Essential Haskell

coping with compiler complexity

Netherlands Organisation for Scientific Research

Universiteit Utrecht

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 design complexity: stepwise grow a language (Haskell in particular)

Feature: Simply typed λ calculus $\rightarrow$ Polymorphic type inference $\rightarrow$ Higher ranked types $\rightarrow$ ...

Example:

Simply typed λ calculus:

$$\begin{array}{l} \textbf{let}\ i :: Int \\ \quad\ i = 5 \\ \textbf{in}\ \ i \end{array}$$

$\rightarrow$ Polymorphic type inference:

$$\begin{array}{l} \textbf{let}\ id = \lambda x \to x \\ \textbf{in}\ \ (id\ 3, id\ \texttt{'x'}) \end{array}$$

$\rightarrow$ Higher ranked types:

$$\begin{array}{l} \textbf{let}\ id :: a \to a \\ \quad\ id = \lambda x \to x \\ \quad\ f\ \ :: (\forall\ a.a \to a) \to \ldots \\ \quad\ f\ \ = \lambda i \to (i\ 3, i\ \texttt{'x'}) \\ \textbf{in}\ \ f\ id \end{array}$$

$\rightarrow$ ...

Semantics:

$$\dfrac{\begin{array}{c} \Gamma; \square \to \sigma^k \vdash^e e_1 : \sigma_a \to \sigma \\ \Gamma; \sigma_a \vdash^e e_2 : \_ \end{array}}{\Gamma; \sigma^k \vdash^e e_1\ e_2 : \sigma} \quad (\text{E.APP}_K)$$

$\rightarrow$

$$\dfrac{\begin{array}{c} v\ \text{fresh} \\ o_{str}; \Gamma; C^k; v \to \sigma^k \vdash^e e_1 : \_ \to \sigma \rightsquigarrow C_f \\ o_{inst-lr}; \Gamma; C_f; v \vdash^e e_2 : \_ \rightsquigarrow C_a \end{array}}{o; \Gamma; C^k; \sigma^k \vdash^e e_1\ e_2 : C_a \sigma \rightsquigarrow C_a} \quad (\text{E.APP}_{I1})$$

$\rightarrow$

$$\dfrac{\begin{array}{c} v\ \text{fresh} \\ o_{str}; \Gamma; \mathbb{C}^k; C^k; v \to \sigma^k \vdash^e e_1 : \sigma_f; \_ \to \sigma \rightsquigarrow \mathbb{C}_f; C_f \\ o_{im} \vdash^{\leqslant} \sigma_f \leqslant \mathbb{C}_f(v \to \sigma^k) : \_ \rightsquigarrow \mathbb{C}_F \\ o_{inst-lr}; \Gamma; \mathbb{C}_F \mathbb{C}_f; C_f; v \vdash^e e_2 : \sigma_a; \_ \rightsquigarrow \mathbb{C}_a; C_a \\ fi^+_{alt}, o_{inst-l} \vdash^{\leqslant} \sigma_a \leqslant \mathbb{C}_a v : \_ \rightsquigarrow \mathbb{C}_A \\ \mathbb{C}_1 \equiv \mathbb{C}_A \mathbb{C}_a \end{array}}{o; \Gamma; \mathbb{C}^k; C^k; \sigma^k \vdash^e e_1\ e_2 : \mathbb{C}_1 \sigma^k; C_a \sigma \rightsquigarrow \mathbb{C}_1; C_a} \quad (\text{E.APP}_{I2})$$

$\rightarrow$ ...

Implementation:

```
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_
```

$\rightarrow$

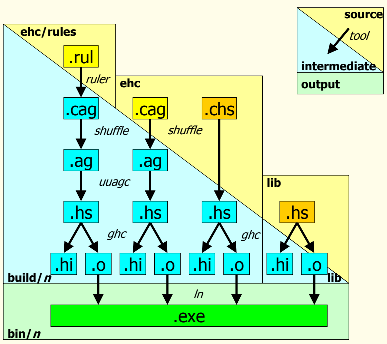
```
sem Expr
  | App (func.gUniq, loc.uniq1)
                         = mkNewLevUID @lhs.gUniq
        loc  .tvarv_     = mkTyVar @uniq1
        func.fiOpts      = o_str
            .knTy        = [@tvarv_] `mkArrow` @lhs.knTy
        (_, loc.ty_)     = tyArrowArgRes @func.ty
        arg .fiOpts      = o_inst-lr
            .knTy        = @tvarv_
        loc  .ty         = @arg.tyCnstr ⊕ @ty_
```

$\rightarrow$ ... $\rightarrow$ ...

Documentation: ...

Coping with maintenance complexity: generate, generate and generate

from common source: guarantees consistency of generated artefacts



- Chunks (.chs, .cag): combine chunks of text for program, documentation, ...
- Attribute Grammar (.ag): domain specific language for tree based computation
- Ruler (.rul): domain specific language for type rules

Coping with formalisation complexity: domain specific languages

$$\dfrac{\begin{array}{c} v\ \text{fresh} \\ o_{str}; \Gamma; \mathbb{C}^k; C^k; v \to \sigma^k \vdash^e e_1 : \sigma_f; \_ \to \sigma \rightsquigarrow \mathbb{C}_f; C_f \\ o_{im} \vdash^{\leqslant} \sigma_f \leqslant \mathbb{C}_f(v \to \sigma^k) : \_ \rightsquigarrow \mathbb{C}_F \\ o_{inst-lr}; \Gamma; \mathbb{C}_F \mathbb{C}_f; C_f; v \vdash^e e_2 : \sigma_a; \_ \rightsquigarrow \mathbb{C}_a; C_a \\ fi^+_{alt}, o_{inst-l} \vdash^{\leqslant} \sigma_a \leqslant \mathbb{C}_a v : \_ \rightsquigarrow \mathbb{C}_A \\ \mathbb{C}_1 \equiv \mathbb{C}_A \mathbb{C}_a \end{array}}{o; \Gamma; \mathbb{C}^k; C^k; \sigma^k \vdash^e e_1\ e_2 : \mathbb{C}_1 \sigma^k; C_a \sigma \rightsquigarrow \mathbb{C}_1; C_a} \quad (\text{E.APP}_{I2})$$

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

More information

- Supported by NWO program ‘Hefboom’ (project 641.000.412)

- Participants: Prof. Dr. S. Doaitse Swierstra, Dr. Bastiaan J. Heeren, Dr. Atze Dijkstra, Drs. Jeroen D. Fokker, Drs. Arie Middelkoop

- See `http://www.cs.uu.nl/wiki/Ehc/WebHome`