Tiger type rules, revision #2

Schema:
   S,D,G |-^b exp : t,S'

G : variable -o-> type[never nil] b is a boolean
S : k -o-> (record (x t) ...) or (array t) e is a tiger AST.
D : id -o-> t k is a number
t is int, string, void, nil, or k

A tiger expression e type checks with type t if (for any S):

0,{int=int,string=string},0 |-^false e: t,S

(In tiger programs "int" is an identifier, but here it is a type, which is separate from the identifier (except in the line just above where it is both, but hopefully that's clear).)

----------[num] ----------[str]
S,D,G |-^b num : int,S S,D,G |-^b str : string,S

----------[void] ----------[var]
S,D,G |-^b () : void,S S,D,G |-^b x : G(t),S

S,D,G |-^b e1 : int,S' S',D,G |-^b e2 : int,S''
----------[binop]
S,D,G |-^b (binop e1 e2) : int,S''

S,D,G |-^b e1 : string S',D,G |-^b e2 : string,S''
----------[eqop]
S,D,G |-^b (eqop e1 e2) : num,S''

S,D,G |-^b e1 : int,S' S',D,G |-^true e2 : t,S''
----------[while] ----------[break]
S,D,G |-^b (while e1 e2) : void,S'' S,D,G |-^true (break) : void,S

S,D,G |-^b e1 : int,S' S',D,G |-^b e2 : t,S''
----------[when]
S,D,G |-^b (when e1 e2) : void,S''

S,D,G |-^b e1 : t1,S' S',D,G |-^b e2: t2,S''
----------[begin2]
S,D,G |-^b (begin e1 e2) : t2,S''

S,D,G |-^b e1 : t1,S' S',D,G |-^b (begin e2 e3 e4 ...) : t,S''
----------[beginN]
S,D,G |-^b (begin e1 e2 e3 e4 ...) : t,S''

S,D,G |-^b e1 : int,S' S',D,G |-^b e2 : t2,S'' S'',D,G |-^b e3 : t3,S'''
----------[if]
S,D,G |-^b (if e1 e2 e3) : same(t2,t3),S'''

S,D,G |-^b e1 : int,S' S',D,G |-^b e2 : int,S'' S'',D,G+{x:int} |-^true e3 : t,S'''
----------[for]
S,D,G |-^b (for (x e1 e2) e3) : void,S'''

```
     S,D+{id=D(id')},G |- e : t, S'
-----------------------------------------[let-tid]
S,D,G |- (let ([type id id']) e) : t, S'


S+{k=array D(id')},D+{id=k},G |- e : t, S'        k \not \in \dom(S)
-------------------------------------------------------------------[let-array]
    S,D,G |- (let ([type id (array id')]) e) : t, S'


S+{k=record{x:D(id'),...}},D+{id=k},G |- e : t, S'   k \not \in \dom(S)
-----------------------------------------------------------------------[let-record]
   S,D,G |- (let ([type id (record (x id') ...)]) e) : t, S'


S,D,G |- e' : t',S'      S',D,G+{id:t'} |- e : t, S''     t' =/= nil
-----------------------------------------------------------------[let-var]
             S,D,G |- (let ([var id e']) e) : t, S''


S,D,G |- e' : t',S'     S',D,G+{id:same(t',D(tid))} |- e : t, S''     t =/= nil
-------------------------------------------------------------------------------[let-var-t]
              S,D,G |- (let ([var id tid e']) e): t, S''


S,D,G |- lvalue : k,S'     S'(k) = array t     S',D,G |- e : int,S''
-------------------------------------------------------------------[aref]
                  S,D,G |- (aref lvalue e) : t,S'''


S,D,G |- lvalue : k,S'       S'(k) = record{...,id:t,...}
-----------------------------------------------------[dot]
              S,D,G |- (dot lvalue id) : t,S'

S,D,G |-^b e1 : k            S',D,G |-^b e2 : k,S''
-------------------------------------------[eqlop]
          S,D,G |-^b (eqop e1 e2) : num,S''


----------------------[nil]
S,D,G |-^b nil : nil,S


S,D,G |-^b lvalue : t,S''     S',D,G |-^b e : t',S''   same(t,t')
-----------------------------------------------------------------[set]
          S,D,G |-^b (:= lvalue e) : void,S''

                same : t t -> t
                same(int,int) = int                      same(k,k) = k
                same(string,string) = string            same(nil,k) = k
                same(void,void) = void                   same(k,nil) = k
```

When the same function is used above, it is assumed to be defined. (In other words, if same is not defined for the types in some rule, you cannot use that rule.)

Similarly, when an environment is used with some variable, the environment must have a binding for the rule in order to use the rule.

The only expression form missing from these pages is let expressions that have multiple bindings. Multiple var bindings are the same as nested let expressions, each with one var binding. To handle those, collect each maximal sequence of type declarations together, ensure there are no cycles in just the 'type id = id' declarations, and then apply the first three rules on this page simultaneously to the sequence of declarations.