Exercise 9: Prolog and Semantics

# Exercise 9.1: Unification

max(X, Y, Z) :- (X >= Y -> Z = X ; Z = Y).

This predicate uses prologs if else statements to find the larger of the two inputs.

# Exercise 9.2: Semantic Proof

## Dynamic-Scoping Natural Semantics

### fn():

The semantics for applying a function fn(x, E2) to an expression E1 in a context C can be written as: <apply(fn(x, E2), E1), C> -> v This means that evaluating the application of the function to E1 in context C results in a value v.

When we evaluate ((fn x => E2) E1), we first evaluate E1 in context C to get a value v1.

<E1, C> -> v1

Next, we substitute v1 for x in E2 and evaluate E2 in the updated context C[x -> v1] to get the final value v.

<E2[x -> v1], C> -> v

### let():

The semantics for the let expression let val x = E1 in E2 end in a context C can be written as: <let(x, E1, E2), C> -> v This means that evaluating the let expression in context C results in a value v.

When we evaluate let val x = E1 in E2 end, we first evaluate E1 in context C to get a value v1.

<E1, C> -> v1

Next, we substitute v1 for x in E2 and evaluate E2 in the updated context C[x -> v1] to get the final value v.

<E2[x -> v1], C> -> v

Since both fn() and let() use the same steps in the context of C to get v, we can say they are equivalent.

## Static-Scoping Natural Semantics

### fn():

The semantics for applying a function fn(x, E2) to an expression E1 in a context C can be written as: <apply(fn(x, E2), E1), C> -> v This means that evaluating the application of the function to E1 in context C results in a value v.

When we evaluate ((fn x => E2) E1), we first evaluate E1 in context C to get a value v1.

<E1, C> -> v1

Next, we substitute v1 for x in E2 and evaluate E2 in the updated context C[x -> v1] to get the final value v.

<E2[x -> v1], C[x -> v1]> -> v

### let():

The semantics for the let expression let val x = E1 in E2 end in a context C can be written as: <let(x, E1, E2), C> -> v This means that evaluating the let expression in context C results in a value v.

When we evaluate let val x = E1 in E2 end, we first evaluate E1 in context C to get a value v1.

<E1, C> -> v1

Next, we substitute v1 for x in E2 and evaluate E2 in the updated context C[x -> v1] to get the final value v.

<E2[x -> v1], C[x -> v1]> -> v

Since both fn() and let() use the same steps in the context of C to get v, we can say they are equivalent.

## Conclusion:

The assertion that let val x = E1 in E2 end is equivalent to ((fn x => E2) E1) is true under both dynamic and static scoping versions of the natural semantics for Language Three.