

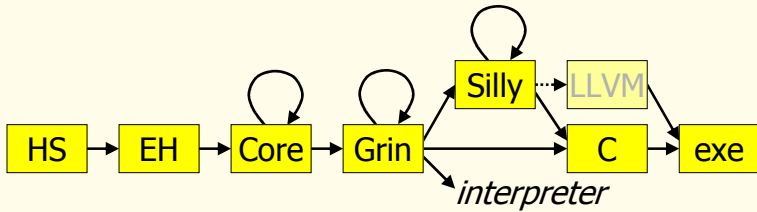
### Observations:

Programmers want programming languages to do as much as possible of their programming job  
Users want guarantees of resulting programs, e.g. no errors

### Resulting problem:

Programming language + compiler become more complex

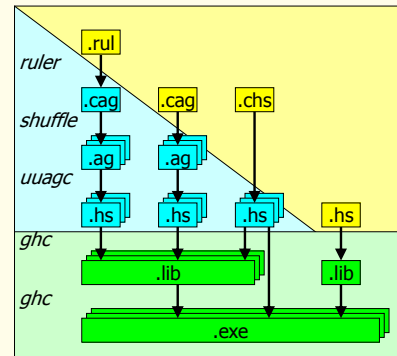
### Coping with implementation complexity: *transform, transform and transform*



- From complex to simple representations

### Coping with maintenance complexity: *generate, generate and generate*

from common source: guarantees consistency of generated artefacts



- Chunks (.chs, .cag): for program, documentation (etc) combination
- Attribute Grammar (.ag): for tree based computation
- Ruler (.rul): for type rules

### Coping with design complexity: *stepwise grow a language*

↓ Higher ranked types (EH4)

↓ Polymorphic type inference (EH3)

↓ Simply typed  $\lambda$  calculus (EH1)

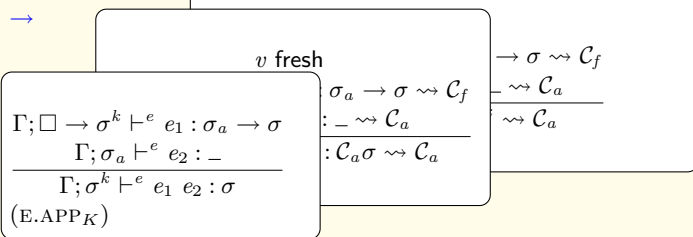
#### Example

```
let id :: a -> a
    id = \x -> x
    f :: (\forall a.a -> a) -> ...
```

```
let id = \x -> x
    id 'x')
```

```
let i :: Int
    i = 5
in i
```

#### Semantics



#### Implementation

```
sem Expr
  | App (func.gUniq, loc.uniql)
    = mkNewLevUID @lhs.gUniq
    loc.uniql = mkTyVar @uniql
```

```
sem Expr
  | App (func.gUniq, loc.uniql)
    = mkNewLevUID @lhs.gUniq
    func.knTy = [mkTyVar @uniql] mkArrow' @lhs.knTy
```

```
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_
```

### Coping with formalisation complexity: *domain specific languages*

$$\begin{array}{l}
 v \text{ fresh} \\
 o_{str}; \Gamma; \mathbb{C}^k; \mathbb{C}^k; v \rightarrow \sigma^k \vdash^e e_1 : \sigma_f; \_ \rightarrow \sigma \rightsquigarrow \mathbb{C}_f; \mathbb{C}_f \\
 o_{im}; \mathbb{C}_f \vdash^e \sigma_f \leq \mathbb{C}_f(v \rightarrow \sigma^k) : \_ \rightsquigarrow \mathbb{C}_F \\
 o_{inst-lr}; \Gamma; \mathbb{C}_F \mathbb{C}_f; \mathbb{C}_f; v \vdash^e e_2 : \sigma_a; \_ \rightsquigarrow \mathbb{C}_a; \mathbb{C}_a \\
 ff_{alt}^+; o_{inst-l}; \mathbb{C}_a \vdash^e \sigma_a \leq \mathbb{C}_a v : \_ \rightsquigarrow \mathbb{C}_A \\
 \hline
 \mathbb{C}_1 \equiv \mathbb{C}_A \mathbb{C}_a \\
 o; \Gamma; \mathbb{C}^k; \mathbb{C}^k; \sigma^k \vdash^e e_1 e_2 : \mathbb{C}_1 \sigma^k; \sigma^k \rightsquigarrow \mathbb{C}_1; \mathbb{C}_a \\
 \text{(E.APP}_{I2}\text{)}
 \end{array}$$

- Specification of type rules
- Implementation of type rules, different strategies
- Pretty printing type rules

### Future plans

- Incremental evaluation
- Parallel compilers
- Use of visual environments (Proxima)
- Efficient analysis
- ...