Assignment 7: Subtyping and Labeled Variants

15-312: Foundations of Programming Languages
Tom Murphy 7
Modified by Kevin Watkins ⟨kw@cmu.edu⟩

Out: Thursday, April 7, 2005
Due: Thursday, April 14, 2005
but hand in Tuesday, April 19, 2005 (10:30 a.m.)

50 points total

In this assignment you will be asked to investigate several aspects of subtyping. For each question, assume that we are employing the subset interpretation of subtyping.

This assignment is nominally due in one week, on April 14, but since there's no class meeting then (owing to Carnival), you will actually be handing it in on the following Tuesday, April 19, in lecture (and that's when the late clock starts). Be advised, though, that Assignment 8 goes out on April 14.

§ 1. Width Subtyping for Products We've seen already (in lecture) some subtyping rules for records, namely width, depth, and permutation subtyping. Now products and records are rather similar; essentially the only difference is that the constituents of a record are labeled. So we might imagine carrying over some of these rules for records to the case of products. (In this assignment, all products are binary—they will have exactly two components.)

It's clear that permutation makes no sense for products; the labels are needed there. And we already have "depth" subtyping for products, given by the rule

$$\frac{\tau_1 \leq \sigma_1 \quad \tau_2 \leq \sigma_2}{\tau_1 \times \tau_2 \leq \sigma_1 \times \sigma_2}$$

from lecture.

But what about a "width" subtyping rule for products? Such a rule might look like

$$\frac{}{\tau_1 \times \tau_2 \leq \tau_1}$$

Q 1.1. [10 pts] Why would adding this rule to MinML be a bad idea? What key property of MinML's static and dynamic semantics would be violated? ("Key properties" might include: type safety, progress, preservation, canonical forms, or determinism. Be as specific as you can.) Show a MinML term that illustrates the problem.

§ 2. Labeled Variants By adding labels to our *products*, we arrived at the useful notion of *records*. What if we add labels to *sums*? Then we get what are called *labeled variants*. A labeled variant is like a sum type, except that it can contain any number of options (not necessarily two), and each option is

labeled. The **in** (injection) and **case** constructs then refer to these labels rather than referring to the various options by number.

The syntax is as follows:

$$\begin{array}{lrcl} \text{Types} & \tau & ::= & \cdots \mid [\ell_1 : \tau_1, \ldots, \ell_n : \tau_n] \\ \text{Terms} & e & ::= & \cdots \mid \mathbf{in}_\ell\, e \\ & & & \quad \mid (\mathbf{case}\ e\ \mathbf{of}\ \mathbf{in}_{\ell_1}\ x_1 \Rightarrow e_1 \mid \cdots \mid \mathbf{in}_{\ell_n}\ x_n \Rightarrow e_n) \\ \text{Values} & v & ::= & \cdots \mid \mathbf{in}_\ell\, v \end{array}$$

So for instance, we could write the type of integer lists as:

$$\mathbf{intlist} = (\mu\alpha.\, [\mathit{nil} : 1, \mathit{cons} : \mathbf{int} \times \alpha]),$$

and the list $[1, 2]$ would be coded as:

$$\mathbf{roll}\ (\mathbf{in}_{\mathit{cons}}\ (1, \mathbf{roll}\ (\mathbf{in}_{\mathit{cons}}\ (2, \mathbf{roll}\ (\mathbf{in}_{\mathit{nil}}\ ())))))),$$

and the operation $\mathit{hd}\, x$ to get the first element of a list could be coded as

$$\mathit{hd}\, x = (\mathbf{case}\ \mathbf{unroll}\ x\ \mathbf{of}\ \mathbf{in}_{\mathit{cons}}\ y \Rightarrow y.1 \mid \mathbf{in}_{\mathit{nil}}\ \_ \Rightarrow \mathbf{fail}).$$

Q 2.1. [10 pts] Come up with typing rules for $\mathbf{in}_\ell$ and **case** on labeled variants. Explain in words why your rules are correct. (It might help to go back and look at the rules for sums, and use them as a model.)

Q 2.2. [10 pts] Come up with subtyping rules for labeled variant types. You should have rules *analogous* to width, depth, and permutation subtyping for records. (But the rules will not be exactly the same!) Explain in words why your rules are correct.

Thinking about how to do the type safety proof for the language extended with labeled variants, it turns out that we need a major change to the *canonical forms lemma*. Usually, this lemma is extremely easy to prove; in fact, it's usually just a single inversion step. With subtyping, though, it turns out that an inductive argument is needed.

Q 2.3. [20 pts] State a canonical forms lemma for labeled variants with subtyping, and prove it.

Hints If you find the proof difficult, it may be that your typing rules or subtyping rules—the answers to questions 2.1 and 2.2—are wrong. Consider this possibility if you run into trouble. Also, it's likely you'll need an extra lemma (with its own induction) saying something about subtyping. It'll probably be easier to try the canonical forms proof first, and in the process of working it out, you'll find out what the other lemma you'll need is.