A. *Haskell Syntax*

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

Type variable  α ::= a, b, c

Type constant  t ::= Int | Bool

         Type  τ ::= α | t | τ₁ → τ₂

   Expression  e ::= v | λx.e | e₁ e₂
                       | let v::τ = e₁ in e₂
```

B. *Prolog Syntax*

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

     Variable  V ::= X, Y, Z

         Atom  A ::= p, q, r

Compound Term  C ::= A(T₁, T₂, ...)

         List  L ::= []|[T₁, T₂, ...]|[T|L]

       Clause  P ::= C ← T₁, T₂, ...Tₙ.
```

C. *Auxiliary Functions*

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

function atom(t):
 return lowercase(t)

function var(v):
 return uppercase(v)
```

D. *Predicate Generation Function*

```
function gen_decl(Γ, x :: τ = e):
  V ← fresh()
  for each vᵢ ∈ Γ
    Vᵢ ← var(vᵢ)
  T₁, T₂, T₃ ..., Tₙ ← gen(Γ, e, V) ∪ gen(Γ, τ, V)
  𝒫 ← 𝒫 ∪ {x(V, [V₁, V₂, V₃ ...|_]) ← T₁, T₂, T₃ ...}
```

E. *Constraints Generation Function*

```
function gen(Γ, v, V):
 if v ∈ Γ
   then
     V' ← var(v)
     return {V = V'}
   else
     for each vᵢ ∈ Γ
       Vᵢ ← var(vᵢ)
     return {x(V, [V₁, V₂, ...|_])}

function gen(Γ, λv.e, V):
  Vₑ ← fresh()
  Vₓ ← var(v)
  return {V = fun(Vₓ, Vₑ)} ∪ gen(Γ ∪ {v}, e, Vₑ)

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

function gen(Γ, α, V):
  Vα ← var(α)
  return {V = Vα}

function gen(Γ, τ₁ → τ₂, V):
  V₁ ← fresh()
  V₂ ← fresh()
  return {V = fun(V₁, V₂)} ∪ gen(Γ, τ₁, V₁) ∪ gen(Γ, τ₂, V₂)

function gen(Γ, let v::τ = e₁ in e₂, V):
  Vₑ ← fresh()
  gen_decl(Γ, v :: τ = e₁)
  return {V = Vₑ} ∪ gen(Γ, e₂, V₂)
```