Discussion of Some of the Interpreter HW Problems

① What needs to be done to add a primitive?

(a) mechanics of adding newprim

Changes needed to atom-to-atom, and to myapply-primitive. But then of course we have to ask: is this enough? What about correctness? What is the criterion for correctness? In fact, how do we know that the currently installed primitives are correctly installed?

(b) correctness?

We'd want to show that every time newprim (or, indeed, any prim) occurs, it is correctly evaluated.

This requires a structural induction. Let's consider how we might argue that the new1 primitive is correctly included in the existing interpreter.

a new primitive

Basis: new1 doesn't occur in numbers or booleans, so it is vacuously true that each occurrence of add1 in these is correctly evaluated. (There are none)

We assume that TLS (as it is currently) works.

Moreover, the only way new! occurs in an identifier is if x actually is new!. In that case, the system's response to

(value 'new!)

is

('primitive new!)

[and we judge this correct]

IH: we assume that any occurrence of new! in a subexpression of the valid TLS input is correctly evaluated.

IS: we consider cases. If the input expression (call it e) is

— a cond, Then e has the form

(cond $(q_1\ a_1)$
$(q_2\ a_2)$
⋮
$(q_n\ a_n)$)

If new1 occurs in e, then it must occur in q_i or a_i, for some i. Now - each of these is a subexpression of e. So by the IH, and knowing that TLS evaluates cond by evaluating $e_1, \ldots, e_k$ until e_k is true, followed by a_k — we see that new1 will be correctly evaluated wherever it occurs in the cond.

because we know that TLS does a full evaluation on the q_i and the a_i

Of course, other cases have to be considered -

(lambda vars e) ✓ even if new1 occurs in e, the system's response to this particular input does not entail evaluating e

(quote e) ✓ somewhere

This all looks suspiciously simple, but we are not out of the woods yet!

What about applications?

Assume new1 is unary

These, too, yield almost entirely to the IH — except for 1 critical case:

what if the application is

(new1 e1) ?

We have to argue specifically that the appropriate R5RS definition of new1 has been added to myapply-primitive, and Then

① IH gives us what we need for e1

② the specific behavior of TLS in this case must stated to correctly compute the result of applying the R5RS def. of new1 to the value of e1

At this point, we'd be done -

---

onto Problem 2

Again, a pf is needed. What needs to be shown is that the changes proposed for the environment subsystem do NOT affect the working of the interpreter.

Specifically: figure out the specification for the env. subsystem and show that your modified subsystem still satisfies this spec.

ie while new ins

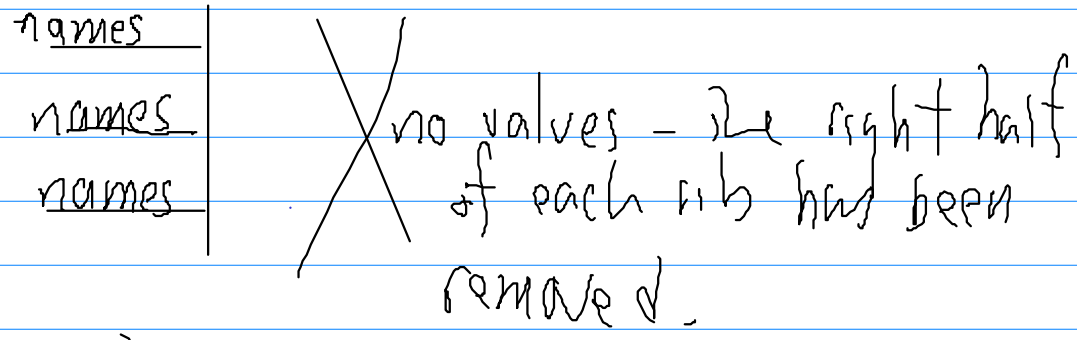
This is NOT an induction.

On to Prob 3

→ simple-check needs no commentary

→ nor does the arity check.

→ but the presence of unbound variables — That's fairly cool. I suggest that you consider modifying the ribs so that they become "half-ribs". By that I mean:



(lambda (x_1) ---

(lambda (x_2) ----

x_3 ----)

its absence of the corresponding lambda which can be

(with no corresponding lambda

checked using the half-rib environment.

Finally, prob 4, adding let to TLS

A clean solution is to write a preprocessor which desugars each occurrence of let

You're defining TLS-LET, which is TLS with another type — namely *let. One idea is to define *let to do the desugaring — for just the one occurrence of let then under consideration.

Then — a structural induction like that for prob 1 — will be needed. You need to show

That every time let occurs, it is correctly evaluated.

Eg - consider cond

(cond (q_1 a_1)
⋮
(q_n a_n))

If let occurs in here, then it occurs within any of the q_i and/or any of the a_i.

So - correctly evaluated by the (structural) IH.

Finally —

① See if you can show for TLS that any reference to a variable x uses the closest enclosing binding of x.

structural induction
(and maybe a subinduction on lambda nesting depth)

② Try to deduce everything you can about lexical scoping in TLS, given that TLS satisfies ①

HINT: don't forget about closures. I try to grok the connection between closures and lexical scope