UNIVERSITY OF BRISTOL

January 2023 Examination Period

FACULTY OF ENGINEERING

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

COMS20007J
Programming Languages and Computation

TIME ALLOWED:
3 Hours

Answers to COMS20007J: Programming Languages and Computation

Intended Learning Outcomes:

Q1. This question is about regular languages.

*(a) Consider each of the following regular expressions over the alphabet $\{a, b, c\}$:

1. $((b+c)^* a(b+c)^* a(b+c)^*)^*$
2. $(a+b+c)^* abb(a+b+c)^*$
3. $(a+b+c)^*(abb+baa)(a+b+c)^*$
4. $b(a+b+c)^* b$
5. $((a+b+c)(a+b+c))^*$
6. $(b+c)^*$
7. $abb(a+b+c)^*$

Match each of the following descriptions of languages to the regular expression above that denotes it:

i. The language of all words that start and end with b
ii. The language of all words that contain abb as a substring
iii. The language of all words that start with abb.
iv. The language of all words that do not contain a.
v. The language of all even length words.
vi. The language of all words containing an even number of a.
vii. The language of all words that either contain abb or bba

[10 marks]

Solution:

i. 4

ii. 2

iii. 7

iv. 6

v. 5

vi. 1

vii. 3

*(b) Design regular expressions that denote the following sets of words over $\{a, b\}$:

i. All non-empty words that start with an a, e.g. aab, a, $abaa$.
ii. All words whose length is a multiple of 3, e.g. abb, $ababba$.
iii. All words that start and end with a different letter, e.g. $abbaab$.
iv. All words that contain a letter b exactly two places from the end, e.g. $aabab$, baa.

v. All words that do not contain the substring aa.

[10 marks]

Solution:

i. $a(a+b)^*$

ii. $((a+b)(a+b)(a+b))^*$

iii. $a(a+b)^*b + b(a+b)^*a$

iv. $(a+b)^*b(a+b)(a+b)$

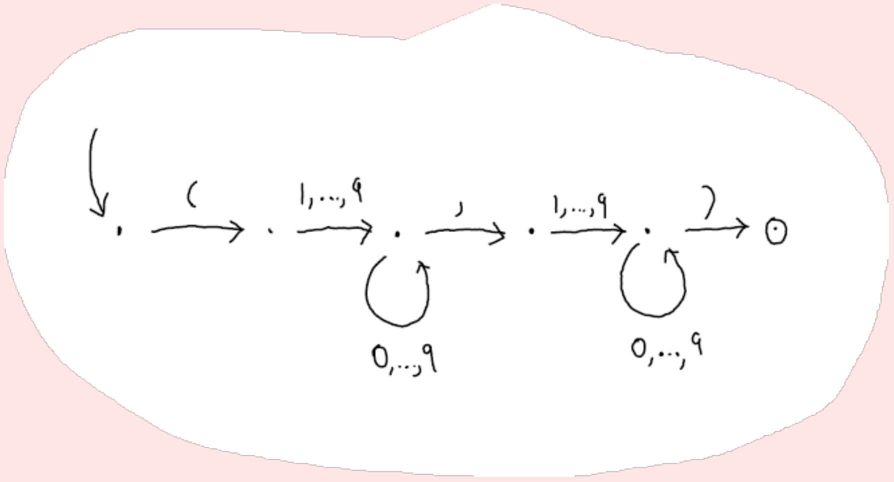
v. $b^*a + b^*(ab^+)^*$

** (c) A pair (n,m) of natural numbers can be written as a string consisting of the following, in sequence:

- a left parenthesis,
- a non-empty sequence of digits not starting with 0,
- a comma,
- a non-empty sequence of digits not starting with 0,
- a right parenthesis.

Construct a finite state automaton to recognise pairs of natural numbers written as strings.

[4 marks]



Solution:

** (d) Using the pumping lemma, show that the language $\{a^n b^m \mid n \geq m\}$ is not regular.

Hint: When you come to "pump" your chosen word, will need to choose the number of repetitions carefully.

[6 marks]

(cont.)

Solution: Let's call this language A. Suppose for contradiction that A were regular. Then, by the pumping lemma, there is a number $p \geq 0$ such that, for all strings $s \in A$ of length at least p, we have that s may be decomposed into uvw satisfying the usual three conditions:

1. $|uv| \leq p$
2. $|v| \geq 1$
3. for all $i \in \mathbb{N}$, $uv^i w \in A$

Then consider $a^p b^p$, which is clearly in A. By (1), $uv = a^{|uv|}$, and hence $w = a^{p-|uv|} b^p$. By condition (3), we have that $uv^0 w \in A$, which is $a^{|u|} a^{p-|uv|} b^p = a^{p-|v|} b^p$. However, by (2), $p - |v| < p$, so this word cannot be in A: a contradiction. Hence, it must be that A is not regular.

*** (e) Let $\Sigma = \{0, 1\}$. Show that the following language is regular over Σ:

$$\{1^k w \mid k \geq 1,\ w \in \Sigma^*,\ \#_1(w) \geq k\}$$

where $\#_1(v)$ counts the number of 1 characters in the word v, e.g. $\#_1(0010110) = 3$.

[5 marks]

Solution: This language can be expressed as $1(1+0)^*1(1+0)^*$. Clearly, every word matched by this regular expression is in the above language, by taking $k = 1$. To see why every word in the above language is matched: suppose I have a word $1^k w$ in the lanugage, then this word can also be written as $1^1 v$ for $v = 1^{k-1} w$. Since $\#_1(w) \geq k \geq 1$, there is at least one 1 in v, hence it is matched by $(1+0)^*1(1+0)^*$, so the whole word $1v$ is indeed matched by the given expression.

*** (f) Give an example of a non-regular language that satisfies the conclusion of the Pumping Lemma. That is, give a language A that is *not* regular, and yet there is a number p such that, for all words $s \in A$ of length at least p, s can be split into three parts $s = uvw$ such that:

1. $|uv| \leq p$
2. $|v| \geq 1$
3. for all $i \in \mathbb{N}$, $uv^i w \in A$

Justify your answer.

[5 marks]

Solution: The language A defined as $\{ab^n c^n \mid n \in \mathbb{N}\} \cup (a^*b^*c^* \setminus ab^*c^*)$ satisfies these requirements. The language cannot be regular, because we can take the intersection with the regular language ab^*c^* to yield $\{ab^n c^n \mid n \in \mathbb{N}\}$, which is clearly not regular. The let $p = 1$. Any word $s \in A$ of length at least 1, has

```
q ← 0
while b < a + 1 {
  a ← a - b
  q ← q + 1
}
r ← a
```

Figure 1: A simple While programme

shape ℓt, for $\ell \in \Sigma$ and $t \in \Sigma^*$ (i.e. it starts with some letter ℓ). Hence it can be decomposed as $u = \epsilon$, $v = \ell$, $w = t$, satisfying conditions (1) and (2). Finally, observe that this also satisfies condition 3, for every word uv^iw with $i \neq 1$ lies inside the second summand of A.

Q2. This question is about the While language, its expressions, and its extensions.

* (a) For each of the following, indicate whether it represents a valid arithmetic expression, a valid boolean expression, or neither.

i. `x + 10 < 6 * (-42 - y)`

ii. `x ← z - 42 + a`

iii. `true && false || 42 * x < 0`

iv. `true = true`

v. `winner * 2 = chicken + dinner`

[5 marks]

Solution:

i. Arithmetic expression

ii. Neither (statement)

iii. Boolean expression

iv. Neither

v. Boolean expression

* (b) Consider the While programme shown in Figure 1. For each of the following states, indicate whether the programme terminates when run in that initial state, and the values of q and r in the final state (if it exists).

i. $\begin{bmatrix} \texttt{a} & \mapsto & 25 \\ \texttt{b} & \mapsto & 3 \end{bmatrix}$

ii. $\begin{bmatrix} \texttt{a} & \mapsto & 25 \\ \texttt{b} & \mapsto & \texttt{-12} \end{bmatrix}$

(cont.)

iii. $\begin{bmatrix} \texttt{a} \mapsto 25 \\ \texttt{b} \mapsto n \end{bmatrix}$, where n is the last digit of your student number

[9 marks]

** (c) Define the following expression constructs as *syntactic sugar*. Recall that syntactic sugar does not extend the formal syntax of expressions, but defines shorthands for existing formal constructs.

For example, we could define a binary operator `!=` such that $[\![a_1\texttt{!=}a_2]\!]^{\mathcal{B}}(\sigma) = [\![a_1]\!]^{\mathcal{A}}(\sigma) \neq [\![a_2]\!]^{\mathcal{A}}(\sigma)$ for every state σ and every pair of arithmetic expressions a_1 and a_2 as follows.

$$a_1 \ \texttt{!=}\ a_2 \equiv \texttt{!}a_1 = a_2$$

i. A unary arithmetic operator `-` such that $[\![\texttt{-}a]\!]^{\mathcal{A}}(\sigma) = -[\![a]\!]^{\mathcal{A}}(\sigma)$ for every state σ and every arithmetic expression a.

ii. A binary operator `<=` such that $[\![a_1\texttt{<=}a_2]\!]^{\mathcal{B}}(\sigma) = [\![a_1]\!]^{\mathcal{A}}(\sigma) \leq [\![a_2]\!]^{\mathcal{A}}(\sigma)$ for every state σ and every pair of arithmetic expressions a_1 and a_2.

iii. A single-branch conditional statement if b then s such that, for every boolean expression b, every statement s and every state σ, the following holds.

$$[\![\text{if } b \text{ then } s]\!](\sigma) = \begin{cases} [\![s]\!](\sigma) & \text{if } [\![b]\!]^{\mathcal{B}}(\sigma) \\ [\![\texttt{skip}]\!](\sigma) & \text{if } \neg [\![b]\!]^{\mathcal{B}}(\sigma) \end{cases}$$

Solution:

i. $\texttt{-}a \equiv 0\ \texttt{- a}$

ii. $a_1\ \texttt{<=}\ a_2 \equiv a_1 < a_2\ \texttt{||}\ a_1 = a_2$

iii. `if` b `then` $s \equiv$ `if` b `then` s `else skip`

[6 marks]

In the remainder of this question, for answers given as programmes in the While language or one of its extensions, we will be forgiving of syntactic issues which could be defined as syntactic sugar—as long as their meaning is clear.

** (d) For each of the following programmes run on the following initial state, either describe their operational semantics as a tree, or otherwise indicate non-termination.

$$\begin{bmatrix} \texttt{x} \mapsto 3 \\ \texttt{n} \mapsto 2 \end{bmatrix}$$

Take care not to waste time trying to write down infinite traces, but to still explicitly indicate when the programme does not terminate.

i.
```
r ← x
while !1 = n {
  r ← r * x
```

```
    n ← n - 1
  }
```

ii.
```
while !0 = n {
    n ← n - x
  }
```

[4 marks]

*** (e) Consider the following relational definition $\sigma \vdash b \Downarrow_{\mathcal{B}} v$ for the semantics of Boolean expressions. (Read "in state σ, b can evaluate to v".)

$$\frac{}{\sigma \vdash \texttt{true} \Downarrow_{\mathcal{B}} \top} \qquad \frac{}{\sigma \vdash \texttt{false} \Downarrow_{\mathcal{B}} \bot}$$

$$\frac{\sigma \vdash b \Downarrow_{\mathcal{B}} v}{\sigma \vdash \texttt{!}b \Downarrow_{\mathcal{B}} \neg b} \qquad \frac{\sigma \vdash b_1 \Downarrow_{\mathcal{B}} v_1 \qquad \sigma \vdash b_2 \Downarrow_{\mathcal{B}} v_2}{\sigma \vdash b_1 \texttt{\&\&} b_2 \Downarrow_{\mathcal{B}} v_1 \wedge v_2}$$

$$\frac{\sigma \vdash b_1 \Downarrow_{\mathcal{B}} v_1 \qquad \sigma \vdash b_2 \Downarrow_{\mathcal{B}} v_2}{\sigma \vdash b_1 \texttt{||} b_2 \Downarrow_{\mathcal{B}} v_1 \vee v_2} \qquad \frac{}{\sigma \vdash a_1 = a_2 \Downarrow_{\mathcal{B}} [\![a_1]\!]^{\mathcal{A}}(\sigma) = [\![a_2]\!]^{\mathcal{A}}(\sigma)}$$

$$\frac{}{\sigma \vdash a_1 < a_2 \Downarrow_{\mathcal{B}} [\![a_1]\!]^{\mathcal{A}}(\sigma) < [\![a_2]\!]^{\mathcal{A}}(\sigma)}$$

By induction, prove that, for every boolean expression b and every state σ, the following holds, where $[\![\cdot]\!]^{\mathcal{B}}$ is the functional semantics defined in the notes.

$$\sigma \vdash b \Downarrow_{\mathcal{B}} [\![b]\!]^{\mathcal{B}}(\sigma)$$

[6 marks]

Q3. This question is about computability.

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

$$f(x) \begin{cases} \simeq 2^x - 1 & \text{if } x \text{ is even} \\ \uparrow & \text{otherwise} \end{cases}$$

is computable. *[5 marks]*

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

```
// First determine whether x is even.
r := x;
// Invariant: r = r_0 mod 2 && r >= 0
while (r >= 2) do { r := r - 2; }
if (r = 0) {
  // Then x is even.
  // Invariant: s = 2^j && 0 <= j <= x
  s := 1; j := 0;
  while (j < x) { s := s * 2; j := j + 1 }
  x := s - 1;
}
else {
  // x is not even, loop forever
  while (true) { }
}
r := 0; s := 0; j := 0
```

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.

- The set of prime numbers is decidable.
- If a function has an inverse, it must be an injection.
- Every surjection has an inverse.
- WHILE programs compute partial functions.
- If a function is computable then it must be an injection.

[5 marks]

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

** (c) Let $f : A \to B$ and $g : B \to C$. Show that if $g \circ f : A \to C$ is injective, then so is f. *[3 marks]*

Solution: Suppose $f(a_1) = f(a_2)$. Then $g(f(a_1)) = g(f(a_2))$, which is to say that $(g \circ f)(a_1) = (g \circ f)(a_2)$. As $g \circ f$ is injective, it follows that $a_1 = a_2$, which is what we wanted to prove.

** (d) Show that the predicate

$$U = \{\ulcorner S \urcorner \mid \text{for all } k \leq 2023 \text{ it is true that } [\![S]\!]_{\mathtt{x}}(k) = [\![S]\!]_{\mathtt{x}}(k+1)\}$$

is semi-decidable. (The use of "=" here means that both sides of the equality must be defined and equal.) *[5 marks]*

Solution: For each $k = 0, \ldots, 2023$ simulate S on inputs k and $k+1$. Whenever one of these simulations terminates check whether the output of the first is equal to the output of the second; if not, return false and halt. Otherwise, after all the simulations are over, return true. Because this is a semi-decision procedure the simulations need not terminate, in which case nothing is returned.

*** (e) Show that the predicate

$$V = \{\ulcorner S \urcorner \mid \text{there exists } k \in \mathbb{N} \text{ such that } [\![S]\!]_{\mathtt{x}}(k) = [\![S]\!]_{\mathtt{x}}(k+1)\}$$

is undecidable (The use of "=" here means that both sides of the equality must be defined and equal.) *[5 marks]*

Solution: Given $\mathtt{D} \in$ **Stmt** and $n \in \mathbb{N}$ let

$$\mathtt{S}_{\mathtt{D},n} = \mathtt{x\ :=\ n;\ D;\ x\ :=\ 0}$$

This program ignores its input, runs $\mathtt{D}$ on n, and if that halts outputs 0.

Construct the code transformation $F : \textbf{Stmt} \times \mathbb{N} \to \textbf{Stmt}$ given by

$$F(\mathtt{D}, n) = \mathtt{S}_{\mathtt{D},n}$$

It is easy to argue that the reflection of this code transformation is computable. We have

$$\langle \ulcorner \mathtt{D} \urcorner, n \rangle \in \text{HALT} \iff \ulcorner S_{\mathtt{D},n} \urcorner \in V$$

As the latter is not decidable, neither is the former.

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.

*** (f) Show that the following predicate is undecidable:

$$P = \{\langle \ulcorner S_1 \urcorner, \ulcorner S_2 \urcorner \rangle \mid \text{for all } n \in \mathbb{N}\text{: } [\![S_1]\!]_x(n) \simeq 1 \text{ iff } [\![S_2]\!]_x(n) \simeq k \text{ where } k \neq 1\ \}$$

[7 marks]

(cont.)

Solution: We construct a reduction f : HALT $\lesssim P$. If P were decidable, then we could also decide the Halting Problem for While programs, which is impossible since this problem is known to be undecidable.

We define a code transformation $F : \textbf{Stmt} \times \mathbb{N} \to \textbf{Stmt} \times \textbf{Stmt}$ by

$$F(\mathtt{D}, n) = (\mathtt{D;\ x\ :=\ 1,\ x\ :=\ 0})$$

:19

We argue that this constitutes a reduction. Suppose $F(D, n) = (S, T)$. Recalling that by convention all our programs are assumed to compute wrt `x`, we see that D halts on input n iff $[\![S]\!]_x(m) \simeq 1$ for all $m \in \mathbb{N}$. We have that $[\![T]\!]_x(n) \simeq 0$ for all $n \in \mathbb{N}$, and clearly $0 \neq 1$. Hence:

- If D halts on n then $\langle \ulcorner S \urcorner, \ulcorner T \urcorner \rangle \in P$ since, for all m:

$$[\![S]\!]_x(m) \simeq 1 \quad \text{iff} \quad [\![T]\!]_x(m) \simeq 0$$

- but otherwise we have $\langle \ulcorner S \urcorner, \ulcorner T \urcorner \rangle \notin P$ since there is an m for which both:

$$[\![S]\!]_x(m) \not\simeq 1 \quad \textit{and} \quad [\![T]\!]_x(m) \simeq 0$$

  In fact, our construction ensures that this is true for every m!

The reflection of this transformation in $\mathbb{N} \to \mathbb{N}$ can be computed by the following algorithm. On input $m \in \mathbb{N}$:

1. Decode m as $\langle \ulcorner \mathtt{D} \urcorner, n \rangle$ to obtain `D` and n.
2. Construct the program $S_{\mathtt{D},n}$ as:

```
D; x := 1
```

3. Return $\langle \ulcorner S_{\mathtt{D},n} \urcorner, \ulcorner \mathtt{x:=1} \urcorner \rangle$

Reminder of Important Definitions

Regex Syntax

$$R, S ::= \emptyset \mid \epsilon \mid a \mid R \cdot S \mid R + S \mid R^*$$

Regex Semantics

$$\text{(MEmpty)}\ \frac{}{\epsilon \text{ matches } \epsilon} \qquad \text{(MChar)}\ \frac{}{a \text{ matches } a} \qquad \text{(StarB)}\ \frac{}{R^* \text{ matches } \epsilon}$$

$$\text{(ChoiceL)}\ \frac{R \text{ matches } w}{R + S \text{ matches } w} \qquad \text{(ChoiceR)}\ \frac{S \text{ matches } w}{R + S \text{ matches } w}$$

$$\text{(StarS)}\ \frac{R \text{ matches } v \qquad R^* \text{ matches } w}{R^* \text{ matches } vw} \qquad \text{(Concat)}\ \frac{R \text{ matches } v \qquad S \text{ matches } w}{R \cdot S \text{ matches } vw}$$

The language *expressed* by a regular expression R over alphabet Σ is:

$$L(R) = \{\ w \in \Sigma^* \mid R \text{ matches } w\ \}$$

Finite Automata

A *deterministic finite state automaton* (DFA) is a tuple $M = (Q, \Sigma, \Delta, q_0, F)$:

- A collection of *states*, Q, which can be any finite set.
- A finite alphabet, Σ, (so we know what kinds of words we are willing to process)
- A transition function, δ, of type $Q \times \Sigma \to Q$.
- An initial state, q_0, which is just a choice of any element of Q.
- A collection of accepting states, F, which is any subset of Q.

A *run* of M on a word w is a finite sequence of transitions: $q_0 \xrightarrow{\ell_1} q_1 \xrightarrow{\ell_2} \cdots \xrightarrow{\ell_n} q_n$ such that both of the following:

- $\ell_1 \ell_2 \cdots \ell_n = w$.
- For all $0 \leq i \leq n-1$: $\delta(q_i,\ \ell_{i+1}) = q_{i+1}$.

The run is said to be *accepting* just if $q_n \in F$. A string w is said to be *accepted* by a finite automaton M just if there is an accepting run of M on w. The language *recognised* by M, written $L(M)$, is the set of all strings accepted by M.

(cont.)

Regular Languages

A language is said to be a *regular language* just if it is the language expressed by some regular expression (equivalently, by some finite automaton).

Pumping Lemma for Regular Languages. Let A be a regular language. Then there is some natural number $p \geq 1$ such that, for any word $s \in A$ of length at least p, there is a splitting of s into three pieces $s = uvw$ such that the following all hold:

(P1) v is not empty

(P2) $|uv| \leq p$

(P3) $\forall i \in \mathbb{N}.\ uv^i w \in A$

While Program Syntax

Syntax of arithmetic expressions

An *integer literal* is a string over the alphabet $\{-, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ that matches the following regular expression:

$$(\texttt{-} \mid \epsilon)(\texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9})^+$$

An *arithmetic expression* is a tree described by the following grammar, where `n` stands for integer literals, and `x` stands for a variable name.

$$A_1, A_2 ::= \texttt{n} \mid \texttt{x} \mid A_1 \texttt{ + } A_2 \mid A_1 \texttt{ - } A_2 \mid A_1 \texttt{ * } A_2$$

We denote the set of arithmetic expressions as $\mathcal{A}$.

Syntax of boolean expressions

A *boolean expression* is a tree described by the following grammar. We exclude strings `true` and `false` from the set of variable names.

$$B, B_1, B_2 ::= \texttt{true} \mid \texttt{false} \mid \texttt{!}B \mid B_1 \texttt{ \&\& } B_2 \mid B_1 \texttt{ || } B_2 \mid A_1 \texttt{ = } A_2 \mid A_1 \texttt{ <= } A_2$$

We denote the set of boolean expressions as $\mathcal{B}$.

Syntax of statements

A *statement* is a tree described by the following grammar. We exclude strings `if`, `then`, `else`, and `while` from the set of variable names.

$$S, S_1, S_2 ::= \epsilon \mid \texttt{x} \leftarrow A \mid S_1\texttt{;}\ S_2 \mid \texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2 \mid \texttt{while } B\ S$$

We denote the set of statements as $\mathcal{S}$.

Qu. continues . . .

While Program Semantics

A *state* is a total function in State = Var $\longrightarrow \mathbb{Z}$. Given a state σ, a variable x and an integer value v, we denote with $[\mathtt{x} \mapsto v]\sigma$ the function that maps x to v, and maps all x' $\neq$ x to $\sigma(\mathtt{x'})$. By convention, when a variable x is not explicitly defined in a given state σ, we will always take $\sigma(\mathtt{x}) = 0$.

Semantics of arithmetic expressions

We define the semantics of arithmetic expressions as a total function $[[\cdot]]^{\mathcal{A}} \in \mathcal{A} \longrightarrow \text{State} \longrightarrow \mathbb{Z}$, inductively as follows, where $[[\mathtt{n}]]^{\mathbb{Z}}$ is the number corresponding to the numeral n.

$$\begin{aligned}
[[\mathtt{n}]]^{\mathcal{A}}(\sigma) &= [[\mathtt{n}]]^{\mathbb{Z}} \\
[[\mathtt{x}]]^{\mathcal{A}}(\sigma) &= \sigma(\mathtt{x}) \\
[[A_1 \;\mathtt{+}\; A_2]]^{\mathcal{A}}(\sigma) &= [[A_1]]^{\mathcal{A}}(\sigma) + [[A_2]]^{\mathcal{A}}(\sigma) \\
[[A_1 \;\mathtt{-}\; A_2]]^{\mathcal{A}}(\sigma) &= [[A_1]]^{\mathcal{A}}(\sigma) - [[A_2]]^{\mathcal{A}}(\sigma) \\
[[A_1 \;\mathtt{*}\; A_2]]^{\mathcal{A}}(\sigma) &= [[A_1]]^{\mathcal{A}}(\sigma) \cdot [[A_2]]^{\mathcal{A}}(\sigma)
\end{aligned}$$

Semantics of boolean expressions

We define the semantics of boolean expressions as a total function $[[\cdot]]^{\mathcal{B}} \in \mathcal{B} \longrightarrow \text{State} \longrightarrow \mathbb{B}$, inductively on the expression's structure.

$$\begin{aligned}
[[\mathtt{true}]]^{\mathcal{B}}(\sigma) &= \top \\
[[\mathtt{false}]]^{\mathcal{B}}(\sigma) &= \bot \\
[[\mathtt{!}B]]^{\mathcal{B}}(\sigma) &= \neg[[B]]^{\mathcal{B}}(\sigma) \\
[[B_1 \;\mathtt{\&\&}\; B_2]]^{\mathcal{B}}(\sigma) &= [[B_1]]^{\mathcal{B}}(\sigma) \wedge [[B_2]]^{\mathcal{B}}(\sigma) \\
[[B_1 \;\mathtt{||}\; B_2]]^{\mathcal{B}}(\sigma) &= [[B_1]]^{\mathcal{B}}(\sigma) \vee [[B_2]]^{\mathcal{B}}(\sigma) \\
[[A_1 \;\mathtt{=}\; A_2]]^{\mathcal{B}}(\sigma) &= [[A_1]]^{\mathcal{A}}(\sigma) = [[A_2]]^{\mathcal{A}}(\sigma) \\
[[A_1 \;\mathtt{<=}\; A_2]]^{\mathcal{B}}(\sigma) &= [[A_1]]^{\mathcal{A}}(\sigma) <= [[A_1]]^{\mathcal{A}}(\sigma)
\end{aligned}$$

Semantics of statements

The set of semantic configurations for the While language is $\mathcal{S} \times \text{State}$.

We define a relation

$$\Downarrow \subseteq (\mathcal{S} \times \text{State}) \times \text{State}$$

using the following inference rules.

(cont.)

$$\text{(BSkip)}\ \frac{}{\langle \epsilon, \sigma \rangle \Downarrow \sigma} \qquad \text{(BAss)}\ \frac{}{\langle \mathtt{x} \ \leftarrow \ A, \sigma \rangle \Downarrow \sigma[\mathtt{x} \mapsto [[A]]^{\mathcal{A}}(\sigma)]}$$

$$\text{(BSeq)}\ \frac{\langle S_1, \sigma \rangle \Downarrow \sigma' \qquad \langle S_2, \sigma' \rangle \Downarrow \sigma''}{\langle S_1;\ S_2, \sigma \rangle \Downarrow \sigma''}$$

$$\text{(BIf}_\top\text{)}\ \frac{[[B]]^{\mathcal{B}}(\sigma) = \top \qquad \langle S_1, \sigma \rangle \Downarrow \sigma'}{\langle \mathtt{if}\ B\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2, \sigma \rangle \Downarrow \sigma'} \qquad \text{(BIf}_\bot\text{)}\ \frac{[[B]]^{\mathcal{B}}(\sigma) = \bot \qquad \langle S_2, \sigma \rangle \Downarrow \sigma'}{\langle \mathtt{if}\ B\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2, \sigma \rangle \Downarrow \sigma'}$$

$$\text{(BWhile}_\top\text{)}\ \frac{[[B]]^{\mathcal{B}}(\sigma) = \top \qquad \langle S, \sigma \rangle \Downarrow \sigma' \qquad \langle \mathtt{while}\ B\ S, \sigma' \rangle \Downarrow \sigma''}{\langle \mathtt{while}\ B\ S, \sigma \rangle \Downarrow \sigma''}$$

$$\text{(BWhile}_\bot\text{)}\ \frac{[[B]]^{\mathcal{B}}(\sigma) = \bot}{\langle \mathtt{while}\ B\ S, \sigma \rangle \Downarrow \sigma}$$

Computable Functions

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

A 'while' program $\mathtt{S}$ *computes* a partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ (with respect to $\mathtt{x}$) 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 $\mathtt{x}$.

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.

Qu. continues . . .

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*.

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:

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

Bijections

A function $f : A \to B$ is *injective* (or 1-1) just if for any $a_1, a_2 \in A$ 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 $b \in B$ there exists $a \in A$ 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 $a \in A$ we have $f^{-1}(f(a)) = a$
- for all $b \in B$ 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

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

(cont.)

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:

$$\begin{gathered} U : \textbf{Stmt} \times \mathbb{N} \rightharpoonup \mathbb{N} \\ U(P, n) = [[P]]_{\mathtt{x}}(n) \end{gathered}$$

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