Check-In Two Summary
CS 364 — Spring 2021

Overview

Student Check-Ins are brief individual meetings for you to ask clarifying questions about the material we've covered in the course and for me to check that you are understanding key ideas. These meetings will offer you a structured way to reflect on the material you've retained as well as content that might still be a bit confusing.

In addition to the individual meetings, you will also turn in a written summary of the topics covered in class. I will provide you with a set of guiding questions for these summaries. The purpose of the written summary is to help you review material and identify aspects of the class that are challenging and/or confusing.

Individual Meetings (25 points)

Individual student check-in meetings will take place **Thursday April 1**. Class will still take place on March 30. Student hours will still be available at their standard times.

Use the sign-up tool on Sakai to sign up for **one twenty minute time slot**. All individual meetings will be hosted using the **Student Hours meeting found on Sakai**. Please be prompt when joining the Zoom meeting at your specified time. There is a waiting room, so you may join early.

Please contact me ASAP if you are unable to schedule a meeting in the available times.

Failure to sign up for a meeting and/or show up on time for your meeting slot may result in a reduced grade.

Written Summary (75 points)

Due: Wednesday March 31 at 11:59 PM Eastern

Submit your written summary to Gradescope by the deadline specified above. To give me time to review your answers prior to the check-in meetings, this is a strict deadline. Please plan your time accordingly. You may either type up your answers or hand-write your answers and scan/take pictures to submit your summary. Space is provided in this outline to record your answers, should you wish to print out a copy of this document.

Collaboration and Resource Policy

You **may** use your notes, class videos, lab assignments, and assigned readings to help you with this summary. **You may not use any other resources**. This summary should be completed **individually**. Please review the academic integrity and professionalism policy in the course syllabus for additional details.

1 Type Checking and Semantic Analysis (20 points)

1.1 Definitions and Background

1. Define the following terms and give examples where appropriate.

   (a) Semantic Analysis:

   (b) Variable Declaration:

(c) Variable Definition:

(d) Scoping Rules:

(e) Symbol Table:

(f) Type System:

(g) Type Checking:

(h) Type Inference:

(i) Soundness:

(j) Type Environment:

(k) Dynamic Type:

(l) Static Type:

2. Describe some key differences between statically- and dynamically-typed languages. What are some advantages of each approach?

1.2 Type Inference

1. Draw the AST for the following Cool snippet, and annotate each node with its associated type (following the Cool typing rules). You may assume that class E has been defined and that it has a method `set_var`, which has a single formal parameter of type `Int`. If the code snippet does not type check, indicate the conflicting types on your AST.

```
let num : Int <- in_int() in
let x : Int <- num + 2 in
     (new E).set_var(x)
```

2. Draw the AST for the following Cool snippet and annotate each node with its associated type (following the Cool typing rules). If the code snippet does not type check, indicate the conflicting types on your AST.

```
let num : Int <- in_int() in
let val : Int <- if num < 3 then num * 2 else false fi in
    num + val
```

2 Operational Semantics (20 points)

2.1 Definitions and Background

1. Define the following terms and give examples where appropriate.

   (a) Environment:

   (b) Store:

(c) Call-by-value:

(d) Call-by-reference:

2. Briefly describe the purpose of operational semantics.

3. What are the constituent parts of the context in a Cool operation semantics rule? Why is each portion of the context necessary?

4. How are side-effects modeled by operational semantics?

5. How is evaluation order enforced by the Cool operational semantics?

2.2 Derivations and Rules

1. Consider these six operational semantics rules:

$$(1)\ \frac{so, E, S \vdash e_1 : Bool(false), S_1}{so, E, S \vdash \text{while } e_1 \text{ loop } e_2 \text{ pool } : void, S_1} \qquad (4)\ \frac{E(id) = l_{id} \quad S(l_{id}) = v}{so, E, S \vdash id : v, S}$$

$$(2)\ \frac{so, E, S \vdash e_1 : Bool(true), S_1 \quad so, E, S_1 \vdash e_2 : v, S_2 \quad so, E, S_2 \vdash \text{while } e_1 \text{ loop } e_2 \text{ pool } : void, S_3}{so, E, S \vdash \text{while } e_1 \text{ loop } e_2 \text{ pool } : void, S_3} \qquad (5)\ \frac{so, E, S \vdash e : v, S_1 \quad E(id) = l_{id} \quad S_2 = S_1[v/l_{id}]}{so, E, S \vdash id \leftarrow e : v, S_2}$$

$$(3)\ \frac{so, E, S \vdash e_1 : v_1, S_1 \quad l_{new} = newloc(S_1) \quad so, E[l_{new}/id], S_1[v_1/l_{new}] \vdash e_2 : v_2, S_2}{so, E, S \vdash \text{let } id : T \leftarrow e_1 \text{ in } e_2 : v_2, S_2}$$

$$(6)\ \frac{so, E, S \vdash e_1 : Int(n_1), S_1 \quad so, E, S_1 \vdash e_2 : Int(n_2), S_2 \quad v = \begin{cases} Bool(true) & \text{if } n_1 < n_2 \\ Bool(false) & \text{if } n_1 \geq n_2 \end{cases}}{so, E, S \vdash e_1 < e_2 : v, S_2}$$

Use these rules to construct a derivation for the following piece of code:

```
let x : Int <- 2 in
while 1 < x loop
 x <- x - 1
pool
```

You may assume reasonable axioms, e.g. it is always true that $so, E, S \vdash 2 - 1 : Int(1), S$. Start your derivation using the `let` rule (3) as follows:

$$\frac{so, E, S \vdash 2 : Int(2), S \qquad \frac{\cdots}{so, E[l_{new}/x], S[Int(2)/l_{new}] \vdash \text{while } 1 < x \text{ loop } x \leftarrow x - 1 \text{ pool } : void, S_{final}}(2)}{so, E, S \vdash \text{let } x : Int \leftarrow 2 \text{ in while } 1 < x \text{ loop } x \leftarrow x - 1 \text{ pool } : void, S_{final}}(3)$$

Note that you only need to expand hypotheses that need to be proved (i.e. those containing $\vdash$).

2. The operational semantics for Cool's `while` expression show that result of evaluating such an expression is always `void`.

   However, we could have used the following alternative semantics:

   - If the loop body executes at least once, the result of the `while` expression is the result from the *last* iteration of the loop body.
   - If the loop body never executes (i.e., the condition is false the first time it is evaluated), then the result of the `while` expression is `void`.

   For example, consider the following expression:

   ```
   while (x < 10) loop x <- x+1 pool
   ```

   The result of this expression would be 10 if, initially, `x` < 10 or `void` if `x` ≥ 10.

   Write new operational rules for the `while` construct that formalize these alternative semantics. HINT: You will need two (2) rules.

3 Runtime Organization and Code Generation (15 points)

3.1 Definitions

Define the following terms and give examples where appropriate.

1. Compiler:

2. Stack Machine:

3. Register:

4. Code Generation:

5. Activation Record (Stack Frame):

6. Liskov Substitution Principle:

7. Dispatch Table (vtable):

3.2 Code Generation Routines

For these questions, consider the simple language grammar and code generation routines we discussed in class:

$$
\begin{array}{rcl}
P & \rightarrow & D; P \mid D \\
D & \rightarrow & \texttt{def id}(ARGS) = E; \\
ARGS & \rightarrow & \texttt{id}, ARGS \mid \texttt{id} \\
E & \rightarrow & \texttt{int} \mid \texttt{id} \mid \texttt{if}(E_1 = E_2)\,\{E_3\}\,\texttt{else}\,\{E_4\} \\
 & & \mid E_1 + E_2 \mid E_1 - E_2 \mid \texttt{id}(E_1, \ldots, E_n)
\end{array}
$$

Answer **ONE** of the following questions.

1. Suppose we wish to support assignment to variables in our simple language:

$$E \rightarrow \texttt{id} \leftarrow E$$

   Write a code generation routine for this expression. You may write your answer in pseudocode or Reason.

2. Suppose we wish to support looping in our simple language:

$$E \quad \rightarrow \quad \texttt{until}\ E_1 = E_2\ \{E_3\}$$

This code will repeatedly evaluate E_3 until E_1 equals E_2. Write a code generation routine for this expression. You may write your answer in pseudocode or Reason.

3.3 Data Layout

1. Draw a suitable object and vtable layout for the following code (Don't forget inherited fields and methods!).

```
class Point {
    vx : Int <- 0;
    vy : Int <- 0;
    move( x : Int, y : Int) : SELF_TYPE {
        {   vx <- x + vx; vy <- y + vy; self;   }
    };
};
class ColorPoint inherits Point {
    r : Int <- 0;
    g : Int <- 0;
    b : Int <- 0;
    changeColor( new_r : Int, new_g : Int, new_b : Int) : SELF_TYPE {
        {   r <- new_r; g <- new_g; b <- new_b; self;   }
    };
};
class Main inherits IO {
    main() : Object {
        let x : ColorPoint <- new ColorPoint in
            { x <- x.changeColor( 25, 25, 25 ); out_string("Hello, world!\n"); }
    };
};
```

4 Special Topics (15 points)

4.1 Debugging

1. Explain the steps needed to set a breakpoint on an expression in the source code of a *high-level* programming language.

2. Describe one advantage and one disadvantage of using a debugger rather than '`printf`' debugging.

4.2 Multi-Language Projects

1. Describe at least two (2) approaches for implementing a program using multiple programming languages.

2. Compare and contrast how Java's JNI and Python's `ctypes` module handle translating (*marshaling*) data and types to/from native C code.

5 General (5 points)

1. Make an impassioned argument for either static type systems or dynamic type systems.

2. What is an implicit bias?

3. Was there anything you were hoping to learn in this class that wasn't covered? Did you learn anything you weren't expecting to learn?

4. What is one piece of advice you wish you were told at the beginning of CS-364?