UNIVERSITY OF BRISTOL

Winter 2024 Examination Period

SCHOOL OF COMPUTER SCIENCE

Second Year Examination for the Degrees
of
Bachelor of Science
Master of Engineering

COMS20007W
Programming Languages and Computation

TIME ALLOWED:
3 Hours

Answers to COMS20007W: Programming Languages and Computation

Intended Learning Outcomes:

Q1. This question is about syntax.

*(a) Consider the following grammar, whose start symbol is S.

$$\begin{array}{lcl} S & \longrightarrow & \text{item} \mid \text{item } T \text{ and item} \\ T & \longrightarrow & \text{, item } T \mid \epsilon \end{array}$$

For each of the following, state if it is true or false.

i. $S \rightarrow^* \epsilon$

ii. $S \rightarrow^*$ item and item

iii. $S \rightarrow$ item

iv. $T \rightarrow^*$ item, item

v. $T \rightarrow \epsilon$

[5 marks]

Solution:

i. false

ii. true

iii. true

iv. false

v. true

*(b) Consider the following context-free grammar:

$$S \longrightarrow aSaS \mid bS \mid \epsilon$$

Give derivations for each of the following strings.

i. aa

ii. $abab$

iii. bb

iv. baa

v. aab

[10 marks]

Solution:

i. $S \rightarrow aSaS \rightarrow aaS \rightarrow aa$

ii. $S \rightarrow aSaS \rightarrow abSaS \rightarrow abaS \rightarrow ababS \rightarrow abab$

iii. $S \rightarrow bS \rightarrow bbS \rightarrow bb$

iv. $S \rightarrow bS \rightarrow baSaS \rightarrow baaS \rightarrow baa$

v. $S \rightarrow aSaS \rightarrow aaS \rightarrow aabS \rightarrow aab$

*(c) Consider the following grammar over 7 terminal symbols:

$$\$ \quad \text{key} \quad \text{val} \quad : \quad \{ \quad \} \quad ;$$

with start symbol S and whose rules are:

$$\begin{array}{rcl} S & \longrightarrow & R\ \$ \\ R & \longrightarrow & \{\ F\ \} \\ F & \longrightarrow & K\ G \\ G & \longrightarrow & ;\ K\ G \mid \epsilon \\ K & \longrightarrow & \text{key} : D \\ D & \longrightarrow & \text{val} \mid R \end{array}$$

The nullable, first and follow maps for this grammar are:

Nonterminal	Nullable	First	Follow
S		{	
R		{	$;
F		key	}
G	✓	;	}
K		key	; }
D		val {	; }

i. Copy the following table into your answer book (make sure the cells are quite large) and use the nullable, first and follow maps above to complete the entries so that it forms a parse table for the grammar.

Nonterminal	$	{	}	;	key	:	val
S							
R							
F							
G							
K							
D							

ii. Is this grammar LL(1)?

[5 marks]

Solution:

i. The parse table is as follows:

NT	$	{	}	;	key	:	val
S		$S \longrightarrow R\$$					
R		$R \longrightarrow \{F\}$					
F					$F \longrightarrow KG$		
G			$G \longrightarrow \epsilon$	$G \longrightarrow ;KG$			
K					$K \longrightarrow \text{key} : D$		
D		$D \longrightarrow R$					$D \longrightarrow \text{val}$

(cont.)

ii. Yes.

** (d) Recall that we write $|u|$ for the length of a given word u. For each of the following languages over the alphabet $\{a, b, \#\}$, design a grammar to express it.

i. $\{w \mid w \in \{a,b\}^*$ and no a occurs after (further to the right of) any b in $w\}$

ii. $\{u\#v \mid u, v \in \{a,b\}^*$ and |u| = |v|$\}$

iii. $\{u\#v\#w \mid u, v, w \in \{a,b\}^*$ and u is the reverse of w $\}$

[6 marks]

Solution:

i.

$$\begin{array}{rcl} S & \longrightarrow & AB \\ A & \longrightarrow & aA \mid \epsilon \\ B & \longrightarrow & bB \mid \epsilon \end{array}$$

ii.

$$\begin{array}{rcl} S & \longrightarrow & XSX \mid \# \\ X & \longrightarrow & a \mid b \end{array}$$

iii.

$$\begin{array}{rcl} S & \longrightarrow & aSa \mid bSb \mid \#T\# \\ T & \longrightarrow & aT \mid bT \mid \epsilon \end{array}$$

** (e) Give an LL(1) grammar equivalent to the following grammar (i.e. expressing the same language).

$$T \longrightarrow \text{'tyvar} \mid (\ T\ ) \mid T\ \text{tycon} \mid T \Rightarrow T \mid T,\ T$$

The 7 terminal symbols of this grammar are:

$$\text{'} \quad \text{tyvar} \quad (\quad) \quad \Rightarrow \quad \text{tycon} \quad ,$$

[4 marks]

Solution:

$$\begin{array}{rcl} T & \longrightarrow & V\ R \\ R & \longrightarrow & \Rightarrow V\ R \mid , V\ R \mid \text{tycon}\ R\epsilon \\ V & \longrightarrow & \text{'tyvar} \mid (\ T\ ) \end{array}$$

*** (f) Consider the following language over $\{0, 1\}$:

$\{w \mid$ any two occurrences of 1 in w are separated by an even number of letters$\}$

So, for example, 1000010 is in this language, because the only occurrences of 1 are separated by 4 letters. However, 1101 is not, because the second and third occurrences of 1 are only separated by 1 letter.

i. Construct a CFG to express this language.

ii. Justify informally why your CFG is correct.

[6 marks]

Solution:

i.

$$\begin{aligned} S &\longrightarrow Z \mid Z1Z \mid Z1E1Z \\ Z &\longrightarrow 0Z \mid \epsilon \\ E &\longrightarrow 00E \mid \epsilon \end{aligned}$$

ii. This works because some thought reveals that a word of the form required by the language may contain at most two occurrences of 1. Symbol Z of this grammar derives all possible words consisting only of zeroes, and S has three cases, corresponding to exactly 0 occurrences of 1, exactly one occurrence or exactly two occurrences. In the latter case, the number of 0s between them must be even (nonterminal E).

***(g) Argue that no word of shape $uauvbv$, with $u, v \in \{a, b\}^*$, can be written as ww for some choice of $w \in \{a, b\}^*$.

[4 marks]

Solution: Let $uaubvb$ be a word as described. Then suppose, for the purpose of contradiction, that there is some w such that $uauvbv = ww$. Suppose the length of u is n and the length of v is m. Then the total length of this word is $n + 1 + n + m + 1 + m = 2(n + m + 1)$. Therefore, it is even and so w has length $n + m + 1$. Therefore, the letter at position $n + 1$ in w (indexing from 1), is a. The second w begins after $n + m + 1$ letters. However, the letter b occurs in $uauvbv$ at position $n + 1 + n + m + 1 = (n + 1 + m) + (n + 1)$, so the letter at position $n + 1$ in w is b. However, $a \neq b$.

Q2. This question is about semantics.

*(a) Consider the following code fragments from the While language:

i. while $0 \leq x$ do $x \leftarrow (x - 1)$

ii. $x * (y - z)$

iii. $12 * x \leq y + z$

iv. if $y \leq 4$ then $x \leftarrow (x - 1)$ else $x \leftarrow y$

For each of these fragments:

- Indicate whether it is an arithmetic expression, Boolean expression, or statement;
- If it is an expression, calculate its value in the state $[x \mapsto -1, y \mapsto 1]$. Your answer should make explicit reference to the denotation function in your answer.
- If it is a statement, determine the final state when the statement is executed in the initial state $[x \mapsto -2, y \mapsto 3, z \mapsto 4]$. You should justify your answer by providing the associated trace.

[10 marks]

Solution:

i. Statement. The final state is $[x \mapsto -2, y \mapsto 3, z \mapsto 4]$, with the trace:

$$\begin{aligned}&\langle \text{while } 0 \leq x \text{ do } x \leftarrow (x - 1), [x \mapsto -2, y \mapsto 3, z \mapsto 4]\rangle \\ &\quad \rightarrow [x \mapsto -2, y \mapsto 3, z \mapsto 4]\end{aligned}$$

ii. Arithmetic expression. This part depends on the convention that states, by default, map variables to zero.

$$[\![x * (y - z)]\!]_A([x \mapsto -1, y \mapsto 1]) = -1$$

iii. Boolean expression. This part depends on the convention that states, by default, map variables to zero.

$$[\![12 * x \leq y + z]\!]_B([x \mapsto -1, y \mapsto 1]) = \top$$

Other representations of truth, e.g. 1 or true are also to be accepted.

iv. Statement. The final state is $[x \mapsto -3, y \mapsto 3, z \mapsto 4]$, with the trace:

$$\begin{aligned}&\langle \text{if } y \leq 4 \text{ then } x \leftarrow (x - 1) \text{ else } x \leftarrow y, [x \mapsto -2, y \mapsto 3, z \mapsto 4]\rangle \\ &\quad \rightarrow \langle x \leftarrow (x - 1), [x \mapsto -2, y \mapsto 3, z \mapsto 4]\rangle \\ &\quad \rightarrow [x \mapsto -3, y \mapsto 3, z \mapsto 4]\end{aligned}$$

**(b) In this part, we will consider an extension of the While language with an additional construct - par$(S_1;\ S_2)$. The grammar defining the statements of this extended language

is thus given as follows:

$$S \longrightarrow \mathsf{par}(S;\ S) \mid \mathsf{skip} \mid x \leftarrow A \mid S; S \mid \mathsf{if}\ B\ \mathsf{then}\ S\ \mathsf{else}\ S \mid \mathsf{while}\ B\ \mathsf{do}\ S$$

where the definition of arithmetic and Boolean expressions are unchanged.
The operational semantics of this statement is given by the following rule:

$$\frac{\langle S_1,\, \sigma_1\rangle \to \sigma_2}{\langle \mathsf{par}(S_1;\ S_2),\, \sigma_1\rangle \to \langle S_2,\, \sigma_2\rangle} \qquad \frac{\langle S_2,\, \sigma_1\rangle \to \sigma_2}{\langle \mathsf{par}(S_1;\ S_2),\, \sigma_1\rangle \to \langle S_1,\, \sigma_2\rangle}$$

$$\frac{\langle S_1,\, \sigma_1\rangle \to \langle S_1',\, \sigma_2\rangle}{\langle \mathsf{par}(S_1;\ S_2),\, \sigma_1\rangle \to \langle \mathsf{par}(S_1';\ S_2),\, \sigma_2\rangle} \qquad \frac{\langle S_2,\, \sigma_1\rangle \to \langle S_2',\, \sigma_2\rangle}{\langle \mathsf{par}(S_1;\ S_2),\, \sigma_1\rangle \to \langle \mathsf{par}(S_1;\ S_2'),\, \sigma_2\rangle}$$

where all other rules still apply.
Here is an example trace using this rule:

$$\begin{aligned}&\langle \mathsf{par}(x \leftarrow x + y;\ y \leftarrow y * 2),\, [y \mapsto 1]\rangle\\ &\quad \to \langle x \leftarrow x + y,\, [y \mapsto 2]\rangle\\ &\quad \to [x \mapsto 2,\, y \mapsto 2]\end{aligned}$$

i. Find a $\sigma \in \mathsf{State}$ such that $\langle \mathsf{par}(x \leftarrow x + y;\ y \leftarrow y * 2),\, [y \mapsto 1]\rangle \to^* \sigma$ and where σ is distinct from the final state given above. Justify your answer by giving a trace.

ii. Prove that, if $\langle S_1,\, \sigma_1\rangle \to^* \sigma_2$, then $\langle \mathsf{par}(S_1;\ S_2),\, \sigma\rangle \to^* \langle S_2,\, \sigma_2\rangle$ for any states $\sigma_1,\, \sigma_2 \in \mathsf{State}$ and any statements $S_1,\, S_2 \in \mathcal{S}$ by induction over the length of the trace.

Solution:

i. The state is $[x \mapsto 1,\, y \mapsto 2]$ with the trace:

$$\begin{aligned}&\langle \mathsf{par}(x \leftarrow x + y;\ y \leftarrow y * 2),\, [y \mapsto 1]\rangle\\ &\quad \to \langle y \leftarrow y * 2,\, [x \mapsto 1,\, y \mapsto 1]\rangle\\ &\quad \to [x \mapsto 1,\, y \mapsto 2]\end{aligned}$$

ii. We shall prove that, if $\langle S_1,\, \sigma_1\rangle \to^n \sigma_2$, then $\langle \mathsf{par}(S_1;\ S_2),\, \sigma_1\rangle \to^* \langle S_2,\, \sigma_2\rangle$ for any states $\sigma_1,\, \sigma_2 \in \mathsf{State}$ and any statements $S_1,\, S_2 \in \mathcal{S}$ by induction over n.

- When $n = 0$, the assumption is absurd.
- Let us assume that if $\langle S_1,\, \sigma_1\rangle \to^n \sigma_2$, then $\langle \mathsf{par}(S_1;\ S_2),\, \sigma_1\rangle \to^* \langle S_2,\, \sigma_2\rangle$ for any states $\sigma_1,\, \sigma_2 \in \mathsf{State}$ and any statements $S_1,\, S_2 \in \mathcal{S}$. Now suppose that $\langle S_1,\, \sigma_1\rangle \to^{n+1} \sigma_2$. There are two cases to consider based on whether $n = 0$ or $n > 0$:
  - If $n = 0$, then $\langle S_1,\, \sigma_1\rangle \to \sigma_2$. In which case, $\langle \mathsf{par}(S_1;\ S_2),\, \sigma_1\rangle \to \langle S_2,\, \sigma_2\rangle$ directly.
  - Otherwise, $\langle S_1,\, \sigma_1\rangle \to \langle S_1',\, \sigma_1'\rangle$ for some S_1' and σ_1' and $\langle S_1',\, \sigma_1'\rangle \to^n \sigma_2$. By induction, $\langle \mathsf{par}(S_1';\ S_2),\, \sigma_1'\rangle \to^* \langle S_2,\, \sigma_2\rangle$. Therefore, $\langle \mathsf{par}(S_1;\ S_2),\, \sigma_1\rangle \to^* \langle S_2,\, \sigma_2\rangle$ as required.

(cont.)

[10 marks]

***(c) Let P be the following While program:

```
while !(a = b) do
  if b ≤ a
    then a ← a − b
    else b ← b − a
```

i. Compute the final state of the program from the initial state $[a \mapsto 15, b \mapsto 6]$. You do not need to provide a trace.

ii. Find an initial state $\sigma \in$ State for which there does *not* exist a final state $\sigma' \in$ State such that $\langle P, \sigma \rangle \rightarrow^* \sigma'$. You should briefly justify your answer.

iii. Find a loop invariant I from which you can conclude the Hoare triple:

$$\{I\}\ P\ \{I \mathbin{\&\&} a = b\}$$

and which demonstrates that the program does not terminate from the initial state given in part ii. You should briefly justify your answer.

Solution:

i. The final state is $[a \mapsto 3, b \mapsto 3]$.

ii. One variable must be less than or equal to zero and they cannot be equal. For example, the initial state $[a \mapsto 1, b \mapsto -1]$ will lead to non-termination. After one iteration of the loop, we will arrive at the state $[a \mapsto 2, b \mapsto -1]$, subsequently the state $[a \mapsto 3, b \mapsto -1]$ will be reached, and so on...

iii. The invariant must be satisfied by the given example in ii, and be such that $I \mathbin{\&\&} a = b$ is unsatisfiable. A possible invariant is $(a \leq 0 \parallel b \leq 0) \mathbin{\&\&} a \neq b$, but more specific invariant may suffice.

[10 marks]

Q3. This question is about computability.

*(a) Show that the function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ defined by

$$f(x) \begin{cases} \simeq x * 2 & \text{if } 0 \leq x \leq 10 \\ \simeq x * 3 & \text{if } 11 \leq x \leq 100 \text{ and } x \text{ is even} \\ \simeq x & \text{if } 11 \leq x \leq 100 \text{ and } x \text{ is odd} \\ \uparrow & \text{otherwise} \end{cases}$$

is computable. *[5 marks]*

Solution: The function is computed by the following code with respect to x.

```
if (0 <= x && x <= 10) { x := x * 2 };
else if (11 <= x && x <= 100) {
  // Determine the parity of x wrt 2
  r := x;
  // Invariant: r = r_0 mod 2 && r >= 0
  while (r >= 2) do { r := r - 2; }
  if (r == 0) then { x := x * 3 } else skip;
  r := 0     // Do not forget to zero out aux variables!
}
else { while (true) do skip }
```

Award 1 mark for correctly stating the input/output variable; 2 marks for a mostly correct program; 1 mark for the infinite loop when the output is undefined; and 1 mark for setting all auxiliary variables to zero at the end.

*(b) State whether each of the following statements is true or false.

(i) The set of numbers that are greater than 42 is decidable.

(ii) Every surjection has an inverse.

(iii) There are bijections that are not injections.

(iv) The set $\{10^x \mid x = 0, 1, 2, \ldots\}$ is decidable.

(v) If a function has an inverse then it must be computable.

[5 marks]

Solution: (i) True (ii) False (iii) False (iv) True (v) False

**(c) State whether each of the following statements is true or false.

(i) If a predicate is semi-decidable then it is also decidable.

(ii) The Church-Turing thesis implies that no computer is more powerful than the While programming language.

(iii) The set of numbers $\{2^{2^{2^{2^n}}} \mid n = 0, 1, 2, \ldots\}$ is undecidable.

(cont.)

(iv) It is possible to implement an interpreter for Java in the While programming language.

(v) A function is computable if and only if it is partial and injective.

[5 marks]

Solution: (i) False (ii) True (iii) False (iv) True (v) False

** (d) Consider the following predicate.

$$U = \{\ulcorner S \urcorner \mid [\![S]\!]_{\mathtt{x}}(k) \text{ is a multiple of } k, \text{ for all } 0 \leq k \leq 10^5\}$$

Is U semi-decidable? Why, or why not? *[5 marks]*

Solution: It is semi-decidable. Simulate the running of S on all $0 \leq k \leq 10^5$. When each simulation terminates, test whether the output is a multiple of its input (this is easily computable). If all simulations succeed with a multiple, return 1. Otherwise return 0. Because this is a semi-decision procedure the simulations need not terminate, in which case nothing is returned.

Award 2 marks for demonstrating understanding of semi-decidability, and 3 further marks for a fully correct argument.

*** (e) Show that the predicate U defined above is undecidable. *[5 marks]*

Solution: By Rice's Theorem. Award 2 mark for stating the name of the theorem, 2 marks for demonstrating that it applies (there is an S that is in the predicate and one that is not), and a further 1 mark for a fully correct argument. Alternative solutions by reduction are also acceptable.

*** (f) Show that the predicate

$$V = \{\langle \ulcorner S \urcorner, \ulcorner T \urcorner \rangle \mid \text{ and for all } k \in \mathbb{N} \text{ we have } [\![S]\!]_{\mathtt{x}}(k) \simeq 2 \times [\![T]\!]_{\mathtt{x}}(k)\}$$

is undecidable. *[5 marks]*

Solution: By reduction from HALT. Define a code transformation $F : \textbf{Stmt} \times \mathbb{N} \to \textbf{Stmt} \times \textbf{Stmt}$ by

$$F(S, n) = (\texttt{x := 2 * x}, (\texttt{y := x; x := n; S; x := y}))$$

Clearly the reflection of F is computable, and

$$\langle \ulcorner S \urcorner, n \rangle \in \text{HALT} \iff F(S, n) \in V$$

As the former is not decidable, the latter is not decidable either.

Award 1 mark for recognising that a reduction is the most appropriate proof method; 3 marks for constructing the reduction and arguing that it is computable; and 1 mark for correctly stating the reduction property in this particular instance.

Reminder of Important Definitions

Grammars

A *Context Free Grammar (CFG)* consists of four components:

- An alphabet of *terminal* symbols.
- A finite, non-empty set of *non-terminal* symbols, disjoint from the terminals.
- A finite set of *production rules*.
- A designated non-terminal called the *start symbol*.

A *sentential form*, usually α, β, γ and so on, is just a finite sequence of terminals and nonterminals.

The sentential form α can make a *derivation step* to β, written $\alpha \to \beta$, just if:

- α has shape $\gamma_1 X \gamma_2$ and β has shape $\gamma_1 \delta \gamma_2$
- and there is a production rule $X ::= \delta$ in the grammar

A *derivation sequence* is a non-empty sequence of sentential forms α_1, α_2, ... α_{k-1}, α_k in which consecutive elements of the sequence are derivation steps:

$$\alpha_1 \to \alpha_2 \to \cdots \to \alpha_{k-1} \to \alpha_k$$

A sentential form β is *derivable* from α, written $\alpha \to^* \beta$ just if there is a derivation sequence starting with α and ending with β.

We say that a word w is in the *language of a grammar* G with start symbol S, and write $w \in L(G)$ just if $S \to^* w$.

Nullable

On nonterminals:

$$\text{Nullable}(X) \text{ iff } X \to^* \epsilon$$

On sentential forms:

$$\text{Nullable}_s(\alpha) = \begin{cases} \text{true} & \text{if } \alpha = \epsilon \\ \text{false} & \text{if } \alpha \text{ is of shape } a\beta \\ \text{Nullable}(X) \wedge \text{Nullable}_s(\beta) & \text{if } \alpha \text{ is of shape } X\beta \end{cases}$$

(cont.)

First

On nonterminals:

$$\mathsf{First}(X) = \{a \mid \exists \beta.\, X \to^* a\beta\}$$

On sentential forms:

$$\mathsf{First}_s(\alpha) = \begin{cases} \emptyset & \text{if } \alpha = \epsilon \\ \{a\} & \text{if } \alpha \text{ is of shape } a\beta \\ \mathsf{First}(X) & \text{if } \alpha \text{ is of shape } X\beta \text{ and } \neg\mathsf{Nullable}(X) \\ \mathsf{First}(X) \cup \mathsf{First}_s(\beta) & \text{if } \alpha \text{ is of shape } X\beta \text{ and } \mathsf{Nullable}(X) \end{cases}$$

Follow

On nonterminals:

$$\mathsf{Follow}(X) = \{a \mid \exists \alpha\beta.\, S \to^* \alpha X a \beta\}$$

Parse Tables and LL(1)

We define the *parsing table*, usually T, for a given grammar as a 2d array indexed by pairs of a nonterminal and a terminal. Each entry $T[X, a]$ is a set of production rules from the grammar, such that some rule $X \longrightarrow \beta$ is in the set $T[X, a]$ just if, either:

1. $a \in \mathsf{First}_s(\beta)$
2. or, $\mathsf{Nullable}_s(\beta)$ and $a \in \mathsf{Follow}(X)$

A grammar whose parsing table contains at most one rule in each cell is called *LL(1)*.

Abstract Syntax of Arithmetic Expressions

An *arithmetic expression* is a tree described by the following grammar:

$$A ::= n \mid x \mid A + A \mid A - A \mid A * A$$

where n ranges over integer literals, and x ranges over variables. Parentheses are used to resolve ambiguity and to indicate the structure of the tree. We write $\mathcal{A}$ for the set of arithmetic expressions.

Abstract Syntax of Boolean Expressions

A *Boolean expression* is a tree described by the following grammar.

$$B ::= \mathsf{false} \mid \mathsf{true} \mid !B \mid B \;\&\&\; B \mid B \parallel B \mid A = A \mid A \leq A$$

Parentheses are used to resolve ambiguity and to indicate the structure of the tree. We write $\mathcal{B}$ for the set of Boolean expressions.

Abstract Syntax of Statements

A *statement* is a tree described by the following grammar:

$$S ::= \text{skip} \mid x \leftarrow A \mid S; S \mid \text{if } B \text{ then } S \text{ else } S \mid \text{while } B \text{ do } S$$

Braces "$\{\cdots\}$" are used to resolve ambiguity and to indicate the structure of the tree. We write $\mathcal{S}$ for the set of statements.

States

A *state* is a total function from the set State $=$ Var $\to \mathbb{Z}$, where Var is the set of variables. We write $[x_1 \mapsto v_1, x_2 \mapsto v_2, \ldots, x_n \mapsto v_n]$ to indicate the state that maps the variable $x_i \in$ Var to the value $v_i \in \mathbb{Z}$ for all $i \leq n$. By convention, any variable not explicitly mentioned by a given state σ is assigned the value 0.

For a given state $\sigma \in$ State, we write $\sigma[x \mapsto v]$ for some variable $x \in$ Var and $v \in \mathbb{Z}$ to denote the state that maps the variable x to v and any other variable y to the value $\sigma(y)$.

Semantics of Arithmetic Expressions

The denotation function for arithmetic expressions $[\![\cdot]\!]_{\mathcal{A}} \in \mathcal{A} \to (\text{State} \to \mathbb{Z})$, which is defined by recursion in Figure 1. We say that two arithmetic expressions $e_1, e_2 \in \mathcal{A}$ are *semantically equivalent* if, and only if, $[\![e_1]\!]_{\mathcal{A}}(\sigma) = [\![e_2]\!]_{\mathcal{A}}(\sigma)$ for all states $\sigma \in$ State.

$$\begin{aligned}
[\![n]\!]_{\mathcal{A}}(\sigma) &= n \\
[\![x]\!]_{\mathcal{A}}(\sigma) &= \sigma(x) \\
[\![e_1 + e_2]\!]_{\mathcal{A}}(\sigma) &= [\![e_1]\!]_{\mathcal{A}}(\sigma) + [\![e_2]\!]_{\mathcal{A}}(\sigma) \\
[\![e_1 - e_2]\!]_{\mathcal{A}}(\sigma) &= [\![e_1]\!]_{\mathcal{A}}(\sigma) - [\![e_2]\!]_{\mathcal{A}}(\sigma) \\
[\![e_1 * e_2]\!]_{\mathcal{A}}(\sigma) &= [\![e_1]\!]_{\mathcal{A}}(\sigma) \cdot [\![e_2]\!]_{\mathcal{A}}(\sigma)
\end{aligned}$$

Figure 1: Definition of the denotational semantics of arithmetic expressions.

Semantics of Boolean Expressions

The denotation function for Boolean expressions $[\![\cdot]\!]_{\mathcal{B}} \in \mathcal{B} \to (\text{State} \to \mathbb{B})$ is defined by recursion in Figure 2. We say that two Boolean expressions $e_1, e_2 \in \mathcal{B}$ are *semantically equivalent* if, and only if, $[\![e_1]\!]_{\mathcal{B}}(\sigma) = [\![e_2]\!]_{\mathcal{B}}(\sigma)$ for all states $\sigma \in$ State.

(cont.)

$$
\begin{aligned}
[\![\mathsf{false}]\!]_{\mathcal{B}}(\sigma) &= \bot \\
[\![\mathsf{true}]\!]_{\mathcal{B}}(\sigma) &= \top \\
[\![!e]\!]_{\mathcal{B}}(\sigma) &= \neg[\![e]\!]_{\mathcal{B}}(\sigma) \\
[\![e_1 \;\&\&\; e_2]\!]_{\mathcal{B}}(\sigma) &= [\![e_1]\!]_{\mathcal{B}}(\sigma) \wedge [\![e_2]\!]_{\mathcal{B}}(\sigma) \\
[\![e_1 \parallel e_2]\!]_{\mathcal{B}}(\sigma) &= [\![e_1]\!]_{\mathcal{B}}(\sigma) \vee [\![e_2]\!]_{\mathcal{B}}(\sigma) \\
[\![e_1 = e_2]\!]_{\mathcal{B}}(\sigma) &= [\![e_1]\!]_{\mathcal{A}}(\sigma) = [\![e_2]\!]_{\mathcal{A}}(\sigma) \\
[\![e_1 \leq e_2]\!]_{\mathcal{B}}(\sigma) &= [\![e_1]\!]_{\mathcal{A}}(\sigma) \leq [\![e_2]\!]_{\mathcal{A}}(\sigma)
\end{aligned}
$$

Figure 2: Definition of the denotational semantics of Boolean expressions.

Semantics of Statements

The small-step operational semantics relation $\rightarrow \subseteq \mathcal{C} \times \mathcal{C}$ is defined by the rules in Figure 3 where the set of configurations $\mathcal{C}$ is $(\mathcal{S} \times \mathsf{State}) \cup \mathsf{State}$.

$$\frac{}{\langle \mathsf{skip},\, \sigma\rangle \rightarrow \sigma} \qquad \frac{}{\langle x \leftarrow e,\, \sigma\rangle \rightarrow \sigma[x \mapsto [\![e]\!]_{\mathcal{A}}(\sigma)]}$$

$$\frac{\langle S_1,\, \sigma_1\rangle \rightarrow \langle S_1',\, \sigma_2\rangle}{\langle S_1; S_2,\, \sigma_1\rangle \rightarrow \langle S_1';\; S_2,\, \sigma_2\rangle} \qquad \frac{\langle S_1,\, \sigma_1\rangle \rightarrow \sigma_2}{\langle S_1; S_2,\, \sigma_1\rangle \rightarrow \langle S_2,\, \sigma_2\rangle}$$

$$\frac{}{\langle \mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2,\, \sigma\rangle \rightarrow \langle S_1,\, \sigma\rangle}\ [\![e]\!]_{\mathcal{B}}(\sigma) = \top$$

$$\frac{}{\langle \mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2,\, \sigma\rangle \rightarrow \langle S_2,\, \sigma\rangle}\ [\![e]\!]_{\mathcal{B}}(\sigma) = \bot$$

$$\frac{}{\langle \mathsf{while}\ e\ \mathsf{do}\ S,\, \sigma\rangle \rightarrow \langle S;\ \mathsf{while}\ e\ \mathsf{do}\ S,\, \sigma\rangle}\ [\![e]\!]_{\mathcal{B}}(\sigma) = \top$$

$$\frac{}{\langle \mathsf{while}\ e\ \mathsf{do}\ S,\, \sigma\rangle \rightarrow \sigma}\ [\![e]\!]_{\mathcal{B}}(\sigma) = \bot$$

Figure 3: Definition of the operational semantics of statements.

Hoare Triples

A Hoare triple $\{P\}\ S\ \{Q\}$ for $P, Q \subseteq \mathsf{State}$ asserts that, for any state $\sigma \in \mathsf{State}$, if $\sigma \in P$ and $\langle S,\, \sigma\rangle \rightarrow^* \sigma'$, then $\sigma' \in Q$. The sets P and Q can be represented as Boolean expressions extended with quantifiers.

The rules for constructing Hoare triples are given in Figure 4.

$$\frac{}{\{P\}\ \mathsf{skip}\ \{P\}} \qquad \frac{}{\{P\}\ x \leftarrow e\ \{\exists x'.\, P[x'/x]\ \&\&\ x = e[x'/x]\}}$$

$$\frac{\{P\}\ S_1\ \{Q\} \quad \{Q\}\ S_2\ \{R\}}{\{P\}\ S_1; S_2\ \{R\}} \qquad \frac{\{P\ \&\&\ e\}\ S_1\ \{Q_1\} \quad \{P\ \&\&\ !e\}\ S_3\ \{Q_2\}}{\{P\}\ \mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \{Q_1 \parallel Q_2\}}$$

$$\frac{\{P\ \&\&\ e\}\ S\ \{P\}}{\{P\}\ \mathsf{while}\ e\ \mathsf{do}\ S\ \{P\ \&\&\ !e\}} \qquad \frac{\{P_1\}\ S\ \{Q_1\} \quad \begin{array}{l} P_2 \subseteq P_1 \\ Q_1 \subseteq Q_2 \end{array}}{\{P_2\}\ S\ \{Q_2\}}$$

Figure 4: Rules of Hoare logic.

Computable Functions

We write $[\mathtt{x} \mapsto n]$ for the state that maps the variable x to the number $n \in \mathbb{N}$, and every other variable to 0.

A 'while' program S *computes* a partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ (with respect to x) just if $f(m) \simeq n$ exactly when $\langle S, [\mathtt{x} \mapsto m]\rangle \Downarrow [\mathtt{x} \mapsto n]$.

A function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ is *computable* just if there is a program S that computes f with respect to the variable x.

Predicates

The *characteristic function* of U is the function

$$\chi_U : \mathbb{N} \to \mathbb{N}$$
$$\chi_U(n) = \begin{cases} 1 & \text{if } n \in U \\ 0 & \text{if } n \notin U \end{cases}$$

The *semi-characteristic function* of U is the partial function

$$\xi_U : \mathbb{N} \rightharpoonup \mathbb{N}$$
$$\xi_U(n) \begin{cases} \simeq 1 & \text{if } n \in U \\ \uparrow & \text{otherwise} \end{cases}$$

A predicate $U \subseteq \mathbb{N}$ is *decidable* just if its characteristic function $\chi_U : \mathbb{N} \to \mathbb{N}$ is computable.

The 'while' program that computes the characteristic function χ_U of a predicate $U \subseteq \mathbb{N}$ is called a *decision procedure*. Any predicate for which there is no decision procedure is called *undecidable*.

(cont.)

A predicate $U \subseteq \mathbb{N}$ is *semi-decidable* just if its semi-characteristic function ξ_U is computable.

The *Halting Problem* is the following predicate:

$$\mathsf{HALT} = \{\langle \ulcorner S \urcorner, n\rangle \mid [\![S]\!]_{\mathtt{x}}(n) \downarrow\}$$

Bijections

A function $f : A \to B$ is *injective* (or 1-1) just if for any $a_1, a_2 \in \mathcal{A}$ we have that $f(a_1) = f(a_2)$ implies $a_1 = a_2$. We sometimes write $f : A \rightarrowtail B$ whenever f is an injection.

A function $f : A \to B$ is *surjective* just if for any $b \in \mathcal{B}$ there exists $a \in \mathcal{A}$ such that $f(a) = b$. We sometimes write $f : A \twoheadrightarrow B$ whenever f is a surjection.

A function $f : A \to B$ is a *bijection* just if it is both injective and surjective.

Let $f : A \to B$ be a function. f is an *isomorphism* just if it has an *inverse*. That is, if there exists a function $f^{-1} : B \to A$ such that:

- for all $a \in \mathcal{A}$ we have $f^{-1}(f(a)) = a$
- for all $b \in \mathcal{B}$ we have $f(f^{-1}(b)) = b$

Encoding Data

A *pairing function* is a bijection $\mathbb{N} \times \mathbb{N} \xrightarrow{\cong} \mathbb{N}$. We assume that we have a fixed pairing function

$$\langle -, - \rangle : \mathbb{N} \times \mathbb{N} \xrightarrow{\cong} \mathbb{N}$$

with the following inverse:

$$\mathsf{split} : \mathbb{N} \xrightarrow{\cong} \mathbb{N} \times \mathbb{N}$$

Reflections

Suppose we have two bijections:

$$\phi : A \xrightarrow{\cong} \mathbb{N} \quad \psi : B \xrightarrow{\cong} \mathbb{N}$$

The *reflection* of $f : A \rightharpoonup B$ under (ϕ, ψ) is the function

$$\tilde{f} : \mathbb{N} \rightharpoonup \mathbb{N}$$
$$\tilde{f}(n) = \psi(f(\phi^{-1}(n)))$$

Gödel Numbering

Let **Stmt** be the set of Abstract Syntax Trees of While. We assume that we have a Gödel numbering

$$\ulcorner - \urcorner : \textbf{Stmt} \xrightarrow{\cong} \mathbb{N}$$

which encodes While programs as natural numbers.

A *code transformation* is a function $f : \textbf{Stmt} \to \textbf{Stmt}$.

Universal Function

The *universal function*, U, is defined as follows:

$$U : \mathbf{Stmt} \times \mathbb{N} \rightharpoonup \mathbb{N}$$
$$U(P, n) = [\![P]\!]_{\mathtt{x}}(n)$$

Reductions

Let $U, W \subseteq \mathbb{N}$ be predicates, and let $f : \mathbb{N} \to \mathbb{N}$. The function f is a *many-one reduction* from U to W just if it is computable, and it is also the case that

$$n \in U \Leftrightarrow f(n) \in W$$

We may write $f : U \lesssim V$ (read "f is a reduction from U to V").

END OF PAPER