Solving Data-flow Problems in Syntax Trees

Sebastian Graf

September 9, 2017

Karlsruhe Institute of Technology

Introduction

- My master's thesis[1]: Call Arity vs. Demand Analysis
  - Result: Usage Analysis generalising Call Arity
  - Precision of Call Arity without co-call graphs
- Requirements led to complex analysis order
- *Specification* of data-flow problem decoupled from its *solution*

[1] https://pp.ipd.kit.edu/uploads/publikationen/graf17masterarbeit.pdf

Strictness Analysis

- Provides lower bounds on *evaluation cardinality*
- Which variables are evaluated at least once?
  - *S* Strict (Yes!)
  - *L* Lazy (Not sure)
- Enables call-by-value, unboxing

```
1  main = do
2    let  x = ... -- S
3    let  y = ... -- S
4    let  z = ... -- L
5    print (x + if odd y then y else z)
```

Strictness Analysis

- Provides lower bounds on *evaluation cardinality*
- Which variables are evaluated at least once?
  - S Strict (Yes!)
  - L Lazy (Not sure)
- Enables call-by-value, unboxing

```
1  main = do
2    let !x = ... -- S
3    let !y = ... -- S
4    let  z = ... -- L
5    print (x + if odd y then y else z)
```

GHC's Demand Analyser

- Performs strictness analysis (among other things)
- Fuels Worker/Wrapper transformation
- Backward analysis
  - Which strictness does an expression place on its free variables?
  - Which strictness does a function place its arguments?

Strictness Signatures

- Interprocedural data-flow
- Looks at the right-hand side of `const` before the `let` body!
- *Strictness type*: $\mathsf{StrType} = \langle \mathsf{FVs} \rightharpoonup \mathsf{Str}, \mathsf{StrSig} \rangle$
- *Unleashes* usage type of `const`'s RHS at call sites

```
1  let const a b = a -- const :: ⟨[], S → L → •⟩
2  in const
3       y            -- S
4       (fac 1000)   -- L
```

Call Context Matters

End