UNIVERSITY OF BRISTOL

Winter 2024 Examination Period

SCHOOL OF COMPUTER SCIENCE

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

COMS20007W
Programming Languages and Computation

TIME ALLOWED:
3 Hours

This paper contains *three* questions, worth *40*, *30* and *30* marks respectively. Answer *all* questions. The maximum for this paper is *100 marks*. Credit will be given for partial answers.

Other Instructions:

Candidates may bring to the exam room 1 double-sided A4 page of notes in any format. A reminder of key definitions is provided at the back of this paper.

TURN OVER ONLY WHEN TOLD TO START WRITING

Q1. This question is about syntax.

*(a) Consider the following grammar over terminal symbols $\{a, b\}$:

$$S \longrightarrow aSa \mid bSb \mid \epsilon$$

i. Give two examples of words over $\{a, b\}$ that are derivable in the grammar.

ii. Give two examples of words over $\{a, b\}$ that are not derivable in the grammar.

iii. Is the following statement true or false? Every word derivable in the grammar has even length.

[5 marks]

*(b) Consider each of the following grammars over the alphabet $\{a, b, c\}$. In each case, the start symbol is S.

1.
$$S \longrightarrow aSaS \mid bS \mid cS \mid \epsilon$$

2.
$$\begin{array}{rcl} S & \longrightarrow & TabbT \mid TbbaT \\ T & \longrightarrow & aT \mid bT \mid cT \mid \epsilon \end{array}$$

3.
$$\begin{array}{rcl} S & \longrightarrow & bTb \\ T & \longrightarrow & aT \mid bT \mid cT \mid \epsilon \end{array}$$

4.
$$\begin{array}{rcl} S & \longrightarrow & XSX \mid \epsilon \\ X & \longrightarrow & a \mid b \mid c \end{array}$$

5.
$$S \longrightarrow bS \mid cS \mid \epsilon$$

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 do not contain a.

iii. The language of all even length words.

iv. The language of all words containing an even number of a.

v. The language of all words that either contain abb or bba as a substring.

[5 marks]

*(c) Consider the following grammar for the syntax of Combinatory Logic:

$$M \longrightarrow \textsf{var} \mid k \mid s \mid M\ M \mid (\ M\ )$$

whose 5 terminal symbols are:

$$\textsf{var} \quad k \quad s \quad (\quad)$$

i. Compute nullable, and the first and follow sets for this grammar.
ii. Draw the parse table for this grammar.
iii. Is the grammar LL(1)?

[10 marks]

** (d) For each of the following sets of words over $\{a, b\}$, design a context-free grammar that expresses the set:

i. All words whose length is a multiple of 3, e.g. abb, $ababba$.
ii. All words that start and end with a different letter, e.g. $abbaab$.
iii. All words that contain a letter b exactly two places from the end, e.g. $aabab$, baa.
iv. All words that do not contain the substring aa.

[6 marks]

** (e) Give an LL(1) grammar equivalent to the following context-free grammar:

$$S \longrightarrow \emptyset \mid (\ S\ ) \mid \mathsf{atom} \mid S \cup S \mid S \cap S \mid S^c$$

whose terminal symbols are:

$$\emptyset \quad (\quad) \quad \mathsf{atom} \quad \cup \quad \cap \quad {}^c$$

[4 marks]

*** (f) Show that the following language over $\{0, 1\}$ can be expressed by a context-free grammar and justify your construction.

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

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

[5 marks]

*** (g) Define the following indexed family of words w_i by recursion on $i \in \mathbb{N}$:

$$\begin{aligned} w_0 &= a \\ w_{k+1} &= a + w_k \end{aligned}$$

For example, $w_3 = a + a + a + a$ and $w_5 = a + a + a + a + a + a$.

Prove that every word in the language $\{w_i \mid i \in \mathbb{N}\}$ is derivable in the following grammar (whose start symbol is S):

$$\begin{array}{rcl} S & \longrightarrow & a\ U \\ U & \longrightarrow & +\ a\ U \mid \epsilon \end{array}$$

[5 marks]

Q2. This question is about semantics.

*(a) For each of the following, indicate whether it represents a valid arithmetic expression, a valid Boolean expression, or neither. In each case, if the expression is valid, evaluate the appropriate denotation function in the state $[x \mapsto 1,\ y \mapsto 2,\ z \mapsto 3]$.

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

ii. $x \leftarrow z - (42 + y)$

iii. true $\&\&$ (false $\|$ $42 * x < 0$)

iv. true $=$ true

v. $w * 2 = c + d$

[5 marks]

**(b) Suppose we add a new form of arithmetic expressions — the *integer exponentiation* operator so that the grammar of arithmetic expressions is now defined as follows:

$$A \longrightarrow n \mid x \mid A + A \mid A - A \mid A * A \mid A \,\hat{}\, A$$

We extended the denotation function for arithmetic expressions with the equation:

$$[\![e_1 \,\hat{}\, e_2]\!]_{\mathcal{A}}(\sigma) = \begin{cases} 0 & \text{if } [\![e_2]\!]_{\mathcal{A}}(\sigma) < 0 \\ [\![e_1]\!]_{\mathcal{A}}(\sigma)^{[\![e_2]\!]_{\mathcal{A}}(\sigma)} & \text{otherwise} \end{cases}$$

i. Find two arithmetic expressions $e_1 \in \mathcal{A}$ and $e_2 \in \mathcal{A}$ such that the arithmetic expression $x \,\hat{}\, (e_1 + e_2)$ is *not* semantically equivalent to the arithmetic expression $(x \,\hat{}\, e_1) \cdot (x \,\hat{}\, e_2)$.

ii. Prove that the arithmetic expression $e\hat{}2$ is semantically equivalent to the arithmetic expression $e * e$ for an any given arithmetic expression $e \in \mathcal{A}$.

iii. Let $S_1 \in \mathcal{S}$ and $S_2 \in \mathcal{S}$ be arbitrary While statements. Prove that the statement "if $x = 1$ then $x \leftarrow x \,\hat{}\, x;\ S_1$ else S_2" and the statement "if $x = 1$ then S_1 else S_2" are semantically equivalent.

[10 marks]

***(c) Consider the While program shown in Figure 1.

while $b \leq a$ do
  $a \leftarrow a - b;$
  $q \leftarrow q + 1$

Figure 1: A simple While program

i. For each of the following states, indicate whether the program terminates when executed in that initial state, and the values of q and a in the final state (if it exists). You do not need to state the corresponding derivation.

1. $[a \mapsto 25,\ b \mapsto 3]$

2. $[a \mapsto 25,\ b \mapsto -12]$
3. $[a \mapsto 25,\ b \mapsto 0]$
4. $[a \mapsto -25,\ b \mapsto 10]$
5. $[a \mapsto 10,\ b \mapsto 3]$

ii. Prove that this program in fact terminates when executed in any initial state in which b is positive. That is, for any $\sigma \in$ State such that $\sigma(b) > 0$, show that there exists some $\sigma' \in$ State such that $P, \sigma \Downarrow \sigma'$ where P is the aforementioned program. You will need to use the strong induction principle.

[15 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]*

*(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]

**(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]*

**(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]*

***(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]*

***(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]

Reminder of Important Definitions

Grammars

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

- An alphabet of *terminal* symbols, which we shall usually write as Σ (capital letter sigma)
- A finite, non-empty set of *non-terminal* symbols, disjoint from the terminals, which we shall usually write as $\mathcal{N}$
- A finite set of *production rules*, which we shall usually write as $\mathcal{R}$, each of which has shape: $X \longrightarrow \alpha$.
- A designated non-terminal from $\mathcal{N}$, called the *start symbol*, which we will usually write as S.

A *sentential form*, usually α, β, γ and so on, is just a finite sequence of terminals (from Σ) and nonterminals (from $\mathcal{N}$).

The *one-step derivation relation* is a binary relation on sentential forms with two sentential forms α and β related, written $\alpha \to \beta$, just if α is of shape $\alpha_1 X \alpha_2$ and there is a production rule $X \longrightarrow \gamma$ and β is exactly $\alpha_1 \gamma \alpha_2$.

We write $\alpha \to^* \beta$, and say β *is derivable from* α just if β can be derived from α in any (finite) number of steps, including zero steps.

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 $S \to^* w$.

While Concrete Syntax

The concrete syntax of the While programming language can be described by the following grammar:

$$
\begin{array}{rcl}
S & \longrightarrow & \text{skip} \mid V \leftarrow A \mid S;S \mid \text{if } B \text{ then } S \text{ else } S \mid \text{while } B \text{ do } S \mid \{\ S\ \} \\
B & \longrightarrow & \text{true} \mid \text{false} \mid A \leq A \mid A = A \mid\ !\,B \mid B\ \&\&\ B \mid B \parallel B \mid (B) \\
A & \longrightarrow & V \mid N \mid A + A \mid A - A \mid A * A \mid (A) \\
D & \longrightarrow & 0 \mid 1 \mid \cdots \mid 9 \\
E & \longrightarrow & D\ E \mid \epsilon \\
L & \longrightarrow & a \mid b \mid \cdots \mid z \\
U & \longrightarrow & A \mid B \mid \cdots \mid Z \mid ' \\
M & \longrightarrow & L\ M \mid U\ M \mid \epsilon \\
V & \longrightarrow & L\ M \\
N & \longrightarrow & D\ E
\end{array}
$$

(cont.)

Nullable

On nonterminals:

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

On sentential forms:

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

To calculate Nullable, first set the approximation Nullable$[X]$ to false for each nonterminal X, then repeatedly perform the following iteration until a fixed point is reached:

- For each production $X \longrightarrow \alpha$:
  - Nullable$[X]$:= Nullable$[X] \vee$ Nullable$_s(\alpha)$

First

On nonterminals:

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

On sentential forms:

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

To calculate First, first set the approximation First$[X]$ to the empty set $\emptyset$ for each nonterminal X. Then repeatedly perform the following iteration until a fixed point is reached:

- For each production $X \longrightarrow \alpha$:
  - First$[X]$:= First$[X] \cup$ First$_s(\alpha)$

Follow

On nonterminals:

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

To calculate Follow, start by initialising Follow$[X]$ to the empty set for each non-terminal X. Then repeatedly perform the following nested iteration until a fixed point is reached:

- For each non-terminal X:
  - For each occurrence of X on the right-hand side of a production $Y \longrightarrow \alpha X\beta$:
    * Follow$[X]$:= Follow$[X] \cup$ First$_s(\beta)$
    * if Nullable$_s(\beta)$ then Follow$[X]$:= Follow$[X] \cup$ Follow$[Y]$

Parse Tables and LL(1)

We define the *parse 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 parse 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:

$$A \longrightarrow n \mid x \mid A + A \mid A - A \mid A * A$$

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.

$$B \longrightarrow \mathsf{false} \mid \mathsf{true} \mid !B \mid B \mathbin{\&\&} B \mid B \mathbin{\|} B \mid A = A \mid A \leq A$$

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:

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

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

While Program Semantics

A *state* is a total function from the set $\mathsf{State} = \mathsf{Var} \to \mathbb{Z}$, 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 $x_i \in \mathsf{Var}$ 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 $\sigma \in \mathsf{State}$, we write $\sigma[x \mapsto v]$ for some variable $x \in \mathsf{Var}$ and $v \in \mathbb{Z}$ to denote the state that maps the variable x to v and any other variable y to the value $\sigma(y)$.

(cont.)

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 2. 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 $\sigma \in \text{State}$.

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

Figure 2: 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 3. 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 $\sigma \in \text{State}$.

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

Figure 3: Definition of the denotational semantics of Boolean expressions.

Semantics of statements

The operational semantics relation $\Downarrow \subseteq \mathcal{S} \times \text{State} \times \text{State}$ is defined inductive by the rules in Figure 4. We say that two statements $S_1,\, S_2 \in \mathcal{S}$ are *semantically equivalent* if, and only if:

$$S_1,\, \sigma_1 \Downarrow \sigma_2 \Leftrightarrow S_2,\, \sigma_1 \Downarrow \sigma_2$$

for any two states $\sigma_1,\, \sigma_2 \in \text{State}$.

Computable Functions

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

$$\frac{}{\text{skip}, \, \sigma \Downarrow \sigma} \qquad \frac{}{x \leftarrow e, \, \sigma \Downarrow \sigma[x \mapsto [\![e]\!]_{\mathcal{A}}(\sigma)]}$$

$$\frac{S_1, \, \sigma_1 \Downarrow \sigma_2 \quad S_2, \, \sigma_2 \Downarrow \sigma_3}{S_1; S_2, \, \sigma_1 \Downarrow \sigma_3} \qquad \frac{S_1, \, \sigma_1 \Downarrow \sigma_2}{\text{if } e \text{ then } S_1 \text{ else } S_2, \, \sigma_1 \Downarrow \sigma_2} \, [\![e]\!]_{\mathcal{B}}(\sigma_1) = \top$$

$$\frac{S_2, \, \sigma_1 \Downarrow \sigma_2}{\text{if } e \text{ then } S_1 \text{ else } S_2, \, \sigma_1 \Downarrow \sigma_2} \, [\![e]\!]_{\mathcal{B}}(\sigma_1) = \bot \qquad \frac{}{\text{while } e \text{ do } s, \, \sigma \Downarrow \sigma} \, [\![e]\!]_{\mathcal{B}}(\sigma) = \bot$$

$$\frac{S, \, \sigma_1 \Downarrow \sigma_2 \quad \text{while } e \text{ do } S, \, \sigma_2 \Downarrow \sigma_3}{\text{while } e \text{ do } S, \, \sigma_1 \Downarrow \sigma_3} \, [\![e]\!]_{\mathcal{B}}(\sigma_1) = \top$$

Figure 4: Definition of the operational semantics of statements.

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*.

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:

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

(cont.)

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:

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

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