

COMP 321 Lecture 12: Substitution/Products Lab

Dan Licata

Fall 2026

1 Substitution

Consider a language with the following expressions and typing rules:

$$e ::= x \mid \text{Num}(k) \mid \text{Plus}(e_1, e_2) \mid \text{Let}(e_1, x.e_2)$$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \quad \frac{}{\Gamma \vdash \text{Num}(k) : \text{int}} \quad \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash \text{Plus}(e_1, e_2) : \text{int}} \quad \frac{\Gamma \vdash e : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{Let}(e_1, x.e_2) : \tau_2}$$

Consider the following definition of substitution (like your `subst_exp` from Homework 3):

$$\begin{aligned} x_0[e_0/x_0] &= e_0 \\ y[e_0/x_0] &= y \text{ if } y \neq x_0 \\ \text{Num}(k)[e_0/x_0] &= \text{Num}(k) \\ \text{Plus}(e_1, e_2)[e_0/x_0] &= \text{Plus}(e_1[e_0/x_0], e_2[e_0/x_0]) \\ \text{Let}(e_1, y.e_2)[e_0/x_0] &= \text{Let}(e_1[e_0/x_0], y.e_2[e_0/x_0]) \text{ if } x_0 \neq y \end{aligned}$$

Task 1. Prove the substitution lemma:

If $\Gamma, x_0 : \tau_0, \Gamma' \vdash e : \tau$ and $\Gamma \vdash e_0 : \tau_0$ then $\Gamma, \Gamma' \vdash e[e_0/x_0] : \tau$.

The proof is by induction on $\Gamma, x_0 : \tau_0, \Gamma' \vdash e : \tau$, so you should start with one case for each typing rule:

- Case for

$$\frac{x : \tau \in (\Gamma, x_0 : \tau_0, \Gamma')}{\Gamma, x_0 : \tau_0, \Gamma' \vdash x : \tau}$$

- Case for

$$\frac{}{\Gamma, x_0 : \tau_0, \Gamma' \vdash \text{Num}(k) : \text{int}}$$

- Case for

$$\frac{\Gamma, x_0 : \tau_0, \Gamma' \vdash e_1 : \text{int} \quad \Gamma, x_0 : \tau_0, \Gamma' \vdash e_2 : \text{int}}{\Gamma, x_0 : \tau_0, \Gamma' \vdash \text{Plus}(e_1, e_2) : \text{int}}$$

- Case for

$$\frac{\Gamma, x_0 : \tau_0, \Gamma' \vdash e : \tau_1 \quad \Gamma, x_0 : \tau_0, \Gamma', x : \tau_1 \vdash e_2 : \tau_2}{\Gamma, x_0 : \tau_0, \Gamma' \vdash \text{Let}(e_1, x.e_2) : \tau_2}$$

State and prove any lemmas that you need.

Have us check your work!

Task 2. Do the case of the type preservation theorem

If $\cdot \vdash e : \tau$ and $e \mapsto e'$ then $\cdot \vdash e' : \tau$.

for the following rule

$$\frac{e_1 \text{ done}}{\text{Let}(e_1, x.e_2) \mapsto e_2[e_1/x]}$$

and show how the above substitution lemma is used.

2 Pattern-Matching for Products

In class, we discussed the following rules for product types:

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \text{Pair}(e_1, e_2) : \tau_1 \times \tau_2} \quad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \text{First}(e) : \tau_1} \quad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \text{Second}(e) : \tau_2}$$

$$\frac{e_1 \mapsto e'_1}{\text{Pair}(e_1, e_2) \mapsto \text{Pair}(e'_1, e_2)} \text{ pair-search-1} \quad \frac{e_2 \mapsto e'_2 \quad e_1 \text{ done}}{\text{Pair}(e_1, e_2) \mapsto \text{Pair}(e_1, e'_2)} \text{ pair-search-2}$$

$$\frac{e_1 \text{ done} \quad e_2 \text{ done}}{\text{Pair}(e_1, e_2) \text{ done}} \text{ pair-done}$$

$$\frac{e \mapsto e'}{\text{First}(e) \mapsto \text{First}(e')} \text{ first-search} \quad \frac{\text{Pair}(e_1, e_2) \text{ done}}{\text{First}(\text{Pair}(e_1, e_2)) \mapsto e_1} \text{ first-instr}$$

$$\frac{e \mapsto e'}{\text{Second}(e) \mapsto \text{Second}(e')} \text{ second-search} \quad \frac{\text{Pair}(e_1, e_2) \text{ done}}{\text{Second}(\text{Pair}(e_1, e_2)) \mapsto e_2} \text{ second-instr}$$

These rules let you access the components of a tuple by projecting them individually. For example, here is what a function left-associates a triple looks like:¹

```
reassociate (t : int x (int x int)) : (int x int) x int =
  ( ( first(t) , first(second(t)) ) , second(second(t)) )
```

A programmer might want a nicer notation in which one can pattern-match on a pair and give names to individual components. To start with, we can add a language feature for pattern-matching on a single pair so that the above example can be written

¹ Assume we have functions whose bodies are expressions which return the value of that expression; we will cover this formally soon.

```

reassociate (t : int x (int x int)) : (int x int) x int =
  split (x,p) = t in
    split (y,z) = p in
      ((x,y),z)

```

Task 1. Give abstract syntax, typing rules, and operational semantics for `split (x,y) = e in e'`.

Have us check your work!

Task 2. Prove the cases of progress and preservation for `split`.

Have us check your work!

Task 3. Show that, when added to the language described in Section 1, the projections `First` and `Second` are *interdefinable* with `Split`. This means:

- In a language with `split`, define `First(e)` and `Second(e)` and show that these definitions satisfy the above typing and operational semantics rules. That is, define

$myFirst(e) := \text{some expression using Split and } e$

and then show in the language with `Split` but without `First` that

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash myFirst(e) : \tau_1} \quad \frac{e \mapsto e'}{myFirst(e) \mapsto myFirst(e')} \quad \frac{Pair(e_1, e_2) \text{ done}}{myFirst(Pair(e_1, e_2)) \mapsto e_1}$$

and then do the same thing for $mySecond$.

- In a language with `First` and `Second` (and `Let` as above), define `Split` and show that the definition satisfies the typing and operational semantics rules that you gave above in Task 1. A single step in the language with `Split` be implemented by several steps in the language with `First`, `Second`, and `Let`.

Have us check your work!

Task 4. Define the syntax, typing, and operational semantics for a more general notion of pattern matching so that the above example can be written like

```

reassociate (t : int x (int x int)) : (int x int) x int =
  split (x, (y,z)) = t in ((x,y),z)

```

Have us check your work!