The University of New South Wales

COMP3161/9161
Concepts of Programming Languages

SAMPLE

Midsession Exam

Semester 2, 2018

Time Allowed: 50 Minutes

Total Number of Questions: 6

Answer **all** questions.

The questions are **not** of equal value.

You are permitted **one** hand-written, single-sided A4 sheet of notes.

Only write your answers on the provided booklets.

Answers must be written in ink, with the exception of diagrams.

Drawing instruments or rules may be used.

Excessively verbose answers may lose marks.

There is a 3% penalty if your name and student number are not filled in correctly.

Question 1 (4 marks)

Given the following expression e, written in the arithmetic expression language we defined in the lectures:

```
let x = 5 in
  let x = 10 + x in
    let y = 5 + x in
      x + let x = 15 in x × y end
    end
  end
end
```

Find a term that is α equivalent where each variable name is used only in one unique binding position (i.e. each variable name only occurs once on the left hand side of a `let`-binding).

Solution: Many solutions are possible. Using a fresh variable name for every variable in the expression is a simple way to arrive at a solution:

```
let a = 5 in
  let b = 10 + a in
    let c = 5 + b in
      b + let d = 15 in d × c end
    end
  end
end
```

Question 2 (6 marks)

In the lecture we discussed the role of the lexer, the parser, and the (static) semantic analyser. For each of these components, give an example of a program error that can be detected by that component. (The error may be in terms of C, Haskell or the language of arithmetic expressions with `let`-bindings discussed in the lecture).

Solution:

Lexer decomposes the programm string into a sequence of tokens. Possible errors it can detect: if a substring is neither a keyword, identifier, nor a constant value of any type. For example, in C, `4rewrcs` would cause an error, as would an invalid symbol like `@`.

Parser analyses the grammatical structure of a program. Possible errors it can detect: if expressions or definitions are not well-formed, e.g. in Haskell, an if-expression without a `then`-branch, mismatching parentheses in any (sane) language.

Semantic analysis checks static semantic properties, for example, if all variables are in scope, no type errors occur (e.g. and `Int` is used where a `String` is expected).

Question 3 (6 marks)

What is the difference between concrete and abstract syntax of a programming language? (keep your answer brief — it may be easiest to describe the difference if you use the possible concrete and abstract syntax of an actual language construct as an example)

Solution: The concrete syntax of a language is a string of symbols, which follows a grammar designed with the human user in mind (at least it should be!), so it has operations in infix notation, symbols which only serve to denote the end of terms like parentheses or semicolons, and may include things like indentation in languages like Python and Haskell.

Abstract syntax are terms in a tree structure designed to store only the information which is necessary for the rest of the compiliation process.

For example, in arithmetic expressions, we have brackets in the concrete syntax to avoid any ambiguity, and allow infix notation, as it is more convenient to use, e.g, $2 \times (3 + 4)$. The abstract syntax for this expression is simply a nested term:

$$\texttt{Times}\ (\texttt{Num}\ 2)\ (\texttt{Plus}\ (\texttt{Num}\ 3)\ (\texttt{Num}\ 4)))$$

Question 4 (4 marks)

What is the β-normal form of this λ-term? What about the $\beta\eta$-normal form? Write down the chain of reductions.

$$(\lambda n.\ \lambda f.\ \lambda x.\ f\ (n\ x\ x))\ (\lambda f.\ \lambda x.\ x)$$

Solution: The β-normal form of the term is $\lambda f.\ \lambda x.\ f\ x$.

$$\begin{array}{lll} (\lambda n.\ \lambda f.\ \lambda x.\ f\ (n\ x\ x))\ (\lambda f.\ \lambda x.\ x) & \mapsto_\beta & (\lambda f.\ \lambda x.\ f\ ((\lambda f.\ \lambda x.\ x)\ x\ x)) \\ & \mapsto_\beta & (\lambda f.\ \lambda x.\ f\ ((\lambda x.\ x)\ x)) \\ & \mapsto_\beta & (\lambda f.\ \lambda x.\ f\ x) \end{array}$$

The $\beta\eta$-normal form is $\lambda f.\ f$, applying η-conversion to the final step of the β-reduction chain above.

Question 5 (7 marks)

Given the following big-step semantics for arithmetic expressions:

$$\frac{}{(\texttt{Num}\ n) \Downarrow n}(1) \qquad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2}{(\texttt{Plus}\ e_1\ e_2) \Downarrow (v_1 + v_2)}(2) \qquad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2}{(\texttt{Times}\ e_1\ e_2) \Downarrow (v_1 \times v_2)}(3)$$

$$\frac{e_1 \Downarrow v_1 \qquad e_2[x := (\texttt{Num}\ v_1)] \Downarrow v_2}{(\texttt{Let}\ e_1\ (x.\ e_2)) \Downarrow v_2}(4)$$

Draw a derivation tree to describe the evaluation of the term:

$$(\texttt{Plus}\ (\texttt{Num}\ 5)\ (\texttt{Let}\ (\texttt{Num}\ 7)\ (x.\ (\texttt{Plus}\ x\ (\texttt{Num}\ 1)))))$$

Solution:

$$\dfrac{\dfrac{}{(\texttt{Num}\ 5) \Downarrow 5}(1) \qquad \dfrac{\dfrac{}{(\texttt{Num}\ 7) \Downarrow 7}(1) \qquad \dfrac{\dfrac{}{(\texttt{Num}\ 7) \Downarrow 7}(1) \qquad \dfrac{}{(\texttt{Num}\ 1) \Downarrow 1}(1)}{(\texttt{Plus}\ (\texttt{Num}\ 7)\ (\texttt{Num}\ 1)) \Downarrow 8}(2)}{(\texttt{Let}\ (\texttt{Num}\ 7)\ (x.\ (\texttt{Plus}\ x\ (\texttt{Num}\ 1)))) \Downarrow 8}(4)}{(\texttt{Plus}\ (\texttt{Num}\ 5)\ (\texttt{Let}\ (\texttt{Num}\ 7)\ (x.\ (\texttt{Plus}\ x\ (\texttt{Num}\ 1))))) \Downarrow 13}(2)$$

The text in red denotes that it is the result of a substitution.

Question 6 (13 marks)

Here are some rules to inductively define the judgements **Nat** and **Unnat**:

$$\frac{}{0\ \mathbf{Nat}}N_1 \quad \frac{n\ \mathbf{Nat}}{(\mathsf{S}\ n)\ \mathbf{Nat}}N_2$$

$$\frac{}{0\ \mathbf{Unnat}}U_1 \quad \frac{}{(\mathsf{S}\ 0)\ \mathbf{Unnat}}U_2 \quad \frac{n\ \mathbf{Unnat}}{(\mathsf{S}\ (\mathsf{S}\ n)\ \mathbf{Unnat}}U_3$$

Using rule induction, show that both definitions are equivalent, that is, for all x, x **Nat** is derivable if and only if x **Unnat** is derivable. Clearly state which cases you have to consider, and what the induction hypothesis is for each case. Annotate derivations with the name of the rule you used. One direction should be comparatively straightforward. For the other direction, you may find it helpful to prove a lemma.

Solution: We must prove two theorems:

$$\frac{x\ \mathbf{Nat}}{x\ \mathbf{Unnat}} \quad \text{and} \quad \frac{x\ \mathbf{Unnat}}{x\ \mathbf{Nat}}$$

Proceeding by rule induction on x **Nat** to prove the first theorem.

Base Case: Here $x = 0$ from rule N_1. We can show 0 **Unnat** via rule U_1.

Inductive Case: Here $x = (\mathsf{S}\ n)$ from rule N_2. We must show $(\mathsf{S}\ n)$ **Unnat** assuming the inductive hypothesis n **Unnat**. We will now prove a lemma:

$$\frac{n\ \mathbf{Unnat}}{(\mathsf{S}\ n)\ \mathbf{Unnat}}\text{Lemma}$$

That, together with our inductive hypothesis, proves our goal.

Now, to prove this lemma, we will do another rule induction on n **Unnat**, and show that $(\mathsf{S}\ n)$ **Unnat** in each case.

Base Case: Here $n = 0$ from rule U_1. We can show $(\mathsf{S}\ 0)$ **Unnat** via rule U_2.

Base Case: Here $n = (\mathsf{S}\ 0)$ from rule U_2. We can show $(\mathsf{S}\ (\mathsf{S}\ 0))$ **Unnat** as follows:

$$\frac{\dfrac{}{0\ \mathbf{Unnat}}U_1}{(\mathsf{S}\ (\mathsf{S}\ 0))\ \mathbf{Unnat}}U_3$$

Inductive Case: Here $n = (\mathsf{S}\ (\mathsf{S}\ k))$ where k **Unnat**, from rule U_3. We get the inductive hypothesis that $(\mathsf{S}\ k)$ **Unnat**. We can show our goal as follows:

$$\frac{\dfrac{}{(\mathsf{S}\ k)\ \mathbf{Unnat}}\text{I.H}}{(\mathsf{S}\ (\mathsf{S}\ (\mathsf{S}\ k)))\ \mathbf{Unnat}}U_3$$

Thus we have completed our lemma proof and therefore the proof of the first direction of the equivalence.

Now, to prove the other direction, we will do rule induction on x **Unnat** and show that x **Nat**.

Base Case: Here $n = 0$ from rule U_1. We can show 0**Nat** via rule N_1.

Base Case: Here $n = (\mathsf{S}\ 0)$ from rule U_2. We can show $(\mathsf{S}\ 0)$ **Nat** using rule N_2 then N_1:

$$\dfrac{\dfrac{}{0\ \mathbf{Nat}}N_1}{(\mathsf{S}\ 0)\ \mathbf{Nat}}N_2$$

Inductive Case: Here $n = (\mathsf{S}\ (\mathsf{S}\ k))$ where k **Unnat**, from rule U_3. We get the inductive hypothesis that k **Nat**. We can show our goal as follows:

$$\dfrac{\dfrac{\dfrac{}{k\ \mathbf{Nat}}\text{I.H}}{(\mathsf{S}\ k)\ \mathbf{Nat}}N_2}{(\mathsf{S}\ (\mathsf{S}\ k))\ \mathbf{Nat}}N_2$$

Thus the two languages are equivalent.