lab27.txt

```
+-----------------------+
| CS 4110 - 10/31/2025  |
| Lab 27 - Type Systems |
+-----------------------+
```

In this lab, we do a few examples using the extension of lambda calculus with sum types we have studied previously.

```
e ::= x
    | \x . e
    | e1 + e2
    | inl_{t1 + t2} e
    | inr_{t1 + t2} e
    | (case e of e1 | e2)
```

```
G |- e : t1
---------------------------[T-Inl]
G |- inl_{t1+t2} e : t1 + t2
```

```
G |- e : t2
---------------------------[T-Inr]
G |- inr_{t1+t2} e : t1 + t2
```

```
G |- e : t1 + t2
G |- e1 : t1 -> t
G |- e2 : t2 -> t
-------------------------[T-Case]
G |- case e of e1 | e2 : t
```

```
E ::= ... | inl_{t1+t2} E | inr_{t1+t2} E | (case E of e1 | e2)
```

```
-------------------------------------[E-Case-Inl]
case inl_{t1+t2} v of e1 | e2 -> e1 v
```

```
-------------------------------------[E-Case-Inr]
case inr_{t1+t2} v of e1 | e2 -> e2 v
```

* Q : Suppose we add the following axiom to our operational semantics:

  ```
  ---------------------------------------[E-Case-Weird]
  case inl_{t1 + t2} e of e1 | e2 -> e1 e
  ```

  Which, if any of, the key properties breaks? Progress? Preservation?

  A: Nothing breaks. In fact, adding operational semantics rules can only make Progress easier to satisfy. And, it so happens that the Preservation theorem still holds here, even though the new axiom changes the evaluation order.

* Q: Now suppose we add this axiom instead:

  ```
  -------------------------[E-Case-Weird]
  case e of e1 | e2 -> e1 e
  ```

  Which, if any of, the key properties breaks? Progress? Preservation?

  A: Again, Progress still holds as adding operational semantics rules can only make it easier to satisfy. But Preservation definitely breaks. Here is a counter-example:

  ```
    e0 = case inl_{unit + int} () of
              (\x : unit. x)
            | (\n:int. ())
  ```

  By T-Case, T-Inl, and other rules we have

  ```
    e0 : unit
  ```

  By E-Case-Weird, we have

  ```
    e0 -> e0'
  ```

```
    where
    e0' = (\x : unit. x) inl_{unit + int} ()
```

And e0' is definitely *not* well typed!

* Q: What if we keep the new axiom,

```
  -------------------------[E-Case-Weird]
  case e of e1 | e2 -> e1 e
```

but also add the following rule to the type system:

```
  G |- e : t1 + t2
  G |- e1 : t1 + t2 -> t
  --------------------------[T-Case-Weird2]
  G |- case e of e1 | e2 : t
```

Which, if any of, the key properties breaks? Progress? Preservation?

A: This is a bit tricky. The new typing rule does seem like it might "patch up" cases where E-Case-Weird caused a problem, by forcing e1 to have a function type whose domain is the entire sum type. However, we can still break Preservation by typing an expression using the original typing rule, T-Case, and then letting it step with E-Case-Weird.

In fact, the same counter-example as above still works.

```
  e0 = case inl_{unit + int} () of
            (\x : unit. x)
          | (\n:int. ())
```

By T-Case, T-Inl, and other rules, we still have

```
  e0 : unit
```

Again, by E-Case-Weird, we still have

```
  e0 -> e0'
  where
  e0' = (\x : unit. x) inl_{unit + int} ()
```

And e0' is still not well typed!