Diagnosing Type Errors with Class

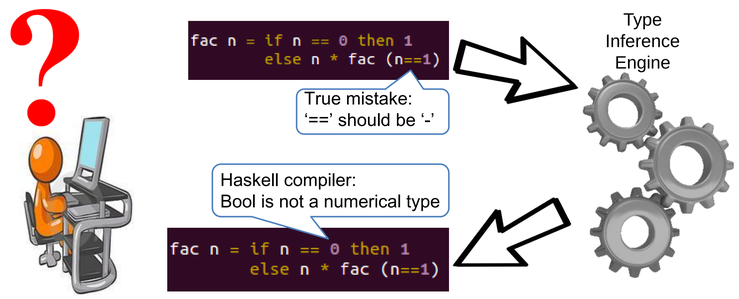
Danfeng Zhang and Andrew C. Myers
Cornell University

Dimitrios Vytiniotis and Simon Peyton-Jones
Microsoft Research Cambridge

A general, expressive, and accurate diagnostic method for Haskell errors

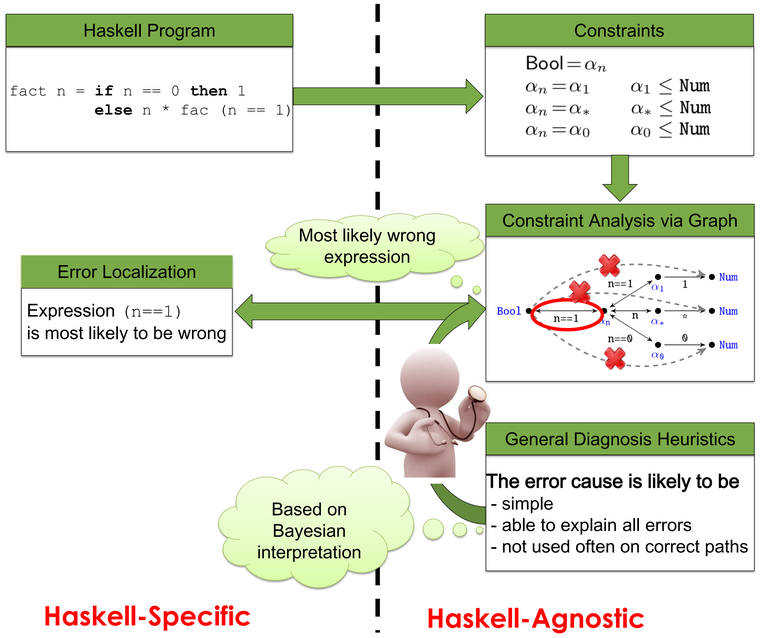
Motivation

Type-inference error messages are sometimes confusing



SHErrLoc Overview

A general approach to diagnosing Haskell errors



Challenges

We handle the highly expressive type system of GHC

Haskell features we handle
- ☐ Type classes
- ☐ Type families
- ☐ GADTs
- ☐ Type signatures

SHErrLoc Constraints

A rich constraint language that models GHC type checking

Syntax of Constraints

Constraint elements $E ::= \alpha \mid \mathrm{a} \mid con(E_1, \ldots, E_n) \mid fun(E_1, \ldots, E_n)$
Quantified Axioms $Q ::= \forall a. \wedge_i I_i \Rightarrow I$
Inequalities $I ::= E_1 \leq E_2$ Constraints $C ::= \wedge_j (\wedge_i Q_i \vdash I_j)$

Haskell Program

```
fact n = if n == 0 then 1
         else n * fac (n == 1)
```

Constraints

Bool = α_n
$\alpha_n = \alpha_1$ $\alpha_1 \leq$ Num
$\alpha_n = \alpha_*$ $\alpha_* \leq$ Num
$\alpha_n = \alpha_0$ $\alpha_0 \leq$ Num

Constraints may have hypotheses and axioms

$(\mathrm{Int} \leq \mathrm{Eq}) \wedge (\forall a.\, a \leq \mathrm{Eq} \Rightarrow [a] \leq \mathrm{Eq}) \vdash [\mathrm{Int}] \leq \mathrm{Eq}$ ✓

Type Classes

Idea: model type classes by partial ordering constraints

```
ghci> 20 :: Int
20
ghci> 20 :: Integer
20
ghci> 20 :: Float
20.0
ghci> 20 :: Double
20.0
```

Instances of a type class, called Num

Intuitively, a set of types

- Modeling type class constraints

$[\![D\ \tau]\!] := \tau \leq \mathrm{D}$
$[\![D\ \overline{\tau}]\!] := (\mathrm{tup}_n\ \overline{\tau}) \leq \mathrm{D}$

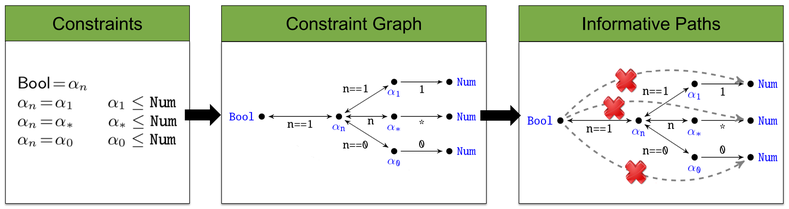
Examples

$[\![\mathrm{Num\ Int}]\!] = \mathrm{Int} \leq \mathrm{Num}$
$[\![\mathrm{Mul}\ \tau_1\ \tau_2\ \tau_3]\!] = (\mathrm{tup}_3\ \tau_1\ \tau_2\ \tau_3) \leq \mathrm{Mul}$

Constraint Analysis Overview

Efficient constraint analysis via graph saturation

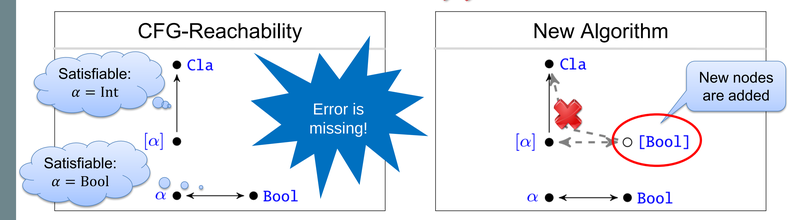
- Graph construction
  - Node: constraint elements
  - Edge: partial orders on elements
- Finding informative paths
  - Saturate constraint graph
  - Test the satisfiability of a partial order on end nodes
  - Trivial paths are ignored (e.g., one end node is a variable)



Graph Saturation

Idea: add new graph nodes and edges during saturation

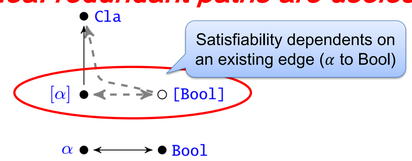
$[\mathrm{Int}] \leq \mathrm{Cla} \vdash \alpha = \mathrm{Bool} \wedge [\alpha] \leq \mathrm{Cla}$ ✗ $[\mathrm{Bool}] \not\leq \mathrm{Cla}$



The new algorithm is decidable and efficient in practice

Inferring Likely Wrong Constraints

Idea: redundant paths are useless in ranking



Ranking metric

$$\underset{E}{\mathrm{argmax}}\ P_1^{|E|}\left(\frac{P_2}{1-P_2}\right)^{k_E}$$

E: set of constraints
k_E: # of sat *non-redundant* paths using E

Lemma: for any redundant path P (see definition in the paper), there exist non-redundant paths P_i, such that (P is satisfiable $\Leftrightarrow \forall i.\, P_i$ is satisfiable).

Evaluation

Our tool identifies error locations more accurately

Correctness metric
- CE Benchmark: well-marked errors in the benchmark
- Helium Benchmark: user's fix with larger time stamp

CE Benchmark

Programs collected by Chen&Erwig, from papers on error diagnosis

Analyzed 77 programs with well-marked errors

Helium Benchmark

Student programs collected by the Helium tool

Analyzed 228 programs with type-checking errors