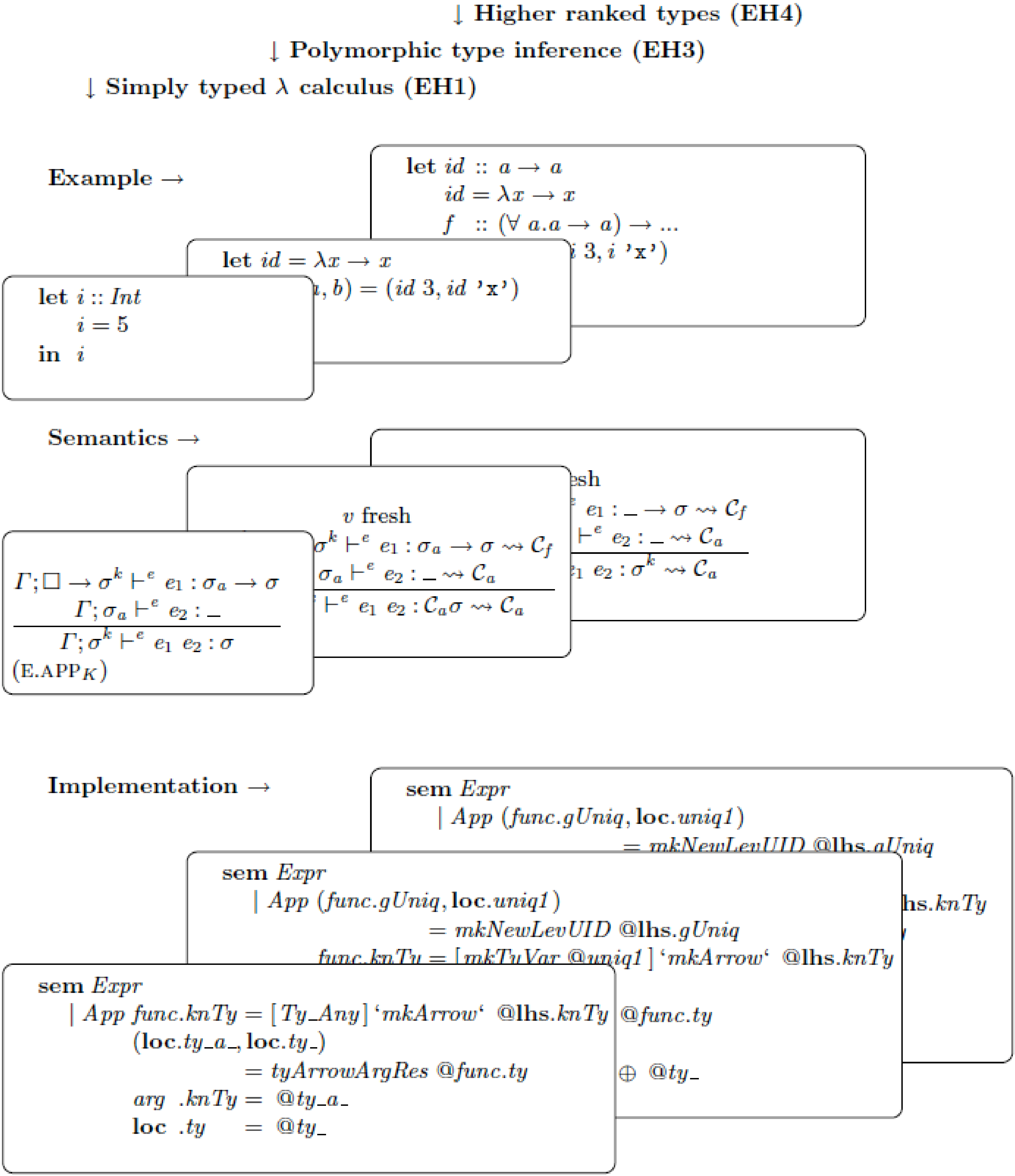
↓ Higher ranked types (EH4)
↓ Polymorphic type inference (EH3)
↓ Simply typed λ calculus (EH1)
Example →
let id :: a → a
id = λx → x
f :: (∀ a.a → a) → ...
i 3, i 'x')
let id = λx → x
a, b) = (id 3, id 'x')
let i :: Int
i = 5
in i
Semantics →
esh
e1 : _ → σ ⇝ C_f
⊢^e e2 : _ ⇝ C_a
e1 e2 : σ^k ⇝ C_a
v fresh
σ^k ⊢^e e1 : σ_a → σ ⇝ C_f
σ_a ⊢^e e2 : _ ⇝ C_a
⊢^e e1 e2 : C_a σ ⇝ C_a
Γ; □ → σ^k ⊢^e e1 : σ_a → σ
Γ; σ_a ⊢^e e2 : _
Γ; σ^k ⊢^e e1 e2 : σ
(E.APP_K)
Implementation →
sem Expr
| App (func.gUniq, loc.uniq1)
= mkNewLevUID @lhs.gUniq
hs.knTy
sem Expr
| App (func.gUniq, loc.uniq1)
= mkNewLevUID @lhs.gUniq
func.knTy = [mkTyVar @uniq1] 'mkArrow' @lhs.knTy
@func.ty
⊕ @ty_
sem Expr
| App func.knTy = [Ty_Any] 'mkArrow' @lhs.knTy
(loc.ty_a_, loc.ty_)
= tyArrowArgRes @func.ty
arg .knTy = @ty_a_
loc .ty = @ty_