Problem 1

empty list does nothing to position/direction:

$$\frac{}{(pos, [\,]) \downarrow pos}$$

home resets position/direction regardless of prior state:

$$\frac{((0,0,0), lst) \downarrow pos}{(pos', \text{home} :: lst) \downarrow pos}$$

forward changes position relative to prior position, depending on direction, leaves direction unaltered:

$$\frac{((x - f\cos(d), y - f\sin(d), d), lst) \downarrow pos}{((x, y, d), \text{forward}\ \ f :: lst) \downarrow pos}$$

turn changes direction, leaves position unaltered:

$$\frac{((x, y, d - f), lst) \downarrow pos}{((x, y, d), \text{turn}\ \ f :: lst) \downarrow pos}$$

for obeys induction for non-zero numbers of steps

$$\frac{i > 0 \qquad (pos, flst) \downarrow (pos_1) \qquad (pos_1, \text{for}\ \ (i-1)(flist) :: lst) \downarrow pos'}{(pos, \text{for}\ \ i(flist) :: lst) \downarrow pos'}$$

for terminates at step zero

$$\frac{(pos, lst) \downarrow pos}{(pos, \text{for}\ \ 0(flist) :: lst) \downarrow pos'}$$

Problem 3b

i. Programs where interp1 would require significantly more space for evaluation than interp2.

Programs that are interpreted with a large number of variable bindings in the environment, where many of these are not actually used, and those that are used are typically used in only one enclosure. This would make interp1 require more space, since the preprocessing done for interp2 would drop all unused variables.

ii. Programs where interp1 would require significantly less space for evaluation than interp2.

Programs that are interpreted with any number of variable bindings in the environment, where most of these are actually used, and those that are used are typically used in many enclosures. This would make interp1 require less space, since the preprocessing done for interp2 would create a copy of the value for each enclosure the variable appears in.

iii. Programs where interp1 would require significantly more time for evaluation than interp2.

If we assume that looking up a variable in the environment is cost O(1), then I don't see how interp1 could require significantly more time than interp1.

iv. Programs where interp1would require significantly less time for evaluation than interp2 .

Programs that have many closures where many of the free variables are referenced, but because of the flow of the program, their values are never actually required (e.g. they're in a branch that is not executed, or they're in an element of a pair that is disregarded). In this case, the function used by interp2 to pre-substitute the values in the closure will be significantly more expensive than looking the value up only when needed, as done by interp1.