Coping with Compiler Complexity

The structure of the Essential Haskell Compiler

Observations:

Programmers want programming languages to do as much as possible of their programming job

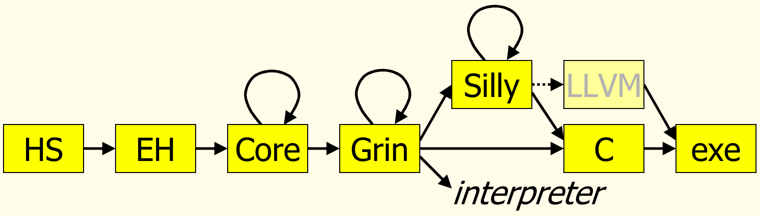
Users want guarantees of resulting programs, e.g. no errors

Resulting problem:

Programming language + compiler become more complex

Coping with implementation complexity:

transform, transform and transform



- From complex to simple representations

Coping with design complexity:

stepwise grow a language

↓ **Higher ranked types (EH4)**

↓ **Polymorphic type inference (EH3)**

↓ **Simply typed λ calculus (EH1)**

Example →

$\textbf{let}\ id :: a \rightarrow a$
$\quad id = \lambda x \rightarrow x$
$\quad f\ :: (\forall\ a.a \rightarrow a) \rightarrow ...$

$\textbf{let}\ id = \lambda x \rightarrow x$
$, id\ \texttt{'x'})$

$\textbf{let}\ i :: Int$
$\quad i = 5$
$\textbf{in}\ \ i$

Semantics →

$\rightarrow \sigma \leadsto \mathcal{C}_f$
$\_ \leadsto \mathcal{C}_a$
$\leadsto \mathcal{C}_a$

v fresh
$: \sigma_a \rightarrow \sigma \leadsto \mathcal{C}_f$
$: \_ \leadsto \mathcal{C}_a$
$: \mathcal{C}_a\sigma \leadsto \mathcal{C}_a$

$$\frac{\Gamma; \square \rightarrow \sigma^k \vdash^e e_1 : \sigma_a \rightarrow \sigma \qquad \Gamma; \sigma_a \vdash^e e_2 : \_}{\Gamma; \sigma^k \vdash^e e_1\ e_2 : \sigma}$$
(E.APP_K)

Implementation →

$\textbf{sem}\ Expr$
$\quad |\ App\ (func.gUniq, \textbf{loc}.uniq1)$
$\qquad\qquad = mkNewLevUID\ @\textbf{lhs}.gUniq$
$\qquad \textbf{loc}\ .tvar\ = mkTyVar\ @uniq1$

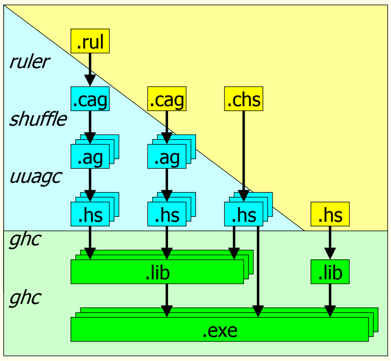
$\textbf{sem}\ Expr$
$\quad |\ App\ (func.gUniq, \textbf{loc}.uniq1)$
$\qquad\qquad = mkNewLevUID\ @\textbf{lhs}.gUniq$
$\qquad func.knTy = [mkTyVar\ @uniq1]\ `mkArrow`\ \ @\textbf{lhs}.knTy$

$\textbf{sem}\ Expr$
$\quad |\ App\ func.knTy = [\,Ty\_Any]\ `mkArrow`\ \ @\textbf{lhs}.knTy$
$\qquad (\textbf{loc}.ty\_a\_, \textbf{loc}.ty\_)$
$\qquad\qquad = tyArrowArgRes\ @func.ty$
$\qquad arg\ \ .knTy = @ty\_a\_$
$\qquad \textbf{loc}\ \ .ty\quad = @ty\_$

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 formalisation complexity:

domain specific languages

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

- 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
- ...

NWO ‘Hefboom’ project 641.000.412,
NWO project 612.063.410,
Microsoft Research Scholarship

Prof. Dr. S. Doaitse Swierstra,
Dr. Atze Dijkstra, Drs. Jeroen Fokker,
Drs. Arie Middelkoop

http://www.cs.uu.nl/wiki/Ehc/WebHome