

Qualifying Exam

Ambrose Bonnaire-Sergeant (0003410123)

January 5, 2018

Question 1

Analyze the space and time complexity of your approach to dynamic tracing & subsequent type inference for Typed Clojure. Are you able to bound space use at all by reducing traces as they are collected? Please analyze a related system, Daikon (<https://plse.cs.washington.edu/daikon/>), along the same lines. How expressive are Daikons invariants compared to yours? How much space and runtime overhead does it impose? If Daikon's inferred invariants were to become part of a (refinement) type system, how powerful would it need to be?

Space/time complexity of dynamic tracing

Space/time complexity of type inference

Can space use be bounded by reducing traces as collected?

Daikon's expressivity

Space/time overhead of Daikon

How to type check Daikon's invariants

Question 2

Examine the use of Clojure's `core.spec` contract system in several real world code bases. Look at what features are used, and how precise specifications are. Analyze how specifications address the lack of higher-order contracts by looking at the frequency of higher-order function contracts vs higher-order functions that omit specifications of higher-order arguments or results.

$d, e ::= x \mid v \mid (e \ e) \mid \lambda x^\tau. e \mid (\text{if } e \ e \ e) \mid (\text{do } e \ e)$	Expressions
$v ::= l \mid \{\} \mid n \mid m \mid [\rho, \lambda x^\tau. e]_c$	Values
$m ::= \{\overline{v \mapsto \vec{v}}\}$	Map Values
$c ::= \text{class} \mid n?$	Constants
$G ::= (\text{get } e \ e) \mid (\text{assoc } e \ e \ e)$	

Figure 1: Syntax of Terms, Types, Propositions, and Objects

What spec features are used in real systems

How precise are spec annotations in practice?

How frequently are higher-order functions annotated with higher-order specs? Why?

Question 3

Write a formal model of Clojure with `core.spec`, and implement it in PLT Redex. Formulate a consistency property between contracted and uncontracted execution, and test it in redex.