CS520 Theory of Programming Languages

Introduction

Hongseok Yang
KAIST

How to analyse programming languages (their constructs, type systems, implementations, etc) formally?

We will study mathematical tools for doing such analysis.

Preview 1: Abstract syntax

- What is a program? What kind of syntactic object is it?

Preview 1:
Abstract syntax

- What is a program? What kind of syntactic object is it?
- Bad answer: a sequence of characters.

Preview 1: Abstract syntax

- What is a program? What kind of syntactic object is it?
- Bad answer: a sequence of characters.
- Our answer: an instance of an abstract syntax.
- Mathematically, an element of an initial algebra.

Preview 2:
Domain theory

```
>>> def F(g): return g
...
```

- Which mathematical object does the program F denote?

Preview 2: Domain theory

```
>>> def F(g): return g
...
```

- Which mathematical object does the program F denote?
- Identity function in [D→D] for some D.

Preview 2: Domain theory

```
>>> def F(g): return g
...
>>> F(F)
<function F at 0x10c573410>
```

- Which mathematical object does the program F denote?
- Identity function in [D→D] for some D.
- But D should include [D→D]. Impossible if D is a set.

Preview 2: Domain theory

```
>>> def F(g): return g
...
>>> F(F)
<function F at 0x10c573410>
```

- Which mathematical object does the program F denote?
- Identity function in [D→D] for some D.
- But D should include [D→D]. Impossible if D is a set.
- Possible if D is a domain & [D→D] has only continuous fns.

Preview 3: Evaluation order

```
>>> def f(x): return (x+x)
...
>>> f(f(3))
12
```

- Should we compute f(3) before applying f to f(3)?

Preview 3: Evaluation order

```
>>> def f(x): return (x+x)
...
>>> f(f(3))
12
```

- Should we compute f(3) before applying f to f(3)?
- Yes. Eager evaluation. Python, OCaml, Scheme, etc.
- No. Normal-order evaluation or lazy evaluation. Haskell.

Preview 3: Evaluation order

```
>>> def f(x): return 4
...
>>> f(f(3))
4
```

- Should we compute f(3) before applying f to f(3)?
- Yes. Eager evaluation. Python, OCaml, Scheme, etc.
- No. Normal-order evaluation or lazy evaluation. Haskell.

Preview 3: Evaluation order

```
>>> def f(x): return 4
...
>>> f(f(3))
4
```

- Should we compute f(3) before applying f to f(3)?
- Yes. Eager evaluation. Python, OCaml, Scheme, etc.
- No. Normal-order evaluation or lazy evaluation. Haskell.
- To be analysed via operational and denotational semantics.

Related topic 4: Type system

```
import typing
from typing import Callable

def twice(f : Callable[[int],int], x : int) -> int:
  return(f(f(x)))
```

- Types help develop correct programs.

Related topic 4: Type system

```
import typing
from typing import Callable

def twice(f : Callable[[int],int], x : int) -> int:
  return(f(f(x)))
```

- Types help develop correct programs.
- Can we infer types automatically?
- What mathematical objects do types denote?

Related topic 4: Type system

```
import typing
from typing import Callable

def twice(f : Callable[[int],int], x : int) -> int:
  return(f(f(x)))
```

- Types help develop correct programs.
- Can we infer types automatically? Type inference algo.
- What mathematical objects do types denote? Partial equivalence relation.

- Predicate Logic (Ch1).
- The Simple Imperative Language (Ch2).
- Program Specification and Their Proofs (Ch3).
- Failure, Input-Output, and Continuation (Ch5).
- Transition Semantics (Ch6).
- An Introduction to Category Theory (Tennent Ch8).
- Recursively-Defined Domains (Tennent Ch10).
- The Lambda Calculus (Ch10).
- An Eager Functional Language (Ch11).
- Continuation in a Functional Language (Ch12).

Imperative Languages

- Predicate Logic (Ch1).
- The Simple Imperative Language (Ch2).
- Program Specification and Their Proofs (Ch3).
- Failure, Input-Output, and Continuation (Ch5).
- Transition Semantics (Ch6).
- An Introduction to Category Theory (Tennent Ch8).
- Recursively-Defined Domains (Tennent Ch10).
- The Lambda Calculus (Ch10).
- An Eager Functional Language (Ch11).
- Continuation in a Functional Language (Ch12).

Imperative Languages

- Predicate Logic (Ch1).
- The Simple Imperative Language (Ch2).
- Program Specification and Their Proofs (Ch3).
- Failure, Input-Output, and Continuation (Ch5).
- Transition Semantics (Ch6).
- An Introduction to Category Theory (Tennent Ch8).
- Recursively-Defined Domains (Tennent Ch10).
- The Lambda Calculus (Ch10).
- An Eager Functional Language (Ch11).
- Continuation in a Functional Language (Ch12).

Functional Languages

Imperative Languages

- Predicate Logic (Ch1).
- The Simple Imperative Language (Ch2).
- Program Specification and Their Proofs (Ch3).
- Failure, Input-Output, and Continuation (Ch5).
- Transition Semantics (Ch6).
- An Introduction to Category Theory (Tennent Ch8). **Math tools**
- Recursively-Defined Domains (Tennent Ch10).
- The Lambda Calculus (Ch10).
- An Eager Functional Language (Ch11).
- Continuation in a Functional Language (Ch12).

Functional Languages

Course webpage

https://github.com/hongseok-yang/graduatePL24

- Primary source of information about the course.

Evaluation

- Final exam — 40%.
- Group project — 40%.
- Homework assignments (3 or 4 problem sheets) — 20%.

Evaluation

- Final exam — 40%.
- Group project — 40%.
- Homework assignments (3 or 4 problem sheets) — 20%.

Final exam

- During the final-exam period.
- Precisely, 13:00 - 14:30 on 19 Dec, at 1101 in E3.
- Closed book.
- All topics are likely to be covered.

Evaluation

- Final exam — 40%.
- Group project — 40%.
- Homework assignments (3 or 4 problem sheets) — 20%.

Group project

- 2-4 students form a group, and study a topic of a paper or papers published in POPL/PLDI 20-24, in depth.
- By 26 Sep (Thu). Form a group, and inform TAs and me.
- By 17 Oct (Thu). Submit a 1-page project proposal.
- By 21 Nov (Thu). Submit a report (at most 4 pages).
- By 28 Nov (Thu). Submit the slides of a 35-min talk.
- Four chosen groups will present their works on 10/12 Dec.

Honour code

- We adopt a strict policy for handling plagiarism and dishonest academic behaviours.
- A student will get F if
  - she or he is found to copy texts from papers and books without paraphrasing them properly; or
  - he or she is found to cheat in a test or copy answers or code from friends' or other sources.

Large language models

- You may use LLMs for your group project and your study in general, as long as you keep the following rules.

1. Don’t ask homework questions directly to an LLM and copy its answers. Doing so will be regarded as cheating.

2. Describe how you or your group gets help from LLMs in a submission, if you or your group does so.

3. LLMs may lie. Ensuring factual correctness is your responsibility.

Teaching staffs

- Prof Hongseok Yang (Lecturer). hongseok00@gmail.com. Office hour: 6pm-7pm on Monday at 3403 in E3-1.
- Mr Gyeongwon Jung (TA1). jgyw0910@kaist.ac.kr
- Mr Sangho Lim (TA2). lim.sang@kaist.ac.kr
- TAs’ office hours will be announced shortly.