MA5219 LOGIC AND FOUNDATION OF MATHEMATICS I

Lecture 1: Programs

Tin Lok Wong

13 August, 2018

The aim of this lecture is to characterize

$$\{S \subseteq \mathbb{N} : \text{some algorithm can tell whether } n \in \mathbb{N} \text{ belongs to } S\}.$$

By an 'algorithm' we mean here an idealized algorithm with no memory or time restriction.

Convention 1.1. The set of all *natural numbers* is $\mathbb{N} = \{0, 1, 2, \dots\}$. *Arithmetic* means arithmetic on $\mathbb{N}$. Unless otherwise stated, the domain of quantification is $\mathbb{N}$. If the length of a tuple $x_1, x_2, \dots, x_n$ is understood or unimportant, then we often abbreviate the tuple as $\bar{x}$.

Let us start by considering some simple decision algorithms on $\mathbb{N}$. These algorithms are specified in a programming language that we call $\mathcal{L}_{\mathrm{A}}(\exp)$.

Definition. The notion of an $\mathcal{L}_{\mathrm{A}}(\exp)$ *term* is defined by recursion as follows.

- Every variable is an $\mathcal{L}_{\mathrm{A}}(\exp)$ term.
- The symbols 0 and 1 are $\mathcal{L}_{\mathrm{A}}(\exp)$ terms.
- If t, s are $\mathcal{L}_{\mathrm{A}}(\exp)$ terms, then $(t + s)$, $(t \times s)$ and (2^t) are $\mathcal{L}_{\mathrm{A}}(\exp)$ terms.
- Every $\mathcal{L}_{\mathrm{A}}(\exp)$ term is obtained by applying the rules above finitely many times.

To make the notation lighter, we may omit some brackets by

- representing $\times$ by juxtaposition, e.g., by writing $x \times y$ as xy; and
- adopting the usual precedence rules for arithmetic operations, e.g., by writing $(x+(y\times z))+x$ as $x + yz + x$.

If $x_1, x_2, \dots, x_k$ is a list of variables with no repetition, and all the variables in an $\mathcal{L}_{\mathrm{A}}(\exp)$ term t appear in this list, then we may write t as $t(x_1, x_2, \dots, x_k)$.

Example 1.2. If w, x, z are variables, then $2^{x+1}z + 2^x + w$ is an $\mathcal{L}_{\mathrm{A}}(\exp)$ term.

Definition. An $\mathcal{L}_{\mathrm{A}}(\exp)$ term $t(x_1, x_2, \dots, x_k)$ can be evaluated on inputs $a_1, a_2, \dots, a_k \in \mathbb{N}$ to give a value $t(a_1, a_2, \dots, a_k) \in \mathbb{N}$. This evaluation is defined by recursion on (the number of steps in the construction of) the term t as follows: whenever $t(x_1, x_2, \dots, x_k), s(x_1, x_2, \dots, x_k)$ are $\mathcal{L}_{\mathrm{A}}(\exp)$ terms and $a_1, a_2, \dots, a_k \in \mathbb{N}$,

- if $t(x_1, x_2, \dots, x_k) = x_i$, then $t(a_1, a_2, \dots, a_k) = a_i$;
- if $t(x_1, x_2, \dots, x_k) = 0$, then $t(a_1, a_2, \dots, a_k) = 0$;
- if $t(x_1, x_2, \dots, x_k) = 1$, then $t(a_1, a_2, \dots, a_k) = 1$;
- $(t + s)(a_1, a_2, \dots, a_k) = t(a_1, a_2, \dots, a_k) + s(a_1, a_2, \dots, a_k)$;
- $(t \times s)(a_1, a_2, \dots, a_k) = t(a_1, a_2, \dots, a_k) \times s(a_1, a_2, \dots, a_k)$;
- $(2^t)(a_1, a_2, \dots, a_k) = 2^{t(a_1, a_2, \dots, a_k)}$.

Example 1.3. If $t(w,x,z) = 2^{x+1}z + 2^x + w$, then

$$t(0,1,2) = 2^{1+1} \times 2 + 2^1 + 0 = 10.$$

Definition. Each (decision) program has finitely many *input variables*. We sometimes indicate the input variables $x_1, x_2, \ldots, x_k$ of a program θ in the notation by writing $\theta(x_1, x_2, \ldots, x_k)$. When using this notation, we implicitly assume that the variables $x_1, x_2, \ldots, x_k$ are mutually distinct. Given inputs $a_1, a_2, \ldots, a_k \in \mathbb{N}$, an $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ program $\theta(x_1, x_2, \ldots, x_k)$ may or may not return an answer, but if it does, then the answer must be either `true` or `false`. The *output* of an $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ program $\theta(x_1, x_2, \ldots, x_k)$ on inputs $a_1, a_2, \ldots, a_k \in \mathbb{N}$ is defined to be

$$[\![\theta(a_1, a_2, \ldots, a_k)]\!] = \begin{cases} \texttt{true}, & \text{if } \theta \text{ returns } \texttt{true} \text{ on inputs } a_1, a_2, \ldots, a_k; \\ \texttt{false}, & \text{if } \theta \text{ returns } \texttt{false} \text{ on inputs } a_1, a_2, \ldots, a_k; \\ \texttt{UNDEF}, & \text{otherwise.} \end{cases}$$

The notion of an $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ *program* is defined by recursion as follows.

- Let $\bar{x}$ be variables. The $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ program $\top(\bar{x})$ is

  return `true`

  For all inputs $\bar{a} \in \mathbb{N}$, we have $[\![\top(\bar{a})]\!] = \texttt{true}$.

- Let $t(\bar{x}), s(\bar{x})$ be $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ terms. The $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ program $(t = s)(\bar{x})$ is

  if $t(\bar{x}) = s(\bar{x})$
  then return `true`
  else return `false`
  end if

  For inputs $\bar{a} \in \mathbb{N}$,

  $$[\![(t = s)(\bar{a})]\!] = \begin{cases} \texttt{true}, & \text{if } t(\bar{a}) = s(\bar{a}); \\ \texttt{false}, & \text{otherwise.} \end{cases}$$

- Let $t(\bar{x}), s(\bar{x})$ be $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ terms. The $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ program $(t < s)(\bar{x})$ is

  if $t(\bar{x}) < s(\bar{x})$
  then return `true`
  else return `false`
  end if

  For inputs $\bar{a} \in \mathbb{N}$,

  $$[\![(t < s)(\bar{a})]\!] = \begin{cases} \texttt{true}, & \text{if } t(\bar{a}) < s(\bar{a}); \\ \texttt{false}, & \text{otherwise.} \end{cases}$$

- If $\theta(\bar{x})$ is an $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ program, then so is $\neg\theta(\bar{x})$, which is defined to be

  if $[\![\theta(\bar{x})]\!] = \texttt{true}$
  then return `false`
  else return `true`
  end if

  For inputs $\bar{a} \in \mathbb{N}$,

  $$[\![\neg\theta(\bar{a})]\!] = \begin{cases} \texttt{true}, & \text{if } [\![\theta(\bar{a})]\!] = \texttt{false}; \\ \texttt{false}, & \text{if } [\![\theta(\bar{a})]\!] = \texttt{true}; \\ \texttt{UNDEF}, & \text{otherwise.} \end{cases}$$

- If $\theta(\bar{x}, \bar{z}), \eta(\bar{y}, \bar{z})$ are $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ programs, where $\{\bar{x}\} \cap \{\bar{y}\} = \varnothing$. then so is $(\theta \vee \eta)(\bar{x}, \bar{y}, \bar{z})$, which is defined to be

if $[\![\theta(\bar{x},\bar{z})]\!] = \texttt{true}$ **par-or** $[\![\eta(\bar{y},\bar{z})]\!] = \texttt{true}$
then return `true`
else return `false`
end if

For inputs $\bar{a},\bar{b},\bar{c} \in \mathbb{N}$ of appropriate lengths,

$$[\![(\theta \vee \eta)(\bar{a},\bar{b},\bar{c})]\!] = \begin{cases} \texttt{true}, & \text{if } [\![\theta(\bar{a},\bar{c})]\!] = \texttt{true} \text{ or } [\![\eta(\bar{b},\bar{c})]\!] = \texttt{true}; \\ \texttt{false}, & \text{if } [\![\theta(\bar{a},\bar{c})]\!] = [\![\eta(\bar{b},\bar{c})]\!] = \texttt{false}; \\ \texttt{UNDEF}, & \text{otherwise.} \end{cases}$$

- Let $t(\bar{x})$ be an $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ term. If $\theta(\bar{x},y)$ is an $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ program, then so is $(\exists y{<}t\ \theta)(\bar{x})$, which is defined to be

  par-for $y \leftarrow 0,1,\ldots,t(\bar{x})-1$ **do**
  if $[\![\theta(\bar{x},y)]\!] = \texttt{true}$
  then return `true`
  end if
  end par-for
  return `false`

  For inputs $\bar{a} \in \mathbb{N}$,

$$[\![(\exists y{<}t\ \theta)(\bar{a})]\!] = \begin{cases} \texttt{true}, & \text{if } [\![\theta(\bar{a},b)]\!] = \texttt{true} \text{ for some } b < t(\bar{a}); \\ \texttt{false}, & \text{if } [\![\theta(\bar{a},b)]\!] = \texttt{false} \text{ for all } b < t(\bar{a}); \\ \texttt{UNDEF}, & \text{otherwise.} \end{cases}$$

- If $\theta(\bar{x},y)$ be an $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ program, then so is $(\exists y\ \theta)(\bar{x})$, which is defined to be

  initialize $y \leftarrow 0$
  par-while $[\![\theta(\bar{x},y)]\!] = \texttt{false}$ **do**
  $y \leftarrow y+1$
  end par-while
  return `true`

  For inputs $\bar{a} \in \mathbb{N}$,

$$[\![(\exists y\ \theta)(\bar{a})]\!] = \begin{cases} \texttt{true}, & \text{if } [\![\theta(\bar{a},b)]\!] = \texttt{true} \text{ for some } b \in \mathbb{N}; \\ \texttt{UNDEF}, & \text{otherwise.} \end{cases}$$

- Every $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ program is obtained by applying the construction rules above finitely many times.

We also introduce the following shorthand.

- Let t,s be $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ terms.
  - $t \leqslant s$ stands for $t < s+1$.
  - $t \neq s$ stands for $\neg(t=s)$.
- Let θ,η be $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ programs.
  - $\bot = \neg\top$.
  - $\theta \wedge \eta = \neg(\neg\theta \vee \neg\eta)$.
  - $\theta \to \eta = \neg\theta \vee \eta$.
  - $\theta \leftrightarrow \eta = (\theta \to \eta) \wedge (\eta \to \theta)$.
- Let $\theta(\bar{x},y)$ be an $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ program and $t(\bar{x})$ be an $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ term.

– $\forall y{<}t\ \theta = \neg\exists y{<}t\ \neg\theta$.

– $\forall y\ \theta = \neg\exists y\ \neg\theta$.

Here $\neg, \exists, \forall$ are more binding than $\vee, \wedge$, which are in turn more binding than $\rightarrow, \leftrightarrow$.

In this module, since we will not study the run-times of programs, we could specify a program entirely using its input–output pairs. Nevertheless, a program written in terms of **if**, **then**, **par-for**, **par-while**, ... is usually easier to understand, especially when unravelled informally.

Example 1.4. Let $\theta(x, y)$ be the $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ program

$$\exists z{<}y\ \exists w{<}2^x\ (y = 2^{x+1}z + 2^x + w).$$

Unravelling the definitions, one sees that $\theta(x, y)$ roughly corresponds to

par-for $z \leftarrow 0, 1, \ldots, y-1$
 par-for $w \leftarrow 0, 1, \ldots, 2^x - 1$
  if $y = 2^{x+1}z + 2^x + w$
   then return `true`
  end if
 end par-for
end par-for
return `false`

For instance, notice $10 = 2^2 \times 2 + 2^1 + 0$. On the one hand, by considering $(z, w) = (2, 0)$, we see that $[\![\theta(1, 10)]\!] = \texttt{true}$. On the other hand, as one can directly verify, we have $[\![\theta(2, 10)]\!] = \texttt{false}$. More generally, for all inputs $a, b \in \mathbb{N}$,

$$[\![\theta(a, b)]\!] = \begin{cases} \texttt{true}, & \text{if the } a\text{th digit in the binary representation of } b \text{ is } 1; \\ \texttt{false}, & \text{if the } a\text{th digit in the binary representation of } b \text{ is } 0. \end{cases}$$

As in the example above, many programs do not need **par-while**. These programs are particularly well-behaved — for instance, they always return an answer. The availability of **par-while** makes the language strictly stronger, but it seems difficult to find a simple example demonstrating this.

Definition. A $\Delta_0(\mathrm{exp})$ *program* is an $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ program with no **par-while**. A Σ_1 *program* is an $\mathcal{L}_{\mathrm{A}}(\mathrm{exp})$ program of the form $\exists \bar{y}\ \theta(\bar{x}, \bar{y})$, where $\theta(\bar{x}, \bar{y})$ is a $\Delta_0(\mathrm{exp})$ program and $\bar{y}$ is a possibly empty tuple of variables.

Although Σ_1 programs are only a very special kind of computer algorithms, they are extremely powerful. In fact, it is postulated that, as far as subsets of $\mathbb{N}$ are concerned, the intuitive notion of algorithms can be captured by Σ_1 programs. This postulate is commonly accepted, but it cannot be proved or refuted mathematically because it involves the intuitive notion of algorithms which by nature cannot have a rigorous definition.

Church–Turing Thesis. For any $k \in \mathbb{N}$ and $S \subseteq \mathbb{N}^k$, the following are equivalent.

- There is an algorithm $\mathbb{A}$ such that

$$S = \{(a_1, a_2, \ldots, a_k) \in \mathbb{N}^k : \mathbb{A} \text{ on input } \bar{a} \text{ returns } \texttt{true}\}.$$

- There is a Σ_1 program $\theta(x_1, x_2, \ldots, x_k)$ such that

$$S = \{(a_1, a_2, \ldots, a_k) \in \mathbb{N}^k : [\![\theta(\bar{a})]\!] = \texttt{true}\}.$$

At various points, we will appeal to the Church–Turing Thesis to obtain Σ_1 programs that replace the algorithms we informally describe.

Challenge. Find a counterexample to the Church–Turing Thesis with reasonable justification.

Definition. Let $k \in \mathbb{N}$. A set $S \subseteq \mathbb{N}^k$ is *recursively enumerable*, or *r.e.* for short, if there is a Σ_1 program $\theta(x_1, x_2, \ldots, x_k)$ such that

$$S = \{(a_1, a_2, \ldots, a_k) \in \mathbb{N}^k : [\![\theta(\bar{a})]\!] = \texttt{true}\}.$$

The set $S \subseteq \mathbb{N}^k$ is *recursive* if both S and $\mathbb{N}^k \setminus S$ are r.e.

Remark 1.5. Some authors write 'computably enumerable', 'c.e.', and 'computable' for 'recursively enumerable', 'r.e.', and 'recursive' respectively.

Strictly speaking, it would be more appropriate to call our Σ_1 programs $\Sigma_1(\exp)$, but the two notions actually coincide in the cases we will consider.

Assignment 1.6. A Δ_0 *program* is a $\Delta_0(\exp)$ program in which no $\mathcal{L}_{\mathrm{A}}(\exp)$ term contains a subterm of the form 2^t, where t is a term. For example, the following are Δ_0 programs:

$$\begin{aligned} \mathrm{divides}(x, y) &= \exists z{\leqslant}y\ (y = xz); \\ \mathrm{prime}(y) &= y \geqslant 2 \wedge \forall x{\leqslant}y\ \big(\mathrm{divides}(x, y) \to x = 1 \vee x = y\big). \end{aligned}$$

Using these examples or otherwise, find a Δ_0 program $\theta(w)$ such that

$$\{a \in \mathbb{N} : [\![\theta(a)]\!] = \texttt{true}\} = \{2^b \in \mathbb{N} : b \in \mathbb{N}\}.$$

[5 points]

It is evident that the output of a program θ is closely related to the truth of θ as a formula. In the next lecture, we will investigate this relationship.