

Supervision 2

Lent Term 2027 · Michael Lee

●●●● difficulty · ★ optional

WORK DUE	A single PDF, 48 hours before the supervision
EXPECTED TIME	6 hours 5 min (+ 35 min for the optional questions)
OPTIONAL	★ marks a question, or a part of one, you may skip

1 The Halting Problem and Decidability

PREREQUISITES Lecture 5 · TIME 1 hour 50 min

QUESTION 1

CONCEPTUAL · 20 MIN

●●●● BASIC

Consider the notion of semi-decidability, also known as recursive enumerability.

The semi-characteristic function of a set A , ψ_A , is defined as:

$$\psi_A(x) = \begin{cases} 1, & x \in A \\ \uparrow, & x \notin A \end{cases}$$

A set A is (register-machine) semi-decidable if its semi-characteristic function is (register machine) computable.

- (a) Show that all decidable sets are semi-decidable.
- (b) Give an example of a semi-decidable set that is not decidable.
- (c) Prove that if a set S is semi-decidable, and its complement S^c is also semi-decidable, then S is decidable.
- (d) Give an example of a set that is not semi-decidable.

QUESTION 2

CONCEPTUAL · 30 MIN

●●●● CORE

Show that decidable sets are closed under union, intersection, and complementation.

In other words, show that, for any decidable sets X and Y

- (a) $X \cup Y$ is decidable
- (b) $X \cap Y$ is decidable

(c) X^c is decidable

Which of these statements is true for undecidable sets?

QUESTION 3

CONCEPTUAL · 20 MIN

● ● ● ● ● CORE

Assume a partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$.

Define the graph of f , $\text{graph}(f)$, as the subset of $\mathbb{N} \times \mathbb{N}$

$$\{(x, y) \in \mathbb{N} \times \mathbb{N} \mid f(x) \downarrow \wedge f(x) = y\}$$

Define the domain of f , $\text{dom}(f)$, as the subset of $\mathbb{N}$ at which f is defined

$$\{x \in \mathbb{N} \mid f(x) \downarrow\}$$

Define the range of f , $\text{range}(f)$, as the subset of $\mathbb{N}$

$$\{y \in \mathbb{N} \mid \exists x \in \mathbb{N}. f(x) \downarrow \wedge f(x) = y\}$$

Note that decidability can also be defined over sets of pairs $\mathbb{N} \times \mathbb{N}$, since these are in bijection (using Gödel encoding and decoding) with numbers $\mathbb{N}$.

- (a) Construct a function f such that $\text{dom}(f)$ and $\text{range}(f)$ are both decidable, but $\text{graph}(f)$ is not decidable.
- (b) Construct a function f such that $\text{graph}(f)$ is decidable, but $\text{dom}(f)$ is not decidable.
- (c) Construct a function f such that $\text{graph}(f)$ is decidable, but $\text{range}(f)$ is not decidable.
- (d) Show that if $\text{graph}(f)$ is decidable, then f is register machine computable.
- (e) Construct a function f which is register machine computable but where $\text{dom}(f)$ and $\text{range}(f)$ are both not decidable.

QUESTION 4

CONCEPTUAL · 20 MIN

● ● ● ● ● BASIC

Let A, B be subsets of $\mathbb{N}$. Write $A \leq B$ if there exists a total computable function $r : \mathbb{N} \rightarrow \mathbb{N}$ that satisfies the following property

$$\forall n \in \mathbb{N}. n \in A \iff r(n) \in B$$

- (a) Assume $A \leq B$. Show that if B is decidable, then A is.
- (b) Consider the converse: If $A \leq B$, and A is decidable, must B be?
- (c) Explain, informally, how to read $\leq$.

QUESTION 5

CONCEPTUAL · 20 MIN

● ● ● ● ● BASIC

Show that the following set E is undecidable. $=$ is extensional equality.

$$E \triangleq \{(e, e') \mid \phi_e = \phi_{e'}\}$$

2 Turing Machines

PREREQUISITES Lecture 7 · TIME 45 min (+ 35 min)

QUESTION 6

CONCEPTUAL · 5 MIN

● ● ● ● ● BASIC

Explain whether the Turing Machine introduced in the notes is deterministic, or non-deterministic.

QUESTION 7

CONCEPTUAL · 5 MIN

● ● ● ● ● BASIC

The notes state

We have seen that a Turing machine's computation can be implemented by a register machine. The converse holds, too: the computation of a register machine can be implemented by a Turing machine. To make sense of this, **we first have to fix a tape representation of register machine configurations and hence of numbers and lists of numbers.**

Why do we need to fix a tape representation of numbers? Why can we not set $\Sigma = \mathbb{N} \cup \{\sqcup, \triangleright\}$?

QUESTION 8

CONCEPTUAL · 10 MIN

● ● ● ● ● BASIC

Some Turing Machines require the head of the machine to move left or right, rather than stay still, during a state transition.

The formalisation in the course allows the head of the machine to remain still during a state transition. Explain how this choice adds no additional expressiveness.

★ QUESTION 9

CONCEPTUAL · 30 MIN

● ● ● ● ● CORE

Define a Turing Machine that accepts all binary palindromes, and rejects all other strings.

QUESTION 10

CONCEPTUAL · 20 MIN

●●●●● CORE

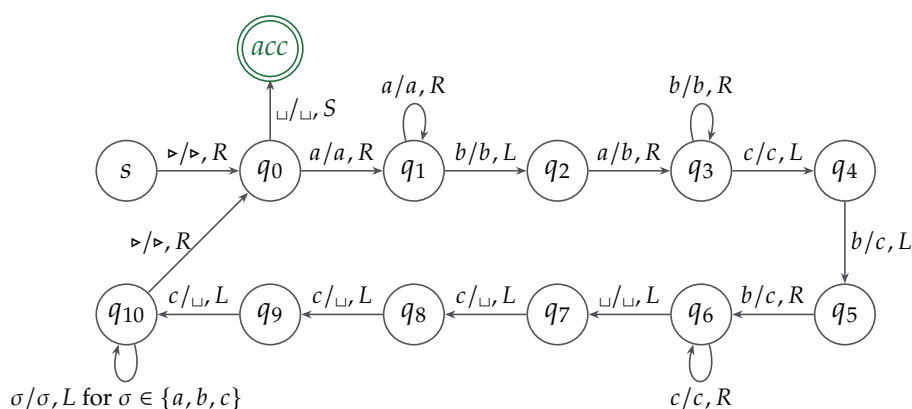
Explain what the following Turing Machine does. It is started in the configuration $(s, \triangleright, u)$ with $u \in \{a, b, c\}^*$.

The Turing Machine has many transitions: the ones that are unreachable are faded out.

$M = (Q, \Sigma, s, \delta)$ where $Q = \{s, q_0, \dots, q_{10}\}$ (s initial), $\Sigma = \{\triangleright, \sqcup, a, b, c\}$, and δ is

δ	$\triangleright$	$\sqcup$	a	b	c
s	$\langle q_0, \triangleright, R \rangle$	$\langle rej, \sqcup, R \rangle$	$\langle rej, a, R \rangle$	$\langle rej, b, R \rangle$	$\langle rej, c, R \rangle$
q_0	$\langle rej, \triangleright, R \rangle$	$\langle acc, \sqcup, S \rangle$	$\langle q_1, a, R \rangle$	$\langle rej, b, S \rangle$	$\langle rej, c, S \rangle$
q_1	$\langle rej, \triangleright, R \rangle$	$\langle rej, \sqcup, S \rangle$	$\langle q_1, a, R \rangle$	$\langle q_2, b, L \rangle$	$\langle rej, c, S \rangle$
q_2	$\langle rej, \triangleright, R \rangle$	$\langle rej, \sqcup, R \rangle$	$\langle q_3, b, R \rangle$	$\langle rej, b, R \rangle$	$\langle rej, c, R \rangle$
q_3	$\langle rej, \triangleright, R \rangle$	$\langle rej, \sqcup, S \rangle$	$\langle rej, a, S \rangle$	$\langle q_3, b, R \rangle$	$\langle q_4, c, L \rangle$
q_4	$\langle rej, \triangleright, R \rangle$	$\langle rej, \sqcup, R \rangle$	$\langle rej, a, R \rangle$	$\langle q_5, c, L \rangle$	$\langle rej, c, R \rangle$
q_5	$\langle rej, \triangleright, R \rangle$	$\langle rej, \sqcup, R \rangle$	$\langle rej, a, R \rangle$	$\langle q_6, c, R \rangle$	$\langle rej, c, R \rangle$
q_6	$\langle rej, \triangleright, R \rangle$	$\langle q_7, \sqcup, L \rangle$	$\langle rej, a, S \rangle$	$\langle rej, b, S \rangle$	$\langle q_6, c, R \rangle$
q_7	$\langle rej, \triangleright, R \rangle$	$\langle rej, \sqcup, R \rangle$	$\langle rej, a, R \rangle$	$\langle rej, b, R \rangle$	$\langle q_8, \sqcup, L \rangle$
q_8	$\langle rej, \triangleright, R \rangle$	$\langle rej, \sqcup, R \rangle$	$\langle rej, a, R \rangle$	$\langle rej, b, R \rangle$	$\langle q_9, \sqcup, L \rangle$
q_9	$\langle rej, \triangleright, R \rangle$	$\langle rej, \sqcup, R \rangle$	$\langle rej, a, R \rangle$	$\langle rej, b, R \rangle$	$\langle q_{10}, \sqcup, L \rangle$
q_{10}	$\langle q_0, \triangleright, R \rangle$	$\langle rej, \sqcup, R \rangle$	$\langle q_{10}, a, L \rangle$	$\langle q_{10}, b, L \rangle$	$\langle q_{10}, c, L \rangle$

You may prefer to view the Turing Machine as a state transition diagram. Rejecting edges are implicit, and have been left out.



QUESTION 11

CONCEPTUAL · 5 MIN (+ 5 MIN)

●●●●● BASIC

(a) Construct a notion of Turing-Machine decidability.

HINT You may find it helpful to fix a concrete Σ , like $\{0, 1, \triangleright, \sqcup\}$.

- ★ (b) List other forms of decidability that are not equivalent to Turing-Machine decidability. You should feel free to use your answers to [Question 9](#) and [Question 10](#).

3 Primitive Recursive Functions

PREREQUISITES Lecture 9 · TIME 2 hours 30 min

QUESTION 12

CONCEPTUAL · 30 MIN

● ● ● ● ● CORE

Recall the fold function for lists, in OCaml.

```
1  let rec fold_r b f = function
2    | [] -> b
3    | x::xs -> f x (fold_r b f xs)
```

OCaml

A very similar concept to a fold is a *recursor*.

```
1  let rec recurse_r b f = function
2    | [] -> b
3    | x::xs -> f xs x (recurse_r b f xs)
```

OCaml

- (a) Show that every fold is a recursor.
 (b) Show that every recursor is a fold.

Consider the OCaml datatype for natural numbers, defined as follows:

```
type nat = Z | S of nat
```

With the following functions given for free:

```
1  let succ n = S n
2  let zero = Z
3  let comp g f = fun x -> g (f x)
```

OCaml

- (c) Design a fold function for naturals. It should have the following type

```
val fold : nat -> (nat -> nat) -> nat -> nat
```

Note that an implementation may have a more general type, but we will ignore this difference for now.

- (d) Design a recurse function for naturals. It should have the following type

```
val recurse : nat -> (nat -> nat -> nat) -> nat -> nat
```

Note that an implementation may have a more general type, but we will ignore this difference for now.

- (e) By instantiating your recurse function in [Part 12\(d\)](#), write the following functions

(i) `val add : nat -> nat -> nat`

- (ii) `val triangle : nat -> nat`
- (iii) `val pred : nat -> nat`
- (iv) `val mul : nat -> nat -> nat`
- (f) Modify the statement of Theorem 19 in the notes to drop the additional parameters $\vec{x}$.
- (g) Compare your answers in [Part 12\(d\)](#) and [Part 12\(f\)](#).
- (h) Extend your recursor over the naturals in [Part 12\(d\)](#) to take a list of additional arguments $\vec{x}$ (call it `xs`). Call this alternative recursor `recurse'`. Explain why, in OCaml, every use of `recurse'` is equivalent to a use of `recurse`.
- (i) Explain why the definition of primitive recursion in the course uses (the equivalent of) `recurse'`, rather than `recurse`.

QUESTION 13

CONCEPTUAL · 20 MIN

● ● ● ● ● CORE

The notes define the set of primitive recursive functions, PRIM , as follows:

A (partial) function f is primitive recursive ($f \in \text{PRIM}$) if it can be built up in finitely many steps from the basic functions by use of the operations of composition and primitive recursion.

- (a) Rephrase this definition as an *inductive* definition, as in Part IB Semantics of Programming Languages.
- (b) Write an OCaml ADT that captures the syntax of primitive recursion

QUESTION 14

CONCEPTUAL · 30 MIN

● ● ● ● ● CORE

- (a) Assume there is a function `ackm` that computes $A_m = \text{ack}(m, -)$. By instantiating `recurse`, write an OCaml function `ackm'` that computes $A_{m+1} = \text{ack}(m+1, -)$

```
val ackm' : nat -> nat
```

The existence of `ackm'` shows that A_{m+1} can be defined by applying primitive recursion to A_m and the basic functions.

- (b) Now make the dependency on `ackm` explicit, by modifying your definition such that `m` and `ackm` are passed as an argument

```
val ackm' : nat -> (nat -> nat) -> nat -> nat
      (*m      ackm      n      ack(m+1, n) *)
```

Convince yourself that `ackm'`, as defined, computes A_{m+1} from A_m

- (c) Consider generalising the return type of the recursor from `nat` to `'a`

```
val recurse : 'a -> (nat -> 'a -> 'a) -> nat -> 'a
```

By choosing suitable `'a`, use `recurse` to write the Ackermann function

```
val ack : nat -> nat -> nat
```

in a primitive recursive style.

HINT How can you use your answer to the previous parts of the question?

- (d) Prove that, for all m , A_m is primitive recursive.
- (e) Explain what your answer to [Part 14\(c\)](#) implies about the relationship between the Ackermann function, recursors, and *primitive recursive functions*.

QUESTION 15

CONCEPTUAL · 40 MIN

● ● ● ● ● CORE

Show that the following functions are all primitive recursive.

You can do this abstractly, by composing together primitive recursive functions (the notes define a set of primitive recursive functions: you can assume these are primitive recursive for free).

You can also do this concretely. I have defined a language for writing primitive recursive functions [here](#). Each of the building blocks of primitive recursion has a corresponding construct in the language:

1. $\text{proj}[n, i]$ is the projection proj_i^n .
2. $\text{zero}[n]$ is the zero function zero^n .
3. succ is the successor function succ .
4. $f \circ (g_1, \dots, g_m)$ is the composition $f \circ [g_1, \dots, g_m]$. When $m = 1$, you may write $f \circ g$.
5. $\text{rho}[n](f, g)$ is the primitive recursion $\rho^n(f, g)$.
6. $\text{let } h = e$ names the function e as h .
7. An expression $h(x_1, \dots, x_n)$ at the end of the file applies h to the numbers $x_1, \dots, x_n$. When $n = 1$, you may write $h\ x$.

I have also provided a standard library of primitive recursive functions, corresponding to each of the primitive recursive functions defined by the notes (the **Primitive Recursive Library** in the **Standard Libraries** sheet). You may use any of these “standard library” functions.

- (a) Truncated subtraction function, $\text{minus}: \mathbb{N}^2 \rightarrow \mathbb{N}$, where

$$\text{minus}(x, y) \triangleq \begin{cases} x - y & \text{if } y \leq x \\ 0 & \text{if } y > x \end{cases}$$

- (b) Exponentiation, $\text{exp}: \mathbb{N}^2 \rightarrow \mathbb{N}$, where $\text{exp}(x, y) = x^y$.

- (c) Conditional branch on zero, $\text{ifzero}: \mathbb{N}^3 \rightarrow \mathbb{N}$, where

$$\text{ifzero}(x, y, z) \triangleq \begin{cases} y & \text{if } x = 0 \\ z & \text{if } x > 0 \end{cases}$$

- (d) Bounded summation: if $f: \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ is primitive recursive, then so is $g: \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ where

$$g(\vec{x}, x) \triangleq \begin{cases} 0 & \text{if } x = 0 \\ f(\vec{x}, 0) & \text{if } x = 1 \\ f(\vec{x}, 0) + \dots + f(\vec{x}, x-1) & \text{if } x > 1 \end{cases}$$

QUESTION 16

CONCEPTUAL · 30 MIN

● ● ● ● ● BASIC

Use minimisation to show that the following functions are partial recursive:

- (a) the binary maximum function $\text{max}: \mathbb{N}^2 \rightarrow \mathbb{N}$.
- (b) the integer square root function $\text{sqrt}: \mathbb{N} \rightarrow \mathbb{N}$ which is only defined if its argument is a perfect square.

Once again, you may do this abstractly. Alternatively, you can use the [programming language](#) given in [Question 15](#). You can write μ^n as `mu[n]`.

Minimisation can result in non-termination. Thus, you need to run the programs with minimisation explicitly turned on, using the `-m` flag, as in `pr -m [FILE]`.

4 The Lambda Calculus: Syntax

PREREQUISITES Lecture 10 · TIME 20 min

QUESTION 17

CONCEPTUAL · 10 MIN

● ● ● ● ● BASIC

Give the syntax of the lambda calculus as an inductive definition, as in the Part IB Semantics of Programming Languages course.

QUESTION 18

CONCEPTUAL · 10 MIN

● ● ● ● ● BASIC

Draw the abstract syntax trees corresponding to each of the following concrete expressions. You can use `@` to mean application.

- (a) $\lambda x. u \ x \ y$
- (b) $\lambda x. u \ (x \ y)$
- (c) $\lambda x. x \ \lambda y. y$
- (d) $(\lambda x. x) (\lambda y. y)$

5 The Lambda Calculus: Equational Systems

PREREQUISITES None · TIME 40 min

QUESTION 19

CONCEPTUAL · 40 MIN

● ● ● ● ● CORE

This question encourages you to explore the relationship between equational systems and reduction systems.

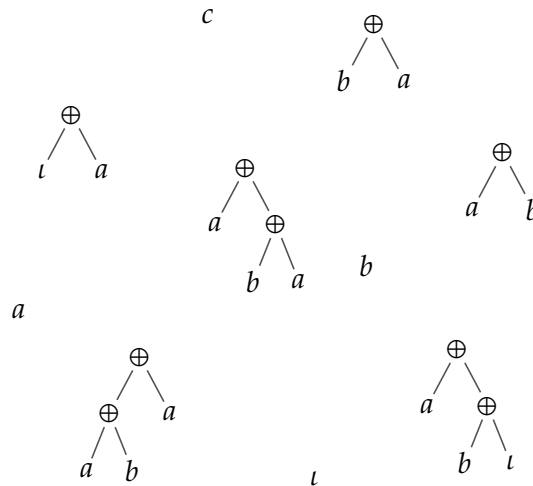
Consider a programming language whose syntax is defined as follows:

$$\begin{aligned} x &::= a \mid b \mid c \\ e &::= \iota \mid x \mid e \oplus e' \end{aligned}$$

with the equations

$$\begin{aligned} \iota \oplus e &= e \\ e \oplus \iota &= e \\ e \oplus (e' \oplus e'') &= (e \oplus e') \oplus e'' \end{aligned}$$

(a) Consider the following abstract syntax trees.



Group ASTs such that all ASTs in a group are equal to one another, and not equal to any AST in any other group.

- Each group in your answer to [Part 19\(a\)](#) contains a selection of elements from the same equivalence class. Why is the notion of a *normal form* useful? What properties should a normal form have?
- Suggest two distinct normal forms for this language. Convert $(a \oplus b) \oplus (\iota \oplus (c \oplus b))$ to each normal form.
- For each of the normal forms in [Part 19\(c\)](#), propose an algorithm for converting an arbitrary expression e into that normal form. Relate your answer to the notion of a *reduction system*.
- Suggest an algorithm for determining if two expressions e and e' are in the same equivalence class.
- Does your algorithm always terminate with a yes/no answer? What can you conclude about the decidability of $=$ as defined above?