MATHEMATICAL LOGIC FOR COMPUTER SCIENCE
(A.Y. 20/21)

HANDOUT N. 9

Abstract. Inseparable sets. Turing Machines. Undecidability results.

1. General Decision Problems

We aim at establishing the two following undecidability results.

Theorem 1.1 (Church's Theorem)**.** *The set of all valid sentences is not decidable. The set of all satisfiable sentences is not decidable.*

What if we are interested only in finite models?

Theorem 1.2 (Trakhtenbrot's Theorem, part 1)**.** *The set of all sentences satisfiable in some finite model is not decidable. The set of all sentences true in all finite models is not decidable.*

Note that FinSat (set of all sentences satisfied in some finite model) is easily seen to be listable by an algorithm (i.e. semi-decidable): enumerate all pairs $(\mathfrak{A}, S)$ where $\mathfrak{A}$ is a finite structure and S is a sentence (both in a fixed relational finite language). For each pair answer algorithmically the question whether $\mathfrak{A} \models S$? If the answer is yes then put S in the list. Else do nothing and proceed.

As already observed, for the above informal algorithm to work we need to make sure that the problem with input $(\mathfrak{A}, S)$ and output YES/NO according to whether $\mathfrak{A} \models S$ is decidable. This problem is known as the Model-Checking Problem. Its decidability can be established by the following naive recursive algorithm, based on Tarski's definition of truth. The recursion is on the syntactical complexity of the sentence S. If S is atomic, it can be easily checked whether it holds in a given finite structure $\mathfrak{A}$. Similarly for the boolean case. The only non-trivial case is the quantifier case. Suppose S is $\exists x F(x)$. We then run the following cycle: for each element $a \in A$ we decide $(\mathfrak{A}, a) \models F^c$. This is a recursive call of the algorithm with inputs $(\mathfrak{A}, a)$ (the expansion of $\mathfrak{A}$ by a) and F^c (the sentence obtained from $F(x)$ by replacing x with a new constant c).

From the above Theorem, observing that if both a set and its complement are listable then the set is decidable we obtain the following corollaries.

Corollary 1.3 (Trakhtenbrot's Theorem, part 2)**.** *The complement of the set of all sentences satisfiable in some finite model is not listable.*

The complement of FinSat and FinVal are equivalent, using the map given by negation: $S \notin$ FinSat iff it has no finite model iff $\neg S$ is true in all finite models iff $\neg S \in$ FinVal.

We obtain this further corollary.

Corollary 1.4 (Trakhtenbrot's Theorem)**.** *The set of sentences valid in all finite structures is not listable (semi-decidable).*

2. Algorithmic undecidability

A model of computation or programming system can be thought of as a modern programming language. By a computable function we can safely mean any function computed by a program in some standard programming language.

From the abstract point of view a model of computation (or programming system) consists in a countable list of programs $(P_i)_{i \in \mathbf{N}}$. For the moment we can restrict our attention to programs operating on one natural

Notes by Lorenzo Carlucci, carlucci@di.uniroma1.it.

number argument. By $P_i(x)$ we denote the application of program number i to argument x. By $P_i(x)\downarrow$ we indicate that the program ends within a finite time and gives an output. If the output is y we denote this by $P_i(x) = y$ or $P_i(x)\downarrow y$. By $P_i(x)\uparrow$ we denote the fact that the program loops or diverges.

For all natural programming systems it is true that the function

$$i, x \mapsto P_i(x)$$

can itself be computed by a program. This corresponds to the very idea of a programmable computer and to Turing's original idea of a universal algorithm simulating all other algorithms.

In this abstract framework we can easily isolate the first examples of algorithmically undecidable problems, i.e., sets whose characteristic function cannot be computed by any program in the system.

The most fundamental one is the Halting Problem, defined as

$$H = \{(i, x) \,:\, P_i(x)\downarrow\}.$$

A typical useful variant is the following

$$K = \{i \,:\, P_i(i)\downarrow\},$$

also denoted by $\varnothing'$, which takes advantage of the fact that (codes of) programs can be given as inputs to programs.

Proposition 2.1 (Undecidability of the Halting Problem)**.** *The sets K and H are algorithmically undecidable.*

Proof. Suppose that K is decidable and let A be an algorithm such that for all $n \in \mathbf{N}$, $A(n) = 1$ if $n \in K$ and $A(n) = 0$ if $n \notin K$.

Consider the following procedure B: on input n, B computes $A(n)$. If $A(n) = 1$ then B loops; if $A(n) = 0$ then B halts and outputs some value, say 4 (this choice is irrelevant).

It is easy to see that, since A is algorithmic, also B is algorithmic. In terms of our system P, given a program P_i that computes A it is easy to write a program P_j that computes B. Let's fix an index e such that P_e computes B.

If $A(e) = 1$ then, by definition of P_e, $P_e(e)\uparrow$. Then by choice of A, $A(e) = 0$.

If $A(e) = 0$ then, by definition of P_e, $P_e(e)\downarrow$. Then by choice of A, $A(e) = 1$.

This contradiction shows that no A can exist deciding K.

The proofs for H and K_0 are left as exercises. □

Many undecidability results in Logic can be obtained by reduction to the Halting Problem. In order to rigorously define the reductions we will need to work with a well-defined suitable model of computation, rather than with our generic $(P_i)_{i\in\mathbf{N}}$.

3. Turing Machines

To prove the previous theorems we will introduce a specific model of computation (Turing Machines). We think of Turing Machines as devices with one "head" that scans the cells of a one-way infinite-to-the-right tape. The formal definition follows.

Figure 1. Alan Turing (1912-1954)

A **Turing Machine** M consists of

- A finite set of states, Q.
- A finite alphabet Σ, containing a blank symbol $\#$. The input alphabet is a subset $\Gamma \subseteq \Sigma \setminus \{\#\}$.
- A special state, $q_0 \in Q$ called the initial state.
- A set $F \subseteq Q$ of final states
- A (partial) function $\delta : (Q \setminus F) \times \Sigma \to Q \times \Sigma \times \{L, 0, R\}$ called the transition function.

We rule out by definition cases in which the transition function would require that the head moves to the left of the leftmost cell.

A **snapshot** is a triple $C = (q, p, w) \in Q \times \mathbf{N} \times \Sigma^*$. This describes the situation in which M is in state q, scanning cell number p, while the tape contains the word w followed by blank symbols.

The transition function δ induces a partial function on snapshots $C \mapsto Next(C)$.

A **computation** of M on x is a sequence $(C_i)_{i \in \mathbf{N}}$ of snapshots such that $C_0 = (q_0, 0, x)$, and $C_{i+1} = Next(C_i)$ for all $i \in \mathbf{N}$.

M **halts** on x if the computation of M on x is finite and the last snapshot ends with a final state (NB This means that we don't consider as halting computations computation that end in a blind corner with some non-final state on which δ is undefined). In this case we write $M(x) \downarrow$ and $M(x) \downarrow = w$ with w the word on the tape at the end of the computation.

In all other cases we say that M **does not halt**, ~~**diverges** on x, and we write $M(x) \uparrow$~~ we will reserve the term "diverge" for infinite computations.

In case F is partitioned in two sets F_a and F_r we speak of a **Turing Acceptor**. We say that M **accepts** x if the computation of M on x ends in a state in F_a and M **rejects** x if the computation of M on x ends in a state in F_r.

In the framework of Turing Machines we have the corresponding notions of decidable and semi-decidable set.

A set $L \subseteq \Gamma^*$ is **decidable** if there exists a Turing Acceptor M such that: if $x \in L$ then M accepts x; and if $x \notin L$ then M rejects x.

A set $L \subseteq \Gamma^*$ is **semi-decidable** (or **computably enumerable**, or **c.e.**) if there is a Turing Machine M such that $M(x) \downarrow$ exactly on the xs that belong to L.

Within this framework Turing isolated the first example of a computably unsolvable problem. This is the Halting Problem, which, in its most general form, asks about halting of a program on some input. In the language of Turing Machines, this is the set

$$H = \{(i, x) \ : \ M_i(x) \downarrow\}$$

We can define some useful variants of the Halting Problem:

$$H_\epsilon = \{i \ : \ M_i(\epsilon) \downarrow\}.$$

$$K = \varnothing' = \{i \ : \ M_i(i) \downarrow\}.$$

Proposition 3.1. *The sets H, H_ϵ and K are undecidable.*

4. Describing the computation by a formula

We show how to effectively associate to each Turing Machine M a formula G_M that describes the computation of M on the empty input ϵ. The precise meaning of "describes" will be apparent from the construction.

Let M be a Turing machine (with accepting and rejecting states) defined by $Q = \{q_0, \ldots, q_r\}$, $\Sigma = \{a_0, \ldots, a_s\}$ with $a_0 = blank$, $F = F_a \cup F_r$, δ.

The formula G_M associated to M is in the language $\mathcal{L} = \{0, f, q, p, w\}$, where f, q, p are function symbols with one argument and w is a binary function symbol. The intended meaning of these symbols is the following.

- $f(t)$ **is the successor** of t,
- $q(t) = i$ indicates M **at time** t **is in state** q_i,
- $p(t) = i$ indicates **at time** t **the head of** M **is under cell** i.

- $w(s,t) = i$ indicates **at time t cell s contains symbol a_i**.

Using f and the constant 0 we can associate a closed term to each natural number: 0 to 0 and $f^n(0)$ to $n > 0$, where $f^n(0)$ denotes the term $f(f(\dots(f(0))\dots))$ with n occurrences of f.

Keeping in mind these intuitive interpretations we define G_M as follows.
N.B. the definition of G_M is algorithmic in M and does not depend on the semantics of the symbols used.

G_M is the conjunction of sentences, $START$ and $STEP$ describing, respectively, the initial configuration and the intermediate configurations

$$G_M := START \wedge STEP.$$

Sentence $START$ describes the initial configuration: at time 0 the machine is in state q_0, the head is under cell 0, and all cells have the blank symbol.

$$START := q(0) = 0 \wedge p(0) = 0 \wedge \forall x(w(x,0) = 0).$$

The formula $STEP$ is the conjunction of two sentences, $NOCHANGE$ e $CHANGE$, describing resp. what happens in the cell involved in a computation step and in the cells that are not involved.

$$STEP := NOCHANGE \wedge CHANGE$$

Sentence $NOCHANGE$ expresses that the content of cells not involved in the computation at step y does not change at the next time step.

$$NOCHANGE := \forall x \forall y(p(y) \neq x \rightarrow w(x, f(y)) = w(x, y))$$

The formula $CHANGE$ describes the content of cells involved in the current computation step. *The sentence has a conjunct for each possible transition and each conjunct is universally quantified over all time steps.*

$$CHANGE := \bigwedge_{\delta:(q_i,a_j)\mapsto(q_k,a_\ell,m)} \forall y(PRE_{i,j}(y) \rightarrow POST_{k,\ell,m}(y))$$

The formula $PRE_{i,j}(y)$ (with one free variable y) expresses *the pre-condition of the transition*, i.e. that at time y machine M is in state q_i reading symbol a_j.

$$PRE_{i,j}(y) := (q(y) = f^i(0) \wedge w(p(y), y) = f^j(0))$$

Formula $POST_{k,\ell,m}(y)$ (with one free variable y) expresses *the post-condition of the transition*, i.e. that at time $f(y)$, the successor of y, machine M is in state q_k, has written a_ℓ and also describes the head position of M at time $f(y)$.
This latter info is in the conjunct $MOVE_m$.

$$POST_{k,\ell,m}(y) := (q(f(y)) = f^k(0) \wedge w(p(y), f(y)) = f^\ell(0) \wedge MOVE_m)$$

$$MOVE_m := \begin{cases} p(f(y)) = p(y) & \text{if } m = 0 \\ p(f(y)) = f(p(y)) & \text{if } m = 1 \\ \exists z(f(z) = p(y) \wedge p(f(y)) = z) & \text{if } m = -1 \end{cases}$$

The above formula G_M describes the computation of M on the empty input ϵ. To make this precise, let's first show that the real computation of M on ϵ can be turned into a model of G_M.

We can associate to M a **canonical model $\mathfrak{A}_M$** describing the real computation of M on ϵ. The association is not effective and we are not able to decide, given the description of M, if $\mathfrak{A}_M$ is infinite or finite!

Model $\mathfrak{A}_M$ has domain $\mathbf{N}$ if the computation of M on ε is divergent (infinite), and the finite set $\{0, 1, \dots, n\}$ if the computation of M on ε is convergent (finite), where n is the max among the number of states, symbols, steps of computation, and cells used during the computation.

We interpret the function symbols in the intuitive way:

- $f^{\mathfrak{A}_M}(t) = t + 1$ if $t + 1 \in A$, else $f(t) = t$.
- $q^{\mathfrak{A}_M}(t) = i$ iff M at time t is in state q_i in the computation of M on ϵ.
- $p^{\mathfrak{A}_M}(t) = i$ iff the head of M at time t is scanning cell i in the computation of M on ϵ.
- $w^{\mathfrak{A}_M}(s,t) = i$ iff at time t cell s contains symbol a_i in the computation of M on ϵ.

It is easy to see that $\mathfrak{A}_M \models G_M$.

The formula G_M says nothing about how the computation of M on ϵ ends. We can express many different scenarios describing the end of the computation. For example, we can express the fact that the computation of M on ϵ never ends; or that it ends in a final state; or that it ends in a non-final state.

Some of these choices will give us a formula that reflects a number of interesting satisfiability/non-satisfiability properties and will allow us to reduce logical problems to computational undecidable problems.

5. Algorithmically inseparable sets

Definition 5.1 (Inseparability)**.** Two sets A, B are *algorithmically (or recursively) inseparable* if there is no algorithm that can discriminate between A and B. In other words, if there is no computable function f such that for every $a \in A$, $f(a) = 1$ and for every $b \in B$, $f(b) = 0$. In other words, there is no decidable set C such that $A \subseteq C$ and $B \subseteq \mathbf{N} \setminus C$. In applications of the concept we are interested in cases where $A \cap B = \varnothing$ (the other case being trivial).

The usefulness of the inseparable sets in undecidability proofs has two reasons.

The first is that if $\bar{A}$ is the complement of A ($\bar{A} = \mathbf{N} \setminus A$) and $A, \bar{A}$ are inseparable, then A is algorithmically undecidable: there is no computable function f such that for every $x \in A$, $f(x) = 1$ and for every $x \notin A$, $f(x) = 0$.

Proposition 5.2. *$(A, \bar{A})$ is inseparable if and only if A is undecidable.*

The second reason is that one can define a good concept of reduction between problems. The property of being algorithmically inseparable transfers from a pair of sets to the other through effective (algorithmic) transformations. Let C and D be the algorithmic images of A and B, i.e., sets such that $C \cap D = \varnothing$, and such that there is a computable function f such that $f(A) \subseteq C$ and $f(B) \subseteq D$. Then if A and B are algorithmically inseparable, so too are C and D.

Figure 2. Emil Post (1897-1954)

Proposition 5.3. *Let A, B be inseparable. Let C, D disjoint. Let f be an algorithm such that $f(A) \subseteq C$ and $f(B) \subseteq D$. Then C, D are inseparable.*

Proof. Exercise! □

We will make use of a particular pair of inseparable sets. Suppose that we have fixed a reasonable model of computation for the computable functions $(P_i)_{i \in \mathbf{N}}$. Assume that each computable function has at least one program P_i that calculates it and that the function computed by the program P_i is computable, for each i. We denote with $P_i(n)$ the result (possibly undefined) of applying the procedure i-th to input n. We assume that the application $i, x \mapsto P_i(x)$ is computable (this is the case in all usual models of computation).

We define two sets as follows.

$$S_0 = \{i : P_i(i) = 0\}, \quad S_1 = \{i : P_i(i) = 1\}.$$

All undecidability results in this section will be obtained by reduction to this pair.

Corollary 5.4. *S_0 and S_1 are algorithmically inseparable.*

Proof. Suppose otherwise and let C be a decidable set that separates S_0 and S_1, i.e.,

$$S_0 \subseteq C$$

and

$$S_1 \subseteq \mathbf{N} \setminus C.$$

Let P_e be a program that computes the characterisic function of C. If $e \in S_0$ then $P_e(e) = 0$ and therefore $e \notin C$. But $S_0 \subseteq C$ by assumption. If $e \in S_1$ then $P_e(e) = 1$ and therefore $e \in C$. But $S_1 \subseteq \mathbf{N} \setminus C$ by assumption. □

We can prove a stronger result.

The assertion that S_0 and S_1 are inseparable implies the following weaker claim: if A and B are two computably enumerable sets such that A and B are disjoint, $S_0 \subseteq A$ and $S_1 \subseteq B$, then there is at least an element in $\mathbf{N} \setminus (A \cup B)$. Otherwise, we would have $B = \mathbf{N} \setminus A$ and therefore A would be a computable separator for S_0 and S_1.

A statement of the form: For all ... there exists ..., as the one obtained always implicitly *defines a function*, taking from the universally quantified parameters (the input) to one of the existing witnesses of the existential (the output). In this cases we can always ask whether the statement *holds effectively*, i.e.: is the function algorithmic? I.e., is there an algorithm that takes from the universally quantified parameters to the witness of the existential?

In the case at hand the statement is: For all sets A and B such that .... there exists a number m neither in A nor in B. How do we deal with (possibly infinite) sets as inputs to algorithms? Note that the sets in questions are computably enumerable, meaning that they are the domain of an algorithm: there exists an $a \in \mathbf{N}$ such that program (or machine) number a has domain A and there exists a $b \in \mathbf{N}$ such that program (or machine) number b has domain B. Thus, we can quantify over computably enumerable As and Bs by quantifying over their numerical codes.

Denote by $dom(P_i)$ the set of inputs on which program P_i terminates. Recall that a set S is computably enumerable iff it's the domain of a computable function, i.e., iff there exists an i such that $S = dom(P_i)$.

The statements becomes: For all $a, b \in \mathbf{N}$ such that $dom(P_a) \cap dom(P_b) = \varnothing$ and $S_0 \subseteq dom(P_a)$ and $S_1 \subseteq dom(P_b)$ there exists $m \notin dom(P_a) \cup dom(P_b)$.

This is implicitly defines a function of type $\mathbf{N} \times \mathbf{N} \to \mathbf{N}$ and we can ask whether this function is algorithmic. In this case we say that we ask for an *effective version* of the statement.

We will prove below that the effective version is true: this means that there is an algorithm that takes as input to codes a, b of programs and, in case $dom(P_a)$ extends S_0 and $dom(P_b)$ extends S_1 and $dom(P_a)$ is disjoint from $dom(P_b)$, outputs a witness to the fact that $\mathbf{N} \neq dom(P_a) \cup dom(P_b)$.

Proposition 5.5. *There is an algorithm α such that, for all $a, b \in \mathbf{N}$, if the following points hold then $\alpha(a, b)$ is defined and neither in the domain of P_a nor in the domain of P_b.*

(1) $dom(P_a) \cap dom(P_b) = \varnothing$.
(2) $S_0 \subseteq dom(P_a)$.
(3) $S_1 \subseteq dom(P_b)$.

Proof. Given two indices a, b, the following procedure ψ is algorithmic:

Given y, enumerate simultaneously $dom(P_a)$ and $dom(P_b)$.

If y first appears in $dom(P_a)$, output 1, if y first appears in $dom(P_b)$, output 0.

If y doesn't appear in $dom(P_a)$ and in $dom(P_b)$, the procedure diverges.

A program for ψ can be obtained algorithmically from programs P_a and P_b, in other words: an index for ψ can be obtained algorithmically from indices a and b in a *uniform* way.

Therefore there exists a computable function f such that $P_{f(a,b)}$ computes ψ.

Suppose $f(a, b) \in dom(P_a)$. Then $\psi(f(a, b)) = 1$ and therefore $P_{f(a,b)}(f(a, b)) = 1$. But then $f(a, b)$ is in S_1. By hypothesis, $S_1 \subseteq dom(P_b)$, which contradicts $dom(P_a) \cap dom(P_b) = \varnothing$.

Suppose $f(a, b) \in dom(P_b)$. Then $\psi(f(a, b)) = 0$ and thus $P_{f(a,b)}(f(a, b)) = 0$. But then $f(a, b)$ is in S_0. By hypotehsis, $S_0 \subseteq dom(P_a)$, which contradicts $dom(P_a) \cap dom(P_b) = \varnothing$.

Therefore $f(a, b)$ is neither in $dom(P_a)$ nor in $dom(P_b)$. □

We aim at the following logical undecidability results.

Theorem 5.6. *The following sets/problems are algorithmically undecidable*

(1) FinSat*, the set of sentences with some finite model.*
(2) InfAx*, the set of sentences with infinite models but no finite model.*
(3) UnSat*, the set of sentences with no model.*

We will obtain these results by showing the following inseparability results:

Theorem 5.7. *The following pairs are inseparable*

(1) (FinSat, UnSat)
(2) (InfAx, FinSat)
(3) (UnSat, InfAx).

Theorem 5.6 follows from Theorem 5.7 by the observation that A is undecidable if and only if $(A, \overline{A})$ is an inseparable pair, and the following observations about reductions:

- The inseparability of (FinSat, UnSat) implies the inseparability of (FinSat, $\overline{\text{FinSat}}$) by reduction since UnSat $\subseteq \overline{\text{FinSat}}$.
- The inseparability of (InfAx, FinSat) implies the inseparability of (InfAx, $\overline{\text{InfAx}}$) by reduction since FinSat $\subseteq \overline{\text{InfAx}}$.
- The inseparability of (UnSat, InfAx) implies the inseparability of (UnSat, $\overline{\text{UnSat}}$) by reduction since InfAx $\subseteq \overline{\text{UnSat}}$.

Consider the following slight variants of the Halting Problem:

$$E^1 = \{i \ : \ M_i \text{ accepts the empty input}\}$$

$$E^0 = \{i \ : \ M_i \text{ rejects the empty input}\}$$

$$E^\infty = \{i \ : \ M_i \text{ the computation of } M \text{ on the empty input is infinite and non periodic}\}$$

M_i indicates the Turing machine with code i.

Proposition 5.8. *The sets E^1, E^0 and E^∞ are pairwise inseparable.*

Proof. We first prove that E^1 and E^∞ are inseparable. We show that if S is a decidable set that separates E^1 and E^∞, then the Halting Set $K = \{i \ : \ M_i \text{ halts on the empty input}\}$ is reducible to S, that is there exists a total computable function $f : \mathbf{N} \to \mathbf{N}$ such that for all $i \in \mathbf{N}$

$$i \in K \Leftrightarrow f(i) \in S.$$

S separates E^1 and E^∞ iff, for all i, if $i \in E^1$ then $i \in S$ and if $i \in E^\infty$ then $i \notin S$.

From a code i of a Turing machine we can obtain algorithmically (and uniformly) a code $h(i)$ such that the Turing machine $M_{h(i)}$ behaves as follows:

- $M_{h(i)}$ simulates M_i *and keeps a counter of the computation steps performed by* M_i, and
- If M_i halts, then $M_{h(i)}$ halts and *accepts*.

If $M_{h(i)}(x)$ does not terminate, then its computation is non periodic (this is ensured by keeping a counter).
If $M_{h(i)}(x)$ terminates, then it also accepts.
The function h gives a reduction from K to S:
$i \in K$ iff M_i converges on the empty input iff $M_{h(i)}$ accepts the empty input iff $h(i) \in E^1$.
Moreover, $h(i) \in E^1$ iff $h(i) \in S$: if $h(i) \in E^1$ then $h(i) \in S$ because S separates E^1 and E^∞. If $h(i) \notin E^1$, then, by construction of $M_{h(i)}$ it must be the case that $h(i) \in E^\infty$ and thus $h(i) \in \bar{S}$ since S separates E^1 and E^∞.
Thus to decide $i \in K$ it is sufficient to decide $h(i) \in S$. So S is not computable.

We next show that E^0 and E^1 are inseparable by reducing them to the pair (K^0, K^1), where

$$K^0 = \{i \ : \ M_i \text{ rejects input } i\},$$

and

$$K^1 = \{i \ : \ M_i \text{ accepts input } i\}.$$

That (K^1, K^0) is inseparable just as we showed that (S_0, S_1) is inseparable.

The reduction $(K^1, K^0) \leq (E^1, E^0)$ is as follows. From a code i we compute a code $h(i)$ such that machine $M_{h(i)}$ on input ε simulates M_i on input i. We then have that if $i \in K^1$ then $h(i) \in E^1$, and if $i \in K^0$ then $h(i) \in E^0$.

□

6. Logical Undecidability Theorems

We show how to add an "end of computation sentence" to G_M so that the following implications are satisfied.

(I) If M accepts the empty input, then F_M has a finite model.
(II) If M rejects the empty input, then F_M is unsatisfiable.
(III) If the computation of M on the empty input does not terminate and is not periodic, then F_M is satisfiable and has no finite model.

In terms of reductions this shows the following.

Proposition 6.1. *The following reductions hold.*

- $(E^1, E^0) \leq (\text{FinSat}, \text{UnSat})$,
- $(E^\infty, E^1) \leq (\text{InfAx}, \text{FinSat})$,
- $(E^0, E^\infty) \leq (\text{UnSat}, \text{InfAx})$,

The idea is that F_M describes the computation of M on the empty string and excludes that this computation terminates in a non-accepting state.

F_M is the conjunction of three sentences, $START$, $STEP$ and END describing, respectively, the initial configuration, the intermediate configurations and the final configuration of the computation of M on the empty string.

$$F_M := START \wedge STEP \wedge END.$$

The sentence END expresses the termination conditions for the computation of M and *excludes that M terminates in a non-accepting state.*

$$END := \bigwedge_{\delta(q_i,a_j)\uparrow, q_i \notin F_a f} \forall y \neg PRE_{i,j}(y).$$

We can associate to M a canonical model $\mathfrak{A}_M$ describing the real computation of M on ϵ. The association is not effective and we are not able to decide, given the description of M, if $\mathfrak{A}_M$ is infinite or finite! We can ensure that if F_M is satisfiable, then $\mathfrak{A}_M$ is a model of F_M.

Model $\mathfrak{A}_M$ has domain $\mathbf{N}$ if the computation of M on ε is divergent (infinite), and the finite set $\{0, 1, \ldots, n\}$ if the computation of M on ε is convergent (finite), where n is the max among the number of states, symbols, steps of computation, and cells used during the computation. We interpret the function symbols in the intuitive way:

- $f^{\mathfrak{A}_M}(t) = t + 1$ if $t + 1 \in A$, else $f(t) = t$.
- $q^{\mathfrak{A}_M}(t) = i$ iff M at time t is in state q_i.
- $p^{\mathfrak{A}_M}(t) = i$ iff the head of M at time t is scanning cell i.
- $w^{\mathfrak{A}_M}(s, t) = i$ iff at time t cell s contains symbol a_i.

If M on ϵ terminates in an accepting state, then the model $\mathfrak{A}_M$ is finite and satisfies F_M. Thus in this case F_M is finitely satisfiable. This proves point (I) of our Theorem.

If M on ϵ diverges (the computation is infinite) then the model $\mathfrak{A}_M$ is infinite and satisfies F_M.

To complete the proof of points (II) and (III) we have to show:

- If M rejects ϵ, then F_M has no model, and
- If the computation of M on ϵ is infinite and not periodic, then F_M has no finite models (we already know it has a infinite model, $\mathfrak{A}_M$).

The contrapositive is as follows:

- If F_M has a model, then M does not reject ϵ, and
- If F_M has finite models then the computation of M on ϵ is not infinite and not-periodic (i.e., it is finite or periodic).

As done previously, we define a notion that allows to reason on arbitrary models of F_M. A model *forces a configuration at time t* if it satisfies the formulas needed to describe the configuration, relatively to a time parameter representing t.

The formal definition is as follows.

Definition 6.2. Let $\mathfrak{B} = (B, 0, f, q, p, w)$ be a structure for the language of F_M. $\mathfrak{B}$ *forces at time t configuration* (q_i, j, w), with $w = a_{i_0} \dots a_{i_m} \in \Sigma^*$, iff the following hold:

(a) $\mathfrak{B} \models q(f^t(0)) = f^i(0)$,
(b) $\mathfrak{B} \models p(f^t(0)) = f^j(0)$,
(c) For all $k \leq m$, $\mathfrak{B} \models w(f^k(0), f^t(0)) = f^{i_k}(0)$,
(d) For all $k > m$, $\mathfrak{B} \models w(f^k(0), f^t(0)) = 0$.

Suppose $\mathfrak{B}$ models F_M. We prove the following points.

(A) $\mathfrak{B}$ forces $C_0 = (q_0, 0, \epsilon)$ at time 0. This follows from $\mathfrak{B} \models START$.

(B) If $\mathfrak{B}$ forces at time t a configuration C_t that is not accepting, then at time $t+1$ $\mathfrak{B}$ forces $C_{t+1} = Next(C_t)$.

Since $\mathfrak{B} \models F_M$, we have that $\mathfrak{B} \models END$. Therefore there is no $b \in B$ such that q_i, a_j exist with $q_i \notin F_a$ and δ undefined on (q_i, a_j) and $\mathfrak{B} \models PRE_{i,j}(y)[\binom{y}{b}]$.

Therefore every non-accepting configuration C_t forced by $\mathfrak{B}$ at time t is of type (q_i, j, w) where $w \in \Sigma^*$ and q_i is a *non-final state* and δ *is defined* on (q_i, a_h), where h is such that at time t the cell number j contains symbol a_h.

Since δ is defined on (q_i, a_h), $Next(C_t)$ is defined. Let k, ℓ, m such that $\delta : (q_i, a_h) \mapsto (q_k, a_\ell, m)$.

Since $\mathfrak{B} \models CHANGE$, and the premise $PRE_{i,j}(f^t(0))$ is satisfied in $\mathfrak{B}$, it must be the case that $\mathfrak{B} \models POST_{k,\ell,m}(f^t(0))$. This implies that $\mathfrak{B}$ forces at time $t+1$ the configuration $Next(C_t)$, i.e., (q_k, j', w') where j' is $j + m$ and w' is obtained from w by substituting a_h with a_ℓ in the relevant position.

~~In particular the two observations above, (A) and (B) hold for the canonical model $\mathfrak{A}_M$ mirroring the true computation of M on ϵ.~~ This comment is out of place: in fact, the above argument on a generic model of F_M implies that if F_M is satisfied, then the real computation of M cannot end in a rejecting state, without assuming that the canonical model satisfies F_M.

We then have:

If M on ϵ halts in a rejecting state, then F_M is unsatisfiable. This proves point (II) of the Theorem.

Suppose that F_M has a finite model but that the computation of M on ε is not finite. We prove that it must be periodic. Let $\mathfrak{B}$ be a finite model of F_M. By the above observations and since $\mathfrak{B} \models F_M$, we have that $\mathfrak{B}$ forces a periodic computation, since it is finite. This proves point (III) of the Theorem.

Remark 6.3. We showed FinSat, InfAx and UnSat are undecidable if the formulas are in a language with three unary and one binary function. For other (simpler) languages the sets can be decided. A complete characterization is known.