MATH3306

Set Theory & Mathematical Logic

Semester 2, 2022

Professor Benjamin Burton

Rohan Boas

Contents

1   Introductory notes

1.1   Gödel's incompleteness theorem

Theorem 1.1.1 (Gödel's incompleteness theorem, informal version)**.** *There are true mathematical statements that cannot be proven.*

"Proof". Take the statement "This statement has no proof."
Assume it is false. This implies that the statement has a proof. If the statement has a proof, it must be true, contradiction!
Assume it is true. This implies that the statement has no proof. Therefore, the statement cannot be proven. $\square$

1.2   The halting problem

We would like to be able to know if an algorithm or program will halt or will loop forever. Can we write an algorithm which can tell us whether or not a given program will halt on a given input? This is known as the halting problem. The halting problem is undecidable.

Proof by contradiction. Say there does exist some program H which solves the halting problem. Let us define H. H takes as inputs a program, x, and an input for that program, y.

$$H(x, y) = \begin{cases} \text{YES}, & \text{if } x \text{ halts on input } y \\ \text{NO}, & \text{if } x \text{ loops forever on input } y \end{cases}$$

Let us define a program Foo.

$$\text{Foo}(x) = \begin{cases} \text{loops forever}, & \text{if } H(x, x) \text{ is YES} \\ \text{halts}, & \text{if } H(x, x) \text{ is NO} \end{cases}$$

TODO: finish proof, maybe rewrite with diagram $\square$

1.3   Defining algorithms

In defining algorithms, Turing machines and recursive functions will be the primary focus. Grammars and code are also alternatives.

Definition 1.1 (Church-Turing thesis, informal version)**.** Any reasonable definitions of "algorithm" are equivalent.

2 Finite and deterministic state automata

2.1 Finite state automata

Definition 2.1 (Alphabet)**.** An alphabet A is a finite set of symbols.

Definition 2.2 (Word)**.** A word is a sequence of symbols from A.

Theorem 2.1.1. *Words can be concatenated. E.g. for words α representing "bob", and β representing "cat", $\alpha\beta\alpha$ represents "bobcatbob".*

Theorem 2.1.2. *The set of words length m is $A^m = A \times A \times ... \times A$.*

Theorem 2.1.3. *The empty word (i.e. of length 0) is ε.*

Theorem 2.1.4. *A^* is the set of all words over A. $A^* = \bigcup_{m \geq 0} A^m$.*

Theorem 2.1.5. *A^+ is the set of all non-empty words over A. $A^+ = \bigcup_{m \geq 1} A^m$.*

Definition 2.3 (Language)**.** A language is a subset of A^*

Definition 2.4 (Finite state automata)**.** A finite state automaton (FSA) can be defined as the 5-tuple (Q, F, A, τ, q_o) where

- Q is a finite set of states,

- $F \subseteq Q$ is the set of final/accepting states,

- A is a finite alphabet,

- $\tau \subseteq Q \times A \times Q$ is the set of transitions, and

- q_0 is the initial state.

Definition 2.5 (Computation)**.** A computation is a sequence $q_0 a_1 q_1 a_2 ... a_n q_n$ such that each $q_i a_{i+1} q_{i+1} \in \tau$.

Definition 2.6 (Successful)**.** A computation is successful if $q_n \in F$.

Definition 2.7 (Accepted)**.** A word $\alpha = a_1 a_2 ... a_n$ is accepted by the FSA if there is a successful computation $q_0 a_1 q_1 a_2 ... a_n q_n$.

Definition 2.8 (Recognised)**.** The language recognised by an FSA is the set of all words it accepts.

Remark 1. A FSA is like a backtracking search.

2.2 Deterministic finite state automata

Definition 2.9 (Deterministic finite state automata)**.** A deterministic state automaton (DFA) is an FSA where $\forall q \in Q, \forall a \in A, \exists! q' \in Q$ s.t. $(q, a, q') \in \tau$

Theorem 2.2.1. *The definition of DFA implies there exists a function $\delta : Q \times A \to A$.*

Theorem 2.2.2. *FSA and DFA solve the same problems.*
A DFA is already an FSA, and an FSA can be represented by a DFA of the reachable sets of states. Though, for an FSA with n states, the corresponding DFA has up to 2^n states.

Theorem 2.2.3. *FSA/DFA can perform addition.*

Theorem 2.2.4. *FSA/DFA cannot perform multiplication.*

Theorem 2.2.5. *To recognise an infinite language of a DFA with a finite complement, find the complement and swap accepting and rejecting states.*

3   Turing machines

Definition 3.1 (Turing machine)**.** A Turing machine (TM) is a FSA with infinite memory in the form of a tape.
A Turing machine is a tuple $(Q, F, {}_\sqcup, A, I, \tau, q_0)$ where

- Q is a finite set of states,

- $F \subseteq Q$ is the set of final/accepting states,

- $\sqcup$ is the "blank" symbol which is present on the tape where no new symbol has been written,

- A is a finite alphabet of all symbols that may be on the tape (including $\sqcup$),

- $I \subseteq A - \{\sqcup\}$ is a finite alphabet of all symbols from the input sequence,

- $\tau \subseteq Q \times A \times Q \times A \times \{L, R\}$ is the set of transitions, and

- q_0 is the initial state.

Definition 3.2 (Tape)**.** A tape is a tuple (a, α, β) where

- $a \in A$,

- $\alpha, \beta : \mathbb{N} \to A$, and

- $\alpha(i), \beta(i) \neq {}_\sqcup$ for only finitely many i.

Definition 3.3 (Configuration)**.** A configuration is a snapshot of the state.
A configuration is a tuple $(q \in Q, a, \alpha, \beta)$.

Definition 3.4 (Reachable)**.** A configuration c' is reachable from c in a single move if for $c = (q, a, \alpha, \beta)$, (in the case of moving to the right) $c' = (q', a(0), \alpha', \beta')$ where

- $\alpha'(i) = \alpha(i+1) \quad \forall i \in \mathbb{N}$,

- $\beta'(i) = \begin{cases} \beta(i-1) & \forall i \geq 1 \\ a' & i = 0 \end{cases}$, and

- $(q, a, q', a', R) \in \tau$.

Definition 3.5 (Computation)**.** A computation is a finite sequence of configurations $c_0 c_1 c_2 \ldots c_n$ s.t. c_i is reachable from c_{i-1} is a single move for all i.

Definition 3.6 (Terminal)**.** A configuration is terminal if no configuration is reachable from it. A Turing machine halts upon reaching such a configuration.

Definition 3.7 (Acceptinng)**.** A Turing machine accepts a word $w \in I^*$ if there exists some computation from the initial state c_w to some final state.

Definition 3.8 (Recognised)**.** The language recognised by a Turing machine is the set of all words $w \in I^*$ that the Turing machine accepts.

Definition 3.9 (Deterministic)**.** A Turing machine is deterministic if for all q and a, there exists at most one tuple $(q, a, q', a', d) \in \tau$.

Theorem 3.0.1. *The transitions of a Turing machine are partial functions of the form* $\delta : Q \times A \to Q \times A \times \{L, R\}$

4