Types & Effects: Overview

Notes based on lectures for CSC 2126H
(Topics in PL: Types and Effects)
at the University of Toronto by Professor Ningning Xie, Fall 2024

Yanning Chen

Course given: **September 09, 2024**
Last updated: **September 11, 2024**

Contents

"The goal of types and effects is **to reason about program properties**."

1. Introduction

This lecture is intended as an general overview of all topics covered in the course.

We'll cover six topics in this course, and generally, they can be classified as follows:

Statics Substructural Types, DT
Modularity Module Systems (OCaml, SML)
Efficiency Modal Types (so there exists specific ways to ask compiler to generate efficient code)
Effects Effect Types (tyck, also include monads), Effect Handlers (algbraic properties, cool!)

2. Questions

There are three basic questions to ask when talking about types and effects:

What is a *program*? syntax, typing, computation
What is its *property*? soundness (progress & preservation), normalization, equivalence, decidability (type checking)
How *decidable*? type inference (e.g. System F is undecidable), type equivalence (e.g. generally undecidable for DT)

In addition, we will also discuss additional questions for each topic.

3. Substructural Types

Motivation: Resource management (e.g. files, memory)

Program where a resource is used in a restricted way (ordering/use count)
Property eliminate invalid states

3.1. Categories

E Exchange **W** Weakening **C** Contraction

Ordered ?
Linear (E) must be used exactly once
Affine (E W) must be used at most once
Relevant (E C)
Normal (E W C) can be used arbitrarily

And, we will try to answer the following questions:
1. How to enforce restrictions with typing?
2. How to design dynamics?
3. Any other useful senario? (ref. ATAPL)

4. Dependent Types

Motivation: Expressive types (e.g. length-indexed Vec)

Program terms in types (`Vec 3 Int`)
Decidability type equivalence is generally undecidable

And, we will try to answer the following questions:
1. How to write programs in DT? (consider C.H., e.g. Rocq, Agda, …)
2. How to compile DT programs? (erasure v.s. proof-preserving transformations?) (*this is what we mainly focus in this course, and papers on this topic could be hard to read. Use caution!*)

5. Module Systems

Motivation: Modularity (e.g. assemble well-specified components, recall ML modules)

Program a modular system assembled from separate components
Property modularity, data abstraction (not allow users to access exact types of a module, and only allow them to access the methods)

And, we will try to answer the following questions:
1. How to design its *core language*?
2. Higher-order modules? (e.g. Module functors in OCaml) First-class modules? (*This is where abstract types could go wrong*)
3. Problems of complex module systems? (*LightQuantum: like what?*)

6. Modal Types (& Code Gen)

Motivation: Tell compilers how to specialize programs (e.g. *staging*)

(*So, we focus more on code generation in this course*)

Program annotated with modal types that indicate *stages*
Property codegen properties (e.g. well-typedness: $\forall c : \text{Program}, \text{well-typed } c \to \text{well-typed } \texttt{translate}\,(c)$)

And, we will try to answer the following questions:
1. Logic foundation? (*temporal logic*/contextual modal logic, *we focus on temporal logic in this course*)
2. Properties? (equivalence between original and generated code)
3. Practical applications? (e.g. efficient compilers by futamura projections)

7. Effect Types

Motivation: Reason about effects (e.g. IO, exceptions) (hsk: monads)

Program track computational effects (e.g. monads)
Property safety (capture effects and ordering)

And, we will try to answer the following questions:

1. What are the different effect systems?
2. How to reason about effects?
3. Applications? (e.g. optimization by leveraging pureness / no entanglement property?)

8. Effect Handlers

Motivation: effectful programs with algebraic properties (e.g. Koka/OCaml)

Program ditto
Property ditto

And, we will try to answer the following questions:

1. What is the *algebraic* part of algeff handlers?
2. monads vs algeffs?
3. How to design a practical language with effect handlers? (e.g. Koka)