if the variable x would appear in Γ or if other variables would appear twice. However, we allow arbitrary other variables in Γ in the variable rule.

In *substructural type systems* we are more precise about such structural properties to control the use of variables. These type systems correspond to substructural logics (such as linear logic). Substructural type systems find applications in memory management, access control, concurrent programming, and resource analysis.

2 Structural Properties

Before we discuss substructural type systems, we define some structural properties that we are interested in. To this end, we are very precise about type contexts. We define, as before

$$\Gamma ::= \cdot \mid \Gamma, x : \tau \ .$$

However, we now take the definition literally and view Γ as list $x_1 : \tau_1, \ldots, x_n : \tau_n$ instead of a function from variables to types. In particular, we do not identify lists with the same variables and type assignments. The only requirement, we still have is that variables appear at most once in a given context. For example, we have

$$x : \tau, y : \tau' \neq y : \tau', x : \tau$$

for well-formed contexts since $x \neq y$.

Exchange We say that a type system allows for *exchange* if the following rule is admissible.

$$\frac{\Gamma_1, x : \tau_1, y : \tau_2, \Gamma_2 \vdash e : \tau}{\Gamma_1, y : \tau_2, x : \tau_1, \Gamma_2 \vdash e : \tau} \text{ (EXCH)}$$

Intuitively, the rule states that the order of variables in type contexts does not matter in type derivations. Type systems in which the exchange rule is not admissible are called *ordered*. In this course, we will only study type systems that enjoy exchange.

Weakening We say that a type system allows for *weakening* if the following rule is admissible.

$$\frac{\Gamma \vdash e : \tau}{\Gamma, x : \tau' \vdash e : \tau} \text{ (WEAK)}$$

From the conclusion of the rule, we know that the variable x does not appear in the context Γ. And since $\Gamma \vdash e : \tau$, the variable x is not free in the expression e. So the rule states that we can always add an unused variable to a context in a type derivation. Type systems without weakening are called *strict*.

Contraction We say that a type systems allows for *contraction* if the following rule is admissible.

$$\frac{\Gamma, x_1 : \tau, x_2 : \tau \vdash e : \tau'}{\Gamma, x : \tau \vdash [x, x/x_1, x_2]e : \tau'} \text{ (CNTR)}$$

A difference in the rule CNTR in comparison with the rules WEAK and EXCH is that the expression in the conclusion is different from the expression in the

Type system	Intuition	Weakening	Contraction
structural	no restriction on variable use	yes	yes
affine	variables are used at most once	yes	no
strict	variables are used at least once	no	yes
linear	variables are used exactly once	no	no

Table 1: Substructural Type Systems

premise. In the expression $e' = [x, x/x_1, x_2]e$, we rename the occurrences of both x_1 and x_2 to x. So if x_1 and x_2 appear free in e then x appears multiple times (free) in e'. In this case, we also say that x is used multiple times. Intuitively, the contraction rule states that it does not affect type judgments if a variable is used more often. A type system that does not enjoy contraction is called *affine*. A type system that does not enjoy contraction and weakening is called *linear*.

3 Substructural Type Systems

The type systems we have studied so far in the course enjoy exchange, weakening, and contraction. Consider a context $\Gamma = x : \tau_1, y : \tau_2$ in a typing such as $\Gamma \vdash e : \tau$. In the previous type judgments, we have been able to

- use x once as in $x : \tau_1, y : \tau_2 \vdash \langle x, \langle\,\rangle\rangle$
- use x multiple times as in $x : \tau_1, y : \tau_2 \vdash \langle x, x\rangle$
- use x not at all as in $x : \tau_1, y : \tau_2 \vdash \langle y, \langle\,\rangle\rangle$

In a substructural type system, we view variables as resources and control how these resources are used. We focus on three types of substructural type systems: linear type systems, affine type systems, and strict type systems. Another important class of substructural type systems are ordered type systems. In ordered type systems, the order in which variables are introduced and used is important. Linear, affine, and strict type systems can be characterized by weakening and contraction as in Table 1.

To study substructural type systems, we use the expressions and types of the simply-typed lambda calculus with the unit type $\mathbf{1}$ as a base type. Expressions are variables, function applications, function abstractions, or

$\boxed{\Gamma \vdash^{\ell} e : \tau \qquad \text{“expression } e \text{ has type } \tau \text{ in context } \Gamma\text{”}}$

$$\frac{}{x : \tau \vdash^{\ell} x : \tau}\ (\text{L:VAR}) \qquad \frac{}{\cdot \vdash^{\ell} \text{triv} : \text{unit}}\ (\text{L:UNIT})$$

$$\frac{\Gamma, x{:}\tau' \vdash^{\ell} e : \tau}{\Gamma \vdash^{\ell} \text{lam}\{\tau'\}(x.e) : \tau' \to \tau}\ (\text{L:ABS})$$

$$\frac{\Gamma_1 \vdash^{\ell} e_1 : \tau' \to \tau \qquad \Gamma_2 \vdash^{\ell} e_2 : \tau'}{\Gamma_1, \Gamma_2 \vdash^{\ell} \text{app}(e_1; e_2) : \tau}\ (\text{L:APP})$$

$$\frac{\Gamma_1, x : \tau_1, y : \tau_2, \Gamma_2 \vdash^{\ell} e : \tau}{\Gamma_1, y : \tau_2, x : \tau_1, \Gamma_2 \vdash^{\ell} e : \tau}\ (\text{L:EXCH})$$

Figure 1: Linear type rules.

the unit value.

$$\begin{array}{rcll} e & ::= & x & x \\ & & \text{app}(e_1; e_2) & e_1(e_2) \\ & & \text{lam}\{\tau\}(x.e) & \lambda(x : \tau)e \\ & & \text{triv} & \langle\rangle \end{array}$$

A type is either an arrow type $\tau_1 \to \tau_2$ or the unit type $\mathbf{1}$.

$$\begin{array}{rcll} \tau & ::= & \text{arr}(\tau_1; \tau_2) & \tau_1 \to \tau_2 \\ & & \text{unit} & \mathbf{1} \end{array}$$

3.1 Linear Type Systems

Our goal is to design a *linear* type system, that is, a type system that ensures that every variable is used exactly once. To this end, we define the rules in Figure 1, which define the type judgment $\Gamma \vdash^{\ell} e : \tau$.

As before, the rules L:VAR and L:UNIT are axioms (leaves in type derivations). To maintain the invariant that every variable is used once, we require that the context in the rule L:VAR contains exactly the variable x. Similarly, we require that the context is empty in L:UNIT since we do not use a variable in the unit value. In the rule L:ABS, the premise $\Gamma, x{:}\tau' \vdash^{\ell} e : \tau$ requires that the variables in Γ and x have to be used exactly once in the function body

e. However, the function itself will be used exactly once in the program. In the rule L:APP, we spit up the context Γ into Γ_1 and Γ_2. The two premises ensure that every variable in Γ_1 is used exactly once in e_1 and every variable in Γ_2 is used exactly once in e_2.

For example, we can derive the judgment $f : \mathbf{1} \to \mathbf{1} \to \tau, x : \mathbf{1}, y : \mathbf{1} \vdash^{\ell} \mathrm{app}(\mathrm{app}(f; x); y) : \tau$ as follows

$$\dfrac{\dfrac{\dfrac{}{f : \mathbf{1} \to \mathbf{1} \to \tau \vdash^{\ell} f : \mathbf{1} \to \mathbf{1} \to \tau}\,(\text{L:VAR}) \quad \dfrac{}{x : \mathbf{1} \vdash^{\ell} x : \mathbf{1}}\,(\text{L:VAR})}{f : \mathbf{1} \to \mathbf{1} \to \tau, x : \mathbf{1} \vdash^{\ell} \mathrm{app}(f; x) : \mathbf{1} \to \tau}\,(\text{L:APP}) \quad \dfrac{}{y : \mathbf{1} \vdash^{\ell} y : \mathbf{1}}\,(\text{L:VAR})}{f : \mathbf{1} \to \mathbf{1} \to \tau, x : \mathbf{1}, y : \mathbf{1} \vdash^{\ell} \mathrm{app}(\mathrm{app}(f; x); y) : \tau}\,(\text{L:APP})$$

Similarly we could derive the judgment

$$f : \mathbf{1} \to \mathbf{1} \to \tau, x : \mathbf{1} \vdash^{\ell} \mathrm{app}(\mathrm{app}(f; x); \langle\,\rangle) : \tau\ .$$

However, we can not derive

$$f : \mathbf{1} \to \mathbf{1} \to \tau, x : \mathbf{1}, y : \mathbf{1} \vdash^{\ell} \mathrm{app}(\mathrm{app}(f; x); \langle\,\rangle) : \tau$$

nor

$$f : \mathbf{1} \to \mathbf{1} \to \tau, x : \mathbf{1} \vdash^{\ell} \mathrm{app}(\mathrm{app}(f; x); x) : \tau\ .$$

Implicit Exchange *In the following lectures, we will just assume the presence of the exchange property without explicitly mentioning it or introducing a specific rule.* Instead, we simply identify contexts that contain the same variable-type pairs, irrespective of their order. For example, we consider the linear type system to only consist of the rules L:VAR, L:UNIT, L:ABS, and L:APP. With this view, the linear type rules are syntax directed, that is, there is exactly one type rule for each syntactic form.

3.2 Affine Type Systems

An *affine* type system ensures that every variable is used at most once. There are two possibilities to turn the linear type system in Figure 1 into an affine one.

The fist option is to leave the existing type rules unchanged and add an additional weakening rule. We write $\Gamma \vdash^{a} e : \tau$ for the judgment that we derive with these rules.

$$\frac{\Gamma \vdash^{a} e : \tau}{\Gamma, x : \tau' \vdash^{a} e : \tau}\,(\text{WEAK})$$

As discussed earlier, the idea of the rule WEAK is that we can add an unused variable x to the context of a type judgment.

Using the rule WEAK and the linear rules, we can derive the judgment $f : \mathbf{1} \to \mathbf{1} \to \tau, x : \mathbf{1}, y : \mathbf{1} \vdash^a \mathrm{app}(\mathrm{app}(f; x); \langle\rangle) : \tau$.

$$\dfrac{\dfrac{\dfrac{}{f : \mathbf{1} \to \mathbf{1} \to \tau \vdash^a f : \mathbf{1} \to \mathbf{1} \to \tau}\,(\text{L:VAR}) \quad \dfrac{}{x : \mathbf{1} \vdash^a x : \mathbf{1}}\,(\text{L:VAR})}{f : \mathbf{1} \to \mathbf{1} \to \tau, x : \mathbf{1} \vdash^a \mathrm{app}(f; x) : \mathbf{1} \to \tau}\,(\text{L:APP}) \quad \dfrac{\dfrac{}{\cdot \vdash^a \langle\rangle : \mathbf{1}}\,(\text{L:UNIT})}{y : \mathbf{1} \vdash^a \langle\rangle : \mathbf{1}}\,(\text{WEAK})}{f : \mathbf{1} \to \mathbf{1} \to \tau, x : \mathbf{1}, y : \mathbf{1} \vdash^a \mathrm{app}(\mathrm{app}(f; x); \langle\rangle) : \tau}\,(\text{L:APP})$$

We can show that every linear typing is also an affine typing. As the previous example shows, the converse is not true.

Theorem 1 *Let e be an expression. If $\Gamma \vdash^\ell e : \tau$ then $\Gamma \vdash^a e : \tau$.*

A disadvantage of this extension of the linear rules is that the rule WEAK is not syntax-directed, which means that it can be applied to every syntactic form. In contrast, the type rules in the linear type system are syntax directed and there is exactly one rule for every syntactic form (if we view the exchange rule as an implicit rule as discussed earlier). Such a syntax-directed type system makes type checking straightforward and simplifies type inference.

The second option is to not add additional rules but to replace the axioms L:VAR and L:UNIT with the rules A:VAR and A:UNIT defined below. The intuition is that we allow to an implicit weakening of all variables in the context Γ. The advantage of this approach is that the rules are syntax directed. A disadvantage is that we have to incorporate implicit waking into multiple rules and that we restrict derivations to have a specific form.

$$\frac{}{\Gamma, x : \tau \vdash^{\mathsf{as}} x : \tau}\,(\text{A:VAR}) \qquad \frac{}{\Gamma \vdash^{\mathsf{as}} \mathrm{triv} : \mathrm{unit}}\,(\text{A:UNIT})$$

We write $\Gamma \vdash^{\mathsf{as}} e : \tau$ for the judgment that we derive with these rules. Using the rule A:UNIT, we can derive the previous type judgment as follows.

$$\dfrac{\dfrac{\dfrac{}{f : \mathbf{1} \to \mathbf{1} \to \tau \vdash^{\mathsf{as}} f : \mathbf{1} \to \mathbf{1} \to \tau}\,(\text{L:VAR}) \quad \dfrac{}{x : \mathbf{1} \vdash^{\mathsf{as}} x : \mathbf{1}}\,(\text{L:VAR})}{f : \mathbf{1} \to \mathbf{1} \to \tau, x : \mathbf{1} \vdash^{\mathsf{as}} \mathrm{app}(f; x) : \mathbf{1} \to \tau}\,(\text{L:APP}) \quad \dfrac{}{y : \mathbf{1} \vdash^{\mathsf{as}} \langle\rangle : \mathbf{1}}\,(\text{A:UNIT})}{f : \mathbf{1} \to \mathbf{1} \to \tau, x : \mathbf{1}, y : \mathbf{1} \vdash^{\mathsf{as}} \mathrm{app}(\mathrm{app}(f; x); \langle\rangle) : \tau}\,(\text{L:APP})$$

We can show that the syntax-directed and declarative approaches are equivalent as formalized by the following theorem.

Theorem 2 *Let e be an expression. Then $\Gamma \vdash^a e : \tau$ if and only if $\Gamma \vdash^{as} e : \tau$.*

3.3 Strict Type Systems

In a strict type system, we want to ensure that each variable is used at least once. This is of course the case in the linear type system but we can be a bit more permissive. Like for the affine type system, we can simply leave the (syntax-directed) linear type rules unchanged and add a structural rule for contraction.

$$\frac{\Gamma, x_1 : \tau, x_2 : \tau \vdash^r e : \tau'}{\Gamma, x : \tau \vdash^r [x, x/x_1, x_2]e : \tau'} \text{ (CNTR)}$$

We write $\Gamma \vdash^r e : \tau$ for the resulting judgment. With the contraction rule we can derive the judgment $f : \mathbf{1} \to \mathbf{1} \to \tau, x : \mathbf{1} \vdash^\ell \mathrm{app}(\mathrm{app}(f; x); x) : \tau$. We simply apply contraction and then use the derivation linear of the linear judgment as before

$$\dfrac{\dfrac{\cdots}{f : \mathbf{1} \to \mathbf{1} \to \tau, x : \mathbf{1}, y : \mathbf{1} \vdash^\ell \mathrm{app}(\mathrm{app}(f; x); y) : \tau}\text{ (T:APP)}}{f : \mathbf{1} \to \mathbf{1} \to \tau, x : \mathbf{1} \vdash^\ell \mathrm{app}(\mathrm{app}(f; x); x) : \tau}\text{ (CNTR)}$$

Clearly, every linear type derivation is also a strict derivation.

Theorem 3 *Let e be an expression. If $\Gamma \vdash^\ell e : \tau$ then $\Gamma \vdash^r e : \tau$.*

As the examples show, the converse is not true and strict and affine type systems are incomparable.

There is also a syntax directed version of the relative type system. If we start again with the linear type rules, a good idea is to modify the rule L:APP for function applications. However, the standard rule

$$\frac{\Gamma \vdash^\ell e_1 : \tau' \to \tau \qquad \Gamma \vdash^\ell e_2 : \tau'}{\Gamma \vdash^\ell \mathrm{app}(e_1; e_2) : \tau} \text{ (L:APP)}$$

is not quite what we want. It would result in a type system in which we have to use each variable in Γ in both, e_1 and e_2. Instead, have to define a sharing judgment $\Gamma \curlyvee (\Gamma_1, \Gamma_2)$ that states that the variables in Γ have to be used in Γ_1, Γ_2, or in both. The idea is formalized in Figure 2.

We can then define the syntax-directed strict type system by replacing the rule L:APP in the linear type system with the rule R:APP below, define the judgment $\Gamma \vdash^{\mathrm{rs}} e : \tau$.

$$\frac{\Gamma_1 \vdash^{\mathrm{rs}} e_1 : \tau' \to \tau \qquad \Gamma_2 \vdash^{\mathrm{rs}} e_2 : \tau' \qquad \Gamma \curlyvee (\Gamma_1, \Gamma_2)}{\Gamma \vdash^{\mathrm{rs}} \mathrm{app}(e_1; e_2) : \tau} \text{ (R:APP)}$$

We can show that this rule is equivalent to adding the contraction rule.

$\boxed{\Gamma \curlyvee (\Gamma_1, \Gamma_2) \qquad \text{“context } \Gamma \text{ is shared as } \Gamma_1 \text{ and } \Gamma_2\text{”}}$

$$\frac{}{\cdot \curlyvee (\cdot, \cdot)}\ (\textsc{Share0}) \qquad \frac{\Gamma \curlyvee (\Gamma_1, \Gamma_2)}{\Gamma, x : \tau \curlyvee (\Gamma_1, x : \tau, \Gamma_2)}\ (\textsc{ShareL})$$

$$\frac{\Gamma \curlyvee (\Gamma_1, \Gamma_2)}{\Gamma, x : \tau \curlyvee (\Gamma_1, \Gamma_2, x : \tau)}\ (\textsc{ShareR})$$

$$\frac{\Gamma \curlyvee (\Gamma_1, \Gamma_2)}{\Gamma, x : \tau \curlyvee (\Gamma_1, x : \tau, \Gamma_2, x : \tau)}\ (\textsc{ShareB})$$

Figure 2: Sharing rules.

Theorem 4 *Let e be an expression. Then $\Gamma \vdash^{r} e : \tau$ if and only if $\Gamma \vdash^{rs} e : \tau$.*

A less elaborate way of obtaining a syntax-directed contraction rule is to introduce a syntactic form that makes multiple uses of a variable explicit in the syntax.

$$\begin{array}{rcll} e & ::= & \ldots & \\ & & \mathrm{share}(e_1; x_1, x_2.e_2) & \mathsf{share}\ e_1\ \mathsf{as}\ x_1, x_2\ \mathsf{in}\ e_2 \end{array}$$

The syntactic form $\mathrm{share}(e_1; x_1, x_2.e_2)$ is like a let binding that binds the result of e_1 to both x_1 and x_2. The rule of evaluation dynamics is as follows.

$$\frac{e_1 \Downarrow v_1 \qquad [v_1, v_1/x_1, x_2]e_2 \Downarrow v}{\mathrm{share}(e_1; x_1, x_2.e_2)}\ (\text{E:}\textsc{Share})$$

The type rule that encodes contraction is as expected.

$$\frac{\Gamma_1 \vdash^{\mathrm{rs}} e_1 : \tau' \qquad \Gamma_2, x_1 : \tau', x_2 : \tau' \vdash^{\mathrm{rs}} e_2 : \tau}{\Gamma_1, \Gamma_2 \vdash^{\mathrm{rs}} \mathrm{share}(e_1; x_1, x_2.e_2) : \tau}\ (\text{E:}\textsc{Share})$$

3.4 Controlling Structural Properties

In the remainder of this course, we will often use a linear type system that is extended with both weakening and contraction. This leads to a “standard” structural type discipline but it enables us to precisely control the structural properties. For example, we can mix linear and unrestricted types in the same context by allowing sharing and weakening for some types only.

If we write $\Gamma \vdash e : \tau$ for the regular typing judgment from the cost-semantics lecture and $\Gamma \vdash^{u} e : \tau$ for the judgment we obtain by extending the linear rules from Figure 1 with the rules WEAK and CNTR then we can prove the following theorem.

Theorem 5 *Let e be an expression. Then $\Gamma \vdash e : \tau$ if and only if $\Gamma \vdash^{u} e : \tau$.*