Computation Assessed Coursework I — Solutions

These solutions cover the WHILEDM part of the first assessed coursework for Models of Computation.

The aim of this coursework was to test the operational semantics part of the course, while extending the ideas beyond the material presented there. In particular, while big-step operational semantics for expressions are covered in the course, the coursework covers big-step semantics for an extended While language. The coursework also gives a flavour of dynamically allocated memory, which is a common feature of most programming languages, as well as testing induction with a moderately complex problem.

This part of the coursework deals with big-step semantics, which differ from small-step semantics in several key ways.

- Big-step semantics define a relation between program configurations and answer configurations: they take a program to its result in a single step.
- If a program never terminates, the big-step semantics does not take it to any answer configuration (typically).
- The derivations for big-step semantics tend to branch following the structure of the program. For example, the derivation for an expression with two sub-expressions will have a branch for each expression (for example, see Figure 1).
- Even programs that require no computation are covered by big-step rules. For instance, $\langle 1, \emptyset, \emptyset \rangle \Downarrow_e \langle 1, \emptyset, \emptyset \rangle$.
- Finally, answer configurations can be different types of things to program configurations. For instance, $\langle \ulcorner 1 \urcorner, \emptyset, (1 \mapsto 0, 2 \mapsto 0) \rangle$ is an answer configuration for expressions of WHILEDM but not a program configuration since $\ulcorner 1 \urcorner$ is not part of the syntax of the language.

On the other hand...

- Small-step semantics define a relation between program configurations and other program configurations: they take a program to a more-evaluated program, reaching a result (if any) through a series of (minimal) steps.
- If a program never terminates, then it has an infinite small-step derivation sequence.
- The derivations of small-step semantics tend not to branch, but focus down the derivation to the next part of the computation to be executed.
- Programs that require no computation have no small-step derivations: they are normal forms.
- Finally, answer configurations and program configurations all have the same type.

1. (a) There are, in fact, two possible derivations, which are given in Figure 1. The first derivation was by far the most popular choice.

   Note that, as the answer shows, it is not generally necessary to include mathematical side conditions such as $1 \notin \mathrm{dom}(\emptyset)$, $1 \in \mathrm{dom}(1 \mapsto 0, 2 \mapsto 0)$ and $0 = 0 \,\underline{+}\, 0$ if it is obvious that they hold, though it is perfectly acceptable to do so — as long as they are correct. In particular, the side condition for E.SND in the first derivation should be $6 \in \mathrm{dom}(1 \mapsto 0, 2 \mapsto 0, 5 \mapsto 0, 6 \mapsto 0)$ (rather than 5). It is also acceptable to use variables (such as h and h') to stand for heaps in the derivation, provided that it is explicit which heaps they stand for.

Note that the E.ADD rule requires that the second argument of the addition should be evaluated from the state given by evaluating the first argument on the initial state. If side-effects of expressions did not propagate in this way, the second use of `newpair` would not be aware that the address 1 had already been allocated, and could therefore allocate it again, which is generally undesirable behaviour. It is therefore a mistake to evaluate $\langle \texttt{newpair.snd}, \emptyset, \emptyset\rangle$ in the derivation.

One mistake is to write things like:

$$\cancel{\text{E.FST}\ \dfrac{\text{E.NEW}\ \dfrac{}{\langle \texttt{newpair}, \emptyset, \emptyset\rangle \Downarrow_e \langle \ulcorner 1 \urcorner, \emptyset, (1 \mapsto 0, 2 \mapsto 0)\rangle}}{\langle \ulcorner 1 \urcorner\texttt{.fst}, \emptyset, \emptyset\rangle \Downarrow_e \langle 0, \emptyset, (1 \mapsto 0, 2 \mapsto 0)\rangle}}$$

The left hand side of conclusion should be the configuration that is being evaluated. The correct configuration in this case is $\langle \texttt{newpair.fst}, \emptyset, \emptyset\rangle$. In fact, $\ulcorner 1 \urcorner\texttt{.fst}$ is not even a syntactically valid expression (since $\ulcorner 1 \urcorner$ is not an expression).

Another mistake is to write 1 and 5 where $\ulcorner 1 \urcorner$ and $\ulcorner 5 \urcorner$ are needed. The $\ulcorner\,\urcorner$ are necessary to differentiate between number values (which can be added together but not dereferenced) and literal address values (which can be dereferenced but not added). Also, writing $\ulcorner 6 \urcorner$ instead of $\ulcorner 5 \urcorner$ is a mistake, because `newpair` always returns the first address of the pair. E`.snd` retrieves the value stored at the address one higher than that returned by E.

[5 marks]

(b) An expression containing `newpair` is a valid answer since it can allocate any two consecutive memory locations that are not already allocated. So we have

$$\langle \texttt{newpair}, \emptyset, \emptyset\rangle \Downarrow_e \langle \ulcorner 1 \urcorner, \emptyset, (1 \mapsto 0, 2 \mapsto 0)\rangle$$

and

$$\langle \texttt{newpair}, \emptyset, \emptyset\rangle \Downarrow_e \langle \ulcorner 5 \urcorner, \emptyset, (5 \mapsto 0, 6 \mapsto 0)\rangle$$

and $\ulcorner 1 \urcorner \neq \ulcorner 5 \urcorner$.

[2 marks]

(c) No, the semantics is not deterministic. Determinacy requires that each *configuration* evaluates in most one way, so it is not enough to say that the same expression may evaluate to different things in different states — it has to be the *same* state. For instance, the expression x evaluates to different things in different states, but cannot be said to be non-deterministic, since it evaluates to at most one thing in any given state. In fact, the only program construct that violates determinacy in WHILEDM is `newpair`. [1 mark]

(d) There are several expressions where this is possible:

- the evaluation requires looking up the value of a variable that is not in the variable store, *e.g.* x with: $\langle x, \emptyset, \emptyset\rangle$ and $\langle x, (x \mapsto 1), \emptyset)\rangle$;

First derivation:

$$\text{E.ADD}\;\dfrac{\text{E.FST}\;\dfrac{\text{E.NEW}\;\dfrac{}{\langle \texttt{newpair}, \emptyset, \emptyset\rangle \Downarrow_e \langle \ulcorner 1 \urcorner, \emptyset, (1 \mapsto 0, 2 \mapsto 0)\rangle}}{\langle \texttt{newpair.fst}, \emptyset, \emptyset\rangle \Downarrow_e \langle 0, \emptyset, (1 \mapsto 0, 2 \mapsto 0)\rangle} \qquad \text{E.SND}\;\dfrac{\text{E.NEW}\;\dfrac{}{\langle \texttt{newpair}, \emptyset, (1 \mapsto 0, 2 \mapsto 0)\rangle \Downarrow_e \langle \ulcorner 5 \urcorner, \emptyset, (1 \mapsto 0, 2 \mapsto 0, 5 \mapsto 0, 6 \mapsto 0)\rangle}}{\langle \texttt{newpair.snd}, \emptyset, (1 \mapsto 0, 2 \mapsto 0)\rangle \Downarrow_e \langle 0, \emptyset, (1 \mapsto 0, 2 \mapsto 0, 5 \mapsto 0, 6 \mapsto 0)\rangle}}{\langle \texttt{newpair.fst} + \texttt{newpair.snd}, \emptyset, \emptyset\rangle \Downarrow_e \langle 0, \emptyset, (1 \mapsto 0, 2 \mapsto 0, 5 \mapsto 0, 6 \mapsto 0)\rangle}$$

Alternative derivation:

$$\text{E.ADD}\;\dfrac{\text{E.FST}\;\dfrac{\text{E.NEW}\;\dfrac{}{\langle \texttt{newpair}, \emptyset, \emptyset\rangle \Downarrow_e \langle \ulcorner 5 \urcorner, \emptyset, (5 \mapsto 0, 6 \mapsto 0)\rangle}}{\langle \texttt{newpair.fst}, \emptyset, \emptyset\rangle \Downarrow_e \langle 0, \emptyset, (5 \mapsto 0, 6 \mapsto 0)\rangle} \qquad \text{E.SND}\;\dfrac{\text{E.NEW}\;\dfrac{}{\langle \texttt{newpair}, \emptyset, (5 \mapsto 0, 6 \mapsto 0)\rangle \Downarrow_e \langle \ulcorner 1 \urcorner, \emptyset, (1 \mapsto 0, 2 \mapsto 0, 5 \mapsto 0, 6 \mapsto 0)\rangle}}{\langle \texttt{newpair.snd}, \emptyset, (5 \mapsto 0, 6 \mapsto 0)\rangle \Downarrow_e \langle 0, \emptyset, (1 \mapsto 0, 2 \mapsto 0, 5 \mapsto 0, 6 \mapsto 0)\rangle}}{\langle \texttt{newpair.fst} + \texttt{newpair.snd}, \emptyset, \emptyset\rangle \Downarrow_e \langle 0, \emptyset, (1 \mapsto 0, 2 \mapsto 0, 5 \mapsto 0, 6 \mapsto 0)\rangle}$$

Figure 1: Derivations for $\langle \texttt{newpair.fst} + \texttt{newpair.snd}, \emptyset, \emptyset\rangle \Downarrow_e \langle 0, \emptyset, (1 \mapsto 0, 2 \mapsto 0, 5 \mapsto 0, 6 \mapsto 0)\rangle$

- the evaluation requires looking up the field which is not allocated in the heap, *e.g.* $x.\mathtt{fst}$ with: $\langle x.\mathtt{fst}, (x \mapsto \ulcorner 7 \urcorner), \emptyset \rangle$ and $\langle x.\mathtt{fst}, (x \mapsto \ulcorner 7 \urcorner), (\ulcorner 7 \urcorner \mapsto 3) \rangle$;

An important thing to note for this question is that literal addresses, such as $\ulcorner 7 \urcorner$, are not expressions, although they are values. This means that $\ulcorner 7 \urcorner + 0$ is not an expression, and so is not a valid answer if presented as an expression. On the other hand $x + 0$ is an expression, and x could be assigned the value $\ulcorner 7 \urcorner$ in the variable store, which is acceptable.

[2 marks]

(e) Such an expression must involve either of the following, irrespective of state:

- the evaluation requires looking up the field of a number, *e.g.* $1.\mathtt{fst}$;
- the evaluation requires performing addition when one of the operands is an address, *e.g.* $\mathtt{newpair} + 0$.

Alternatively, convoluted examples that make use of different states are possible, such as $x + x.\mathtt{fst}$, which are undefined for different reasons in different states. In this case, if x is not in the variable store, the semantics is undefined because x cannot be looked up; if x has an address value, the semantics is undefined because an address cannot be added to anything; if x has a number value, the semantics is undefined because a number cannot be dereferenced (in any state).

An important thing to note for this question is that literal addresses, such as $\ulcorner 7 \urcorner$, are not expressions, although they are values. This means that $\ulcorner 7 \urcorner + 0$ is not an expression, and so is not a valid answer if presented as an expression.

[2 marks]

(f) No, the semantics is not total. This directly follows from both the previous two answers, and the definition of totality. Make particular note that the totality of a semantics also depends upon the state of a particular configuration as well as the expression.

[1 mark]

[Total for question: 13 marks]

2. (a) An appropriate choice of rules is:

$$\textsc{b.true}\ \frac{}{\langle \mathtt{true}, s, h \rangle \Downarrow_b \langle \mathtt{true}, s, h \rangle} \qquad \textsc{b.false}\ \frac{}{\langle \mathtt{false}, s, h \rangle \Downarrow_b \langle \mathtt{false}, s, h \rangle}$$

$$\textsc{b.and.tt}\ \frac{\langle B_1, s, h \rangle \Downarrow_b \langle \mathtt{true}, s', h' \rangle \qquad \langle B_2, s', h' \rangle \Downarrow_b \langle \mathtt{true}, s'', h'' \rangle}{\langle B_1 \,\&\, B_2, s, h \rangle \Downarrow_b \langle \mathtt{true}, s'', h'' \rangle}$$

$$\textsc{b.and.tf}\ \frac{\langle B_1, s, h \rangle \Downarrow_b \langle \mathtt{true}, s', h' \rangle \qquad \langle B_2, s', h' \rangle \Downarrow_b \langle \mathtt{false}, s'', h'' \rangle}{\langle B_1 \,\&\, B_2, s, h \rangle \Downarrow_b \langle \mathtt{false}, s'', h'' \rangle}$$

$$\textsc{b.and.ft}\ \frac{\langle B_1, s, h \rangle \Downarrow_b \langle \mathtt{false}, s', h' \rangle \qquad \langle B_2, s', h' \rangle \Downarrow_b \langle \mathtt{true}, s'', h'' \rangle}{\langle B_1 \,\&\, B_2, s, h \rangle \Downarrow_b \langle \mathtt{false}, s'', h'' \rangle}$$

$$\textsc{b.and.ff}\ \frac{\langle B_1, s, h \rangle \Downarrow_b \langle \mathtt{false}, s', h' \rangle \qquad \langle B_2, s', h' \rangle \Downarrow_b \langle \mathtt{false}, s'', h'' \rangle}{\langle B_1 \,\&\, B_2, s, h \rangle \Downarrow_b \langle \mathtt{false}, s'', h'' \rangle}$$

$$\textsc{b.not.true}\ \frac{\langle B, s, h \rangle \Downarrow_b \langle \mathtt{false}, s', h' \rangle}{\langle \neg B, s, h \rangle \Downarrow_b \langle \mathtt{true}, s', h' \rangle} \qquad \textsc{b.not.false}\ \frac{\langle B, s, h \rangle \Downarrow_b \langle \mathtt{true}, s', h' \rangle}{\langle \neg B, s, h \rangle \Downarrow_b \langle \mathtt{false}, s', h' \rangle}$$

Another possibility is to give the semantics of & using three rules, such as:

$$\text{B.AND.TRUE}\ \frac{\langle B_1, s, h\rangle \Downarrow_b \langle \mathtt{true}, s', h'\rangle \qquad \langle B_2, s', h'\rangle \Downarrow_b \langle \mathtt{true}, s'', h''\rangle}{\langle B_1 \mathbin{\&} B_2, s, h\rangle \Downarrow_b \langle \mathtt{true}, s'', h''\rangle}$$

$$\text{B.AND.FALSE.L}\ \frac{\langle B_1, s, h\rangle \Downarrow_b \langle \mathtt{false}, s', h'\rangle \qquad \langle B_2, s', h'\rangle \Downarrow_b \langle b, s'', h''\rangle}{\langle B_1 \mathbin{\&} B_2, s, h\rangle \Downarrow_b \langle \mathtt{false}, s'', h''\rangle}$$

$$\text{B.AND.FALSE.R}\ \frac{\langle B_1, s, h\rangle \Downarrow_b \langle \mathtt{true}, s', h'\rangle \qquad \langle B_2, s', h'\rangle \Downarrow_b \langle \mathtt{false}, s'', h''\rangle}{\langle B_1 \mathbin{\&} B_2, s, h\rangle \Downarrow_b \langle \mathtt{false}, s'', h''\rangle}$$

The following alternative rule is also acceptable:

$$\text{B.AND.FALSE.R}\ \frac{\langle B_1, s, h\rangle \Downarrow_b \langle b, s', h'\rangle \qquad \langle B_2, s', h'\rangle \Downarrow_b \langle \mathtt{false}, s'', h''\rangle}{\langle B_1 \mathbin{\&} B_2, s, h\rangle \Downarrow_b \langle \mathtt{false}, s'', h''\rangle}$$

Having this rule gives more than one possible derivation for Boolean expressions such as `false` & `false` (conjunctions of two Booleans which both evaluate to `false`), but not with different outcomes.

It is also possible to give the semantics of & with just the following two rules:

$$\text{B.AND.L.TRUE}\ \frac{\langle B_1, s, h\rangle \Downarrow_b \langle \mathtt{true}, s', h'\rangle \qquad \langle B_2, s', h'\rangle \Downarrow_b \langle b, s'', h''\rangle}{\langle B_1 \mathbin{\&} B_2, s, h\rangle \Downarrow_b \langle b, s'', h''\rangle}$$

$$\text{B.AND.L.FALSE}\ \frac{\langle B_1, s, h\rangle \Downarrow_b \langle \mathtt{false}, s', h'\rangle \qquad \langle B_2, s', h'\rangle \Downarrow_b \langle b, s'', h''\rangle}{\langle B_1 \mathbin{\&} B_2, s, h\rangle \Downarrow_b \langle \mathtt{false}, s'', h''\rangle}$$

A rare approach for defining semantics for & and ¬ was to give the definitions for the `true` and `false` cases, and then give the definition for general expressions in terms of these. For instance:

$$\text{B.NOT.TRUE}\ \frac{}{\langle \neg\mathtt{true}, s, h\rangle \Downarrow_b \langle \mathtt{false}, s, h\rangle} \qquad \text{B.NOT.FALSE}\ \frac{}{\langle \neg\mathtt{false}, s, h\rangle \Downarrow_b \langle \mathtt{true}, s, h\rangle}$$

$$\text{B.NOT}\ \frac{\langle B, s, h\rangle \Downarrow_b \langle b, s', h'\rangle \qquad \langle \neg b, s', h'\rangle \Downarrow_b \langle b', s'', h''\rangle}{\langle \neg B, s, h\rangle \Downarrow_b \langle b', s'', h''\rangle}$$

This is technically correct (since boolean values are themselves boolean expressions), however, the premise $\langle \neg b, s', h'\rangle \Downarrow_b \langle b', s', h'\rangle$ is quite close to being a side-condition, which is not in the spirit of the question. Also, it is quite easy to get this approach wrong; for example, the rule

$$\text{B.NOT}\ \xcancel{\frac{\langle B, s, h\rangle \Downarrow_b \langle b, s', h'\rangle}{\langle \neg B, s, h\rangle \Downarrow_b \langle \neg b, s', h'\rangle}}$$

is not correct, because $\neg b$ is not a Boolean value.

It is important to note that evaluating a Boolean expression can have side-effects, since it may involve evaluating an expression, so it is important that side-effects are handled correctly, as specified in the question. It can be tempting to short-circuit the second boolean expression in an & if the first evaluates to `false`. However, the question required strict evaluation, so rules such as the following are not acceptable:

$$\xcancel{\text{B.AND.SHORT}\ \frac{\langle B_1, s, h\rangle \Downarrow_b \langle \texttt{false}, s', h'\rangle}{\langle B_1 \mathbin{\&} B_2, s, h\rangle \Downarrow_b \langle \texttt{false}, s', h'\rangle}}$$

An occasional mistake was to define the rules for & on the Boolean values (`true` and `false`) such as

$$\frac{}{\langle \texttt{true} \mathbin{\&} \texttt{false}, s, h\rangle \Downarrow_b \langle \texttt{false}, s, h\rangle}$$

and then try to extend these to cover all expressions, but gave rules in which the right side of $\Downarrow_b$ contained expressions, such as

$$\xcancel{\frac{\langle B_1 \mathbin{\&} B_2, s, h\rangle \Downarrow_b \langle \texttt{true} \mathbin{\&} B_2, s', h'\rangle}{\langle B_1 \mathbin{\&} B_2, s, h\rangle \Downarrow_b \langle B_2, s', h'\rangle}}$$

This kind of rule is no good because $\texttt{true} \mathbin{\&} B_2$ is not a $BVal$ (and B_2 need not be, either).

[8 marks]

(b) The only answer is the expression `newpair`, for which we have the following derivation (among others):

$$\text{B.EQ.FALSE}\ \frac{\text{E.NEW}\ \dfrac{}{\langle \texttt{newpair}, \emptyset, \emptyset\rangle \Downarrow_e \langle \ulcorner 1 \urcorner, \emptyset, h\rangle} \qquad \text{E.NEW}\ \dfrac{}{\langle \texttt{newpair}, \emptyset, h\rangle \Downarrow_e \langle \ulcorner 3 \urcorner, \emptyset, h'\rangle}}{\langle \texttt{newpair} = \texttt{newpair}, \emptyset, \emptyset\rangle \Downarrow_b \langle \texttt{false}, \emptyset, h'\rangle}$$

where $h = (1 \mapsto 0, 2 \mapsto 0)$ and $h' = (1 \mapsto 0, 2 \mapsto 0, 3 \mapsto 0, 4 \mapsto 0)$.

The simplest way to answer the question was with a concrete derivation such as the above, where all values are given explicitly. Answering the question with a derivation that has undetermined variables in it risks losing marks. For example,

$$\xcancel{\frac{\dfrac{a \notin \mathrm{dom}(h) \quad a+1 \notin \mathrm{dom}(h)}{\langle \texttt{newpair}, s, h\rangle \Downarrow_e \langle \ulcorner a \urcorner, s', h'\rangle} \quad \dfrac{b \notin \mathrm{dom}(h'') \quad b+1 \notin \mathrm{dom}(h'')}{\langle \texttt{newpair}, s'', h''\rangle \Downarrow_e \langle \ulcorner b \urcorner, s''', h'''\rangle} \quad \ulcorner a \urcorner \neq \ulcorner b \urcorner}{\langle \texttt{newpair} = \texttt{newpair}, s, h\rangle \Downarrow_b \langle \texttt{false}, s''', h'''\rangle}}$$

would not be sufficient on its own. It would also be necessary to explain the relationships between the different states and the addresses a and b. In particular:

- since h is a finite partial function there must be some address a with $a \notin \mathrm{dom}(h)$ and $a+1 \notin \mathrm{dom}(h)$
- $s' = s$ and $h' = h[a \mapsto 0, a+1 \mapsto 0]$ (which is required by E.NEW)
- $s'' = s'$ and $h'' = h'$ (which this is required by B.EQ.FALSE)
- since h'' is a finite partial function there must be some address b with $b \notin \mathrm{dom}(h'')$ and $b+1 \notin \mathrm{dom}(h'')$
- $s''' = s''$ and $h''' = h''[b \mapsto 0, b+1 \mapsto 0]$
- since $a \in \mathrm{dom}(h') = \mathrm{dom}(h'')$ but $b \notin \mathrm{dom}(h'')$, it must be that $a \neq b$ and so $\ulcorner a \urcorner \neq \ulcorner b \urcorner$.

A take-away message from this question is that stateful behaviour in programming languages can constrain possible optimisations. For example, it would be tempting to optimise the expression $E = E$ to `true` for any E, but this example shows that this is not a valid optimisation. (Another reason for not optimising this expression is that the semantics is not total, so evaluation should get stuck in the case where $E = 1.\texttt{fst}$, for instance.)

[4 marks]

[Total for question: 12 marks]

3. (a) The state is

$$s' \equiv (c \mapsto \ulcorner 3 \urcorner, x \mapsto 3, y \mapsto 12) \equiv s[c \mapsto \ulcorner 3 \urcorner, x \mapsto 3]$$
$$h' \equiv (1 \mapsto 7, 2 \mapsto \ulcorner 5 \urcorner, 3 \mapsto \ulcorner 1 \urcorner, 4 \mapsto 0, 5 \mapsto 12, 6 \mapsto \ulcorner 3 \urcorner) \equiv h[5 \mapsto 12]$$

The derivation is given in Figure 2.

Some common mistakes with this question include:

- incorrectly identifying the states at different points;
- interpreting E.SND as if it were E.FST.
- interpreting the command as $(x := c.\texttt{fst}; \texttt{fst}[c] \leftarrow y); c := c.\texttt{snd}$ rather than $x := c.\texttt{fst}; (\texttt{fst}[c] \leftarrow y; c := c.\texttt{snd})$ as stipulated by the question: "You may assume that ; is right-associative: that is, $C_1; C_2; C_3$ is interpreted as $C_1; (C_2; C_3)$."
- reversing the order of execution of sequentially composed commands, presumably by misunderstanding right-associativity.
- all we mean by right associativity is that we want you to treat $C_1; C_2; C_3$ as $C_1; (C_2; C_3)$ rather than $(C_1; C_2); C_3$. This means your derivation tree should be of the shape:

$$\dfrac{\dfrac{\vdots}{C_1} \qquad \dfrac{\dfrac{\vdots}{C_2} \qquad \dfrac{\vdots}{C_3}}{C_2; C_3}}{C_1; (C_2; C_3)}$$

  . . . not of the shape:

$$\xcancel{\dfrac{\dfrac{\dfrac{\vdots}{C_1} \qquad \dfrac{\vdots}{C_2}}{C_1; C_2} \qquad \dfrac{\vdots}{C_3}}{(C_1; C_2); C_3}}$$

- performing an additional dereference on $\ulcorner 3 \urcorner$ in $c := c.\texttt{snd}$

[12 marks]

(b) The state is

$$s' \equiv (hd \mapsto \ulcorner 1 \urcorner, x \mapsto \ulcorner 1 \urcorner, c \mapsto 0, y \mapsto 3)$$
$$h' \equiv (1 \mapsto 7, 2 \mapsto \ulcorner 5 \urcorner, 3 \mapsto 3, 4 \mapsto 0, 5 \mapsto 12, 6 \mapsto \ulcorner 3 \urcorner)$$

The program essentially operates on a linked list starting at the address stored in hd and shifting the values (stored in the first component of each pair) one step along the list. The second component of each pair is the address of the next element of the list, or 0 if it is the last element in the list.

After the first iteration of the loop, the state will be

$$(hd \mapsto \ulcorner 1 \urcorner, x \mapsto 12, c \mapsto \ulcorner 5 \urcorner, y \mapsto 7)$$
$$(1 \mapsto 7, 2 \mapsto \ulcorner 5 \urcorner, 3 \mapsto \ulcorner 1 \urcorner, 4 \mapsto 0, 5 \mapsto 3, 6 \mapsto \ulcorner 3 \urcorner).$$

$$(\dagger) \equiv \;\textsc{C.STOR.FST}\,\dfrac{\textsc{E.VAR}\,\dfrac{}{\langle c, s[x \mapsto 3], h\rangle \Downarrow_e \langle \ulcorner 5 \urcorner, s[x \mapsto 3], h\rangle} \qquad \textsc{E.VAR}\,\dfrac{}{\langle y, s[x \mapsto 3], h\rangle \Downarrow_e \langle 12, s[x \mapsto 3], h\rangle}}{\langle \texttt{fst}[c] \leftarrow y, s[x \mapsto 3], h\rangle \Downarrow_c \langle s[x \mapsto 3], h'\rangle}$$

$$\textsc{C.SEQ}\,\dfrac{\textsc{C.ASSIGN}\,\dfrac{\textsc{E.FST}\,\dfrac{\textsc{E.VAR}\,\dfrac{}{\langle c, s, h\rangle \Downarrow_e \langle \ulcorner 5 \urcorner, s, h\rangle}}{\langle c.\texttt{fst}, s, h\rangle \Downarrow_e \langle 3, s, h\rangle}}{\langle x := c.\texttt{fst}, s, h\rangle \Downarrow_c \langle s[x \mapsto 3], h\rangle} \qquad \textsc{C.SEQ}\,\dfrac{(\dagger) \qquad \textsc{C.ASSIGN}\,\dfrac{\textsc{E.SND}\,\dfrac{\textsc{E.VAR}\,\dfrac{}{\langle c, s[x \mapsto 3], h'\rangle \Downarrow_e \langle \ulcorner 5 \urcorner, s[x \mapsto 3], h'\rangle}}{\langle c.\texttt{snd}, s[x \mapsto 3], h'\rangle \Downarrow_e \langle \ulcorner 3 \urcorner, s[x \mapsto 3], h'\rangle}}{\langle c := c.\texttt{snd}, s[x \mapsto 3], h'\rangle \Downarrow_c \langle s', h'\rangle}}{\langle \texttt{fst}[c] \leftarrow y; c := c.\texttt{snd}, s[x \mapsto 3], h\rangle \Downarrow_c \langle s', h'\rangle}}{\langle x := c.\texttt{fst}; \texttt{fst}[c] \leftarrow y; c := c.\texttt{snd}, s, h\rangle \Downarrow_c \langle s', h'\rangle}$$

Figure 2: Derivation of $\langle x := c.\texttt{fst}; \texttt{fst}[c] \leftarrow y; c := c.\texttt{snd}, s, h\rangle \Downarrow_c \langle s', h'\rangle$

After the second iteration of the loop, the state will be

$$(hd \mapsto \ulcorner 1 \urcorner, x \mapsto 3, c \mapsto \ulcorner 3 \urcorner, y \mapsto 12)$$
$$(1 \mapsto 7, 2 \mapsto \ulcorner 5 \urcorner, 3 \mapsto \ulcorner 1 \urcorner, 4 \mapsto 0, 5 \mapsto 12, 6 \mapsto \ulcorner 3 \urcorner).$$

After the third iteration of the loop, the state will be

$$(hd \mapsto \ulcorner 1 \urcorner, x \mapsto \ulcorner 1 \urcorner, c \mapsto 0, y \mapsto 3)$$
$$(1 \mapsto 7, 2 \mapsto \ulcorner 5 \urcorner, 3 \mapsto 3, 4 \mapsto 0, 5 \mapsto 12, 6 \mapsto \ulcorner 3 \urcorner).$$

Since c now has value 0, the loop exits, and this is the final state.
A common error was to omit the values of c and y from the variable store.

[4 marks]

[Total for question: 16 marks]

4. (a) One thing to note is that the type of a value will be nat if it's a number and a pair type if it is an address of a pair of things that are themselves typeable.

   i. $\tau \equiv ((\mathsf{nat}, (\mathsf{nat}, \mathsf{nat})), \mathsf{nat})$. To illustrate (not required by the question), this type is derived as follows:

$$\dfrac{\dfrac{\dfrac{}{h \Vdash 6 : \mathsf{nat}} \qquad \dfrac{\dfrac{}{h \Vdash 3 : \mathsf{nat}} \qquad \dfrac{}{h \Vdash 0 : \mathsf{nat}}}{h \Vdash \ulcorner 7 \urcorner : (\mathsf{nat}, \mathsf{nat})}}{h \Vdash \ulcorner 5 \urcorner : (\mathsf{nat}, (\mathsf{nat}, \mathsf{nat}))} \qquad \dfrac{}{h \Vdash 2 : \mathsf{nat}}}{h \Vdash \ulcorner 3 \urcorner : ((\mathsf{nat}, (\mathsf{nat}, \mathsf{nat})), \mathsf{nat})}$$

   ii. There is no such τ'. Because $h(9)$ is undefined, it is not possible to give any type to $\ulcorner 9 \urcorner$.

   iii. Again, there is no such type, because in order to derive a type for $\ulcorner 1 \urcorner$ we must *already have* derivation of a type for $\ulcorner 1 \urcorner$:

$$\dfrac{h \Vdash \ulcorner 3 \urcorner : ((\mathsf{nat}, (\mathsf{nat}, \mathsf{nat})), \mathsf{nat}) \qquad \dfrac{\vdots}{h \Vdash \ulcorner 1 \urcorner : ?}}{h \Vdash \ulcorner 1 \urcorner : (((\mathsf{nat}, (\mathsf{nat}, \mathsf{nat})), \mathsf{nat}), ?)}$$

   Because derivations and types are necessarily finite (they are inductively defined), it is therefore impossible to derive any type of $\ulcorner 1 \urcorner$ in h. (Note: Haskell's type system similarly rejects this kind of circular pair structure. For instance the program `f = (0, f)` is rejected with the error "cannot construct the infinite type: `t1 = (t0, t1)`".)
   A quick proof that, for all τ, τ', $\tau \neq (\tau', \tau)$:
   Proceed by induction on the structure of τ. For the base case, $\tau = \mathsf{nat}$ and clearly $\mathsf{nat} \neq (\tau', \mathsf{nat})$. For the inductive case, $\tau = (\tau_1, \tau_2)$ and as inductive hypotheses we have that $\forall \tau'. \tau_1 \neq (\tau', \tau_1)$ and $\forall \tau'. \tau_2 \neq (\tau', \tau_2)$. Suppose for a contradiction that $\tau = (\tau', \tau)$. It must be that $\tau_1 = \tau'$ and $\tau_2 = \tau$; hence $\tau_2 = (\tau', \tau_2)$, but this violates the second inductive hypothesis.
   (In Haskell, you might be used having infinite data structures, such as the list `[1..]`. These are not inductive data structures, but rather coinductive.)

[3 marks]

(b) It suffices to check that the value of each variable mentioned in the typing context can be given its assigned type in the heap. Note that a context need not type everything.

i. Yes.

ii. No. x is a number, and so cannot be typed as a pair.

iii. Yes.

[3 marks]

(c) Fix some arbitrary typing context Γ. We wish to show $\forall E.\, P(E)$, where

$$P(E) \equiv \forall s, h, \tau.\, [\Gamma; s; h \vdash \mathsf{well\text{-}typed} \wedge \Gamma \vdash E : \tau \implies \exists v, s', h'.\, \langle E, s, h\rangle \Downarrow_e \langle v, s', h'\rangle \wedge \Gamma; s'; h' \vdash \mathsf{well\text{-}typed} \wedge h' \Vdash v : \tau]$$

Proceed by induction on the structure of the expression E.

Base Case: $E = n \in \mathbb{N}$. Assume $\Gamma; s; h \vdash \mathsf{well\text{-}typed}$ and $\Gamma \vdash n : \tau$. From the latter, it must be that $\tau = \mathsf{nat}$.

Let $v = n$, $s' = s$ and $h' = h$. We have $\langle n, s, h\rangle \Downarrow_e \langle v, s', h'\rangle$ by E.NUM, and $\Gamma; s'; h' \vdash \mathsf{well\text{-}typed}$ and $h' \Vdash v : \tau$ as required.

Base Case: $E = x \in \mathit{Var}$. Assume $\Gamma; s; h \vdash \mathsf{well\text{-}typed}$ and $\Gamma \vdash x : \tau$. From the latter, it must be that $\Gamma(x) = \tau$, so from the former it must be that $h \Vdash s(x) : \tau$.

Let $v = s(x)$, $s' = s$ and $h' = h$. We have $\langle x, s, h\rangle \Downarrow_e \langle v, s', h'\rangle$ by E.VAR, and $\Gamma; s'; h' \vdash \mathsf{well\text{-}typed}$ and $h' \Vdash v : \tau$ as required.

Base Case: $E = \texttt{newpair}$. Assume $\Gamma; s; h \vdash \mathsf{well\text{-}typed}$ and $\Gamma \vdash \texttt{newpair} : \tau$. From this, it must be that $\tau = (\mathsf{nat}, \mathsf{nat})$. Since $\mathrm{dom}(h)$ is finite, we can pick some address a that is greater than every $a' \in \mathrm{dom}(h)$. In particular, $a, a+1 \notin \mathrm{dom}(h)$.

Let $v = \ulcorner a \urcorner$, $s' = s$ and $h' = h[a \mapsto 0][a{+}1 \mapsto 0]$. We have $\langle \texttt{newpair}, s, h\rangle \Downarrow_e \langle v, s', h'\rangle$ by E.NEW. By the lemma, we have $\Gamma; s'; h' \vdash \mathsf{well\text{-}typed}$. Finally, we have

$$\frac{h' \Vdash h'(a) : \mathsf{nat} \qquad h' \Vdash h'(a+1) : \mathsf{nat}}{h' \Vdash \ulcorner a \urcorner : (\mathsf{nat}, \mathsf{nat})}$$

as required.

Inductive Case: $E = E_1 + E_2$. As the inductive hypotheses, we assume $P(E_1)$ and $P(E_2)$. Assume $\Gamma; s; h \vdash \mathsf{well\text{-}typed}$ and $\Gamma \vdash E_1 + E_2 : \tau$. From the latter, it must be that $\tau = \mathsf{nat}$, $\Gamma \vdash E_1 : \mathsf{nat}$ and $\Gamma \vdash E_2 : \mathsf{nat}$. Consequently, from $P(E_1)$ there must be some v_1, h'_1, s'_1 with $\langle E_1, s, h\rangle \Downarrow_e \langle v_1, s'_1, h'_1\rangle$ and $\Gamma; s'_1; h'_1 \vdash \mathsf{well\text{-}typed}$ and $h'_1 \Vdash v_1 : \mathsf{nat}$. From the type of v_1, we have that $v_1 = n_1 \in \mathbb{N}$. Now from $P(E_2)$ there must be some v_2, h'_2, s'_2 with $\langle E_2, s'_1, h'_1\rangle \Downarrow_e \langle v_2, h'_2, s'_2\rangle$ and $\Gamma; s'_2; h'_2 \vdash \mathsf{well\text{-}typed}$ and $h'_2 \Vdash v_2 : \mathsf{nat}$. As before, we have that $v_1 = n_2 \in \mathbb{N}$.

Let $v = n_1 \underline{+} n_2$, $s' = s'_2$ and $h' = h'_2$ We have

$$\text{E.ADD}\ \frac{\langle E_1, s, h\rangle \Downarrow_e \langle v_1, s'_1, h'_1\rangle \qquad \langle E_2, s'_1, h'_1\rangle \Downarrow_e \langle v_2, h'_2, s'_2\rangle}{\langle E_1 + E_2, s, h\rangle \Downarrow_e \langle v, s', h'\rangle}$$

and $\Gamma; s'; h' \vdash \mathsf{well\text{-}typed}$ and $h' \Vdash v : \mathsf{nat}$.

Inductive Case: $E = E_1.\texttt{fst}$. As the inductive hypothesis, we assume $P(E_1)$. Assume $\Gamma; s; h \vdash \mathsf{well\text{-}typed}$ and $\Gamma \vdash E_1.\texttt{fst} : \tau$. From this, it must be that $\Gamma \vdash E_1 : (\tau, \tau')$ for some τ'. From $P(E_1)$, there must be some v_1, h'_1, s'_1 with $\langle E_1, s, h\rangle \Downarrow_e$

$\langle v_1, s_1', h_1' \rangle$ and $\Gamma; s_1'; h_1' \vdash \mathsf{well\text{-}typed}$ and $h_1' \Vdash v_1 : (\tau, \tau')$. Consequently, $v_1 = \ulcorner a \urcorner$ for some a with $h_1' \Vdash h_1'(a) : \tau$.
Let $v = h_1'(a)$, $s' = s_1'$ and $h' = h_1'$. We have

$$\textsc{e.fst}\ \frac{\langle E_1, s, h \rangle \Downarrow_e \langle \ulcorner a \urcorner, s', h' \rangle}{\langle E_1.\mathtt{fst}, s, h \rangle \Downarrow_e \langle v, s', h' \rangle}$$

and $\Gamma; s'; h' \vdash \mathsf{well\text{-}typed}$ and $h' \Vdash v : \tau$.
Inductive Case: $E = E_1.\mathtt{snd}$. As the inductive hypothesis, we assume $P(E_1)$. Assume $\Gamma; s; h \vdash \mathsf{well\text{-}typed}$ and $\Gamma \vdash E_1.\mathtt{snd} : \tau$. From this, it must be that $\Gamma \vdash E_1 : (\tau', \tau)$ for some τ'. From $P(E_1)$, there must be some v_1, h_1', s_1' with $\langle E_1, s, h \rangle \Downarrow_e \langle v_1, s_1', h_1' \rangle$ and $\Gamma; s_1'; h_1' \vdash \mathsf{well\text{-}typed}$ and $h_1' \Vdash v_1 : (\tau', \tau)$. Consequently, $v_1 = \ulcorner a \urcorner$ for some a with $h_1' \Vdash h_1'(a+1) : \tau$.
Let $v = h_1'(a+1)$, $s' = s_1'$ and $h' = h_1'$. We have $\langle E_1.\mathtt{snd}, s, h \rangle \Downarrow_e \langle v, s', h' \rangle$ by E.SND, and $\Gamma; s'; h' \vdash \mathsf{well\text{-}typed}$ and $h' \Vdash v : \tau$, as required. $\square$
Some common problems with this question include:

- For the variable case, it is necessary to justify the fact that $s(x)$ is well defined in order to use the E.VAR rule (which explicitly requires that $x \in \mathrm{dom}(s)$. This follows from the two assumptions.
- In the `newpair` case, you should explain how a can be picked to be fresh from $\mathrm{dom}(h)$ (because h is finite). Also, it is clearly not the case that $h \Vdash \ulcorner a \urcorner : (\mathsf{nat}, \mathsf{nat})$ because $a \notin \mathrm{dom}(h)$.
- The inductive hypotheses should be an implication, with s, h, τ universally quantified on the outside. (Γ could also be quantified in this way, although it is not necessary.) Part of the reason for this is that E occurs in the assumption $\Gamma \vdash E : \tau$, so we need to assume this for the specific E of each case, rather than in general. Also, the s, h and τ we use the inductive hypotheses with are not always the same as the ones we're proving the case for. For instance, in the addition case we use the second hypothesis with the s' and h' generated from the first inductive hypothesis.
- For the E.`fst` and E.`snd` cases, it is really important for us to know that E evaluates to an address that is in the heap (with the appropriate type), otherwise the evaluation will get stuck. We have to use the inductive hypothesis for this.
- Some students seemed to misunderstand how rules such as E.VAR can be used in proofs. All our big-step operational rules are of the form:

  $$\textsc{Name}\ \frac{\textit{Premise}}{\textit{Conclusion}}$$

  This means in a proof, if you have already shown *Premise*, then you can appeal to rule NAME in order to prove *Conclusion*. This does *not* mean that if you find yourself in a case where you want *Conclusion* then you get to assume *Premise*. So, a common instance of this mistake was in the case where $E = x$. Students sometimes mistakenly said "by E.VAR we have $\langle x, s, h \rangle \Downarrow_e \langle s(x), s, h \rangle$ and $x \in \mathrm{dom}(s)$". In fact, to use this rule correctly you need to find some other way to show $x \in \mathrm{dom}(s)$ – only then can you say "Since $x \in \mathrm{dom}(s)$, we have $\langle x, s, h \rangle \Downarrow_e \langle s(x), s, h \rangle$ by E.VAR"

[8 marks]

[Total for question: 14 marks]

5. (a) The relation is: $R = \{(33,5),(15,29),(21,1),(7,7)\}$.

Recall,

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

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

In order to find R it is easiest to start with the empty relation $\emptyset$ and add pairs whenever the conditions for $\sim_R$ require. Recall that $(s_1, h_1) \sim_R (s_2, h_2)$ if

- 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 $h(a + 1) \sim_R h'(a' + 1)$.

The first two conditions are met by our first approximation ($\emptyset$). The third condition, however, requires that

$$\begin{aligned} s_1(x) = \ulcorner 33 \urcorner \sim_R \ulcorner 5 \urcorner = s_2(x) \\ s_1(y) = \ulcorner 15 \urcorner \sim_R \ulcorner 29 \urcorner = s_2(y) \\ s_1(z) = 20 \sim_R 20 = s_2(z) \end{aligned}$$

For the first of these, we require $(33, 5) \in R$; for the second, we require $(15, 29) \in R$; for the third, we require $20 = 20$ (which is true). So we add these two new pairs to give a second approximation for R: $\{(33, 5), (15, 29)\}$.

This new approximation now meets the first three conditions. The fourth condition requires that

$$\begin{aligned} h_1(33) = \ulcorner 15 \urcorner \sim_R \ulcorner 29 \urcorner = h_2(5) \\ h_1(15) = \ulcorner 33 \urcorner \sim_R \ulcorner 5 \urcorner = h_2(29) \end{aligned}$$

which hold. The fifth condition requires that

$$\begin{aligned} h_1(34) = \ulcorner 21 \urcorner \sim_R \ulcorner 1 \urcorner = h_2(6) \\ h_1(16) = \ulcorner 21 \urcorner \sim_R \ulcorner 1 \urcorner = h_2(30) \end{aligned}$$

which we can satisfy by adding the pair $(21, 1)$ to R to give a third approximation: $\{(33, 5), (15, 29), (21, 1)\}$.

Since we have added another pair to R, we need to check the conditions still hold. We have not violated invertibility, so it is enough to check that the last two conditions hold for this additional pair. The fourth condition requires that

$$h_1(21) = \ulcorner 33 \urcorner \sim_R \ulcorner 5 \urcorner = h_2(1)$$

which holds. The fifth condition requires that

$$h_1(22) = \ulcorner 7 \urcorner \sim_R \ulcorner 7 \urcorner = h_2(2)$$

which we can satisfy by adding the pair $(7, 7)$ to R to give a fourth approximation $\{(33, 5), (15, 29), (21, 1), (7, 7)\}$.

Again, we have to check whether this additional pairing meets the last two conditions. The fourth condition requires that

$$h_1(7) = 4 \sim_R 4 = h_2(7)$$

which holds. The fifth condition requires that

$$h_1(8) = \ulcorner 21 \urcorner \sim_R \ulcorner 1 \urcorner = h_2(8)$$

which also holds. Therefore the choice $R = \{(33, 5), (15, 29), (21, 1), (7, 7)\}$ meets all the conditions for the equivalence.

It is tempting to add the pairs $(34, 5), (16, 30), (22, 2), (8, 8)$ to the relation since it seems that these addresses are related. However, doing so violates the fifth condition because, for instance $35 \notin \text{dom}(h_1)$.

Adding any other pairs will also violate one or other of the conditions.

(b) Assume that

$$(s_1, h_1) \sim (s_1', h_1') \tag{1}$$

$$\langle \texttt{newpair}, s_1, h_1\rangle \Downarrow_e \langle \ulcorner a \urcorner, s_2, h_2\rangle \tag{2}$$

$$\text{and} \qquad \langle \texttt{newpair}, s_1', h_1'\rangle \Downarrow_e \langle \ulcorner a' \urcorner, s_2', h_2'\rangle \tag{3}$$

By (1), we know there exists some R' such that

$$(s_1, h_1) \sim_{R'} (s_1', h_1'). \tag{4}$$

Since (2) must have been derived using the E.NEW rule, we know that

$$a \notin \text{dom}(h_1) \tag{5}$$

$$(a + 1) \notin \text{dom}(h_1) \tag{6}$$

$$s_2 = s_1 \tag{7}$$

$$h_2 = h_1[a \mapsto 0][a + 1 \mapsto 0] \tag{8}$$

Similarly, from (3) we know that

$$a' \notin \text{dom}(h_1') \tag{9}$$

$$(a' + 1) \notin \text{dom}(h_1') \tag{10}$$

$$s_2' = s_1' \tag{11}$$

$$h_2' = h_1'[a' \mapsto 0][a' + 1 \mapsto 0] \tag{12}$$

Choose $R = R' \cup \{(a, a')\}$. We have by construction that

$$(a, a') \in R. \tag{13}$$

We show the conditions for $(s_2, h_2) \sim_R (s_2', h_2')$ by using the conditions which establish that $(s_1, h_1) \sim_{R'} (s_1', h_1')$:

- R is an invertible mapping:
  - Suppose that $(a_1, a_2) \in R$ and $(a_1, a_3) \in R$.
    - If $a_1 \neq a$ then it must be that $(a_1, a_2) \in R'$ and $(a_1, a_3) \in R'$ and so $a_2 = a_3$ by (4).
    - Otherwise, $a_1 = a$, and either $a_2 = a_3 = a'$ or there is some a_0 with $(a, a_0) \in R'$. The latter case would mean that $h_1(a) = n$ or $h_1(a) = \ulcorner b \urcorner$ for some n or b, but this is impossible since $a \notin \mathrm{dom}(h_1)$ (5). Thus it must be that $a_2 = a_3$.
  - If $(a_2, a_1) \in R$ and $(a_3, a_1) \in R$ then $a_2 = a_3$ by a similar argument.
- $\mathrm{dom}(s_2) = \mathrm{dom}(s_2')$, since $\mathrm{dom}(s_2) = \mathrm{dom}(s_1) = \mathrm{dom}(s_1') = \mathrm{dom}(s_2')$ by (7), (4) and (11).
- The condition on variables holds since $s_2 = s_1$ (7) and $s_2' = s_1'$ (11) and $R' \subseteq R$ (by construction).
- The first condition on addresses holds for all $(a_1, a_1') \in R'$ since it must be that $h_2(a_1) = h_1(a_1)$ and $h_2'(a_1') = h_1'(a_1')$; we only require to show that it holds for (a, a'), which is the case since $h_2(a) = 0 = h_2'(a')$.
- Similarly, the second condition on addresses holds for all $(a_1, a_1') \in R'$ since it must be that $h_2(a_1 + 1) = h_1(a_1 + 1)$ and $h_2'(a_1' + 1) = h_1'(a_1' + 1)$; we only require to show that it holds for (a, a'), which is the case since $h_2(a+1) = 0 = h_2'(a'+1)$.

Thus we have established

$$(s_2, h_2) \sim_R (s_2', h_2'). \tag{14}$$

Together, (13) and (14) establish what we require to show.

Note that since we have to prove the *existence* of some R, we have to construct that R ourselves, and then show that this choice meets the conditions. A common mistake is not to explain how R is constructed, or how R' (from which it is constructed) is derived. Another common mistake is to give insufficient detail as to why that choice of R establishes the equivalence: it is necessary to establish that each of the points in the definition holds. A further mistake is to add the pair $(a+1, a'+1)$ to R, which prevents us from establishing that the fifth point of the definition of $\sim_R$ holds.

(c) We prove $\forall E.\, P(E)$ by induction on the structure of the expression E, where

$$\begin{aligned} P(E) \equiv\ & \forall s_1, h_1, s_1', h_1', s_2, h_2, s_2', h_2', v, v'. \\ & (s_1, h_1) \sim (s_1', h_1') \wedge \langle E, s_1, h_1 \rangle \Downarrow_e \langle v, s_2, h_2 \rangle \wedge \langle E, s_1', h_1' \rangle \Downarrow_e \langle v', s_2', h_2' \rangle \\ & \qquad \implies \exists R.\, v \sim_R v' \wedge (s_2, h_2) \sim_R (s_2', h_2') \end{aligned}$$

In otherwords, $P(E)$ means: whenever

$$\begin{gathered} (s_1, h_1) \sim (s_1', h_1') \\ \langle E, s_1, h_1 \rangle \Downarrow_e \langle v, s_2, h_2 \rangle \\ \langle E, s_1', h_1' \rangle \Downarrow_e \langle v', s_2', h_2' \rangle \end{gathered}$$

it is the case that there is some R such that

$$\begin{gathered} v \sim_R v' \\ (s_2, h_2) \sim_R (s_2', h_2'). \end{gathered}$$

Base Case: $E = x$ for some variable x. Assume that

$$\begin{gathered}(s_1, h_1) \sim (s'_1, h'_1)\\ \langle x, s_1, h_1\rangle \Downarrow_e \langle v, s_2, h_2\rangle\\ \langle x, s'_1, h'_1\rangle \Downarrow_e \langle v', s'_2, h'_2\rangle\end{gathered}$$

Both derivations must be by the E.VAR rule, and so

$$\begin{aligned} v &= s_1(x) \qquad & (s_2, h_2) &= (s_1, h_1)\\ v' &= s'_1(x) \qquad & (s'_2, h'_2) &= (s'_1, h'_1).\end{aligned}$$

Pick R to be the one such that $(s_1, h_1) \sim_R (s'_1, h'_1)$. It must be that

$$\begin{gathered} v = s_1(x) \sim_R s'_1(x) = v' \text{ and}\\ (s_2, h_2) = (s_1, h_1) \sim_R (s'_1, h'_1) = (s'_2, h'_2)\end{gathered}$$

as required.
A common mistake for this case is to assume that $v = v'$. In general, this is not true. For instance, it could be that $v = \ulcorner 33 \urcorner$ and $v' = \ulcorner 5 \urcorner$. What is required is that $v \sim_R v'$.
Base Case: $E = n$ for some number n. Assume that

$$\begin{gathered}(s_1, h_1) \sim (s'_1, h'_1)\\ \langle n, s_1, h_1\rangle \Downarrow_e \langle v, s_2, h_2\rangle\\ \langle n, s'_1, h'_1\rangle \Downarrow_e \langle v', s'_2, h'_2\rangle\end{gathered}$$

Both derivations must be by the E.NUM rule, and so

$$\begin{aligned} v &= n \qquad & (s_2, h_2) &= (s_1, h_1)\\ v' &= n \qquad & (s'_2, h'_2) &= (s'_1, h'_1).\end{aligned}$$

Pick R to be the one such that $(s_1, h_1) \sim_R (s'_1, h'_1)$. It must be that

$$\begin{gathered} v = n \sim_R n = v' \text{ and}\\ (s_2, h_2) = (s_1, h_1) \sim_R (s'_1, h'_1) = (s'_2, h'_2)\end{gathered}$$

as required.
Base Case: $E = \texttt{newpair}$. Assume that

$$\begin{gathered}(s_1, h_1) \sim (s'_1, h'_1)\\ \langle \texttt{newpair}, s_1, h_1\rangle \Downarrow_e \langle v, s_2, h_2\rangle\\ \langle \texttt{newpair}, s'_1, h'_1\rangle \Downarrow_e \langle v', s'_2, h'_2\rangle\end{gathered}$$

Since both derivations must be by the E.NEW rule, $v = \ulcorner a \urcorner$ and $v' = \ulcorner a' \urcorner$ for some a and a'. By part (b), it must be that, for some R, $(s_2, h_2) \sim_R (s'_2, h'_2)$ and $(a, a') \in R$. Hence also $v \sim_R v'$, as required.
Inductive Case: $E = E_1 + E_2$ for some E_1, E_2. For the inductive hypotheses, we assume $P(E_1)$ and $P(E_2)$. Namely (renaming bound variables):

$$\begin{aligned} P(E_1) \equiv\ & \forall s_1, h_1, s'_1, h'_1, s_3, h_3, s'_3, h'_3, v_1, v'_1.\\ & \quad (s_1, h_1) \sim (s'_1, h'_1) \wedge \langle E_1, s_1, h_1\rangle \Downarrow_e \langle v_1, s_3, h_3\rangle \wedge \langle E_1, s'_1, h'_1\rangle \Downarrow_e \langle v'_1, s'_3, h'_3\rangle\\ & \qquad\qquad\qquad\qquad\qquad \implies \exists R_1.\, v_1 \sim_{R_1} v'_1 \wedge (s_3, h_3) \sim_{R_1} (s'_3, h'_3)\end{aligned}$$

$$P(E_2) \equiv \forall s_3, h_3, s_3', h_3', s_2, h_2, s_2', h_2', v_2, v_2'.$$
$$(s_3, h_3) \sim (s_3', h_3') \wedge \langle E_2, s_3, h_3 \rangle \Downarrow_e \langle v_2, s_2, h_2 \rangle \wedge \langle E_2, s_3', h_3' \rangle \Downarrow_e \langle v_2', s_2', h_2' \rangle$$
$$\implies \exists R_2.\, v_2 \sim_{R_2} v_2' \wedge (s_2, h_2) \sim_{R_2} (s_2', h_2')$$

Assume that

$$(s_1, h_1) \sim (s_1', h_1')$$
$$\langle E_1 + E_2, s_1, h_1 \rangle \Downarrow_e \langle v, s_2, h_2 \rangle$$
$$\langle E_1 + E_2, s_1', h_1' \rangle \Downarrow_e \langle v', s_2', h_2' \rangle$$

Since both derivations must be by the E.ADD rule, we have

$$v = n_3$$
$$\langle E_1, s_1, h_1 \rangle \Downarrow_e \langle n_1, s_3, h_3 \rangle$$
$$\langle E_2, s_3, h_3 \rangle \Downarrow_e \langle n_2, s_2, h_2 \rangle$$
$$n_3 = n_1 \underline{+} n_2$$

for some n_1, n_2, n_3, s_3, h_3, and

$$v' = n_3'$$
$$\langle E_1, s_1', h_1' \rangle \Downarrow_e \langle n_1', s_3', h_3' \rangle$$
$$\langle E_2, s_3', h_3' \rangle \Downarrow_e \langle n_2', s_2', h_2' \rangle$$
$$n_3' = n_1' \underline{+} n_2'$$

for some $n_1', n_2', n_3', s_3', h_3'$.
We have

$$(s_1, h_1) \sim (s_1', h_1') \wedge \langle E_1, s_1, h_1 \rangle \Downarrow_e \langle n_1, s_3, h_3 \rangle \wedge \langle E_1, s_1', h_1' \rangle \Downarrow_e \langle n_1', s_3', h_3' \rangle$$

so by the inductive hypothesis $P(E_1)$, it must be that, for some R_1,

$$n_1 \sim_{R_1} n_1' \quad \text{and} \quad (s_3, h_3) \sim_{R_1} (s_3', h_3')$$

and hence

$$n_1 = n_1' \quad \text{and} \quad (s_3, h_3) \sim (s_3', h_3')$$

Consequently, we have

$$(s_3, h_3) \sim (s_3', h_3') \wedge \langle E_2, s_3, h_3 \rangle \Downarrow_e \langle n_2, s_2, h_2 \rangle \wedge \langle E_2, s_3', h_3' \rangle \Downarrow_e \langle n_2', s_2', h_2' \rangle$$

so by the inductive hypothesis $P(E_2)$, it must be that, for some R_2,

$$n_2 \sim_{R_2} n_2' \quad \text{and} \quad (s_2, h_2) \sim_{R_2} (s_2', h_2')$$

and hence $n_1 = n_1'$.
Let $R = R_2$. It must be that $n_3 = n_1 \underline{+} n_2 = n_1' + n_2' = n_3'$ and so

$$v \sim_R v'.$$

Furthermore,

$$(s_2, h_2) \sim_R (s_2', h_2')$$

as required.

The most common mistake with this case was in not realising that the inductive hypothesis must be an implication (since what we are proving is an implication). In order to use an implication, we prove the antecedent and can thereby deduce the consequent. The antecedent of $P(E_1)$ is established from the assumptions. However, the antecedent of $P(E_2)$ must be established using the consequent of $P(E_1)$.

In more concrete terms, we know the initial states are related, which allows us to use $P(E_1)$. This establishes that the intermediate states are also related (they may have changed in evaluating E_1) which allows us to use $P(E_2)$.

(As a general note, having a proof of an implication $A \implies B$ is very much like having a function that, given a proof of A, returns a proof of B. In fact, there is a very strong relationship between proofs and programs and between propositions and types called the Curry-Howard correspondence.)

Inductive Case: $E = E_1.\texttt{fst}$ for some E_1. For the inductive hypothesis, we assume $P(E_1)$.

Assume that

$$(s_1, h_1) \sim (s_1', h_1')$$
$$\langle E_1.\texttt{fst}, s_1, h_1\rangle \Downarrow_e \langle v, s_2, h_2\rangle$$
$$\langle E_1.\texttt{fst}, s_1', h_1'\rangle \Downarrow_e \langle v', s_2', h_2'\rangle$$

Since both derivations must be by the E.FST rule,

$$v = h_1(a) \qquad \langle E_1, s_1, h_1\rangle \Downarrow_e \langle \ulcorner a \urcorner, s_2, h_2\rangle$$

for some $a \in \mathrm{dom}(h_1)$, and

$$v' = h_1'(a') \qquad \langle E_1, s_1', h_1'\rangle \Downarrow_e \langle \ulcorner a' \urcorner, s_2', h_2'\rangle$$

for some $a' \in \mathrm{dom}(h_2)$. By the inductive hypothesis, it must be that, for some R, $\ulcorner a \urcorner \sim_R \ulcorner a' \urcorner$ (and so $(a, a') \in R$) and $(s_2, h_2) \sim_R (s_2', h_2')$. Now, it must be that $v = h_2(a) \sim_R h_2'(a') = v'$ (using the fourth condition for $(s_2, h_2) \sim_R (s_2', h_2')$), as required.

Inductive Case: $E = E_1.\texttt{snd}$ for some E_1. For the inductive hypothesis, we assume $P(E_1)$.

Assume that

$$(s_1, h_1) \sim (s_1', h_1')$$
$$\langle E_1.\texttt{snd}, s_1, h_1\rangle \Downarrow_e \langle v, s_2, h_2\rangle$$
$$\langle E_1.\texttt{snd}, s_1', h_1'\rangle \Downarrow_e \langle v', s_2', h_2'\rangle$$

Since both derivations must be by the E.SND rule,

$$v = h_1(a+1) \qquad \langle E_1, s_1, h_1\rangle \Downarrow_e \langle \ulcorner a \urcorner, s_2, h_2\rangle$$

for some a with $a + 1 \in \mathrm{dom}(h_1)$, and

$$v' = h_1'(a'+1) \qquad \langle E_1, s_1', h_1'\rangle \Downarrow_e \langle \ulcorner a' \urcorner, s_2', h_2'\rangle$$

for some a' with $a' + 1 \in \mathrm{dom}(h_2)$. By the inductive hypothesis, it must be that, for some R, $\ulcorner a \urcorner \sim_R \ulcorner a' \urcorner$ (and so $(a, a') \in R$) and $(s_2, h_2) \sim_R (s_2', h_2')$. Now, it must be that $v = h_2(a+1) \sim_R h'_2(a'+1) = v'$ (by the fifth condition for $(s_2, h_2) \sim_R (s'_2, h'_2)$), as required.

Although this case is very similar to the E_1.`fst` case, it is a mistake to say that they are the same: the sub-expressions evaluate in the same way, but a different condition is used to establish the relationship between the resulting values.