### A. *Haskell Syntax*

```
Term Variable  v ::= x, y, z

Type variable  α ::= a, b, c

Type constant  t ::= Int | Bool

         Type  τ ::= α | t | $τ_1$ → $τ_2$

   Expression  e ::= v | λx.e | $e_1$ $e_2$
                     | let v::τ = $e_1$ in $e_2$
```

### B. *Prolog Syntax*

```
         Term  T ::= V | A | C | L

     Variable  V ::= X, Y, Z

         Atom  A ::= p, q, r

Compound Term  C ::= A($T_1$, $T_2$,...)

         List  L ::= []|[$T_1$, $T_2$,...]|[T|L]

       Clause  P ::= C ← $T_1$, $T_2$,...$T_n$.
```

### C. *Auxiliary Functions*

```
function fresh():
 return concat('_', randomInteger())

function atom(t):
 return lowercase(t)

function var(v):
 return uppercase(v)

function append_clause(P):
 $\mathcal{P}$ ← $\mathcal{P}$ ∪ {P}
```

### D. *Predicate Generation Function*

```
function gen_decl(Γ, x :: τ = e):
  $V_d$ ← fresh()
  for each $v_i$ ∈ Γ
    $V_i$ ← var($v_i$)
  $T_1$, $T_2$, ..., $T_n$ ← gen(Γ, e, $V_d$) ∪ gen(Γ, τ, $V_d$)
  return x(V, [$V_1$, $V_2$ ..., $V_i$|_]) ← $T_1$, $T_2$ ..., $T_n$.
```

### E. *Constraints Generation Function*

```
function gen(Γ, v, V):  # Var
 if v ∈ Γ
   then
     V' ← var(v)
     return {V = V'}
   else
     for each $v_i$ ∈ Γ
       $V_i$ ← var($v_i$)
     return {x(V, [$V_1$, $V_2$, ..., $V_i$|_])}

function gen(Γ, λv.e, V): # Lambda Expression
  $V_e$ ← fresh()
  $V_v$ ← var(v)
  return {V = fun($V_v$, $V_e$)} ∪ gen(Γ ∪ {v}, e, $V_e$)

function gen(Γ, $e_1$ $e_2$, V): # Application
  $V_1$ ← var($e_1$)
  $V_2$ ← var($e_2$)
  return {$V_1$ = fun($V_2$, V)} ∪ gen(Γ, $e_1$, $V_1$) ∪ gen(Γ, $e_2$, $V_2$)

function gen(Γ, let v::τ = $e_1$ in $e_2$, V): # Let Expression
  $V_e$ ← fresh()
  P ← gen_decl(Γ, v :: τ = $e_1$)
  append_clause(P)
  return {V = $V_e$} ∪ gen(Γ, $e_2$, $V_2$)

function gen(Γ, t, V): # Concrete Type
  A ← atom(t)
  return {A = V}

function gen(Γ, α, V): # Type Variable
  $V_α$ ← var(α)
  return {V = $V_α$}

function gen(Γ, $τ_1$ → $τ_2$, V): # Function Type
  $V_1$ ← fresh()
  $V_2$ ← fresh()
  return {V = fun($V_1$, $V_2$)} ∪ gen(Γ, $τ_1$, $V_1$) ∪ gen(Γ, $τ_2$, $V_2$)
```