Now run R on $\langle M_1 \rangle$
If R accepts then accept
If R rejects then reject"

We claim that it decides $Halt_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that halts on } w\}$

If M halts on w $\Rightarrow$ M_1 halts on every input
If M does not halt on w $\Rightarrow$ M_1 does not halt on w.
$\Rightarrow$ M halts on w $\Leftrightarrow \langle M_1 \rangle \in L$

$\Rightarrow$ R accepts $\langle M_1 \rangle \Rightarrow$ M halts on w
R rejects $\langle M_1 \rangle \Rightarrow$ M does not halt on w.

(4) Example: Show that $E_{TM} = \{\langle M \rangle \mid M \text{ is a TM with } L(M) = \emptyset\}$ is NOT decidable.
Answer (by using A_{TM}):
Suppose the contrary: that there exists a TM called R that decides E_{TM}.
Then N = "on input $\langle M, w \rangle$
Construct the TM:
M_1 = "on input x,
If $x \neq w$, reject.
If $x = w$, then run M on x.
If it accepts $\Rightarrow$ accept".
Run R on $\langle M_1 \rangle$.
If it accepts, then reject.
If it rejects, then accept"

If R accepts $\langle M_1 \rangle \Rightarrow \langle M_1 \rangle \in L \Rightarrow L(M_1) = \emptyset \Rightarrow$ M does not accept w $\Rightarrow \langle M, w \rangle \notin A_{TM}$.
If R rejects $\langle M_1 \rangle \Rightarrow \langle M_1 \rangle \notin L \Rightarrow L(M_1) \neq \emptyset \Rightarrow$ M accepts w $\Rightarrow \langle M, w \rangle \in A_{TM}$

[PAUL HUSSMAN]

COMP 330
NOV 3 2015

Recall: The following languages are NOT DECIDABLE:

So Far
- $A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM accepting } w\}$ (proved by diagonalization)
- $Halt_{TM} = \{\langle M, w\rangle \mid M \text{ halts on } w\}$ (proved by reduction)
- $Decider = \{\langle M \rangle \mid M \text{ halts on every input}\}$ (proved by reduction)
- $E_{TM} = \{\langle M \rangle \mid M \text{ is a TM with } L(M) = \emptyset\}$ (proved by reduction)

Reduction: Suppose that we know A is undecidable. We want to show that B is undecidable.
We show that (B is decidable $\Rightarrow$ A is decidable)
We find an algorithm that converts x to f(x) such that
$x \in A \Leftrightarrow f(x) \in B$
Then if R is a TM that decides B then we would be able to decide A:
On input x:
Compute f(x)
Run R on f(x) and: if it accepts, accept
if it rejects, reject.

[Picture: x → f → f(x) → R]

(5) Example: Show that $EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are Turing Machines and } L(M_1) = L(M_2)\}$ is NOT decidable.
Answer: Suppose that R is a TM that decides EQ_{TM}. Then the following TM would decide E_{TM}: N = "On input $\langle M \rangle$:
Construct the TM M_1 as M_1 = "reject every input"
Then run R on $\langle M, M_1 \rangle$. If it accepts → accept
rejects → reject

$\langle M \rangle \rightarrow$ [f] $\xrightarrow{\langle M, M_1 \rangle}$ [R] →

Then $L(M_1) = \emptyset$ so $\langle M, M_1 \rangle \in EQ_{TM} \Leftrightarrow L(M) = L(M_1) = \emptyset \Leftrightarrow \langle M \rangle \in E_{TM}$

COMP 330
NOV 3 CONT'D

PAUL HUSSMAN

DEFINITION: Let M be a TM and w be an input string.
An ACCEPTING COMPUTATIONAL HISTORY of M on w is the following sequence of configurations:

$$C_1 \Rightarrow C_2 \Rightarrow C_3 \Rightarrow \dots \Rightarrow C_\ell$$

C_1: Initial Configuration of M on w; C_ℓ: an accept configuration.

- REJECTING COMPUTATIONAL HISTORY is similar but C_ℓ would be a reject configuration.

- Remark: If M loops on w then there is NO computational history of M on w.
- Remark: Every deterministic Turing Machine has at most one computational history on every input w.
  - Non-Deterministic Turing Machines can have several computational histories on every input.

DEFINITION: A Linear Bound Automaton (LBA) is a restricted TM in which the tape head is not allowed to move off the input area. If the tape head tries to move further to the right, it will stay in its place.

We want to investigate the decidability of

$$A_{LBA} = \{\langle M, w\rangle \mid M \text{ is an LBA that accepts } w\}$$

Lemma: Let M be an LBA with q states and g symbols on its tape. There are exactly $q \cdot n \cdot g^n$ possible configurations for M on an input w with $|w| = n$.

Proof: every configuration consists of
- the current state ⇒ (q choices)
- the position of the tape head ⇒ (n choices)
- the content of the tape. ⇒ (g^n possibilities)

Multiplying these numbers gives us qng^n.

Theorem: $A_{LBA} = \{\langle M, w\rangle \mid M \text{ is an LBA that accepts } w\}$ is decidable.

Proof: First, we notice that a computational history can not contain repeated configurations.

$C_1 \Rightarrow \dots \Rightarrow C \Rightarrow \dots \Rightarrow C \Rightarrow \dots \Rightarrow C \Rightarrow \dots \Rightarrow C \Rightarrow \dots$ etc. (ie: it would loop if that happened).

A repetition of a configuration results in a loop.
Every accepting computational history for w will have at most qng^n number of steps.
Then the following TM will decide A_{LBA}.

On input $\langle M, w\rangle$
"Run M on w for qng^n number of steps.
If it accepts, accept"
If it rejects, reject."

Theorem: $E_{LBA} = \{\langle M\rangle \mid M \text{ is an LBA with } L(M) = \varnothing\}$ IS NOT DECIDABLE.

COMP 330
NOV 5 2010

PAUL HUSSMAN

Theorem: $E_{LBA} = \{\langle M\rangle \mid M \text{ is an LBA with } L(M) = \varnothing\}$ is NOT decidable.

Proof: (By reduction from A_{TM}) ▲

Proof: - Let M be a TM and w be a string. First we note that there is an LBA called $D_{M,w}$ such that for any sequence of configurations $c_1, \dots c_\ell$ it can decide whether $c_1 \Rightarrow c_2 \Rightarrow \dots \Rightarrow c_\ell$ is a valid accepting computational history of M on w.

- That is it accepts input $\#\underbrace{c_1}\#\underbrace{c_2}\#\dots\#\underbrace{c_\ell}\#$ if & only if $c_1 \Rightarrow c_2 \Rightarrow \dots \Rightarrow c_\ell$ is an accepting computational history of M on w.

- The LBA $D_{M,w}$ functions in the following way:
  - It first checks whether c_1 is the correct starting configuration
  - It moves back & forth between c_1 & c_2 to see whether c_1 follows from c_2 or not. Then it continues verifying $c_2 \Rightarrow c_3$ and $c_3 \Rightarrow c_4$ and so on.
  - Finally it checks that c_ℓ is an accept configuration.
  - Note that $D_{M,w}$ is an LBA and it never moves the tapehead off the input area

- Now suppose the contrary of the theorem: There is a Turing machine R that decides $E_{LBA} = \{\langle M \rangle \mid M \text{ is an LBA}, L(M) = \emptyset\}$
- Then the following Turing machine would decide A_{TM}

$N =$ "on input $\langle M, w \rangle$
construct the LBA $D_{M,w}$
Run R on $\langle D_{M,w} \rangle$
If it accepts, reject. //
If it rejects, accept.

- N decides A_{TM} as $L(D_{M,w}) = \emptyset \iff$ There is no accepting computational history of M on w. $\iff$ M does not accept w. □

The post-correspondence Problem:

We are given a finite collection of dominos where each domino contains two strings (top and bottom). eg: $\left\{\left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right]\right\}$

We want to make a finite list of these dominos (repetition is allowed) such that the top string matches the bottom string. eg: $\left[\frac{a}{ab}\right], \left[\frac{b}{ca}\right], \left[\frac{ca}{a}\right], \left[\frac{a}{ab}\right], \left[\frac{abc}{c}\right]$ Top: abcaaabc Bottom: abcaaabc

Let $PCP = \{\langle P \rangle \mid P \text{ is an instance of the post-correspondance problem with at least match}\}$

and $MPCP = \{\langle P \rangle \mid P \text{ is an instance of the post correspondance problem with at least a match that starts with the first domino}\}$

Theorem: MPCP is undecidable.

Proof: (by reduction of A_{TM})
Suppose the contrary: That R is a TM that decides MPCP. Consider a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ and string $w = w_1 w_2 w_3 \dots w_n$
We will construct an instance of PCP such that if M accepts w then there will be a matching solution of $\#\underbrace{q_0 w_1 \dots w_n}_{c_1}\#\underbrace{\quad\quad}_{c_2}\#\dots\#\underbrace{u_1 \dots u_i q_{accept} v_1 \dots v_j}_{c_\ell}\#\dots\#q_{accept}\#\#$

where $c_1 \Rightarrow c_2 \Rightarrow \dots \Rightarrow c_\ell$ is the accepting conditional history of M on w.

① Put the domino $\left[\frac{\#}{\#q_0 w_1 \dots w_n}\right]$ as the first domino.

② If $\delta(q_i, a) = (r, b, R)$ (where $r, q \in Q$ and $a, b \in \Gamma$) $\dots q_a \dots \Rightarrow \dots br \dots$
So if $r \neq q_{reject}$, in this case we put $\left[\frac{qa}{br}\right]$ in the set.

③ If $\delta(q, a) = (r, b, L)$ where $r \neq q_{reject}$ then for every $c \in \Gamma$ add $\left[\frac{cqa}{rcb}\right]$

COMP 330
NOV 5 CONT'D

PAUL HUSSMAN

④ For every $a \in \Gamma$ we put $\left[\frac{a}{a}\right]$

⑤ Put $\left[\frac{\#}{\#}\right]$ and $\left[\frac{\#}{\sqcup \#}\right]$

⑥ For every $a \in \Gamma$ add $\left[\frac{a\, q_{accept}}{q_{accept}}\right]$ and $\left[\frac{q_{accept}\, a}{q_{accept}}\right]$

⑦ add $\left[\frac{q_{accept}\, \#\, \#}{\#}\right]$

Why does this work? ⇒ $\frac{\# \; q_0 w_1 \cdots w_n \ldots \ldots}{\# \; q_0 w_1 \cdots w_n}$

COMP 330
NOV. 8, 2010

[office hours for A4 on wed @ 3-4 - Linda MC232]

PAUL HUSSMAN.

Reductions:

Definition: A function $f: \Sigma^* \to \Sigma^*$ is called **COMPUTABLE** if & only if there is a TM that on every input w halts with just $f(w)$ left on the tape.
Roughly speaking $f: \Sigma^* \to \Sigma^*$ is computable iff there is an algorithm that takes w as input and outputs $f(w)$.

Definition: A language A is **MAPPING REDUCABLE** to language B, written $A \leq_m B$, if and only if there is a computable map $f: \Sigma^* \to \Sigma^*$ such that
$w \in A \Leftrightarrow f(w) \in B$
In this case the map f is called a reduction of A to B.

Theorem: If $A \leq_m B$ and B is decidable then A is decidable

Proof: Since B is decidible, there is a Turing Machine R that decides B.
Since A is mapping reducible to B ($A \leq_m B$), there is a reduction f of A to B.
So the following TM decides A:

N = "on input w,
compute $f(w)$
run R on $f(w)$
If it accepts accept.
If it rejects reject."

Note that $w \in A \Leftrightarrow f(w) \in B$. (by def of mapping reducibility)
$\Leftrightarrow R$ accepting $f(w)$

(trivial) So N correctly decides whether or not $w \in A$.

Corollary: If $A \leq_m B$ and A is undecidable, then B is undecidable.

Theorem: If $A \leq_m B$ and B is recursively enumerable then A is also recursively enumerable.

Proof: Since B is recursively enumerable, there is a TM R that accepts B.
Let $f(B)$ be a reduction from A to B.
Now we construct the following TM that ACCEPTS A:

N = "on input w,
compute $f(w)$
Run R on $f(w)$
If it accepts → accept."

N accepts A because:
If $w \in A$ then $f(w) \in B \Rightarrow R$ accepts $f(w) \Rightarrow N$ accepts w.
If $w \notin A$ then $f(w) \notin B \Rightarrow R$ does not accept $f(w) \Rightarrow N$ does not accept w. □

Corollary: If A is mapping reducible to B ($A \leq_m B$) and A is NOT recursively enumerable then B is not recursively enumerable.

Example: Give a mapping reduction of $A_{TM} = \{\langle M,w\rangle \mid M \text{ accepts } w\}$ to $Halt_{TM} = \{\langle M,w\rangle \mid M \text{ halts on } w\}$

Answer: Let $u_0 \in Halt_{TM}^C$ (so $u_0 \notin Halt_{TM}$)

Now define $f: \Sigma^* \to \Sigma^*$ in the following way:

$$f(u) = \begin{cases} u_0 & \text{if } u \text{ is not of the form } \langle M,w\rangle \\ \langle M',w\rangle & \text{if } u = \langle M,w\rangle \text{ where } M \text{ is a TM and } w \text{ is a string.} \end{cases}$$

where M' = "On input x:
Run M on w
If it accepts accept
If it rejects enter a loop"

Note that M' is defined in a way that
M accepts $w \iff M'$ halts on w.
M does not accept $w \iff M'$ enters a loop on $w \Rightarrow M'$ does NOT halt on w.

So: $\langle M,w\rangle \in A_{TM} \iff f(\langle M,w\rangle) = \langle M',w\rangle \in Halt_{TM}$

Also: If u is not in A_{TM} because it's not the right format $\Rightarrow f(u) = u_0 \notin Halt_{TM}$.

Question: Does it hold that $A_{TM}^C \leq_m A_{TM}$?

Answer: No, A_{TM}^C is not recursively enumerable and since A_{TM} is recursively enumerable we cannot have $A_{TM}^C \leq_m A_{TM}$ □

Note that if we could decide $w \in A_{TM}$ then we could also decide $w \in A_{TM}^C$

Definition: An **ORACLE** for a language B is an external device that can tell us whether any given string w belongs to B or not.
An "oracle Turing Machine" is a modified turing machine that has access to an oracle.
NOTATION: M^B denotes a TM that has access to an oracle for the language B.

Definition: The language A is **decidable RELATIVE TO B** if there is an oracle Turing Machine M^B that decides A.

Example: A_{TM}^C is decidable relative to A_{TM}

$T^{A_{TM}}$ = "On input $\langle M,w\rangle$
Query the oracle for $\langle M,w\rangle \in A_{TM}$
If yes → reject
otherwise → accept."

Definition: We write $A \leq_T B$ if A is decidable relative to B. We say A is **Turing Reducible** to B.

---

comp 330
NOV 10 2010

PAUL HUSSMAN

Example: Show that $E_{TM} \leq_T A_{TM}$ where $A_{TM} = \{\langle M,w\rangle \mid M \text{ accepts } w\}$, $E_{TM} = \{\langle M\rangle \mid L(M) = \emptyset\}$

Answer: The following oracle Turing Machine decides E_{TM}. Let $s_1, s_2, \ldots$ be an enumeration of all strings in Σ^*

$T^{A_{TM}}$ = "on input $\langle M\rangle$
1) Construct a TM called N as
N = "on any input
for $i = 1,2,3\ldots$
Run M on $s_1, s_2, \ldots, s_i$ for i steps
If M accepts any of them, accept."
2) Query the oracle to determine whether $\langle N, 0\rangle \in A_{TM}$.
If yes, reject.
If No, accept."

Note that $\langle M\rangle \in E_{TM} \Rightarrow L(M) = \emptyset \Rightarrow L(N) = \emptyset \Rightarrow N$ does Not accept $0 \Rightarrow \langle N, 0\rangle \notin A_{TM}$.
$\Rightarrow$ we accept M correctly.

Other case: $\langle M\rangle \notin E_{TM} \Rightarrow \exists j \mid M$ accepts $s_j \Rightarrow L(N) = \Sigma^* \Rightarrow \langle N, 0\rangle \in A_{TM} \Rightarrow$ we reject $\langle M\rangle$ correctly.

PAUL HUSSMAN

Theorem: If $A \leq_M B \Rightarrow A \leq_T B$

Proof: Let $f: \Sigma^* \to \Sigma^*$ be a mapping reduction of A to B. So $w \in A \Leftrightarrow f(w) = B \; \forall w \in \Sigma^*$

The following oracle TM will decide A:

T^B = "On input w
compute f(w)
Query the oracle to determine whether $f(w) \in B$ holds or not.
If Yes, accept
If No, reject"

so: $w \in A \Rightarrow f(w) \in B \Rightarrow$ Oracle will say Yes $\Rightarrow$ we accept w.
and $w \notin A \Rightarrow f(w) \notin B \Rightarrow$ Oracle will say No $\Rightarrow$ we reject w.

Exercise

True or False:

① If $A \leq_T B$ and B is decidable $\Rightarrow$ A is decidable
TRUE: If B is decidable, there is a TM that works as an oracle for B and this can be used to construct a TM that decides A.

② $A \leq_M B$ and B is recursively enumerable $\Rightarrow$ A is recursively enumerable.
TRUE: If M is a TM that accepts B, then:
N = "On input w
Run M on f(w) and if it accepts, accept"
will accept A.

③ $A \leq_T B$ and B is recursively enumerable $\Rightarrow$ A is recursively enumerable
FALSE: Trivially $A_{TM}^c \leq_T A_{TM}$ but A_{TM} is recursively enumerable and A_{TM}^c is NOT.

④ $A \leq_T B \Rightarrow A \leq_M B$
FALSE $A_{TM}^c \leq_T A_{TM}$ but $A_{TM}^c \not\leq_M A_{TM}$ as otherwise it would be recursively enumerable.

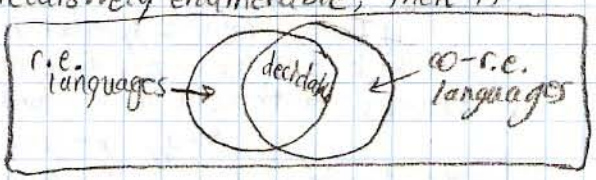
⑤ $A \leq_M B \Rightarrow A \leq_T B$
TRUE.

Definition: A language is **CO-RECURSIVELY ENUMERABLE** $\Leftrightarrow$ Its complement L^c is recursively enumerable.

Example: A_{TM} is recursively enumerable but not co-recursively-enumerable.

Remark: Every decidable language is co-recursively enumerable because the complement of a decidable language is decidable.

Remark: If L is both recursively enumerable and co-recursively enumerable, then it is decidable.



Theorem: $EQ_{TM} = \{\langle M_1, M_2 \rangle \mid L(M_1) = L(M_2)\}$ is neither recursively enumerable, nor co-recursively enumerable.

Proof: First we show that EQ_{TM} is not recursively enumerable by showing that $A_{TM}^c \leq_M EQ_{TM}$
The mapping reduction works as follows:

"On input $\langle M, w \rangle$
Construct TM's M_1 and M_2
M_1 = "on any input, reject"
M_2 = "on any input
Run M on w and if it accepts, accept"
Output $\langle M_1, M_2 \rangle$"

$\langle M, w \rangle \in A_{TM}^c \Rightarrow$ M does not accept w $\Rightarrow \begin{cases} L(M_1) = \emptyset \\ L(M_2) = \emptyset \end{cases} \Rightarrow \langle M_1, M_2 \rangle \in EQ_{TM}$.

$\langle M, w \rangle \notin A_{TM}^c \Rightarrow$ M accepts w $\Rightarrow \begin{matrix} L(M_1) = \emptyset \\ L(M_2) = \Sigma^* \end{matrix} \Rightarrow \langle M_1, M_2 \rangle \notin EQ_{TM}$,

Next we show that EQ_{TM} is NOT co-recursively enumerable by proving $A_{TM}^{C} \leq_M EQ_{TM}^{C}$.
The mapping reduction works as follows:

"On input $\langle M, w\rangle$:
construct TM's M_1 and M_2
M_1 = "on any input accept"
M_2 = "on any input
run M on w and if it accepts, accept"
Output $\langle M_1, M_2\rangle$"

$\langle M_1, M_2\rangle \in A_{TM}^{C} \Rightarrow M$ doesn't accept $w \Rightarrow \begin{matrix} L(M_1)=\Sigma^* \\ L(M_2)=\emptyset \end{matrix} \Rightarrow \langle M_1, M_2\rangle \in EQ_{TM}^{C}$

$\langle M_1, M_2\rangle \notin A_{TM}^{C} \Rightarrow M$ accepts $w \Rightarrow \begin{matrix} L(M_1)=\Sigma^* \\ L(M_2)=\Sigma^* \end{matrix} \Rightarrow \langle M_1, M_2\rangle \notin EQ_{TM}^{C}$ □

COMP 330
NOV 12 2010

Today's stuff → very important.

PAUL HUSSMAN

First-Order Logic

Formulas in first logic are similar to this form:
$\forall a,b,c,n\ [(n>2) \wedge (a,b,c>0) \Rightarrow (a^n+b^n \neq c^n)]$
How do we construct a formula?
5 ingredients:
(1) For every integer $n \in \mathbb{N}$, there is a set of n-ary function symbols (Possibly empty). We use meta symbols $f, g, h, \ldots$ and also $+, \cdot, \ldots$ as function symbols.
(2) For every integer n, there is a set of memory relation symbols (at least one of these sets is non-empty). We use $P, Q, R, \ldots$ and also $=$ as memory relation symbols.
(3) connectives: $\neg, \wedge, \vee$ (not, and, or)
(4) Quantifiers: $\forall, \exists$ (for all, there exists).
(5) Paranthesis:
(6) Infinite set of variable symbols, usually denoted by metasymbols $x, y, z, \ldots$ & $a, b, c \ldots$

Example: The standard vocabulary for arithmetic is: $\{0, +, \cdot, S, =\}$
$+$ is a binary function symbol
$\cdot$ is a binary function symbol.
S is a unary function symbol (called a successor $\Rightarrow S(0)=1, S(1)=2, \ldots$)
0 is a zero-ary function symbol. (aka a constant)
$=$ is a binary relation symbol.

Definition: TERM (recursive definition):
(1) Every variable symbol is a term.
(2) If f is an n-ary function symbol, and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

Example: Arithmetic with standard vocabulary:
x, $x \cdot y$, $x + x \cdot y$ are terms. [Note: $x \cdot y \Leftrightarrow \cdot(x,y)$, $x + x \cdot y \Leftrightarrow +(x, \cdot(x,y))$]

Definition: First-order Logic Formula:
(1) If P is an n-ary relation and $t_1, \ldots, t_n$ are terms, then $P(t_1, \ldots, t_n)$ is a formula.
(2) If A and B are formulas, then $\neg A$, $A \wedge B$, $A \vee B$ are formulas.
(3) If A is a formula and x is a variable, then $\forall x A$ and $\exists x A$ are formulas □

Example: $\neg((\forall x\, P(x)) \vee (\exists x\, P(x)))$

Example: $\forall x\, \neg Q(x,y) \wedge \forall z\, Q(x, f(y))$

Example: $\forall x\, \exists y\, (y > x \wedge Prime(y) \wedge Prime(y+2))$
Here, Prime is a unary relation
$>$ is a binary relation
$+$ is a binary function
2 is a zero-ary function.

COMP 330
NOV 12, 2010

PAUL HUSSMAN

Some other notation we often use

- $A \Rightarrow B$ meaning $\neg A \vee B$
- $A \Leftrightarrow B$ meaning $(\neg A \vee B) \wedge (\neg B \vee A)$

Example: $\forall x\,(Even(x) \wedge (x>2)) \Rightarrow \exists y \exists z\,(Prime(y) \wedge Prime(z) \wedge (x+y=z))$

Solution worth 1 million ↓ (GOLDBACH'S CONJECTURE)

Definition: An occurrence of a variable x in a formula A is called **BOUND** if it appears in a subformula of the form $\forall x\,B$ or $\exists x\,B$.
Otherwise this occurrence is called **FREE**

Example: In $\exists y\,(x=y+y)$ the occurrence of x is FREE while both occurrences of y are BOUND.

A formula with no free variables is called a **SENTENCE**.

So far we talked about syntax, and now we discuss the INTERPRETATIONS of these formulas.

Definition: A **MODEL** M for a vocabulary consists of:

- Universe: a set U (possibly infinite), the variables range over U.
- M assigns meanings to function symbols in the following way:
  For every n-ary function, f and every n elements $u_1, \ldots, u_n$ in the universe, it assigns a value from U to $f(u_1, \ldots, u_n)$
- M assigns values to relation symbols for every n-ary relation symbol R and every $u_1, \ldots, u_n \in U$ it assigns a true or false to $R(u_1, \ldots, u_n)$.

Remark: The relation symbol $=$ always gets its natural meaning. (It holds if two elements are the same element in the universe)

Remark: Note that every model imposes a true or false value on every sentence.

Example: Consider the sentence: $\forall x\,\exists y\,(x=y+y)$

(i) Consider the model M_1 which has $\mathbb{Z}$ universe and $+, =$ are defined in the natural way.
This sentence is FALSE in the model M_1.

(ii) Consider the model M_2 with $\mathbb{R}$ as its universe and $+, =$ are defined normally.
This sentence is TRUE in the model M_2