Chapter 5

Records, Variants, and Subtypes [Lecture 9]

Version of March 8, 2021

Summary We extend FUN with record and variant types, and introduce the notion of subtyping. We then show type soundness for the resulting system.

5.1 Records and variants

5.1.1 Records

The FUN language introduced so far has only binary tuples, in the form of pair constructors and first/second projections. It should be obvious how these notions can be generalized to n-ary tuples and corresponding component selections. However, most realistic languages also offer a notion of *records* (possibly packaged with additional functionality as *objects*), whose fields are not indexed by numbers, but by arbitrary labels. Let us extend FUN with such a facility.

Throughout this chapter, $n..m$ will denote the finite set $\{i \in \mathbf{N} \mid n \leq i \leq m\}$. We also use the *meta*notation $(\phi)^{i \in n..m}$ (where ϕ is any syntactic phrase in which i occurs) to abbreviate the sequence $(\phi[n/i], \phi[(n+1)/i], \ldots, \phi[m/i])$. (When $n > m$, the sequence is empty.) For example, $(x_i = \overline{2 \times i})^{i \in 3..5}$ is the same as $(x_3 = \overline{6}, x_4 = \overline{8}, x_5 = \overline{10})$.

Further, let the metavariable $\ell \in \mathbf{Label}$ range over a class of *labels*, with individual labels written in a sans-serif font; then we introduce a pair of syntactic forms, *record constructors* and *field selectors*:

$$t ::= \cdots \mid \{(\ell_i = t_i)^{i \in 1..n}\} \mid t_0.\ell \ .$$

All labels in a record constructor are required to be distinct. For example, if n and b are labels, then $\{\mathsf{n} = \overline{4}, \mathsf{b} = \mathbf{true}\}$ and $\lambda r.\,(r.\mathsf{b})$ are well-formed terms. Note that labels are not explicitly declared, and the same label can be used in multiple, unrelated records.

The definition of canonical forms is also extended, by saying that a record constructor in which all the component terms are canonical, is itself canonical:

$$c ::= \cdots \mid \{(\ell_i = c_i)^{i \in 1..n}\} \, .$$

The small-step semantics of the new record-related terms are now given by the following rules:

$$\text{S-Rcd1}: \frac{t_k \to t_k'}{\{(\ell_i = c_i)^{i\in 1..k-1}, \ell_k = t_k, (\ell_i = t_i)^{i\in k+1..n}\} \to \{(\ell_i = c_i)^{i\in 1..k-1}, \ell_k = t_k', (\ell_i = t_i)^{i\in k+1..n}\}} \; (k \in 1..n)$$

$$\text{S-Sel1}: \frac{t_0 \to t_0'}{t_0.\ell \to t_0'.\ell} \qquad \text{S-Sel}: \frac{}{\{(\ell_i = c_i)^{i\in 1..n}\}.\ell_k \to c_k} \; (k \in 1..n)$$

Note how attempting to select a field from a record constructor that does not have it results in a stuck term. (Our type system later will guarantee that this cannot happen.) Because S-Rcd1 only allows reducing the leftmost unevaluated component in a record constructor, and because the labels in the record constructor in S-Sel are required to be distinct, the stepping judgment remains deterministic.

5.1.2 Variants

Where records can be seen as a generalization of binary products, *variants* provide the corresponding notion of labeled, n-ary sums (also known as *disjoint unions*). These correspond fairly closely to (non-recursive) *datatypes* in ML or Haskell. The relevant extensions of the term syntax (adopting the notation from Pierce's *Types and Programming Languages*) are:

$$t ::= \cdots \mid \langle \ell = t_0 \rangle \mid \textbf{case}\ t_0\ \textbf{of}\ (\langle \ell_i = x_i \rangle \Rightarrow t_i)^{i\in 1..n}\,.$$

The form $\langle \ell = t_0 \rangle$ is a *variant constructor*; note that, in contrast to records, it always contains exactly one component. The **case** form distinguishes cases based on the value of the *discriminator* t_0; again, to ensure determinism, the labels $\ell_1, ..., \ell_n$ in the branches are required to be distinct. For example, $t_1 = \langle \mathsf{b} = \textbf{true} \rangle$ and

$$\begin{aligned} t_2 = \lambda v.\, &\textbf{case}\ v\ \textbf{of} \\ &\langle \mathsf{n} = x \rangle \Rightarrow x + \overline{7}, \\ &\langle \mathsf{b} = y \rangle \Rightarrow \textbf{if}\ y\ \textbf{then}\ \overline{3}\ \textbf{else}\ \overline{5} \end{aligned}$$

are well-formed terms.

A variant constructor is a canonical form iff its term component is itself canonical:

$$c ::= \cdots \mid \langle \ell = c_0 \rangle\,.$$

The operational semantics of variants is given by the following rules:

$$\text{S-Vnt1}: \frac{t_0 \to t_0'}{\langle \ell = t_0 \rangle \to \langle \ell = t_0' \rangle}$$

$$\text{S-Case1}: \frac{t_0 \to t_0'}{\textbf{case}\ t_0\ \textbf{of}\ (\langle \ell_i = x_i \rangle \Rightarrow t_i)^{i\in 1..n} \to \textbf{case}\ t_0'\ \textbf{of}\ (\langle \ell_i = x_i \rangle \Rightarrow t_i)^{i\in 1..n}}$$

$$\text{S-Case}: \frac{}{\textbf{case}\ \langle \ell_k = c_0 \rangle\ \textbf{of}\ (\langle \ell_i = x_i \rangle \Rightarrow t_i)^{i\in 1..n} \to t_k[c_0/x_k]} \; (k \in 1..n)$$

With these rules, and the sample terms above, the application $t_2\, t_1$ evaluates (in three steps) to $\overline{3}$. Like for S-Sel, if the evaluated discriminator in a **case** does not match any of the branch labels (or if it isn't a variant constructor at all), the term is stuck.

5.2 Typing records and variants

We extend the syntax of types with record and variant types:

$$\tau ::= \cdots \mid \{(\ell_i : \tau_i)^{i \in 1..n}\} \mid \langle (\ell_i : \tau_i)^{i \in 1..n} \rangle \,.$$

All the ℓ_i in a single record or variant type must be distinct from each other, but the same labels may be freely used (possibly with different types) in other record and/or variant types. In a record, the labels and types represent fields, *all* of which must be present to construct the record; whereas in a variant, they represent alternatives, *exactly one* of which must be used when constructing a variant term.

The typing rules for the terms introduced in the previous section are thus:

$$\text{T-Rcd} : \frac{(\Gamma \vdash t_i : \tau_i)^{i \in 1..n}}{\Gamma \vdash \{(\ell_i = t_i)^{i \in 1..n}\} : \{(\ell_i : \tau_i)^{i \in 1..n}\}} \qquad \text{T-Sel} : \frac{\Gamma \vdash t_0 : \{(\ell_i : \tau_i)^{i \in 1..n}\}}{\Gamma \vdash t_0.\ell_k : \tau_k} \, (k \in 1..n)$$

$$\text{T-Vnt} : \frac{\Gamma \vdash t_0 : \tau_k}{\Gamma \vdash \langle \ell_k = t_0 \rangle : \langle (\ell_i : \tau_i)^{i \in 1..n} \rangle} \, (k \in 1..n)$$

$$\text{T-Case} : \frac{\Gamma \vdash t_0 : \langle (\ell_i : \tau_i)^{i \in 1..n} \rangle \qquad (\Gamma[x_i \mapsto \tau_i] \vdash t_i : \tau)^{i \in 1..n}}{\Gamma \vdash \textbf{case}\ t_0\ \textbf{of}\ (\langle \ell_i = x_i \rangle \Rightarrow t_i)^{i \in 1..n} : \tau}$$

Note that we can now remove binary products and sums from the language, by re-classifying them as syntactic sugar:

$$\begin{array}{rcl}
\tau_1 \times \tau_2 & \equiv & \{\mathsf{first} : \tau_1, \mathsf{second} : \tau_2\} \\
(t_1, t_2) & \equiv & \{\mathsf{first} = t_1, \mathsf{second} = t_2\} \\
\textbf{fst}(t_0) & \equiv & t_0.\mathsf{first} \\
\textbf{snd}(t_0) & \equiv & t_0.\mathsf{second} \\
 & & \\
\tau_1 + \tau_2 & \equiv & \langle \mathsf{left} : \tau_1, \mathsf{right} : \tau_2 \rangle \\
\textbf{inl}(t_0) & \equiv & \langle \mathsf{left} = t_0 \rangle \\
\textbf{inr}(t_0) & \equiv & \langle \mathsf{right} = t_0 \rangle \\
\textbf{case}\ t_0\ \textbf{of}\ \textbf{inl}(x_1) \Rightarrow t_1, \textbf{inr}(x_2) \Rightarrow t_2 & \equiv & \textbf{case}\ t_0\ \textbf{of}\ \langle \mathsf{left} = x_1 \rangle \Rightarrow t_1, \langle \mathsf{right} = x_2 \rangle \Rightarrow t_2
\end{array}$$

It is easy to check that these definitions exactly preserve the typing and operational semantics of products and sums. We can even get rid of booleans as a base type:

$$\begin{array}{rcl}
\textbf{bool} & \equiv & \langle \mathsf{true} : \{\}, \mathsf{false} : \{\} \rangle \\
\textbf{true} & \equiv & \langle \mathsf{true} = \{\} \rangle \\
\textbf{false} & \equiv & \langle \mathsf{false} = \{\} \rangle \\
\textbf{if}\ t_0\ \textbf{then}\ t_1\ \textbf{else}\ t_2 & \equiv & \textbf{case}\ t_0\ \textbf{of}\ \langle \mathsf{true} = u \rangle \Rightarrow t_1, \langle \mathsf{false} = u \rangle \Rightarrow t_2
\end{array}$$

where u is a dummy variable (of type $\{\}$), not occurring in t_1 or t_2.

We can still show the Progress and Preservation lemmas, with the new cases as fairly straightforward generalizations of the ones for binary products and sums. Recall the statement of Progress (Lemma 4.8): If $[] \vdash t : \tau$ (by $\mathcal{T}$), then either $t = c$ for some canonical form c, or $t \to t'$ for some other term t'. The new cases in the proof are:

- Case $\mathcal{T} = \dfrac{\begin{pmatrix}\mathcal{T}_i \\ [] \vdash t_i : \tau_i\end{pmatrix}^{i \in 1..n}}{[] \vdash \{(\ell_i = t_i)^{i \in 1..n}\} : \{(\ell_i : \tau_i)^{i \in 1..n}\}}$, so $t = \{(\ell_i = t_i)^{i \in 1..n}\}$.

  If all of $t_1, ..., t_n$ are canonical forms, then t is itself a canonical form, and we are done. Otherwise, let k be the smallest index such that $t_1, ..., t_{k-1}$ are all canonical, but t_k is not. Then by IH on $\mathcal{T}_k$, since t_k is *not* canonical, we must have that $t_k \to t'_k$ for some t'_k, and thus the whole term steps by S-Rcd1.

- Case $\mathcal{T} = \dfrac{\begin{matrix}\mathcal{T}_0 \\ [] \vdash t_0 : \{(\ell_i : \tau_i)^{i \in 1..n}\}\end{matrix}}{[] \vdash t_0.\ell_k : \tau_k}$, where $k \in 1..n$.

  By IH on $\mathcal{T}_0$, t_0 is canonical or can take a step. In the latter case, so can $t_0.\ell_k$ (by S-Sel1), so we are done. Otherwise, since the only canonical forms of a record type are record constructors with canonical components and field labels exactly matching those in the type, we must have $t_0 = \{(\ell_i = c_i)^{i \in 1..n}\}$, for some canonical forms $(c_i)^{i \in 1..n}$. And this record constructor in particular includes a field ℓ_k, so the whole term reduces by S-Sel to c_k.

- (The cases related to variant types are left as an exercise.)

The statement of Preservation (Lemma 4.11) was: If $[] \vdash t : \tau$ (by $\mathcal{T}$) and $t \to t'$ (by $\mathcal{S}$), then also $[] \vdash t' : \tau$. The new cases for the induction on $\mathcal{S}$ are:

- (The cases related to record types are left as an exercise.)

- The cases for S-Vnt1 and S-Case1 proceed like the other context cases in the original proof.

- Case $\mathcal{S} = \dfrac{}{\textbf{case}\ \langle \ell_k = c_0 \rangle\ \textbf{of}\ (\langle \ell_i = x_i \rangle \Rightarrow t_i)^{i \in 1..n} \to t_k[c_0/x_k]}$, where $k \in 1..n$.

  From the shape of t, we know that $\mathcal{T}$ must look as follows:

$$\mathcal{T} = \text{T-Case}\ \frac{\text{T-Vnt}\ \dfrac{\begin{matrix}\mathcal{T}_0 \\ [] \vdash c_0 : \tau_k\end{matrix}}{[] \vdash \langle \ell_k = c_0 \rangle : \langle (\ell_i : \tau_i)^{i \in 1..n} \rangle} \qquad \begin{pmatrix}\mathcal{T}_i \\ [x_i \mapsto \tau_i] \vdash t_i : \tau\end{pmatrix}^{i \in 1..n}}{[] \vdash \textbf{case}\ \langle \ell_k = c_0 \rangle\ \textbf{of}\ (\langle \ell_i = x_i \rangle \Rightarrow t_i)^{i \in 1..n} : \tau}.$$

  Thus, by the Substitution lemma (4.10, straightforwardly extended to the new term forms) on $\mathcal{T}_k$ and $\mathcal{T}_0$, we get that $[] \vdash t_k[c_0/x_k] : \tau$, as required.

5.3 Subtyping

In our language so far, the order of labels in a record is significant: $\{\mathsf{l} = \overline{3}, \mathsf{m} = \textbf{true}\}$ is a different canonical form from $\{\mathsf{m} = \textbf{true}, \mathsf{l} = \overline{3}\}$. However, they are essentially equivalent operationally, since each of them allows selection of (only) labels l or m; and in particular, if evaluation does not get stuck with one, it will not get stuck with the other.

Judgment $\boxed{\tau' \leq \tau}$:

$$\text{ST-Refl} : \frac{}{\tau \leq \tau} \qquad \text{ST-Trans} : \frac{\tau' \leq \tau'' \qquad \tau'' \leq \tau}{\tau' \leq \tau}$$

$$\text{ST-Fun} : \frac{\tau_1 \leq \tau_1' \qquad \tau_2' \leq \tau_2}{\tau_1' \to \tau_2' \leq \tau_1 \to \tau_2}$$

$$\text{ST-RcdPerm} : \frac{}{\{(\ell_i : \tau_i)^{i \in 1..n}\} \leq \{(\ell_{\pi(i)} : \tau_{\pi(i)})^{i \in 1..n}\}} \; (\pi \text{ a permutation on } 1..n)$$

$$\text{ST-RcdWidth} : \frac{}{\{(\ell_i : \tau_i)^{i \in 1..n}\} \leq \{(\ell_i : \tau_i)^{i \in 1..m}\}} \; (n \geq m)$$

$$\text{ST-RcdDepth} : \frac{(\tau_i' \leq \tau_i)^{i \in 1..n}}{\{(\ell_i : \tau_i')^{i \in 1..n}\} \leq \{(\ell_i : \tau_i)^{i \in 1..n}\}}$$

$$\text{ST-VntPerm} : \frac{}{\langle(\ell_i : \tau_i)^{i \in 1..n}\rangle \leq \langle(\ell_{\pi(i)} : \tau_{\pi(i)})^{i \in 1..n}\rangle} \; (\pi \text{ a permutation on } 1..n)$$

$$\text{ST-VntWidth} : \frac{}{\langle(\ell_i : \tau_i)^{i \in 1..n}\rangle \leq \langle(\ell_i : \tau_i)^{i \in 1..m}\rangle} \; (n \leq m)$$

$$\text{ST-VntDepth} : \frac{(\tau_i' \leq \tau_i)^{i \in 1..n}}{\langle(\ell_i : \tau_i')^{i \in 1..n}\rangle \leq \langle(\ell_i : \tau_i)^{i \in 1..n}\rangle}$$

Figure 5.1: Subtyping rules for FUN with records and variants

This strict ordering of record fields is reflected in the type system. For example, the following term is untypable, because the branches of the **if** have different types:

$$[x \mapsto \mathbf{int}] \vdash (\mathbf{if}\ x \leq \overline{5}\ \mathbf{then}\ \{\mathsf{l} = \overline{3}, \mathsf{m} = \mathbf{true}\}\ \mathbf{else}\ \{\mathsf{m} = \mathbf{false}, \mathsf{l} = \overline{7}\}).\mathsf{l} : ?$$

Likewise, the function application in

$$[f \mapsto (\{\mathsf{l} : \mathbf{int}, \mathsf{m} : \mathbf{bool}\} \to \mathbf{int})] \vdash f\ \{\mathsf{l} = \overline{3}, \mathsf{m} = \mathbf{true}, \mathsf{k} = \lambda x.\, x\} : ?$$

is ill-typed, even though the typing of f indicates that it can only access fields l and m from its argument, and thus will necessarily ignore k.

In the subtyping discipline, we relax the well-typedness conditions, while maintaining type soundness. We say that, for a term to have type $\{\ell_1 : \tau_1, ..., \ell_n : \tau_n\}$, the term must evaluate to a record with *at least* the fields mentioned, and with the indicated types (or subtypes, see later), but not necessarily in the same order. Conversely, the result alternatives for a term of a variant type must be *at most* those indicated by its type.

We capture these notions with a judgment $\tau' \leq \tau$ (often also written $\tau' <: \tau$), pronounced "τ' is a subtype of τ" (or, equivalently, "τ is a supertype of τ'"), defined by the rules in Figure 5.1. The intuition is that every term of type τ' can also be safely regarded as having type τ. Rules ST-Refl and ST-Trans say that $\leq$ forms a *preorder* on syntactic types. It is not a *partial order*, because we can have types τ and τ' such that $\tau' \leq \tau$ and $\tau \leq \tau'$, but $\tau \neq \tau'$. (Such types are called *equivalent*; an example would be two record types differing only in the order of their fields.)

Rule ST-Fun displays the principle of *contravariant* subtyping: when checking whether a function type is a subtype of another, the ordering on the domain (input) types is reversed. That is, if $\tau_1 \leq \tau_1'$, then a function expecting τ_1'-typed arguments will also accept

τ_1-typed ones. The ordering on codomain (output) types, on the other hand, is the non-reversed, or *covariant*, one.

There are three rules specific to record subtyping. ST-RcdPerm says that permuting the order of fields in a record type, gives a supertype of the original one. (But since permutations are by definition invertible, the original type is also supertype of the new one, i.e., the types are equivalent, though not identical.)

ST-RcdWidth says that the supertype can omit fields that were present in the subtype, but only from the end. Of course, together with the permutation rule, this allows us to omit fields from anywhere in the list, but the rule also makes sense by itself to model classic C++-style single inheritance, where a (reference to a) structure with additional fields at the end can be used where a shorter structure was expected, without requiring any representation change at runtime. Finally, ST-RcdDepth allows subtyping on the types associated with individual fields in the record.

The rules for variant subtyping are analogous to those for records, with the exception that ST-VntWidth allows the supertype to include *more* alternatives than the subtype. Again, it would often be combined with ST-VntPerm, but in some languages it may make sense by itself: it says that we can add new alternatives to an enumeration type, but if we want to avoid changing representations, we must add the new possibilities at the end of the list.

To make use of the subtyping judgment, we include an additional typing rule, called *subsumption*:

$$\text{T-Sub}: \frac{\Gamma \vdash t : \tau' \qquad \tau' \leq \tau}{\Gamma \vdash t : \tau}.$$

This says that a term of a type τ' can always be used where one of a supertype τ is expected. Note that this represents the programmer's view; it is up to the language implementer whether a use of subsumption involves any representation change at runtime. For example, if we have a subtyping axiom $\textbf{int} \leq \textbf{real}$ (as found in many languages), then the language implementation must either represent all numbers as floating-point values to begin with, or perform the appropriate conversion during evaluation.

With the addition of subtyping, we can actually "tighten" the rules for selection and variant construction. That is, we replace the previous rules T-Sel and T-Vnt with the following special cases:

$$\text{T-Sel'}: \frac{\Gamma \vdash t_0 : \{\ell : \tau\}}{\Gamma \vdash t_0.\ell : \tau} \qquad \text{T-Vnt'}: \frac{\Gamma \vdash t_0 : \tau_0}{\Gamma \vdash \langle \ell = t_0 \rangle : \langle \ell : \tau_0 \rangle}$$

These may look overly restrictive at first glance, but T-Sel' together with T-Sub, ST-RcdWidth, and ST-RcdPerm allows us to type everything that the original T-Sel did, and analogously for T-Vnt'.

5.4 Type safety with subtyping

Proving type safety in the system with subsumption follows the same progress+preservation strategy as the original proof. The details are much more involved, however, because now typing derivations can contain uses of T-Sub anywhere. This means that a number of properties that were immediate in the original type system suddenly require separate inductive proofs.

We start by two lemmas that will be useful for showing both Progress and Preservation. The parts in [brackets] are only needed for Preservation.

Lemma 5.1 (Subtype characterization) *Suppose $\tau' \le \tau$ is derivable. Then:*

a. If $\tau = \mathbf{int}$, then $\tau' = \mathbf{int}$.

b. If $\tau = \tau_1 \to \tau_2$, then $\tau' = \tau_1' \to \tau_2'$ for some τ_1' and τ_2' [with $\tau_1 \le \tau_1'$ and $\tau_2' \le \tau_2$].

c. If $\tau = \{(\ell_j : \tau_j)^{j \in 1..m}\}$, then $\tau' = \{(\ell_i' : \tau_i')^{i \in 1..n}\}$, for some $n \ge m$, $(\ell_i')^{i \in 1..n}$, and $(\tau_i')^{i \in 1..n}$, such that for each $j \in 1..m$ there exists an $i \in 1..n$ with $\ell_i' = \ell_j$ [and $\tau_i' \le \tau_j$].

d. If $\tau = \langle(\ell_j : \tau_j)^{j \in 1..m}\rangle$, then $\tau' = \langle(\ell_i' : \tau_i')^{i \in 1..n}\rangle$, for some $n \le m$, $(\ell_i')^{i \in 1..n}$, and $(\tau_i')^{i \in 1..n}$, such that for each $i \in 1..n$ there exists a $j \in 1..m$ with $\ell_j = \ell_i'$ [and $\tau_i' \le \tau_j$].

Proof. Each part is shown separately, by induction on the subtyping derivation.

a. Since there are no subtyping rules specific to **int**, there are only two possible rules that could have been used in the derivation $\mathcal{ST}$ of $\tau' \le \mathbf{int}$, namely ST-Refl and ST-Trans:

- Case $\mathcal{ST} = \dfrac{}{\mathbf{int} \le \mathbf{int}}$. Then $\tau' = \mathbf{int}$ and we are done.
- Case $\mathcal{ST} = \dfrac{\begin{array}{cc}\mathcal{ST}_0 & \mathcal{ST}_1\\ \tau' \le \tau'' & \tau'' \le \mathbf{int}\end{array}}{\tau' \le \mathbf{int}}$. Then by IH on $\mathcal{ST}_1$, we get that $\tau'' = \mathbf{int}$. Thus $\mathcal{ST}_0$ is actually also a derivation of $\tau' \le \mathbf{int}$, and so by IH on $\mathcal{ST}_0$, we get $\tau' = \mathbf{int}$, as required.

b. There are three possibilities for the final rule in $\mathcal{ST}$:

- Case $\mathcal{ST} = \dfrac{}{\tau_1 \to \tau_2 \le \tau_1 \to \tau_2}$.
  Then $\tau' = \tau_1' \to \tau_2'$, where $\tau_1' = \tau_1$ and $\tau_2' = \tau_2$. [We get $\tau_1 \le \tau_1'$ and $\tau_2' \le \tau_2$ by ST-Refl.]
- Case $\mathcal{ST} = \dfrac{\begin{array}{cc}\mathcal{ST}_0 & \mathcal{ST}_1\\ \tau' \le \tau'' & \tau'' \le \tau_1 \to \tau_2\end{array}}{\tau' \le \tau_1 \to \tau_2}$.
  By IH on $\mathcal{ST}_1$, $\tau'' = \tau_1'' \to \tau_2''$ for some τ_1'' and τ_2'' [with $\tau_1 \le \tau_1''$ and $\tau_2'' \le \tau_2$]. Hence, by IH on $\mathcal{ST}_0$, $\tau' = \tau_1' \to \tau_2'$ for some τ_1' and τ_2' [with $\tau_1'' \le \tau_1'$ and $\tau_2' \le \tau_2''$]. [Moreover, $\tau_1 \le \tau_1'' \le \tau_1'$ and $\tau_2' \le \tau_2'' \le \tau_2$ by ST-Trans.]
- $\mathcal{ST} = \dfrac{\begin{array}{cc}\mathcal{ST}_1 & \mathcal{ST}_2\\ \tau_1 \le \tau_1' & \tau_2' \le \tau_2\end{array}}{\tau_1' \to \tau_2' \le \tau_1 \to \tau_2}$.
  Here we get the claimed shape of τ' directly from the conclusion of the rule [and the subtypings from its premises].

c. For record types, there are 5 applicable rules: reflexivity, transitivity, and the 3 record rules:

- Case $\mathcal{ST} = \dfrac{}{\{(\ell_j : \tau_j)^{j\in 1..m}\} \le \{(\ell_j : \tau_j)^{j\in 1..m}\}}$.
  Immediate: we take $n = m$, and with $i = j$, we have $\ell'_i = \ell_i = \ell_j$ and $\tau'_i = \tau_i = \tau_j$. [We get $\tau'_i \le \tau_i$ by reflexivity.]
- Case $\mathcal{ST} = \dfrac{\mathcal{ST}_0 \quad \tau' \le \tau'' \qquad \mathcal{ST}_1 \quad \tau'' \le \{(\ell_j : \tau_j)^{j\in 1..m}\}}{\tau' \le \{(\ell_j : \tau_j)^{j\in 1..m}\}}$.
  By IH on $\mathcal{ST}_1$, we get that $\tau'' = \{(\ell''_k : \tau''_k)^{k\in 1..p}\}$ for some $p \ge m$, where for each j there exists a k with $\ell''_k = \ell_j$ [and $\tau''_k \le \tau_j$]. But then, by IH on $\mathcal{ST}_0$, we get $\tau' = \{(\ell'_i : \tau'_i)^{i\in 1..n}\}$ for some $n \ge p$, such that for each k, there exists an i with $\ell'_i = \ell''_k$ [and $\tau'_i \le \tau''_k$]. Hence, for each j we have an i with $\ell'_i = \ell''_k = \ell_j$ [and $\tau'_i \le \tau''_k \le \tau_j$ by transitivity].
- Case $\mathcal{ST} = \dfrac{}{\{(\ell'_j : \tau'_j)^{j\in 1..m}\} \le \{(\ell_j : \tau_j)^{j\in 1..m}\}}$, where $\ell_j = \ell'_{\pi(j)}$ and $\tau_j = \tau'_{\pi(j)}$.
  Then evidently $n = m$ and for each j, if we take $i = \pi(j)$, then $\ell'_i = \ell'_{\pi(j)} = \ell_j$ [and $\tau'_i = \tau'_{\pi(j)} \le \tau_j$ by reflexivity].
- Case $\mathcal{ST} = \dfrac{}{\{(\ell_j : \tau_j)^{j\in 1..m}, (\ell'_i : \tau'_i)^{i\in m+1..n}\} \le \{(\ell_j : \tau_j)^{j\in 1..m}\}}$.
  Then clearly for each $j \in 1..m$ we can take $i = j$ to get $\ell'_i = \ell_i = \ell_j$ directly [and $\tau'_i = \tau_i \le \tau_j$ by reflexivity].
- Case $\mathcal{ST} = \dfrac{\left(\mathcal{ST}_j \quad \tau'_j \le \tau_j\right)^{j\in 1..m}}{\{(\ell_j : \tau'_j)^{j\in 1..m}\} \le \{(\ell_j : \tau_j)^{j\in 1..m}\}}$.
  Here, like for reflexivity, $n = m$ and $\ell'_i = \ell_i$ [but we now get $\tau'_i \le \tau_i$ from the premise $\mathcal{ST}_i$ instead of by ST-Refl].

d. (The proof for variants is left as an exercise.)

■

Using subtype characterization, we can show the following lemma, whose proof was completely immediate in the system without subtyping, and thus silently inlined in those cases of the Progress and Preservation that needed it. Again, the parts in [brackets] are relevant only for Preservation.

Lemma 5.2 (Canonical forms) *Suppose* $[] \vdash c : \tau$. *Then:*

a. If $\tau = \mathbf{int}$, *then* $c = \overline{n}$ *for some* n.

b. If $\tau = \tau_1 \to \tau_2$, *then* $c = \lambda x.\, t_0$ *for some* x *and* t_0, *[and* $[x \mapsto \tau_1] \vdash t_0 : \tau_2$*].*

c. If $\tau = \{(\ell_j : \tau_j)^{j\in 1..m}\}$, *then* $c = \{(\ell'_i = c_i)^{i\in 1..n}\}$ *for some* $(\ell'_i)^{i\in 1..n}$ *and* $(c_i)^{i\in 1..n}$, *such that for each* $j \in 1..m$, *there exists an* $i \in 1..n$ *with* $\ell'_i = \ell_j$, *[and* $[] \vdash c_i : \tau_j$*].*

d. If $\tau = \langle(\ell_j : \tau_j)^{j\in 1..m}\rangle$, *then* $c = \langle \ell = c_0 \rangle$ *for some* ℓ *and* c_0, *and there exists a* $k \in 1..m$ *with* $\ell_k = \ell$, *[and* $[] \vdash c_0 : \tau_k$*].*

Proof. Each part proceeds by induction on the typing derivation $\mathcal{T}$. This derivation can end with the original, syntax-directed typing rule for canonical forms of type τ; or it can end with the subsumption rule, with a subderivation $\mathcal{T}'$ of $[] \vdash c : \tau'$ for some $\tau' \leq \tau$.

a. If $[] \vdash c : \mathbf{int}$ by T-Num, we immediately get $c = \overline{n}$, and we are done.

   In the T-Sub case, by Lemma 5.1(a), we must also have $\tau' = \mathbf{int}$, so the result follows by IH on $\mathcal{T}'$.

b. Similar to the **int** case. If $\mathcal{T}$ ends in T-Lam, we know that c is a lambda-abstraction, [and the premise of the typing rule is exactly what we need to show].

   If $\mathcal{T}$ ends in T-Sub, we use Lemma 5.1(b) to conclude that τ' must also be a function type, $\tau' = \tau_1' \to \tau_2'$ [with $\tau_1 \leq \tau_1'$ (by $\mathcal{ST}_1$) and $\tau_2' \leq \tau_2$ (by $\mathcal{ST}_2$)]. Hence, we can use the IH on $\mathcal{T}'$, to get that $c = \lambda x.\, t_0$, [and $[x \mapsto \tau_1'] \vdash t_0 : \tau_2'$ (by $\mathcal{T}_0'$)]. [Using a straightforward inner induction on $\mathcal{T}_0'$, we can rebuild a derivation $\mathcal{T}_0$ of $[x \mapsto \tau_1] \vdash t_0 : \tau_2'$, because whenever the original derivation $\mathcal{T}_0'$ used T-Var with x typed as τ_1', in the corresponding $\mathcal{T}_0$, we can use T-Var with x typed as τ_1, followed immediately by a subsumption using $\mathcal{ST}_1$. Then, by a final application of subsumption to $\mathcal{T}_0$ using $\mathcal{ST}_2$, we get $[x \mapsto \tau_1] \vdash t_0 : \tau_2$, as required.]

c. If $\mathcal{T}$ ends in T-Rcd, we must have $c = \{(\ell_j = c_j)^{j \in 1..m}\}$ [where $[] \vdash c_j : \tau_j$ for each j], so the conclusion follows immediately, taking $n = m$ and $\ell_i' = \ell_i$ for each i.

   Otherwise, we have derivations $\mathcal{T}'$ of $[] \vdash c : \tau'$ and $\mathcal{ST}$ of $\tau' \leq \{(\ell_j : \tau_j)^{j \in 1..m}\}$. Then, by Lemma 5.1(c) on $\mathcal{ST}$, we get that $\tau' = \{(\ell_i' : \tau_i')^{i \in 1..n}\}$, where each ℓ_j in τ is present as some ℓ_i' in τ', [and $\tau_i' \leq \tau_j$ (by $\mathcal{ST}_j$)]. By IH on $\mathcal{T}'$, c must be a record constructor containing at least the field bindings $(\ell_i' = c_i)^{i \in 1..n}$ [where $[] \vdash c_i : \tau_i'$ by $\mathcal{T}_i$ for each i], and thus in particular, also including all fields $(\ell_j)^{j \in 1..m}$. [And by subsumption using $\mathcal{T}_i$ and $\mathcal{ST}_j$, we also get $[] \vdash c_i : \tau_j$ for each of those fields.]

d. If $\mathcal{T}$ ends in T-Vnt' (remember that we use this restricted rule in the subtyping system), we must have $\tau = \langle \ell_1 : \tau_1 \rangle$ and $c = \langle \ell_1 = c_1 \rangle$ [where $[] \vdash c_1 : \tau_1$], so the conclusion is immediate (with $\ell = \ell_1$ and $c_0 = c_1$).

   Otherwise, by Lemma 5.1(d), $\tau' = \langle (\ell_i' : \tau_i')^{i \in 1..n} \rangle$, where each ℓ_i' in τ' is also present as some ℓ_j in τ [and $\tau_i' \leq \tau_j$ (by $\mathcal{ST}_i$)]. By IH on $\mathcal{T}'$ of $[] \vdash c : \langle (\ell_i' : \tau_i')^{i \in 1..n} \rangle$, we get that $c = \langle \ell_l' = c_0 \rangle$ for some $l \in 1..n$ and canonical form c_0 [where $[] \vdash c_0 : \tau_l'$ (by $\mathcal{T}_0$)]. But since $\ell_l' = \ell_k$ for some $k \in 1..m$, we also have $c = \langle \ell_k = c_0 \rangle$ [and $[] \vdash c_0 : \tau_k$ by subsumption using $\mathcal{T}_0$ and $\mathcal{ST}_l$], as required.

■

5.4.1 Progress

Using the canonical-forms lemma, the proof of Progress is now quite straightforward.

Lemma 5.3 (Progress) *If* $[] \vdash t : \tau$ *then either* $t = c$ *for some canonical form* c*, or* $t \to t'$ *for some term* t'*.*

Proof. The proof is essentially as before, by induction on the typing derivation $\mathcal{T}$. There is one new case, for T-Sub, and a couple of modified ones:

- Case $\mathcal{T} = \dfrac{\begin{array}{cc}\mathcal{T}_0 & \mathcal{ST} \\ [] \vdash t : \tau_0 & \tau_0 \le \tau\end{array}}{[] \vdash t : \tau}$.

  By IH on $\mathcal{T}_0$, the term t is either already canonical or can take a step, which was exactly what we needed to show.

- Case T-Plus. The subcases where at least one of the summands can step proceed exactly as before, but the final argument for when $t = c_0 + c_1$ must now use Lemma 5.2(a) to conclude that each c_i of type **int** must actually be a numeral, so $t = \overline{n_0} + \overline{n_1}$, which again can step by S-Plus.

- Case T-App. The cases for t_1 or t_2 stepping proceed as before, but when $t = c_1\, c_2$, we use Lemma 5.2(b) to conclude that $c_1 = \lambda x.\, t_0$, so that the application can be reduced by the S-App rule.

- Case $\mathcal{T} = \dfrac{\begin{array}{c}\mathcal{T}_0 \\ [] \vdash t_0 : \{\ell : \tau\}\end{array}}{[] \vdash t_0.\ell : \tau}$.

  (Note that, since we are using the subtyping system, we are considering only the restricted T-Sel' rule.) By IH on $\mathcal{T}_0$, either $t_0 \to t_0'$, in which case $t = t_0.\ell \to t_0'.\ell$ by S-Sel1; or t_0 is a canonical form. In the latter case, by Lemma 5.2(c), we must have $t_0 = \{(\ell_i' = c_i)^{i \in 1..n}\}$, with $\ell = \ell_1 = \ell_i'$ for some i, and thus $t_0.\ell$ reduces to c_i by S-Sel.

- (The cases for variant constructors and **case** are left as an exercise.)

■

5.4.2 Preservation

For preservation, we first need to re-prove the substitution lemma:

Lemma 5.4 (Substitution) *If* $\Gamma[x \mapsto \tau'] \vdash t : \tau$ *and* $[] \vdash t' : \tau'$*, then* $\Gamma \vdash t[t'/x] : \tau$*.*

Proof. We proceed as before, by induction on the first typing derivation $\mathcal{T}$, but with one additional case, which is completely analogous to all the other inductive cases that don't directly involve variables:

- Case $\mathcal{T} = \dfrac{\begin{array}{cc}\mathcal{T}_0 & \mathcal{ST} \\ \Gamma[x \mapsto \tau'] \vdash t : \tau_0 & \tau_0 \le \tau\end{array}}{\Gamma[x \mapsto \tau'] \vdash t : \tau}$.

  By IH on $\mathcal{T}_0$, $\Gamma \vdash t[t'/x] : \tau_0$, and hence by T-Sub with the original $\mathcal{ST}$, also $\Gamma \vdash t[t'/x] : \tau$, as required.

■

Lemma 5.5 (Preservation) *If* $[] \vdash t : \tau$ *(by* $\mathcal{T}$*) and* $t \to t'$ *(by* $\mathcal{S}$*), then* $[] \vdash t' : \tau$*.*

Proof. The proof is by an overall induction on the typing derivation $\mathcal{T}$, distinguishing just two cases: when $\mathcal{T}$ ends in T-Sub, and when it ends in any other rule (including axioms). In the former case, we have:

$$\mathcal{T} = \frac{\begin{array}{cc}\mathcal{T}' & \mathcal{S}\mathcal{T} \\ [] \vdash t : \tau' & \tau' \leq \tau\end{array}}{[] \vdash t : \tau}$$

By IH on $\mathcal{T}'$ with $\mathcal{S}$, we get $[] \vdash t' : \tau'$; and hence by T-Sub with $\mathcal{S}\mathcal{T}$, also $[] \vdash t' : \tau$.

In the other cases, we proceed by a case analysis of the stepping derivation $\mathcal{S}$, where we can now assume that $\mathcal{T}$ does not end in T-Sub. The proof cases are essentially as before, but now using the bracketed parts of Lemma 5.2 to reason about the typing properties of the relevant canonical forms:

- All the context-rule cases go through without significant modifications. For example, consider $\mathcal{S} = \dfrac{\begin{array}{c}\mathcal{S}_0 \\ t_0 \to t_0'\end{array}}{t_0.\ell \to t_0'.\ell}$. Since the typing derivation for $t_0.\ell$ cannot end with T-Sub (which was already taken care of above), we must have

  $$\mathcal{T} = \text{T-Sel'}\,\frac{\begin{array}{c}\mathcal{T}_0 \\ [] \vdash t_0 : \{\ell : \tau\}\end{array}}{[] \vdash t_0.\ell : \tau}$$

  Thus, by IH on $\mathcal{T}_0$ with $\mathcal{S}_0$, $[] \vdash t_0' : \{\ell : \tau\}$, and thus again by T-Sel', $[] \vdash t_0'.\ell : \tau$, as required.

- Case $\mathcal{S} = \dfrac{}{(\lambda x.\, t_0)\, c_2 \to t_0[c_2/x]}$.
  Since the typing derivation cannot end in T-Sub, it must look as follows:

  $$\mathcal{T} = \text{T-App}\,\frac{\begin{array}{cc}\mathcal{T}_0 & \mathcal{T}_1 \\ [] \vdash \lambda x.\, t_0 : \tau_1 \to \tau & [] \vdash c_2 : \tau_1\end{array}}{[] \vdash (\lambda x.\, t_0)\, c_2 : \tau}$$

  Unlike the corresponding case in the original Lemma 4.11, we *cannot* argue that $\mathcal{T}_0$ must end in T-Lam; but Lemma 5.2(b) plays essentially the same role: by using it on $\mathcal{T}_0$, we get to conclude that $[x \mapsto \tau_1] \vdash t_0 : \tau$ (by some $\mathcal{T}_{00}$); and then the extended Substitution Lemma 5.4 on $\mathcal{T}_{00}$ and $\mathcal{T}_1$ gives us $[] \vdash t_0[c_2/x] : \tau$, as required.

- (The case for S-Sel is left as an exercise.)

- Case $\mathcal{S} = \dfrac{}{\mathbf{case}\ \langle \ell_k = c_0 \rangle\ \mathbf{of}\ (\langle \ell_i = x_i \rangle \Rightarrow t_i)^{i \in 1..n} \to t_k[c_0/x_k]}$, where $k \in 1..n$.
  Since the typing derivation cannot end in T-Sub, it must end with a use of T-Case:

  $$\mathcal{T} = \frac{\begin{array}{cc}\mathcal{T}_0 & \left(\begin{array}{c}\mathcal{T}_i \\ [x_i \mapsto \tau_i] \vdash t_i : \tau\end{array}\right)^{i \in 1..n} \\ [] \vdash \langle \ell_k = c_0 \rangle : \langle (\ell_i : \tau_i)^{i \in 1..n} \rangle & \end{array}}{[] \vdash \mathbf{case}\ \langle \ell_k = c_0 \rangle\ \mathbf{of}\ (\langle \ell_i = x_i \rangle \Rightarrow t_i)^{i \in 1..n} : \tau}$$

Using Lemma 5.2(d) on $\mathcal{T}_0$ (where m in the lemma is taken as n, and c is taken as $\langle \ell_k = c_0 \rangle$), we get in the first instance that there exists a $k' \in 1..n$ such that $\ell_{k'} = \ell_k$, and $[] \vdash c_0 : \tau_{k'}$ (by some $\mathcal{T}_{00}$). But since labels in variant types are unique, we must actually have $k' = k$, so $\mathcal{T}_{00}$ also shows $[] \vdash c_0 : \tau_k$. And using the Substitution Lemma on $\mathcal{T}_k$ and $\mathcal{T}_{00}$, we again get the required $[] \vdash t_k[c_0/x_k] : \tau$.

- Case $\mathcal{S} = \dfrac{}{\mathbf{rec}\ x.\, t_0 \to t_0[(\mathbf{rec}\ x.\, t_0)/x]}$.

  Since $\mathcal{T}$ cannot end in T-Sub, it must end in T-Rec, and so the case proceeds exactly as in the original proof (only appealing to the new Substitution Lemma).

■

5.5 Exercises

1. In the system with subtyping, prove the cases for subtype characterization (Lemma 5.1(d)) and progress (Lemma 5.3) related to variant types, and those for preservation (Lemma 5.5) related to record types. Be sure to consider the correct rules.

2. Consider the system with just T-Var, T-Lam, T-App, T-Sub, ST-Refl, ST-Trans, and ST-Fun, together with some unspecified collection of subtyping axioms for base types (e.g., $\overline{\mathbf{bool} \leq \mathbf{int}}$). Show that the following typing rule is *admissible*, i.e., if the premise is derivable (without using the rule), then so is the conclusion:

$$\text{T-Eta}: \frac{\Gamma \vdash \lambda x.\, (t\, x) : \tau}{\Gamma \vdash t : \tau}\ (x \notin FV(t))$$

   (Conversely, one can show that any term typable in the original system can also be typed in the system without ST-Refl, ST-Trans, or ST-Fun, but with just T-Eta added. The proof of this is substantially more involved, though, and is not part of the exercise.)

   You may use, without separate proof, the following general result (*strengthening*): If $\Gamma[x \mapsto \tau'] \vdash t : \tau$ and $x \notin FV(t)$, then also $\Gamma \vdash t : \tau$.

   Hint: Show first the following, by induction on derivations:

   > **Lemma S.** If $\mathcal{T}$ is a derivation of $\Gamma \vdash t : \tau$, then for some τ_0, there exists a derivation $\mathcal{T}'$ of $\Gamma \vdash t : \tau_0$ where $\mathcal{T}'$ does not *end* in a use of T-Sub, and a derivation $\mathcal{ST}$ of $\tau_0 \leq \tau$.