PERFORMANCE EVALUATION FOR GRADUAL TYPING

Asumu Takikawa, Daniel Feltey, Ben Greenman, Max S. New, Jan Vitek, Matthias Felleisen

GRADUAL TYPING is for software maintenance

Fact 1: developers use untyped languages
Fact 2: type annotations enable safety checks and serve as documentation
Thesis: stable untyped code + type annotations = happy future maintainers

EVALUATION METHOD

Report the relative performance of the untyped and fully-typed configurations

- with "deliverable" overhead (at most $N_x$ slowdown)
- with "usable" overhead (at most $M_x$ slowdown)
- within $L$ conversion steps from an $N_x$ or $M_x$ configuration

PROMISES

Freedom to add types incrementally
Soundness: type invariants are enforced at runtime . . . What about performance?

PERFORMANCE LATTICE

Visualizing all possible gradually-typed configurations

Example: FSM benchmark
4 modules, 16 configurations
- A. automata.rkt Interface & basic strategies
- M. main.rkt Runs a simulation
- P. population.rkt Models groups of automata
- U. utilities.rkt Helper functions

Untyped runtime: 182ms

Is this "good" performance?

Yes
- Fully typed is 2x faster
- 50% of all configurations have < 3x overhead
- Can avoid > 2,000x overhead by typing both main.rkt and population.rkt

No
- Maximum overhead: 8,500x (26 minutes to run)
- Average overhead: 2,700x
- Median overhead: 470x
- No smooth migration paths: Impossible to convert module-by-module and avoid 2,000x overhead

Open Question
How to help developers avoid performance valleys (without exploring the whole lattice)?

L-N/M FIGURES

Summarizing performance lattices

Legend:
- #Configs < Overhead
- 60% ofConfigs

FSM economy simulator 4 modules 182ms untyped 85ms untyped

GREGOR time & date library 13 modules 666ms untyped 815ms typed

SYNTH music maker 10 modules 263ms untyped 272ms typed

Each call to step wraps pop with a higher-order contract After N calls, each vector operation suffers N indirections

A. automata.rkt
require "automata.rkt"

M. main.rkt
require "automata.rkt"
require "population.rkt"
require "utilities.rkt"

define (evolve pop count) if (zero? count) null evolve (step pop) (count - 1) evolve (create 100) 5

P. population.rkt
require/typed "automata.rkt"
require/typed "population.rkt"
require/typed "utilities.rkt"

define-type Population (Vectorof (Vectorof Automaton))

provide:
  step (-> Population Population)
  create (-> Natural Population)