COMP 321 Lecture 14: Sum Types Lab

Dan Licata

Fall 2026

Recall the rules for binary sum types from Monday's class. A binary sum type is essentially a datatype

```
datatype T1 + T2 =
    left of T1
  | right of T2
```

so it has two constructors

```
left  : T1 -> T1 + T2
right : T2 -> T1 + T2
```

The new concrete syntax for these is `left(e)` and `right(e)`. The typing for these is

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathsf{Left}(e) : \tau_1 + \tau_2} \qquad \frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \mathsf{Right}(e) : \tau_1 + \tau_2}$$

Next, one can case-analyze a value of sum type to see if it was constructed with `left` or `right`, and in each case you get the data stored in that constructor. In concrete syntax, we write

```
case e of left(x) => e1 | right(y) => e2
```

where `x` is a new immutable variable that is in scope in `e1` and `y` is a new immutable variable that is in scope in `e2`. The abstract syntax is $\mathsf{Case}(e, x.e_1, y.e_2)$, which notates that the subtrees of a Case are e and e_1 and e_2, where e_1 has x in scope and e_2 has y in scope. The typing rule says that both branches must have the same type, and that x and y must have the type of the data that comes with Left and Right, respectively:

$$\frac{\Gamma \vdash e : \tau_1 + \tau_2 \quad \Gamma, x : \tau_1 \vdash e_1 : \tau \quad \Gamma, y : \tau_2 \vdash e_2 : \tau}{\Gamma \vdash \mathsf{Case}(e, x.e_1, y.e_2) : \tau}$$

The most important cases of the operational semantics say that if we case-analyze a constructed value, we step to the indicated branch, substituting the actual data in the constructor for the variable that stood for such data:

$$\frac{\mathsf{Left}(e)\ \mathsf{done}}{\mathsf{Case}(\mathsf{Left}(e), x.e_1, y.e_2) \mapsto e_1[e/x]}\ \texttt{case-instr-left} \qquad \frac{\mathsf{Right}(e)\ \mathsf{done}}{\mathsf{Case}(\mathsf{Right}(e), x.e_1, y.e_2) \mapsto e_2[e/y]}\ \texttt{case-instr-ri}$$

We also have search rules for each:

$$\frac{e \mapsto e'}{\mathsf{Left}(e) \mapsto \mathsf{Left}(e')}\ \texttt{Left-search} \qquad \frac{e\ \mathsf{done}}{\mathsf{Left}(e)\ \mathsf{done}}$$

$$\frac{e \mapsto e'}{\mathsf{Right}(e) \mapsto \mathsf{Right}(e')}\ \texttt{Right-search} \qquad \frac{e\ \mathsf{done}}{\mathsf{Right}(e)\ \mathsf{done}}$$

$$\frac{e \mapsto e'}{\mathsf{Case}(e, x.e_1, y.e_2) \mapsto \mathsf{Case}(e', x.e_1, y.e_2)}\ \texttt{Case-search}$$

1 Representing Datatypes

In this problem, you will show how a language with sum types can encode various datatypes. For each of the following datatypes, we will represent the type using sum and product types, and show how to define the constructors and case analysis.

For example, consider the datatype

```
datatype bool = True of unit | False of unit
```

This defines constructors and case analysis as follows:

$$\frac{}{\Gamma \vdash \mathsf{True}() : \mathsf{bool}} \qquad \frac{}{\Gamma \vdash \mathsf{False}() : \mathsf{bool}}$$

$$\frac{}{\mathsf{True}()\ \mathsf{done}} \qquad \frac{}{\mathsf{False}()\ \mathsf{done}}$$

$$\frac{\Gamma \vdash e : \mathsf{bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{Case}(e, e_1, e_2) : \tau}$$

$$\frac{e \mapsto e'}{\mathsf{Case}(e, e_1, e_2) \mapsto \mathsf{Case}(e', e_1, e_2)}$$

$$\frac{}{\mathsf{Case}(\mathsf{True}(), e_1, e_2) \mapsto e_1} \qquad \frac{}{\mathsf{Case}(\mathsf{False}(), e_1, e_2) \mapsto e_2}$$

The encoding is

```
bool = unit + unit

True() = Left()

False() = Right()

(case e of True() => e1 | False () => e2) =
  case e of Left _ => e1 | Right _ => e2
```

This shows how to define the datatype to be a particular use of binary sum types, how to represent each of the constructors using the sum type constructors, and how the expected case analysis is defined using the case for binary sum types. You can think of the encoding as like an *elaborator* (as in Homework 3) from a syntax that has `bool`,`True()`,`False()`, and `case` for booleans to a syntax that has binary sum types. The encoding satisfies the typing and operational semantics.

Traffic light Consider the datatype

```
datatype trafficlight =
   Red of unit
 | Green of unit
 | Yellow of unit
```

This has three constructors, `Red()` and `Green()` and `Yellow()`, each of which takes a single input `()` of type unit. The expected case analysis is

```
case (e : trafficlight) of
  Red() => e1
| Green() => e2
| Yellow() => e3
```

For example, here is some code using these language constructs:

```
fun nextColor(t : trafficlight) : trafficlight =
  case t of
    Red() => Green()
  | Green() => Yellow()
  | Yellow() => Red()
```

Task 1. Give a typing rule and operational semantics for the three constructors and the `case`.

$$\frac{}{\Gamma \vdash \mathsf{Red}() : \mathsf{TrafficLight}} \qquad \frac{}{\Gamma \vdash \mathsf{Yellow}() : \mathsf{TrafficLight}} \qquad \frac{}{\Gamma \vdash \mathsf{Green}() : \mathsf{TrafficLight}}$$

$$\frac{\Gamma \vdash e : \mathsf{TrafficLight} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \mathsf{TrafficLightCase}(e, e_1, e_2, e_3) : \tau}$$

$$\frac{}{\mathsf{Red}() \ \mathsf{done}} \qquad \frac{}{\mathsf{Yellow}() \ \mathsf{done}} \qquad \frac{}{\mathsf{Green}() \ \mathsf{done}}$$

$$\frac{e \mapsto e'}{\mathsf{TrafficLightCase}(e, e_1, e_2, e_3) \mapsto \mathsf{TrafficLightCase}(e', e_1, e_2, e_3)}$$

$$\frac{}{\mathsf{TrafficLightCase}(\mathsf{Red}(), e_1, e_2, e_3) \mapsto e_1}$$

$$\frac{}{\mathsf{TrafficLightCase}(\mathsf{Yellow}(), e_1, e_2, e_3) \mapsto e_2}$$

$$\frac{}{\mathsf{TrafficLightCase}(\mathsf{Green}(), e_1, e_2, e_3) \mapsto e_3}$$

Task 2. Give an encoding of the constructors and case for traffic lights using binary sum types. The encoding should implement the typing and operational semantics from the previous part.

$$\begin{aligned}
&\mathsf{TrafficLight} = \mathsf{unit} + (\mathsf{unit} + \mathsf{unit}) \\
&\mathsf{Red}() = \mathsf{Left}() \\
&\mathsf{Yellow}() = \mathsf{Right}(\mathsf{Left}()) \\
&\mathsf{Green}() = \mathsf{Right}(\mathsf{Right}()) \\
&\mathsf{TrafficLightCase}(e, e_1, e_2, e_3) = \mathsf{case}(e, _.e_1, z.\mathsf{case}(z, _.e_2, _.e_3))
\end{aligned}$$

Step Results Here is a datatype like one you have used in the homework:

```
datatype result =
   Done
 | Stepped of exp
```

Task 1. Give a typing rule and operational semantics for the two constructors and the `case` for `result`.

$$\frac{}{\Gamma \vdash \mathsf{Done}() : \mathsf{result}} \qquad \frac{\Gamma \vdash e : \mathsf{exp}}{\Gamma \vdash \mathsf{Stepped}(e) : \mathsf{result}}$$

$$\frac{\Gamma \vdash e : \mathsf{result} \quad \Gamma \vdash e_1 : \tau \quad \Gamma, x : \mathsf{exp} \vdash e_2 : \tau}{\Gamma \vdash \mathsf{ResultCase}(e, e_1, x.e_2) : \tau}$$

$$\frac{}{\mathsf{Done}()\ \mathsf{done}} \qquad \frac{e\ \mathsf{done}}{\mathsf{Stepped(e)}()\ \mathsf{done}}$$

$$\frac{e \mapsto e'}{\mathsf{ResultCase}(e, e_1, x.e_2) \mapsto \mathsf{ResultCase}(e', e_1, x.e_2)}$$

$$\frac{}{\mathsf{ResultCase}(\mathsf{Done}(), e_1, e_2, e_3) \mapsto e_1}$$

$$\frac{e\ \mathsf{done}}{\mathsf{ResultCase}(\mathsf{Stepped}(e), e_1, x.e_2) \mapsto e_2[e/x]}$$

Task 2. Give an encoding of the constructors and case for `result` using binary sum types. The encoding should implement the typing and operational semantics from the previous part.

$$\begin{aligned}
&\mathsf{result} = \mathsf{unit} + \mathsf{exp} \\
&\mathsf{Done}() = \mathsf{Left}() \\
&\mathsf{Stepped}(\mathsf{e}') = \mathsf{Right}(e') \\
&\mathsf{ResultCase}(e, e_1, x.e_2) = \mathsf{case}(e, _.e_1, x.e_2)
\end{aligned}$$

2 Properties of Product and Sum Types

In this problem, you will write some functions in a functional language FUN with product and sum types. For precision, we can use the following typing and operational semantics rules for functions:

$$\frac{x : \tau_1 \vdash e : \tau_2}{(f(x : \tau_1) : \tau_2 = e)\ \mathsf{ok}} \qquad \frac{(f(x : \tau_1) : \tau_2 = \ldots) \in p \quad \Gamma \vdash e : \tau_1}{\Gamma \vdash f(e) : \tau_2}$$

$$\frac{e \mapsto e'}{fe \mapsto fe'} \qquad \frac{(f(x : \tau_1) : \tau_2 = e_2) \in p}{fe_1 \mapsto e_2[e_1/x]}$$

Task 1. In each of the following, τ_1, τ_2, τ_3 stand for arbitrary types. Write a function of the indicated type:

- Sums are commutative

$$(\tau_1 + \tau_2) \to (\tau_2 + \tau_1)$$

$$comm(x : \tau_1 + \tau_2) : \tau_2 + \tau_1 = \mathsf{case}\, x\, \mathsf{of}\, \mathsf{Left}(x) \Rightarrow \mathsf{Right}(x) \mid \mathsf{Right}(y) \Rightarrow \mathsf{Left}(y)$$

- Sums are associative

$$((\tau_1 + \tau_2) + \tau_3) \to (\tau_1 + (\tau_2 + \tau_3))$$

saved for possible homework problems

- Sums are associative part 2

$$(\tau_1 + (\tau_2 + \tau_3)) \to ((\tau_1 + \tau_2) + \tau_3)$$

- Products distribute over sums:

$$(\tau_1 \times (\tau_2 + \tau_3)) \to ((\tau_1 \times \tau_2) + (\tau_1 \times \tau_3))$$

- Products distribute over sums part 2:

$$((\tau_1 \times \tau_2) + (\tau_1 \times \tau_3)) \to (\tau_1 \times (\tau_2 + \tau_3))$$

- Sums distribute over products:

$$(\tau_1 + (\tau_2 \times \tau_3)) \to ((\tau_1 + \tau_2) \times (\tau_1 + \tau_3))$$

- Sums distribute over products part 2:[1]

$$((\tau_1 + \tau_2) \times (\tau_1 + \tau_3)) \to (\tau_1 + (\tau_2 \times \tau_3))$$

[1] Thought question: How is this one different from all of the others?

- Products are commutative

$$(\tau_1 \times \tau_2) \to (\tau_2 \times \tau_1)$$

- Products are associative

$$((\tau_1 \times \tau_2) \times \tau_3) \to (\tau_1 \times (\tau_2 \times \tau_3))$$

- Products are associative part 2

$$(\tau_1 \times (\tau_2 \times \tau_3)) \to ((\tau_1 \times \tau_2) \times \tau_3)$$

Task 2. Define the cardinality of a type to be the number of done expressions of that type. What is the cardinality of a type made up of of any number of uses of the type constructors unit, $\times$, and $+$?

3 Void

The unit type with single value $()$ is a *nullary product* (a product of no types). Give syntax and typing rules for a type void that is a *nullary sum* (a datatype with no constructors). Define functions $(\mathsf{void} + \tau) \to \tau$ and $(\tau + \mathsf{void}) \to \tau$.