Models of Computation: Assessed Coursework I

This is the first assessed coursework for Models of Computation. You should submit your answers to these questions in hard copy by 12 noon on Friday, the 18th of November, 2016. This work may be submitted individually or completed in pairs. If you opt to work as a pair, please set up appropriate groups in CaTE in the usual way.

While with Dynamically Allocated Memory

The While language that we have considered so far in this course lacks a number of features that are common to most programming languages. For this coursework, we consider an extension of While that supports dynamically allocated memory.

The language WhileDM extends While by allowing pairs of values to be dynamically created and manipulated. A pair is analogous to an object in Java with two fields, `fst` and `snd`. Pairs are constructed by the `newpair` expression, which returns the address of a freshly allocated pair. If E is an expression that evaluates to the address of a pair, E.`fst` returns the contents of its `fst` field and E.`snd` returns the contents of its `snd` field. The command $\texttt{fst}[E] \leftarrow E'$ stores the value of E' in the `fst` field of the pair addressed by E; similarly, the command $\texttt{snd}[E] \leftarrow E'$ stores the value of E' in the `snd` field of E.

The syntax of the language WhileDM is defined by:

$$\begin{aligned}
E \in \mathit{Expr} &::= x \mid n \mid E + E \mid \texttt{newpair} \mid E.\texttt{fst} \mid E.\texttt{snd} \\
B \in \mathit{Bool} &::= \texttt{true} \mid \texttt{false} \mid E = E \mid E < E \mid B \,\&\, B \mid \neg B \\
C \in \mathit{Comm} &::= \texttt{skip} \mid x := E \mid C; C \mid \texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ C \\
&\ \texttt{while}\ B\ \texttt{do}\ C \mid \texttt{fst}[E] \leftarrow E \mid \texttt{snd}[E] \leftarrow E
\end{aligned}$$

where

- $x \in \mathit{Var}$ ranges over the set of program variables, and
- $n \in \mathbb{N}$ ranges over natural numbers.

Expressions in WhileDM can evaluate either to numbers or to addresses of pairs. To represent the results obtained from evaluating an expression, which include values that may be stored in variables, we define the set of WhileDM *values*, denoted *Val*, as follows:

$$v \in \mathit{Val} ::= n \;\Big|\; \ulcorner a \urcorner$$

where $a \in \mathit{Addr}$ ranges over addresses and $\ulcorner a \urcorner$ denotes the literal address corresponding to a. The set of addresses is defined to be the set of positive natural numbers: $\mathit{Addr} = \mathbb{N}^+ = \{1, 2, 3, \dots\}$. A value is, by definition, either a number n or a literal address $\ulcorner a \urcorner$. In particular 7 and $\ulcorner 7 \urcorner$ denote *different* values, the first being a number and the second a literal address.

In While, the program state was simply a partial function from variables to numbers. For WhileDM, the program state is extended in two ways. Firstly, variables can store literal addresses as well as numbers. Secondly, the state must record the values stored in pairs that have been allocated. To this end, *states* of WhileDM are pairs (s, h), where

- $s : \mathit{Var} \rightharpoonup \mathit{Val}$ is a *variable store*, which is a partial function mapping program variables to values, and
- $h : \mathit{Addr} \rightharpoonup_{\mathit{fin}} \mathit{Val}$ is a *heap*, which is a finite partial function mapping addresses to values.

For a partial function $f : A \rightharpoonup B$, we write $\mathrm{dom}(f)$ for the subset of A on which f is defined. We write $\emptyset$ for the partial function which is undefined everywhere.

If a is the address of a pair in the heap h, then we define the `fst` field of the pair to be $h(a)$ and the `snd` field of the pair to be $h(a+1)$. That is, a pair is represented by two consecutive locations in the heap that store the values of its fields. The address of the pair is the address of the first of these locations.

We will define a big-step semantics for WhileDM.

1. (**Expressions**) The big-step semantics for WhileDM expressions is given by the relation $\Downarrow_e \subseteq (\mathit{Expr} \times \mathit{Store} \times \mathit{Heap}) \times (\mathit{Val} \times \mathit{Store} \times \mathit{Heap})$. For an expression E, a value v, variable stores s, s' and heaps h, h',

$$\langle E, s, h \rangle \Downarrow_e \langle v, s', h' \rangle$$

means that E evaluates to v in the state (s, h), resulting in the updated state (s', h').

The rules defining $\Downarrow_e$ are as follows:

(E.NUM) In the state (s, h), a number n evaluates to itself. The state remains unchanged.

$$\textsc{e.num}\ \frac{}{\langle n, s, h \rangle \Downarrow_e \langle n, s, h \rangle}$$

(E.VAR) If, in the state (s, h), the variable x is in the domain of the store s, then the expression x evaluates to the value of x in the store. The state remains unchanged.

$$\textsc{e.var}\ \frac{x \in \mathrm{dom}(s)}{\langle x, s, h \rangle \Downarrow_e \langle s(x), s, h \rangle}$$

(E.ADD) If, in the state (s, h), an expression E_1 evaluates to the number n_1, updating the state to (s', h') and if, in the state (s', h'), an expression E_2 evaluates to the number n_2, updating the state to (s'', h''), then, in the state (s, h), the expression $E_1 + E_2$ evaluates to the sum of n_1 and n_2, updating the state to (s'', h'').

$$\textsc{e.add}\ \frac{\langle E_1, s, h \rangle \Downarrow_e \langle n_1, s', h' \rangle \qquad \langle E_2, s', h' \rangle \Downarrow_e \langle n_2, s'', h'' \rangle}{\langle E_1 + E_2, s, h \rangle \Downarrow_e \langle n_1 \mathbin{\underline{+}} n_2, s'', h'' \rangle}$$

(E.NEW) If, in the state (s, h), both the address a and the address $a+1$ are not in the domain of the heap h, then, in the state (s, h), the expression `newpair` evaluates to the literal address $\ulcorner a \urcorner$, updating the state to $(s, h[a \mapsto 0][a+1 \mapsto 0])$, that is, extending the heap with the pair $(0, 0)$ at address a.

$$\textsc{e.new}\ \frac{a \notin \mathrm{dom}(h) \qquad (a+1) \notin \mathrm{dom}(h)}{\langle \texttt{newpair}, s, h \rangle \Downarrow_e \langle \ulcorner a \urcorner, s, h[a \mapsto 0][a+1 \mapsto 0] \rangle}$$

(E.FST) If, in the state (s, h), an expression E evaluates to the literal address $\ulcorner a \urcorner$, updating the state to (s', h'), and if the address a is in the domain of the heap h', then, in the state (s, h), the expression E.`fst` evaluates to the value that is at address a in the heap h', updating the state to (s', h').

$$\text{E.FST}\ \frac{\langle E, s, h\rangle \Downarrow_e \langle \ulcorner a \urcorner, s', h'\rangle \qquad a \in \mathrm{dom}(h')}{\langle E.\mathtt{fst}, s, h\rangle \Downarrow_e \langle h'(a), s', h'\rangle}$$

(E.SND) If, in the state (s, h), an expression E evaluates to the literal address $\ulcorner a \urcorner$, updating the state to (s', h'), and if the address $a+1$ is in the domain of the heap h', then, in the state (s, h), the expression E.`snd` evaluates to the value that is at address $a+1$ in the heap h', updating the state to (s', h').

$$\text{E.SND}\ \frac{\langle E, s, h\rangle \Downarrow_e \langle \ulcorner a \urcorner, s', h'\rangle \qquad a+1 \in \mathrm{dom}(h')}{\langle E.\mathtt{snd}, s, h\rangle \Downarrow_e \langle h'(a+1), s', h'\rangle}$$

Given these big-step rules for expressions, answer the following questions.

(a) Give a derivation of

$$\langle (\mathtt{newpair.fst}) + (\mathtt{newpair.snd}), \emptyset, \emptyset\rangle \Downarrow_e \langle 0, \emptyset, (1 \mapsto 0, 2 \mapsto 0, 5 \mapsto 0, 6 \mapsto 0)\rangle$$

(b) Give an expression and initial state that can be evaluated to more than one final configuration, and give at least two such final configurations.

(c) Is the semantics of expressions deterministic? Give a proof that it is, or a counterexample showing that it isn't.

(d) Find an expression that can be evaluated in one state but cannot be evaluated in another state. Give that expression, the state in which it can be evaluated, and the state in which it cannot be evaluated.

(e) Give an expression that cannot be evaluated in any state. Briefly justify your answer.

(f) Is the semantics of expressions total? Give a proof that it is, or a counterexample showing that it isn't.

2. (**Boolean Expressions**) Let $BVal = \{\mathtt{true}, \mathtt{false}\}$ be the set of Boolean values. The big-step semantics for WhileDM Boolean expressions is given by the relation

$$\Downarrow_b \subseteq (Bool \times Store \times Heap) \times (BVal \times Store \times Heap).$$

For a boolean expression B, a boolean value b, variable stores s, s' and heaps h, h',

$$\langle B, s, h\rangle \Downarrow_b \langle b, s', h'\rangle$$

means that B evaluates to b in the state (s, h), resulting in the updated state (s', h').

The rules for $=$ and $<$ are as follows:

(B.EQ.TRUE) If, in the state (s, h), an expression E_1 evaluates to the value v, updating the state to (s', h') and if, in the state (s', h'), an expression E_2 evaluates to the same value v, updating the state to (s'', h''), then, in the state (s, h), the expression $E_1 = E_2$ evaluates to `true`, updating the state to (s'', h'').

$$\text{B.EQ.TRUE}\ \frac{\langle E_1, s, h\rangle \Downarrow_e \langle v, s', h'\rangle \qquad \langle E_2, s', h'\rangle \Downarrow_e \langle v, s'', h''\rangle}{\langle E_1 = E_2, s, h\rangle \Downarrow_b \langle \mathtt{true}, s'', h''\rangle}$$

(B.EQ.FALSE) If, in the state (s, h), an expression E_1 evaluates to the value v_1, updating the state to (s', h') and if, in the state (s', h'), an expression E_2 evaluates to the value v_2, updating the state to (s'', h''), and if $v_1 \neq v_2$, then, in the state (s, h), the expression $E_1 = E_2$ evaluates to `false`, updating the state to (s'', h'').

$$\textsc{b.eq.false}\ \frac{\langle E_1, s, h\rangle \Downarrow_e \langle v_1, s', h'\rangle \qquad \langle E_2, s', h'\rangle \Downarrow_e \langle v_2, s'', h''\rangle \qquad v_1 \neq v_2}{\langle E_1 = E_2, s, h\rangle \Downarrow_b \langle \mathtt{false}, s'', h''\rangle}$$

(B.LT.TRUE) If, in the state (s, h), an expression E_1 evaluates to the number n_1, updating the state to (s', h') and if, in the state (s', h'), an expression E_2 evaluates to the number n_2, updating the state to (s'', h''), and if $n_1 < n_2$ in the sense of natural numbers, then, in the state (s, h), the expression $E_1 < E_2$ evaluates to `true`, updating the state to (s'', h'').

$$\textsc{b.lt.true}\ \frac{\langle E_1, s, h\rangle \Downarrow_e \langle n_1, s', h'\rangle \qquad \langle E_2, s', h'\rangle \Downarrow_e \langle n_2, s'', h''\rangle \qquad n_1 < n_2}{\langle E_1 < E_2, s, h\rangle \Downarrow_b \langle \mathtt{true}, s'', h''\rangle}$$

(B.LT.FALSE) If, in the state (s, h), an expression E_1 evaluates to the number n_1, updating the state to (s', h') and if, in the state (s', h'), an expression E_2 evaluates to the number n_2, updating the state to (s'', h''), and if $n_1 \geq n_2$ in the sense of natural numbers, then, in the state (s, h), the expression $E_1 < E_2$ evaluates to `false`, updating the state to (s'', h'').

$$\textsc{b.lt.false}\ \frac{\langle E_1, s, h\rangle \Downarrow_e \langle n_1, s', h'\rangle \qquad \langle E_2, s', h'\rangle \Downarrow_e \langle n_2, s'', h''\rangle \qquad n_1 \geq n_2}{\langle E_1 < E_2, s, h\rangle \Downarrow_b \langle \mathtt{false}, s'', h''\rangle}$$

Given these big-step rules for booleans, answer the following questions.

(a) Complete the big-step semantics of booleans by adding appropriate rules for `true`, `false`, & and ¬. Make sure that your rules:
   - *do not have side conditions* — they should only have premises of the form $- \Downarrow_b -$;
   - *enforce strict evaluation* — both arguments of & should always be evaluated, from left to right.

(b) Find an expression E such that, for some states $(s, h), (s', h')$,

$$\langle E = E, s, h\rangle \Downarrow_b \langle \mathtt{false}, s', h'\rangle$$

   Justify your answer with a derivation.

3. (**Commands**) The big-step semantics for WhileDM commands is given by the relation

$$\Downarrow_c \subseteq (\mathit{Comm} \times \mathit{Store} \times \mathit{Heap}) \times (\mathit{Store} \times \mathit{Heap}).$$

For command C, variable stores s, s' and heaps h, h',

$$\langle C, s, h\rangle \Downarrow_c \langle s', h'\rangle$$

means that executing C in the state (s, h) results in the state (s', h').

The semantic rules for commands are as follows:

(C.SKIP) In the state (s, h), executing the command `skip` results in the state (s, h).

$$\text{C.SKIP}\ \frac{}{\langle \texttt{skip}, s, h\rangle \Downarrow_c \langle s, h\rangle}$$

(C.ASSIGN) If, in the state (s, h), an expression E evaluates to a value v, updating the state to (s', h'), then, in the state (s, h), executing the command $x := E$ results in the state $(s'[x \mapsto v], h')$, i.e. the value of the variable x in the store s' is updated to v.

$$\text{C.ASSIGN}\ \frac{\langle E, s, h\rangle \Downarrow_e \langle v, s', h'\rangle}{\langle x := E, s, h\rangle \Downarrow_c \langle s'[x \mapsto v], h'\rangle}$$

(C.SEQ) If, in the state (s, h), executing the command C_1 results in the state (s', h') and if, in the state (s', h'), executing the command C_2 results in the state (s'', h''), then, in the state (s, h), executing the command $C_1; C_2$ results in the state (s'', h'').

$$\text{C.SEQ}\ \frac{\langle C_1, s, h\rangle \Downarrow_c \langle s', h'\rangle \qquad \langle C_2, s', h'\rangle \Downarrow_c \langle s'', h''\rangle}{\langle C_1; C_2, s, h\rangle \Downarrow_c \langle s'', h''\rangle}$$

(C.IF.TRUE) If, in the state (s, h), the boolean B evaluates to `true`, updating the state to (s', h') and if, in the state (s', h'), executing the command C_1 results in the state (s'', h''), then, in the state (s, h), executing the command `if` B `then` C_1 `else` C_2 results in the state (s'', h'').

$$\text{C.IF.TRUE}\ \frac{\langle B, s, h\rangle \Downarrow_b \langle \texttt{true}, s', h'\rangle \qquad \langle C_1, s', h'\rangle \Downarrow_c \langle s'', h''\rangle}{\langle \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2, s, h\rangle \Downarrow_c \langle s'', h''\rangle}$$

(C.IF.FALSE) If, in the state (s, h), the boolean B evaluates to `false`, updating the state to (s', h') and if, in the state (s', h'), executing the command C_2 results in the state (s'', h''), then, in the state (s, h), executing the command `if` B `then` C_1 `else` C_2 results in the state (s'', h'').

$$\text{C.IF.FALSE}\ \frac{\langle B, s, h\rangle \Downarrow_b \langle \texttt{false}, s', h'\rangle \qquad \langle C_2, s', h'\rangle \Downarrow_c \langle s'', h''\rangle}{\langle \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2, s, h\rangle \Downarrow_c \langle s'', h''\rangle}$$

(C.WHILE.FALSE) If, in the state (s, h), the boolean B evaluates to `false`, updating the state to (s', h'), then, in the state (s, h), executing the command `while` B `do` C results in the state (s', h').

$$\text{C.WHILE.FALSE}\ \frac{\langle B, s, h\rangle \Downarrow_b \langle \texttt{false}, s', h'\rangle}{\langle \texttt{while}\ B\ \texttt{do}\ C, s, h\rangle \Downarrow_c \langle s', h'\rangle}$$

(C.WHILE.TRUE) If, in the state (s, h), the boolean B evaluates to `true`, updating the state to (s', h') and if, in the state (s', h'), executing the command C results in the state (s'', h''), and if, in the state (s'', h''), executing the command `while` B `do` C results in the state (s''', h'''), then, in the state (s, h), executing the command `while` B `do` C results in the state (s''', h''').

$$\text{C.WHILE.TRUE}\ \frac{\langle B, s, h\rangle \Downarrow_b \langle \texttt{true}, s', h'\rangle \qquad \langle C, s', h'\rangle \Downarrow_c \langle s'', h''\rangle \qquad \langle \texttt{while}\ B\ \texttt{do}\ C, s'', h''\rangle \Downarrow_c \langle s''', h'''\rangle}{\langle \texttt{while}\ B\ \texttt{do}\ C, s, h\rangle \Downarrow_c \langle s''', h'''\rangle}$$

(C.STOR.FST) If, in the state (s, h), the expression E_1 evaluates to a literal address $\ulcorner a \urcorner$, updating the state to (s', h'), and if, in the state (s', h'), the expression E_2 evaluates to a value v, updating the state to (s'', h''), and if the address a is in the domain of the heap h'', then, in the state (s, h), executing the command $\texttt{fst}[E_1] \leftarrow E_2$ results in the state $(s'', h''[a \mapsto v])$, i.e. the value at address a in the heap h'' is updated to v.

$$\text{C.STOR.FST}\ \frac{\langle E_1, s, h\rangle \Downarrow_e \langle \ulcorner a \urcorner, s', h'\rangle \qquad \langle E_2, s', h'\rangle \Downarrow_e \langle v, s'', h''\rangle \qquad a \in \mathrm{dom}(h'')}{\langle \texttt{fst}[E_1] \leftarrow E_2, s, h\rangle \Downarrow_c \langle s'', h''[a \mapsto v]\rangle}$$

(C.STOR.SND) If, in the state (s, h), an expression E_1 evaluates to a literal address $\ulcorner a \urcorner$, updating the state to (s', h'), and if, in the state (s', h'), an expression E_2 evaluates to a value v, updating the state to (s'', h''), and if the address $a + 1$ is in the domain of the heap h'', then, in the state (s, h), executing the command $\texttt{snd}[E_1] \leftarrow E_2$ results in the state $(s'', h''[a + 1 \mapsto v])$, i.e. the value at $a + 1$ in the heap h'' is updated to v.

$$\text{C.STOR.SND}\ \frac{\langle E_1, s, h\rangle \Downarrow_e \langle \ulcorner a \urcorner, s', h'\rangle \qquad \langle E_2, s', h'\rangle \Downarrow_e \langle v, s'', h''\rangle \qquad a + 1 \in \mathrm{dom}(h'')}{\langle \texttt{snd}[E_1] \leftarrow E_2, s, h\rangle \Downarrow_c \langle s'', h''[a + 1 \mapsto v]\rangle}$$

Given these big-step rules for commands, answer the following questions.

(a) Consider the configuration $\langle C, s, h\rangle$ given by

$$\begin{aligned} C &\equiv x := c.\texttt{fst}; \texttt{fst}[c] \leftarrow y; c := c.\texttt{snd} \\ s &\equiv (c \mapsto \ulcorner 5 \urcorner, x \mapsto 12, y \mapsto 12) \\ h &\equiv (1 \mapsto 7, 2 \mapsto \ulcorner 5 \urcorner, 3 \mapsto \ulcorner 1 \urcorner, 4 \mapsto 0, 5 \mapsto 3, 6 \mapsto \ulcorner 3 \urcorner) \end{aligned}$$

Find the state (s', h'), such that $\langle C, s, h\rangle \Downarrow_c \langle s', h'\rangle$, and give the full corresponding derivation. You may assume that the sequencing operator ; is right-associative, i.e. that $C_1; C_2; C_3$ is interpreted as $C_1; (C_2; C_3)$.

(b) Consider the configuration $\langle C, s, h\rangle$ given by

```
C  ≡  c := hd;
       while(¬(c = 0)){
           y := x;
           x := c.fst;
           fst[c] ← y;
           c := c.snd
       }
```

$$\begin{aligned} s &\equiv (hd \mapsto \ulcorner 1 \urcorner, x \mapsto 7) \\ h &\equiv (1 \mapsto 12, 2 \mapsto \ulcorner 5 \urcorner, 3 \mapsto \ulcorner 1 \urcorner, 4 \mapsto 0, 5 \mapsto 3, 6 \mapsto \ulcorner 3 \urcorner) \end{aligned}$$

Find the state (s', h'), such that $\langle C, s, h\rangle \Downarrow_c \langle s', h'\rangle$.

4. (**Some Simple Types**) Consider a simple type system for WHILEDM. The types are defined by the following grammar:

$$\tau \in \mathit{Type} ::= \mathsf{nat} \;\Big|\; (\tau, \tau)$$

A value v has type nat if it is a number. A value v has type (τ_1, τ_2) if it is a literal address whose first field is mapped to a value of type τ_1 and whose second field is mapped to a value of type τ_2. To formalise this, we define the relation $h \Vdash v : \tau$, which should be read as "value v has type τ in heap h" with the following rules:

$$\frac{n \in \mathbb{N}}{h \Vdash n : \mathsf{nat}} \qquad \frac{h \Vdash h(a) : \tau_1 \qquad h \Vdash h(a+1) : \tau_2}{h \Vdash \ulcorner a \urcorner : (\tau_1, \tau_2)}$$

In order to type all WHILEDM expressions, we must be somehow able to type variables. This is accomplished with a *typing context* $\Gamma \in \mathit{Var} \rightharpoonup \mathit{Type}$, which is a partial function from variables to their types. We say that a state (s, h) is compatible with a typing context Γ if, for every variable that Γ associates with a type, the value of that variable (interpreted in s and h) actually has the type that Γ said it would have. We write this as $\Gamma; s; h \vdash \mathsf{well\text{-}typed}$, which is formally defined as:

$$\Gamma; s; h \vdash \mathsf{well\text{-}typed} \iff \forall x \in \mathrm{dom}(\Gamma).\, h \Vdash s(x) : \Gamma(x)$$

Implicitly, $\Gamma; s; h \vdash \mathsf{well\text{-}typed}$ requires that every variable in $\mathrm{dom}(\Gamma)$ is also in $\mathrm{dom}(s)$.

Now, we can define the typing judgement for expressions, $\Gamma \vdash E : \tau$, which should be read as "expression E has type τ in context Γ". The rules for this typing judgement are as follows:

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{n \in \mathbb{N}}{\Gamma \vdash n : \mathsf{nat}} \qquad \frac{\Gamma \vdash E_1 : \mathsf{nat} \qquad \Gamma \vdash E_2 : \mathsf{nat}}{\Gamma \vdash E_1 + E_2 : \mathsf{nat}}$$

$$\frac{}{\Gamma \vdash \texttt{newpair} : (\mathsf{nat}, \mathsf{nat})} \qquad \frac{\Gamma \vdash E : (\tau_1, \tau_2)}{\Gamma \vdash E.\texttt{fst} : \tau_1} \qquad \frac{\Gamma \vdash E : (\tau_1, \tau_2)}{\Gamma \vdash E.\texttt{snd} : \tau_2}$$

Given these typing rules, answer the following questions.

(a) Consider the heap

$$h \equiv (1 \mapsto \ulcorner 3 \urcorner, 2 \mapsto \ulcorner 1 \urcorner, 3 \mapsto \ulcorner 5 \urcorner, 4 \mapsto 2, 5 \mapsto 6, 6 \mapsto \ulcorner 7 \urcorner, 7 \mapsto 3, 8 \mapsto 0)$$

i. For what type τ, if any, does $h \Vdash \ulcorner 3 \urcorner : \tau$ hold?
ii. For what type τ, if any, does $h \Vdash \ulcorner 9 \urcorner : \tau$ hold?
iii. For what type τ, if any, does $h \Vdash \ulcorner 1 \urcorner : \tau$ hold?

(b) Consider the state

$$s \equiv (x \mapsto 5, y \mapsto \ulcorner 3 \urcorner, z \mapsto \ulcorner 7 \urcorner)$$
$$h \equiv (1 \mapsto \ulcorner 3 \urcorner, 2 \mapsto \ulcorner 1 \urcorner, 3 \mapsto \ulcorner 5 \urcorner, 4 \mapsto 2, 5 \mapsto 6, 6 \mapsto \ulcorner 7 \urcorner, 7 \mapsto 3, 8 \mapsto 0)$$

For each of the following typing contexts Γ, does $\Gamma; s; h \vdash \mathsf{well\text{-}typed}$?

i. $\Gamma \equiv (x \mapsto \mathsf{nat}, z \mapsto (\mathsf{nat}, \mathsf{nat}))$
ii. $\Gamma \equiv (x \mapsto (\mathsf{nat}, (\mathsf{nat}, \mathsf{nat})), y \mapsto ((\mathsf{nat}, (\mathsf{nat}, \mathsf{nat})), \mathsf{nat}), z \mapsto (\mathsf{nat}, \mathsf{nat}))$
iii. $\Gamma \equiv (y \mapsto ((\mathsf{nat}, (\mathsf{nat}, \mathsf{nat})), \mathsf{nat}), z \mapsto (\mathsf{nat}, \mathsf{nat}))$

(c) The guiding principle of type systems is that "well-typed programs can't go wrong". In particular, we want that every expression E of type τ can always evaluate to an answer of type τ. Prove, by induction on the structure of E, that, for all typing contexts Γ, expressions E, types τ, and states (s, h), if

$$\Gamma; s; h \vdash \mathsf{well\text{-}typed} \quad \text{and} \quad \Gamma \vdash E : \tau$$

then there exist a value v and a state (s', h'), such that

$$\langle E, s, h\rangle \Downarrow_e \langle v, s', h'\rangle, \quad \Gamma; s'; h' \vdash \mathsf{well\text{-}typed}, \quad \text{and} \quad h' \Vdash v : \tau.$$

Attention! This proof **<u>must not</u>** be longer than **three sides of A4** and all cases must be given explicitly. All content beyond the first three sides will be disregarded.

You may use the following lemma without proof:

Lemma 1. *If* $\Gamma; s; h \vdash \mathsf{well\text{-}typed}$ *and* $a, a+1 \notin \mathrm{dom}(h)$*, then* $\Gamma; s; h[a \mapsto 0][a+1 \mapsto 0] \vdash \mathsf{well\text{-}typed}$.

The following question is *not assessed.* Please do not include it in your submission.

5. (**Unassessed: Garbage Collection**) WHILEDM does not have any commands for deallocating memory. In order to efficiently implement such a language, it is generally necessary to have a garbage collector, which periodically deallocates memory cells that can no longer be reached by the program. As well as deallocating memory, the garbage collector is allowed to move already-allocated memory, provided that it updates all references to the memory.

To model the effects of a garbage collector, we will consider a (partial) equivalence on states. We will identify states as equivalent if there is a one-to-one mapping between the reachable portions of the heap. The garbage collector may only change the state to an equivalent state.

Given a relation $R \subseteq \mathit{Addr} \times \mathit{Addr}$ and values v, v', we say that v and v' are *equivalent with respect to* R and write $v \sim_R v'$ if either:

- $v = v' = n$ for some number n, or
- $v = \ulcorner a \urcorner$ and $v' = \ulcorner a' \urcorner$ for some $(a, a') \in R$.

Given a relation $R \subseteq \mathit{Addr} \times \mathit{Addr}$ and states (s, h) and (s', h'), we say the states are *equivalent with respect to* R and write $(s, h) \sim_R (s', h')$ if the following conditions hold:

- R is an invertible mapping, that is:
  - for all $a_1, a_2, a_3 \in \mathit{Addr}$, if $(a_1, a_2) \in R$ and $(a_1, a_3) \in R$ then $a_2 = a_3$, and
  - for all $a_1, a_2, a_3 \in \mathit{Addr}$, if $(a_2, a_1) \in R$ and $(a_3, a_1) \in R$ then $a_2 = a_3$;
- $\mathrm{dom}(s) = \mathrm{dom}(s')$
- for every variable $x \in \mathrm{dom}(s)$, $s(x) \sim_R s'(x)$;
- for all $(a, a') \in R$, it is the case that $a \in \mathrm{dom}(h)$, $a' \in \mathrm{dom}(h')$ and $h(a) \sim_R h'(a')$;
- for all $(a, a') \in R$, it is the case that $a+1 \in \mathrm{dom}(h)$, $a'+1 \in \mathrm{dom}(h')$ and $h(a+1) \sim_R h'(a'+1)$.

We say that states (s, h) and (s', h') are equivalent, and write $(s, h) \sim (s', h')$, if, for some R, $(s, h) \sim_R (s', h')$.

(a) Let

$$s_1 = (x \mapsto \ulcorner 33 \urcorner, y \mapsto \ulcorner 15 \urcorner, z \mapsto 20) \qquad s_2 = (x \mapsto \ulcorner 5 \urcorner, y \mapsto \ulcorner 29 \urcorner, z \mapsto 20)$$

$$h_1 = \begin{pmatrix} 1 \mapsto \ulcorner 15 \urcorner, & 2 \mapsto \ulcorner 21 \urcorner, \\ 7 \mapsto 4, & 8 \mapsto \ulcorner 21 \urcorner, \\ 15 \mapsto \ulcorner 33 \urcorner, & 16 \mapsto \ulcorner 21 \urcorner, \\ 21 \mapsto \ulcorner 33 \urcorner, & 22 \mapsto \ulcorner 7 \urcorner, \\ 33 \mapsto \ulcorner 15 \urcorner, & 34 \mapsto \ulcorner 21 \urcorner \end{pmatrix} \qquad h_2 = \begin{pmatrix} 1 \mapsto \ulcorner 5 \urcorner, & 2 \mapsto \ulcorner 7 \urcorner, \\ 5 \mapsto \ulcorner 29 \urcorner, & 6 \mapsto \ulcorner 1 \urcorner, \\ 7 \mapsto 4, & 8 \mapsto \ulcorner 1 \urcorner, \\ 13 \mapsto 7, & 14 \mapsto \ulcorner 17 \urcorner, \\ 17 \mapsto 17, & 18 \mapsto \ulcorner 13 \urcorner, \\ 29 \mapsto \ulcorner 5 \urcorner, & 30 \mapsto \ulcorner 1 \urcorner \end{pmatrix}$$

Find a relation R (a set of pairs of addresses) such that $(s_1, h_1) \sim_R (s_2, h_2)$.

(b) Show that, for all states $(s_1, h_1), (s_1', h_1'), (s_2, h_2), (s_2', h_2')$ and addresses a, a' if

$$\begin{aligned} & (s_1, h_1) \sim (s_1', h_1') \\ & \langle \texttt{newpair}, s_1, h_1 \rangle \Downarrow_e \langle \ulcorner a \urcorner, s_2, h_2 \rangle \\ \text{and} \qquad & \langle \texttt{newpair}, s_1', h_1' \rangle \Downarrow_e \langle \ulcorner a' \urcorner, s_2', h_2' \rangle \end{aligned}$$

then there is some R with $(a, a') \in R$ and $(s_2, h_2) \sim_R (s_2', h_2')$. [Partial marks will be awarded for just giving R. More marks will be awarded for explaining in English why $(s_2, h_2) \sim_R (s_2', h_2')$. Full marks will be awarded for *proving* that $(s_2, h_2) \sim_R (s_2', h_2')$.]

(c) Prove by induction on the structure of E that, for all expressions E, states (s_1, h_1), (s_1', h_1'), (s_2, h_2), (s_2', h_2') and values v, v', if

$$\begin{aligned} & (s_1, h_1) \sim (s_1', h_1') \\ & \langle E, s_1, h_1 \rangle \Downarrow_e \langle v, s_2, h_2 \rangle \\ \text{and} \qquad & \langle E, s_1', h_1' \rangle \Downarrow_e \langle v', s_2', h_2' \rangle \end{aligned}$$

then there is some R with $v \sim_R v'$ and $(s_2, h_2) \sim_R (s_2', h_2')$. This result establishes that the semantics of expressions is deterministic up to equivalence.