6.044J: Theory of Computation and Notational Techniques

3-0-7

Lecturer: A. Meyer
Prerequisites: 18.063
Difficulty: 7/10
Lecturer's Rating: 4.2/7

Next Term: Not Offered
Evaluated: Fall 88
Pace: 4.8/7
Overall Rating: 3.8/7

6.044J is a heavy dose of theory of computability and logic with a lot of proofs added in. The course intends to reach all computer science majors, but those with a stronger math background are likely to do better.

Grades are determined by problem sets and four quizzes. Quiz grading is somewhat unfair in that partial credit is not given. In general, quizzes were thought to be difficult. Problem sets introduced new material rather than reinforcing old material.

Professor Meyer knows his stuff but occasionally has trouble explaining it to the class. Lectures sometimes went too fast, leaving some students confused. Most felt that it was important to attend lecture. I. Saias, the TA, was enthusiastic, but many students could not understand his English.

Both the textbook, *Mathematical Theory of Computation* (by Z. Manna), and the class notes were considered helpful; however, they did not have enough examples and discussion. The recitation handouts were considered illegible by most students.

There was no syllabus; administration was done on a day-to-day basis, making it difficult to plan ahead.

"The most abstract theoretical course possible."

"Flossing with barbed wire is more fun. And much easier to understand."

"The relationship between Meyer/Saias is much like that of Bush/Quayle."

Course Information

Staff.

Lecturer:	Prof. Albert R. Meyer	NE43-315 x3-6024 meyer@theory.lcs.mit.edu
Teaching Assistant:	TBA	
Secretary:	Sally Bemus	NE43-316 x3-~~6025~~ 5936 bemus@theory.lcs.mit.edu

Send email to 6044-admin@theory.

Lectures and Tutorials. Class meets MWF 1:00–2:00PM in 24-115. There will be no recitation sections, but tutorial/review sessions may be organized in response to requests. No regularly scheduled office hours; meetings with the instructor (or TA if we have one) by appointment.

Prerequisites. The official requirement for the course is either 18.063 *Introduction to Algebraic Systems*, or 18.310 *Principles of Applied Mathematics*. If you know the basic vocabulary of mathematics and how to do elementary proofs, then you may take this course with the permission of the instructor.

Contrarequisites. There will be up to a 40% overlap in topics (namely, basic computability theory) between 6.045J/18.400J and this course. For this reason, Course 6 students are discouraged from taking both courses. There will be a smaller overlap with 6.840J/18.404J; students, especially Math majors, *may* take both this course and 6.840J/18.404J.

Grading. There will be regular problem sets, quizzes, and most likely a regular three hour final exam. The problem sets, quizzes, and final each count about equally toward the final grade. Some exam problems are typically adaptations of earlier homework problems. Quizes and final are *open book*.

Problem Sets. There will be four to six problem sets. Homework will usually be assigned on a Friday and due 7–10 days later. Late homeworks should be submitted and will be accepted until such time as solutions for that assignment are given out. If late problems can be graded without inconvenience, they will be, otherwise they will be kept for reference until final grades are given. More information will be supplied with the first assignment.

Collaboration. You must *write* your own problem solutions and other assigned course work in your own words and entirely *alone*. On the other hand, you are encouraged to *discuss* the problems with one or two classmates *before* you write your solutions. If you do so, **please be sure to indicate the members of your discussion group on your solution.**

(OVER)

Textbooks. The required text is:

Z. Manna, *Mathematical Theory of Computation*, McGraw Hill, 1974.

Possibly helpful, but not essential supplemental texts include:

J. Loeckx and K. Sieber, *Foundations of Program Verification*, Prentice-Hall, 1986.

H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, 1981.

Other well written, more advanced texts covering portions of the course include:

Davis and Weyuker, *Computability, Complexity, and Languages*, Academic Press, 1983.

J. W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, 1984.

Boolos and Jeffrey, *Computability and Logic*, Cambridge Univ. Press, 1980 (Second Edition).

Handouts and Notebook. You may find it useful to get a loose-leaf notebook for use with the course, since all handouts and homework will be on standard three-hole punched paper. If you fail to obtain a handout in lecture, you can get a copy from the file cabinet outside Sally's office (NE43-316). If you take the last copy of a handout, please inform Sally so that more copies can be made.

Pictures. You can help us learn who you are by giving us your photograph with your name on it. This is especially helpful if you later need a recommendation.

Notes on Computability Theory

The following notes outline the main definitions and results about computability theory indicated in the course lectures and homework which go beyond Manna's text.

1 Simulation

The "simulation thesis" says that all general computation models are capable of simulating one another, ignoring issues of efficiency (*e.g.*, time or space). Thus, an apparently very weak Turing machine model computes the same set of functions over, say, $\{\mathtt{a},\mathtt{b}\}^*$, as are computable by Scheme programs, register machines, or Post machines. For historical reasons these notes, like Manna's text, will regularly refer to Turing machines, but the reader can safely think of "Turing Machine" as replaced by, say, "Scheme procedure text" if that seems more familiar. This class of functions is the class of *Turing computable* functions.

The class of partial recursive functions as defined by Manna (section 1-4.2) coincides with the class of Turing computable functions. So a function is partial recursive iff there is a program (in some language—by the simulation thesis it does not matter which language) that computes it.

2 Coding Functions

We will want to talk about computations on strings, integers, graphs, lists, flowcharts, ordered pairs and finite sets of these, and various other finite or finitely representable mathematical objects. In each case we assume, without going into detail, that standard encodings of these objects into finite "binary" strings over $\{\mathtt{a},\mathtt{b}\}$ are adopted, and that a function on, say, graphs, is "partial recursive" iff the string function on the graph codes is partial recursive.

For example, one might code integers by their binary representations or perhaps their unary representations, *i.e.*, a string of n a's represents the natural number n. Since it is easy to write a unary to binary translation program, or vice-versa, it follows that the class of computable functions on the integers is unaffected by the choice of coding.

In a basic argument below, we will speak of the Turing Machine (Self-)Halting Problem, K_1. The "problem" is represented by the set of Turing machines that halt "when given themselves as input". More precisely, we must define a coding function

$$d : \{\text{Turing machines}\} \to \{\mathtt{a},\mathtt{b}\}^*$$

under which every Turing machine is coded as a binary string. It is straightforward enough, though a little tedious, to do this; we'll skip the details and take it as done. It is technically convenient if every string in $\{\mathtt{a},\mathtt{b}\}^*$ is the code of some Turing machine. We can always

make this hold by invoking the convention that every string not in the range of d is to be interpreted as the code of a particular fixed Turing machine which, say, halts with output a on any input. We use the notation M_x for the Turing machine denoted by $x \in \{\mathtt{a},\mathtt{b}\}^*$.

So, in summary, we have

$$M_{d(N)} = N \text{ for all Turing Machines } N,$$

$$d(M_x) = x \text{ for all } x \in \text{range}(d),$$

and there is a certain fixed string $x_0 \in \{\mathtt{a},\mathtt{b}\}^*$ such that M_{x_0} computes the constant function a, and

$$d(M_x) = x_0 \text{ for all } x \notin \text{range}(d).$$

We now define

$$K_1 = \{d(M) \mid M \text{ is a Turing machine that halts on input } d(M)\} .$$

Similarly, we can straightforwardly code strings over an arbitrary countably infinite alphabet into binary strings, and of course we can code a pair of binary strings into a single string. Thus, when we say the (General) Turing Machine Halting Problem, K_0, is

$$K_0 = \{(M,x) \mid \text{Turing machine } M \text{ halts on input } x\} ,$$

we really mean

$$\begin{aligned} K_0 = \{z \in \{\mathtt{a},\mathtt{b}\}^* \mid\ & z \text{ codes the pair } (y_1,y_2), \\ & \text{where } y_1 = d(M) \text{ and } y_2 \text{ codes } x \in \Sigma_M^*, \\ & \text{and } M \text{ halts on input } x \} . \end{aligned}$$

The precise set of binary words equal to K_0 or K_1 depends of course on how we choose the coding function d, but the salient properties we establish about these sets are independent of the details of the coding.

3 Axioms for Coding Computable Functions (Optional)

There is a simple set of axioms which characterize the properties of the set of codewords abstractly without having to mention d or Turing machines at all; only the general notion of partial computable function need be known. For simplicity we'll state the axioms for computable functions on strings over the alphabet $\{\mathtt{a},\mathtt{b}\}$. First, saying that d is a coding certainly implies that $d(M)$ and x uniquely determine the behavior of M on x. The important part of d is thus the mapping from $d(M)$ and x to the output of M on x. This is captured the idea of a *coder* function.

Definition 1. A *partial-computable-function coder* is a partial function $\upsilon : \{\mathtt{a},\mathtt{b}\}^* \times \{\mathtt{a},\mathtt{b}\}^* \to \{\mathtt{a},\mathtt{b}\}^*$ such that for every partial computable function $\varphi : \{\mathtt{a},\mathtt{b}\}^* \to \{\mathtt{a},\mathtt{b}\}^*$, there is a $z_\varphi \in \{\mathtt{a},\mathtt{b}\}^*$ with the property that for all $x \in \{\mathtt{a},\mathtt{b}\}^*$,

$$\upsilon(z_\varphi, x) = \varphi(x).$$

Such a z_φ is called a *code* or *Gödel number* for φ. Coders are also called *universal functions* for the partial computable functions.

Having chosen a partial-computable-function coder v, the axiomatic definition of K_1 becomes

Definition 2.

$$K_1 = \{x \in \{\mathtt{a},\mathtt{b}\}^* \mid (x,x) \in \text{domain}(v)\}.$$

The intuitive requirement that d be computable serves to guarantee that the coding is effectively decipherable—there is some computable way to recover information about M from $d(M)$. This is abstractly captured by the *universal machine theorem*:

Theorem 1. There is a partial-computable-function coder v which is itself a partial computable function.

One other property of the coder will be needed for some of the later results. In addition to determining the input/output behavior of M, the code $z = d(M)$ can be modified to obtain the code for simple variants of M. For example, from z and $y \in \{\mathtt{a},\mathtt{b}\}^*$, one can easily construct a machine $M_{z,y}$ which replaces every input x given to it by the word $\mathit{pair}(y,x) \in \{\mathtt{a},\mathtt{b}\}^*$ which codes the pair (y,x), and then acts like M on the new input $\mathit{pair}(y,x)$. This is abstractly captured by thinking of the function $s(z,y) = d(M_{z,y})$ and saying it is total and computable. For historical reasons, this is known as the *s_n^m-Theorem*:

Theorem 2. There is a total computable function $s : \{\mathtt{a},\mathtt{b}\}^* \times \{\mathtt{a},\mathtt{b}\}^* \to \{\mathtt{a},\mathtt{b}\}^*$ such that for all $x, y, z \in \{\mathtt{a},\mathtt{b}\}^*$

$$v(z, \mathit{pair}(y,x)) = v(s(z,y),x).$$

In the rest of these notes we continue to give explanations in terms of a function d coding Turing machines into binary strings. We confidently omit the details of how d is defined because a careful reading of the arguments will reveal that Theorems 1 and 2 are the only facts we need about the coding to obtain all the results below. In fact, there is an elegant "recursive isomorphism" theorem due to Hartley Rogers which explains why all reasonable codings—namely those that satisfy the universal machine and s_n^m-theorems—have the same properties with respect to computability.

Theorem 3. Let v_1 and v_2 be two coders satisfying the universal machine Theorem 1 and the s_n^m-Theorem 2. Then there is a total computable one-one and onto function $t : \{\mathtt{a},\mathtt{b}\}^* \to \{\mathtt{a},\mathtt{b}\}^*$ such that

$$v_1(z,x) = v_2(t(z),x)$$

for all $z, x \in \{\mathtt{a},\mathtt{b}\}^*$.

The proof is not very hard but a bit long, and to save time we'll skip it.

Theorem 3 can be understood as saying that there is a *one-one onto* computable function t translating, say, Scheme programs into equivalent CLU programs. So Scheme and CLU (and Turing machines, Post machines, *etc.*) are indistinguishable from the point of view of general computability theory. This is a clear warning that the conclusions of the theory will *not* bear on some central Computer Science issues—such as which language features which make Scheme or CLU more desirable for certain problems. On the other hand, the conclusions of the theory, especially negative conclusions about the noncomputability of certain functions, will hold with great generality and can't be gotten around by changing programming languages.

4 Terminology

Let Σ be an arbitrary alphabet and let $* \notin \Sigma$ be a new symbol.

Definition 3. A partial function $\varphi : (\Sigma^*)^n \to \Sigma^*$ is called *Turing computable* iff there is a Turing machine M with input alphabet $\Sigma \cup \{\$\}$ that computes it. Namely, if $\varphi(x_1, \ldots, x_n)$ is defined, then M on input $x_1\$\ldots\$x_n$ halts, leaving on its tape the string $\varphi(x_1, \ldots, x_n)$ followed only by blanks, with the leftmost tape cell of M containing the first symbol of $\varphi(x_1, \ldots, x_n)$. On the other hand, if $\varphi(x_1, \ldots, x_n)$ is not defined, then M on input $x_1*\ldots*x_n$ either never halts, or halts without leaving its head and tape looking like a well-formed output string in Σ^*.

A synonym for Turing computable function is *partial recursive function.* Actually, there is another inductive definition of the class of partial recursive functions (given in Manna's text), and it is a long programming exercise to show that the two definitions are equivalent. We will skip this.

Definition 4. A partial recursive function that happens to be totally defined (on $(\Sigma^*)^n$) is called *total recursive.*

A language $D \subseteq \Sigma^*$ is *decidable* iff its characteristic function $c_D : \Sigma^* \to \{\mathtt{a}, \mathtt{b}\}$ is total recursive, where

$$c_D(x) = \begin{cases} \mathtt{a} & \text{if } x \in D, \\ \mathtt{b} & \text{otherwise.} \end{cases}$$

For any Turing machine M, let

$$\text{domain}(M) = \{x \in \Sigma_M^* \mid M \text{ halts on input } x\}.$$

A language $R \subseteq \Sigma^*$ is *recursively enumerable* (r.e.) iff $R = \text{domain}(M)$ for some Turing machine M. for some Turing machine M_R.

A *set* is r.e., recursive, *etc.*, iff the language consisting of binary codes of elements of the set is r.e., *etc.*

Synonyms:

- Recursive set = decidable set = Turing-decidable set.
- R.e. set = recursively enumerable set = Turing-acceptable set.
- [Turing] computable partial function = partial recursive function.
- Total recursive function = recursive function = [Turing] computable total function.

If P is a property of sets, then "A is co-P" means $P(\overline{A})$ holds.

5 Basic Properties of R.E. and Recursive Sets

Lemma 1. Recursive sets are closed under complement. (That is, if A is recursive, then $\overline{A}$ is recursive.)

Theorem 4. A is recursive iff both A and $\overline{A}$ are r.e., *i.e.*, iff A is both r.e. and co-r.e.

Theorem 5. Recursive sets are closed under union, intersection, and complementation.

Theorem 6. R.e. sets are closed under union and intersection.

Theorem 7. The following are equivalent for a language A:

- A is r.e.
- A is the domain of a partial recursive function of one argument.
- A is the range of a partial recursive function.
- $A = \emptyset$ or A is the range of a total recursive function.

Definition 5. The canonical order $<_{\text{can}}$ of strings in Σ^* (where Σ is ordered) is defined for all $w, u \in \Sigma^*$ as follows: $w <_{\text{can}} u$ iff $|w| < |u|$ or $|w| = |u|$ and w precedes u alphabetically.

Note that canonical order is different from alphabetical (dictionary) order. For example, $b <_{\text{can}} ab$ even though ab precedes b alphabetically. The canonical ordering of Σ^* is $\Lambda, \mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{ab}, \mathtt{ba}, \mathtt{bb}, \mathtt{aaa}, \mathtt{aab}, \ldots$.

Definition 6. A function $f : \Sigma^* \to \Sigma^*$ is an *increasing* function iff, for all strings $x, y \in$ domain(f), if $x <_{\text{can}} y$, then $f(x) <_{\text{can}} f(y)$.

Theorem 8. A language A is recursive iff it is finite or the range of a total increasing recursive function.

6 Undecidability of the Halting Problem

Definition 7.

$$K_0 = \{(M, x) \mid \text{Turing machine } M \text{ halts on input } x\}$$
$$K_1 = \{M \mid \text{Turing machine } M \text{ halts on input } d(M)\}$$

Theorem 9. $\overline{K_1}$ is not r.e.

Proof: Assume that $\overline{K_1}$ is r.e. Then there is some Turing machine M_1 that halts on precisely the binary words in $\overline{K_1}$. By the definition of $\overline{K_1}$, we have for any Turing machine M that

$$M \in \overline{K_1} \quad \text{iff} \quad M \text{ does not halt on input } d(M).$$

By the definition of M_1, we have for every Turing machine M that

$$M \in \overline{K_1} \quad \text{iff} \quad M_1 \text{ halts on } d(M).$$

Hence,

$$M_1 \text{ halts on } d(M) \quad \text{iff} \quad M \text{ does not halt on } d(M).$$

Now, let $M = M_1$ and we obtain an immediate contradiction. ■

Exercise 1. Let $L_1 = \{x \in \{\mathtt{a}, \mathtt{b}\}^* \mid x \in \text{domain}(M_x)\}$. (This is essentially the same as the set L_1 of section 1-3.1 of Manna.)

a. Briefly explain why, if $\text{range}(d) \neq \{\mathtt{a}, \mathtt{b}\}^*$, then L_1 contains but does not equal K_1.

b. Prove that $\overline{L_1}$ is not r.e.

Corollary 1. K_1 is not decidable.

Proof: If it were, then $\overline{K_1}$ would be recursive too by Lemma 1, and so would be r.e. by Theorem 4, contradicting what we just proved. ■

Corollary 2. K_0 is not decidable.

Proof: A simple modification of any program which decided membership in K_0, would yield a program which decided membership K_1, a contradiction. ■

Remark.

It's worth noting that Theorem 9 and Corollaries 1 and 2 depend only on the fact that the coding function d : {Turing Machines} $\rightarrow \{\mathtt{a}, \mathtt{b}\}^*$ is *one-to-one*—it does not even need to be "computable", *viz.*, it need not yield a computable coder function as in Theorem 1.

Claim 1. K_0 and K_1 are r.e.

Proof: Left to reader. Note that here we do assume that the coding function d is computable. ■

Corollary 3. K_0 is not co-r.e.

7 Many-one Reducibility

Definition 8. Given two languages $A \subseteq \Sigma_A^*$, $B \subseteq \Sigma_B^*$, we say A is *many-one reducible* to B, in symbols $A \leq_m B$, iff there is a total computable function $f : \Sigma_A^* \to \Sigma_B^*$ such that

$$x \in A \quad \text{iff} \quad f(x) \in B$$

for all $x \in \Sigma_A^*$. We say $A \equiv_m B$ iff $[A \leq_m B$ and $B \leq_m A]$.

The following properties are easily verified:

Transitivity. $A \leq_m B$ and $B \leq_m C$ implies $A \leq_m C$.

Recursiveness Inherits Down. $A \leq_m B$ and B recursive implies A recursive.

Non-recursiveness Inherits Up. $A \leq_m B$ and A not recursive implies B not recursive.

R.e. Inherits Down. $A \leq_m B$ and B r.e. implies A r.e.

Non-r.e. Inherits Up. $A \leq_m B$ and A not r.e. implies B not r.e.

Symmetry w.r.t. Complement. $A \leq_m B$ iff $\overline{A} \leq_m \overline{B}$.

An Incomparability. If A is r.e. but not recursive, then A and $\overline{A}$ are $\leq_m$-incomparable.

Note that the ability to decide membership in a set obviously implies the ability to decide membership in the complement of the set. But the last property above reveals that a set and its complement may be $\leq_m$-incomparable! So we recognize that $\leq_m$ is a technical notion of reducibility which is more restricted than the general intuitive notion of "reducing" one problem to another.

Corollary 4. $K_0 \leq_m A$ implies A is not co-r.e.

Corollary 5. $K_0 \times \overline{K_0}$ is neither r.e. nor co-r.e.

Lemma 2. If A is r.e., then $A \leq_m K_0$.

Proof: Suppose M accepts A; let $f(x) = (M, x)$. Then $x \in A$ iff $f(x) \in K_0$. ■

Definition 9. If R is a class of sets and $\leq$ is a relation on sets, then a set K is $\leq$-hard for R iff $C \leq K$ for all $C \in R$. A set K is $\leq$-complete iff K is both $\leq$-hard and $K \in R$.

In particular, K_0 is $\leq_m$-complete for r.e. sets.

Lemma 3. *Any* set other than the empty set and Σ^* is $\leq_m$-hard for recursive sets.

If $K_0 \leq_m A$, then by the transitivity of $\leq_m$, A is $\leq_m$-hard for r.e. sets, and if A is r.e. too, then it is an $\leq_m$-complete r.e. set. We will see next that K_1 (and K_2 given below) is also an $\leq_m$-complete r.e. set.

8 Undecidability of K_2

Theorem 10. The language $K_2 = \{M \mid M \text{ halts on blank tape}\}$ is a $\leq_m$-complete r.e. set.

Proof: Clearly K_2 is r.e., so we need only show that it is $\leq_m$-hard for r.e. sets.

Let $R \subseteq \Sigma^*$ be any r.e. set and say $R = \text{domain}(M_R)$ for some Turing machine M_R. We show that $R \leq_m K_2$ as follows.

For any string $x \in \Sigma^*$, we can define a new Turing machine $M_{f(x)}$ (that is, $f(x)$ is the code of this Turing machine) that operates as follows:

> "On input w, erase w, print x on the tape as input, and then act exactly like M_R."

By definition, the behavior of $M_{f(x)}$ does not depend on its input—it always halts or it never halts. In fact,

$$\begin{aligned} x \in R \quad &\text{iff} \quad M_R \text{ halts on } x \\ &\text{iff} \quad M_{f(x)} \text{ halts on some input} \\ &\text{iff} \quad M_{f(x)} \text{ halts on input } \Lambda \\ &\text{iff} \quad f(x) \in K_2 \ . \end{aligned}$$

But it is not hard to see that the function f is total recursive. (Think of writing a Scheme procedure that, when applied to character string x, prints out a Turing machine flow-chart for $M_{f(x)}$. The Scheme program has a flowchart for M_R as a "built-in" constant. Alternatively, we could justify this claim using the s_n^m-Theorem 2. This is the first result in these notes other than the recursive isomorphism Theorem 3 which depends on the s_n^m-Theorem.) Hence, $R \leq_m K_2$ ■

Theorem 11. K_1 is a $\leq_m$-complete r.e. set.

Proof: Replace "2" by "1" in the preceding proof. ■

9 Rice's Theorem

Definition 10. A property of languages is *nontrivial* iff there is some r.e. language that has the property and some r.e. language that does not.

For example, the property of being an r.e. language is trivial (since all r.e. languages have it); the properties of

- containing the empty word,
- being empty, or
- being infinite

are each nontrivial.

Theorem 12. The set $K_P = \{M \mid P(\text{domain}(M))\}$ is not decidable for any nontrivial property P of r.e. sets. In fact, if $\neg P(\emptyset)$, then K_P is $\leq_{\text{m}}$-hard for r.e. sets.

Proof: Suppose that $P(\emptyset)$ is not true. Since P is nontrivial, there exists a machine M_1 with $\text{domain}(M_1) \neq \emptyset$ such that $P(\text{domain}(M_1))$.

Let $R \subseteq \Sigma^*$ be any r.e. set and say $R = \text{domain}(M_R)$ for some Turing machine M_R. We show that $R \leq_{\text{m}} K_P$ as follows.

For any string $x \in \Sigma^*$, we can define a new Turing machine $M_{f(x)}$ (that is, $f(x)$ is the code of this Turing machine) that operates as follows

> "On input w, save w temporarily, and simulate M_R on input x. If this simulation halts, then act exactly like M_1 on input w."

By definition,

$$\begin{aligned} x \in R \quad &\text{iff} \quad M_R \text{ halts on } x \\ &\supset M_{f(x)} \text{ acts like } M_1 \text{ on every input} \\ &\supset \text{domain}(M_{f(x)}) = \text{domain}(M_1) \\ &\supset f(x) \in K_P \ , \end{aligned}$$

and also

$$\begin{aligned} x \notin R \quad &\text{iff} \quad M_R \text{ does not halt on } x \\ &\supset \text{domain}(M_{f(x)}) = \emptyset \\ &\supset f(x) \notin K_P \ . \end{aligned}$$

Moreover, as in the proof for K_2, the function f is total recursive. Hence, $R \leq_{\text{m}} K_P$. ■

10 Total Functions

Definition 11.

$$K_{\text{tot}} = \{M \mid \text{domain}(M) = \Sigma^*\}$$

Theorem 13. K_{tot} is neither r.e. nor co-r.e.

11 Turing Reducibility and Oracle Machines

Many-one reducibility is a good technical tool for establishing that one language is no harder than another, but as we noted, it does not completely capture the idea of reducing one problem to another. We introduce *Turing reducibility*, $\leq_{\text{T}}$, as a formulation of this general notion.

Informally, A is Turing-reducible to B, or equivalently A is *decidable in* B, in symbols, $A \leq_T B$, iff there is some program computing the characteristic function of A, where the program is allowed to repeatedly "call a subroutine" to answer questions about membership in B. It may use the answers however it likes. (For contrast, many-one reducibility allows only a single question about membership in B, *viz.*, "$f(x) \in B$", and must return the same answer as that question.)

We formalize "calling a subroutine" for B by defining *Turing machines with (string and) language inputs.* This is a Turing machine with one extra tape, called the *language tape.* The head on the ordinary tape operates as usual. The language tape is a read-only tape over B's alphabet plus the blank symbol. Unlike the Turing machine tapes considered so far, the language tape will start with useful information in every cell. This is a mathematical trick to allow the Turing machine access to the whole language B. There may not be any reasonable physical way to initialize the entire infinite language tape nor any effective way to grow it as computations proceed.

To run a generalized Turing machine on string input $x \in \Sigma_1^*$ and $B \subseteq \Sigma_2^*$, we start with $\Delta x \Delta$ on the work tape in the normal way. We initialize the language tape with the values of the characteristic function, c_B, of B on the successive words of Σ_2^* in canonical order. For example, if $\Sigma_2 = \{\mathtt{a}, \mathtt{b}\}$, ordered with a coming before b, the language tape looks like:

Δ	$c_B(\Lambda)$	$c_B(a)$	$c_B(b)$	$c_B(aa)$	$c_B(ab)$	$c_B(ba)$	$c_B(bb)$	$c_B(aaa)$	...

In general, the leftmost symbol on the tape is a blank, Δ, and the n^{th} symbol to the right of the Δ is an $\mathtt{a}$ if the n^{th} string in the canonical order of Σ_2^* is in B, and a $\mathtt{b}$ otherwise.

When we run a generalized machine M on inputs x and B, we are essentially doing a computation as if we had a subroutine for deciding membership in B.

Now we say that $A \leq_T B$ iff there is a generalized Turing machine M which, given fixed language input B, halts on all string inputs x, printing *yes* or *no* as $x \in A$ or $x \notin A$.

Similarly, we say that A is *r.e. in* B if there is a generalized Turing machine M such that $x \in A$ iff M halts on inputs x and B.

Theorem 14. Basic Facts about Turing Reducibility:

- A is r.e. iff A is r.e. in $\emptyset$.
- A is decidable iff A is decidable in $\emptyset$.
- If R is a recursive set, then A is recursive iff A is decidable in R.
- $A \leq_T B$ and $B \leq_T C$ imply $A \leq_T C$.
- $A \leq_m B$ implies that $A \leq_T B$.
- It is not the case that $A \leq_T B$ implies $A \leq_m B$. (For example, $K_0 \leq_T \overline{K}_0$, but it is false that $K_0 \leq_m \overline{K}_0$.)

- $A \leq_T \overline{A}$ for any set A.
- $A \leq_T B$ iff $A \leq_T \overline{B}$ iff $\overline{A} \leq_T B$.

We define the *relativized Halting Problem in B* to be

$$B' = \{(M, x) \mid M \text{ halts on input } x \text{ and } B\} .$$

So K_0 amounts to $\emptyset'$. B' is also called the *jump* of B, for short.

Theorem 15. (Relativized Halting Problem) B' is r.e. but not recursive in B.

The proof of this theorem is the same as the proof that for the ordinary halting problem $K_0 = \emptyset'$, with all the ordinary Turing machines in the original proof replaced by language input Turing machines with fixed language input B.

We say $A <_T B$ iff $A \leq_T B$ and $B \not\leq_T A$. Thus, we have

Corollary 6. $B <_T B'$

Theorems (and proofs) about Turing machines that carry over without change to Turing machines with language inputs are said to ***relativize***. Most of our theorems relativize. For example, the remark that a set A is r.e. iff $A \leq_m K_0$ (which follows immediately from the facts that K_0 is a $\leq_m$-complete r.e. set and r.e. inherits down $\leq_m$) relativizes to:

Theorem 16. A is r.e. in B iff $A \leq_m B'$.

Likewise Theorem 4 relativizes to:

Theorem 17. $A \leq_T B$ iff A is both r.e. and co-r.e. in B.

Some further relativizations:

Theorem 18. The following are equivalent:

- A is r.e. in B.
- $A = \text{domain}(\varphi)$, where φ is a partial recursive function in B.
- $A = \text{range}(f)$, where f is a total recursive function in B, or $A = \emptyset$.

Lemma 4. $B' \equiv_m \{M \mid M \text{ halts on blank string input with language input } B\}$. (That is, the relativized halting problem is $\equiv_m$ to the relativized blank-tape halting problem.)

Theorem 19. A'' is neither r.e. nor co-r.e. in A.

Proof: To show a language B is not r.e. in A, it suffices to show that $\overline{A'} \leq_m B$ (since $\overline{A'}$ is not r.e. in A and non-r.e.-in inherits up $\leq_m$).

But $A' \leq_T A'$ trivially, so by Theorem 17, A' is r.e. in A' and $\overline{A'}$ is r.e. in A'. Therefore, by Theorem 16, $A' \leq_m A''$ so $\overline{A'} \leq_m \overline{A''}$, and also $\overline{A'} \leq_m A''$. ■

Corollary 7. K_0' is neither r.e. nor co-r.e.

Define

$$B^{(0)} = \emptyset ,$$
$$B^{(n+1)} = (B^{(n)})' .$$

By the preceding theorems, $B^{(n)} <_T B^{(n+1)}$ for all n. So the sequence of sets

$$\emptyset <_T \emptyset' <_T \emptyset^{(2)} <_T \cdots$$

has strictly more difficult successive membership problems. This sequence, or more precisely the sequence of families $\{L \mid L \leq_m \emptyset^{(n)}\}$ for $n = 0, 1, \ldots$, is called the *Arithmetic Hierarchy.*

Thus, there is a rich classification possible among undecidable problems. Various natural decision lie along the arithmetic hierarchy. For example, the problem of proving "partial correctness" of program schemes turn out to be $\equiv_m \emptyset^{(2)}$. Since such problems are not even r.e. in the Halting Problem, it will follow that there is no complete axiom system for proving partial correctness even assuming availability of a completely effective decision procedure for first-order logic. More about this later in the course....

More Course Information

Teaching Assistant.

Mr. Isaac Saias 2-229 x3-1589 saias@theory.lcs.mit.edu
Send email to 6044-admin@theory.

Homework Instructions. Each problem is to be done on a *separate sheet* of *three-hole punched paper.* If a problem requires more than one sheet, staple these sheets together, but keep each problem separate. Do *not* write in red. Mark the top of the paper with:

- Your name,
- "6.044J/18.423J",
- the assignment number,
- the problem number, and
- the date.

Try to be as clear and precise as possible in your presentations. Problem grades are based not only on getting the right answer or otherwise demonstrating that you understand how a solution goes, but also on your ability to explain the solution or proof in a way helpful to a reader.

If you have doubts about the way your homework has been graded, first see the TA. Other questions and suggestions will be welcomed by both the instructor and the TA.

Late homeworks should be submitted to the TA. If they can be graded without inconvenience, they will be. Late homeworks that are not graded will be kept for reference until after the final. No homework will be accepted after the solutions have been given out.

Problem sets will be collected at the beginning of class; graded problem sets will be returned at the end of class. Solutions will generally be available with the graded problem sets, one week after their submission.

Collaboration and References. You must *write* your own problem solutions and other assigned course work in your own words and entirely *alone.* On the other hand, you are encouraged to *discuss* the problems with one or two classmates *before* you write your solutions. If you do so, please be sure to *indicate the members of your discussion group on your solution.*

Similarly, you are welcome to use other texts and references in doing homework, but if you find that a solution to an assigned problem has been given in such a reference, you should nevertheless rewrite the solution in your own words and *cite your source.*

Problem Set 1

Reading assignment. For this assignment: Notes on Computability, Handout 2. For supplement and lookahead: Manna's text, sections 1-2 through 1-5.2.

Remark. We follow the convention that all functions are total unless partiality is explicitly mentioned. Total functions are usually denoted by $f, g, h, \ldots$, while partial functions are denoted by $\phi, \psi, \eta, \ldots$. We will also give mnemonic identifiers as names for interesting partial and total functions.

Problem 1. A *pairing function* on $\{\mathtt{a},\mathtt{b}\}^*$ is a function $pair : \{\mathtt{a},\mathtt{b}\}^* \times \{\mathtt{a},\mathtt{b}\}^* \to \{\mathtt{a},\mathtt{b}\}^*$ together with two functions $left : \{\mathtt{a},\mathtt{b}\}^* \to \{\mathtt{a},\mathtt{b}\}^*$ and $right : \{\mathtt{a},\mathtt{b}\}^* \to \{\mathtt{a},\mathtt{b}\}^*$ such that

$$left(pair(x,y)) = x \text{ and } left(pair(x,y)) = y$$

for all $x, y \in \{\mathtt{a},\mathtt{b}\}^*$.

A pairing function is *computable* iff all three functions $pair, left, right$ are total computable functions.

A simple way to define a computable pairing function, $pair_0$ is to map (x,y) into the string $z = xcy \in \{\mathtt{a},\mathtt{b},\mathtt{c}\}^*$ and then code z into $\{\mathtt{a},\mathtt{b}\}^*$ by mapping the individual letters `a`, `b` and `c` into the two bit codes `aa`, `bb` and `ab`.

1(a). Describe suitable functions $left_0$ and $right_0$ and explain why $pair_0$ is computable.

1(b). Let *pair*, with *left* and *right* be some computable pairing function. Explain why range(*pair*) is recursive.

1(c). Use the previous result to show that a one-to-one computable function $pair : \{\mathtt{a},\mathtt{b}\}^* \times \{\mathtt{a},\mathtt{b}\}^* \to \{\mathtt{a},\mathtt{b}\}^*$ has functions *left*, *right* which make it a compuable pairing function iff range(*pair*) is recursive.

1(d). Describe a computable pairing function which is *onto* as well as one-to-one, namely, $pair(left(z), right(z)) = z$ for all $z \in \{\mathtt{a},\mathtt{b}\}^*$.

1(e). Note that $|\, pair_0(x,y)| = 2(|x|+|y|+1)$. Describe another "more succint" computable pairing function $pair_1$ such that for any $\epsilon > 0$ and all but finitely many x, y

$$|\, pair_1(x,y)| < (1+\epsilon)(|x|+|y|).$$

(over)

Problem 2. Let D be a decidable language over the alphabet $\{\mathtt{a},\mathtt{b}\}$, and let R be an r.e. language over $\{\mathtt{a},\mathtt{b}\}$. Let f be a total recursive function from $\{\mathtt{a},\mathtt{b}\}^*$ to $\{\mathtt{a},\mathtt{b}\}^*$, and let ψ be a partial recursive function from $\{\mathtt{a},\mathtt{b}\}^*$ to $\{\mathtt{a},\mathtt{b}\}^*$.

(a) Show that $\psi(R) = \{\psi(x) \in \{\mathtt{a},\mathtt{b}\}^* \mid x \in R\}$ is r.e.

(b) Show that $\psi^{-1}(R) = \{x \in \{\mathtt{a},\mathtt{b}\}^* \mid \psi(x) \in R\}$ is r.e.

(c) Show that $f^{-1}(D) = \{x \in \{\mathtt{a},\mathtt{b}\}^* \mid f(x) \in D\}$ is recursive.

(d) For any Turing machine, M, define

$$pad(M) = \{pair(x, \mathtt{a}^n) \mid x \in \{\mathtt{a},\mathtt{b}\}^* \text{ and } M \text{ halts on input } x \text{ in exactly } n \text{ steps}\}.$$

Briefly explain why $pad(M)$ is decidable. Conclude that there is an simple to compute function f and a decidable set D such that $f(D)$ is not decidable.

Problem 3. For each of the sets below, state whether it is recursive,r.e. but not co-r.e., or co-r.e. but not r.e., and briefly explain why.

(a) The set of TM's that halt on every input.

(b) The set of TM's that halt on blank tape in a number of steps between 20073 and 997631.

(c) The set of TM's that halt on input aba in $\geq$ 749 steps.

(d) $\{M \mid 10^9 < |d(M)|\}$.

(e) The set of integers n such that there are at least n occurrences of the successive digits '38' after the n^{th} place in the decimal expansion of $(2.7)^{\ln \pi}$. (*Hint*: Case 1: there are infinitely many '38's in the decimal expansion of $(2.7)^{\ln \pi}$, Case 2: there are finitely many '38's in the decimal expansion of $(2.7)^{\ln \pi}$.)

Handout 5: Review material of last year (1)

6.044J/18.423J: Computability, Programming, and Logic
Massachusetts Institute of Technology

Handout 6
05 October 1987

Practice for Quiz 1

The following is a sample of the kind of problems you will see on Quiz 1. We present it here as an optional practice problem.

Problem 1. Let

$$A = \{M \mid M \text{ diverges on input } \Lambda \text{ and writes an infinite number of a's during its computation on input } \Lambda\} .$$

(a) Show that $K_2 \leq_m A$.

(b) Show that $\overline{K}_2 \leq_m A$.

(c) Conclude that A is neither r.e. nor co-r.e.

($K_2 = \{M \mid M \text{ halts on input } \Lambda\}$ is the blank tape halting problem.)

Handout 5 ②

Quick Solution to the practice to Quiz 1 of last year's handout 6.

a) Show that $K_2 \leq_m A$.

Consider a Turing Machine M.
Construct a machine N as follows:
". Construct machine M' which works exactly as M, except that M' is using some new letter, say for instance α, instead of a, if a is in the working alphabet of M.
. N is the machine that for any input ~~given~~ operates as follows:
: Have M' working on blank input Λ.
. if M' halts on Λ, then diverge writing at each step "a" on the working tape.
. if M' does not halt on Λ -- then nothing more is happening! "

Then we see that: • $M \in K_2$ iff $M' \in K_2$
iff $N \in A$.

• Writing $N = f(M)$, f is total, computable

Hence $\boxed{K_2 \leq_m A}$.

b) $\overline{K_2} \leq_m A$:

Given machine M construct machine N that works as follows.

machine N

" on all inputs do the following:

- have M working on Λ and:
  at each step, if the head of M is on the cell: $\boxed{\alpha}$ (α is the letter of the cell at this time)
  Then:
  - remember α for a while
  - write a
  - erase a
  - rewrite α.
  - go ahead with the normal procedure of M. "

We see that a) M halts on Λ iff N halts on Λ

b) If M diverges then actually N writes an infinite number of a's during its computation on input Λ.

Call f the function s.t.: $N = f(M)$.

We see that: • f is total and computable.

• $M \in \overline{K_2}$ iff $f(M) \in A$.

(re) $\boxed{\overline{K_2} \leq_m A}$

c) $K_1 \leq_m K_2 \leq_m A \Rightarrow$ A not core

(thm 10 of course) (a)

K_1 not core

- if A was re then from (b) K_2 would be core which contradicts the fact that $K_1 \leq_m K_2$

Handout 5 ④

6.044J/18.423J: Computability, Programming, and Logic — Handout 8
Massachusetts Institute of Technology — 07 October 1987

Quiz 1

Instructions. Do all 4 problems; a total of 100 points is allocated as shown on each problem. This exam is *open book*. There is a short glossary and summary of notation on the last page. You have two hours. Good luck.

Problem 1 [30 points]. Let

$$\mathit{OddSquare} = \{M \mid \text{Turing machine } M \text{ halts on input } \Lambda \text{ with its head on an odd numbered tape square}\}.$$

(a) [10 points] Explain why Rice's theorem doesn't apply to *OddSquare*. More precisely, show that $\mathit{OddSquare} \neq K_P$ for any property P of r.e. sets.

(b) [20 points] Show that *OddSquare* is a $\leq_m$-complete r.e. set anyway.

Problem 2 [20 points]. For any total function $T : \{a, b\}^* \to \mathbf{N}$ define the set

$$K^T = \{M \mid M \text{ halts on input } d(M) \text{ in} \leq T(d(M)) \text{ steps}\}$$

and the class of languages

$$\mathit{Accept}(T) = \{L \subseteq \{a, b\}^* \mid \exists M \text{ such that domain}(M) = L \text{ and } \forall x \in \{a, b\}^*, \text{ if } M \text{ halts on input } x \text{ then it does so in} \leq T(x) \text{ steps}\}.$$

Prove that $\overline{K^T} \notin \mathit{Accept}(T)$.

Problem 3 [25 points]. For each of the sets below, indicate with a single capital letter whether it is (D) decidable, (R) r.e. but not co-r.e., (C) co-r.e. but not r.e., or (N) neither. No explanation is required and there is no penalty for guessing.

(a) The set of Thue systems S such that there exist $x \neq y \in \Sigma_S^*$ for which $x \not\leftrightarrow_S^* y$.

(b) The set of TMs that accept languages containing only strings of even length.

(c) $\{M \mid \exists M' \text{such that } d(M) \neq d(M') \text{ but } M \text{ and } M' \text{ accept the same language}\}$.

(d) $\{M \mid M \text{ writes a } \Delta \text{ symbol during its computation on some input}\}$.

(e) $K^T \times \overline{K^T}$ where K^T is given in Problem 2, and $T(w) = 2^{|w|}$.

Handout 5 (5)

Problem 4 [25 points]. Let

$$A = \{M \mid M \text{ halts on input a and doesn't halt on input b}\}$$

Show that A is neither r.e. nor co-r.e.

Glossary and notation

$d(M) \in \{\mathtt{a}, \mathtt{b}\}^*$ is the code of the Turing machine M.

$\mathbf{N} = \{0, 1, 2, ...\}$ is the set of natural numbers.

$K_P = \{M \mid P(\text{domain}(M))\}$ where P is a property of r.e. sets.

A set A is $\leq_{\mathrm{m}}$-complete for r.e. sets if and only if

(i) A is an r.e. set

(ii) $B \leq_{\mathrm{m}} A$ for any r.e. set B.

Δ is the blank symbol for Turing machines.

$x \not\rightarrow_S^* y$ means that y is not derivable from x for strings x, y over the alphabet Σ_S of the Thue system S.

Handout 5 : Review Material of last year (6)

6.044J/18.423J: Computability, Programming, and Logic
Massachusetts Institute of Technology

Handout 10
14 October 1987

Quiz 1 Solutions

Problem 1 [30 points]. Let

$$\textit{OddSquare} = \{M \mid \text{Turing machine } M \text{ halts on input } \Lambda \text{ with its head on an odd numbered tape square } \}\ .$$

(a) [10 points] Explain why Rice's theorem doesn't apply to *OddSquare*. More precisely, show that $\textit{OddSquare} \neq K_P$ for any property P of r.e. sets.

One can find two distinct machines M and M' which have the same domain but only one of which is in *OddSquare*. Let M be the machine with the description "on input x halt with the head in the first square of the tape", and let M' be the machine with the description "on input x halt with the head in the second square of the tape". Then

$$\text{domain}(M) = \text{domain}(M') = \{\mathtt{a}, \mathtt{b}\}^*\ ,$$

but $M \in \textit{OddSquare}$ and $M' \notin \textit{OddSquare}$. If *OddSquare* were equal to K_P for some property P of r.e. sets we would have both

$$P(\text{domain}(M)) = P(\{\mathtt{a}, \mathtt{b}\}^*)$$

and

$$\neg P(\text{domain}(M')) = \neg P(\{\mathtt{a}, \mathtt{b}\}^*)\ ,$$

which is impossible.

(b) [20 points] Show that *OddSquare* is a $\leq_{\mathtt{m}}$-complete r.e. set anyway.

OddSquare is r.e. because it is easy to write a program which given M runs M on input Λ and halts iff the computation of M on Λ terminated with M's head in an odd numbered square of its tape.

We show that $K_2 \leq_{\mathtt{m}} \textit{OddSquare}$. Since K_2 is a $\leq_{\mathtt{m}}$-complete r.e. set this will imply that *OddSquare* is $\leq_{\mathtt{m}}$-hard for the class of r.e. sets.

For any machine M let M' be the machine with the following description: "on input x run M on x. If M halts then return to the first square of the tape and halt." Let the function f mapping Turing machines to Turing machines be defined by $f(M) = M'$. It is easy to see that f is a total computable function. M halts on x iff M' halts on x with its head in an odd numbered tape square, by definition of M'. So

$$x \in K_2 \quad \text{iff} \quad f(x) \in \textit{OddSquare}\ .$$

So $K_2 \leq_{\mathtt{m}} \textit{OddSquare}$ via f.

Handout 5 (7)

Problem 2 [20 points]. For any total function $T : \{a, b\}^* \to \mathbf{N}$ define the set

$$K^T = \{M \mid M \text{ halts on input } d(M) \text{ in } \leq T(d(M)) \text{ steps}\}$$

and the class of languages

$$Accept(T) = \{L \subseteq \{a, b\}^* \mid \exists M \text{ such that domain}(M) = L \text{ and } \forall x \in \{a, b\}^*, \text{ if } M \text{ halts on input } x \text{ then it does so in } \leq T(x) \text{ steps } \} .$$

Prove that $\overline{K^T} \notin Accept(T)$.

We will use a diagonal argument to derive a contradiction from the assumption that $\overline{K^T} \in Accept(T)$.

Suppose $\overline{K^T} \in Accept(T)$. By definiton of $Accept(T)$ this means there is a Turing machine M whose domain is $\overline{K^T}$ and which for all $x \in \overline{K^T}$ halts on input x in $\leq T(x)$ steps. We consider the action of M on input $d(M)$. We have

$$\begin{aligned} M \in \overline{K^T} \quad &\text{iff} \quad M \text{ halts on input } d(M) \text{ in } \leq T(d(M)) \text{ steps} \\ &\text{iff} \quad M \in K^T , \end{aligned}$$

a contradiction (the first iff is by the definition of M and the second is by the definition of K^T).

Problem 3 [25 points]. For each of the sets below, indicate with a single capital letter whether it is (D) decidable, (R) r.e. but not co-r.e., (C) co-r.e. but not r.e., or (N) neither. No explanation is required and there is no penalty for guessing.

(a) The set of Thue systems S such that there exist $x \neq y \in \Sigma_S^*$ for which $x \not\to_S^* y$.

This problem contained a misprint: we meant to say $x \to_S^* y$ rather than $x \not\to_S^* y$. For the question as we meant it the answer is that the set is decidable. For there are distinct x, y such that $x \to_S^* y$ iff the Thue system has a rewrite rule with its right hand side different from its left hand side. For the question as it appeared on the quiz we think the answer is "C", but the proof that K_0 is many-one reducible to this set seems hard to work out. *All* answers to this question got full credit because of our misprint.

(b) The set of TMs that accept languages containing only strings of even length.

Call this set A. The complement of A is

$$\{M \mid \text{domain}(M) \text{ contains a string of odd length .}\}$$

This set is r.e.: it is easy to write a program which given M searches the domain of M for a string of odd length and halts iff it finds one. So A is co-r.e.. Rice's theorem implies that A is not recursive. So A is not r.e..

Handout 5 (8)

(c) $\{M \mid \exists M'$ such that $d(M) \neq d(M')$ but M and M' accept the same language$\}$.

Given any Turing machine M one can add superfluous instructions to its flowchart to create a different machine M' which accepts exactly the same set of strings as M. So our set is just the set of all strings $\{\mathtt{a}, \mathtt{b}\}^*$ and is certainly decidable.

(d) $\{M \mid M$ writes a Δ symbol during its computation on some input$\}$.

Call this set A. A is r.e. because we can write a program which given M runs M in parallel on all inputs, and halts iff M writes a Δ during its computation on some input. A is not co-r.e. because $K_2 \leq_m A$ and non-co-r.e. inherits up. The reduction takes a machine M to the machine M' whose description is as follows: "on input x, ignore x and run M on input Λ, using a new symbol in place of the blank symbol Δ. If M halts, print a Δ and halt".

(e) $K^T \times \overline{K^T}$ where K^T is given in Problem 2, and $T(w) = 2^{|w|}$.

K^T is decidable for any total computable function T because we can write a program which given M runs M on input $d(M)$ and sees whether or not it halts in $T(d(M))$ steps. Being the complement of a decidable language, $\overline{K^T}$ is also decidable. $K^T \times \overline{K^T}$ can be decided by checking for a given pair of inputs (M_1, M_2) whether $M_1 \in K^T$ and $M_2 \in \overline{K^T}$.

Problem 4 [25 points]. Let

$$A = \{M \mid M \text{ halts on input } \mathtt{a} \text{ and doesn't halt on input } \mathtt{b}\}$$

Show that A is neither r.e. nor co-r.e.

Claim 1. $K_2 \leq_m A$.

Proof. For any Turing machine M let M' be the machine whose description is: "on input x, if $x = \mathtt{b}$ then diverge. Else behave like M on input Λ". The function f defined by $f(M) = M'$ is total computable. M' never halts on input $\mathtt{b}$, and halts on input $\mathtt{a}$ iff M halts on Λ. So

$$M \in K_2 \quad \text{iff} \quad M' \in A\,,$$

and $K_2 \leq_m A$ via f. ■

Claim 2. $\overline{K_2} \leq_m A$.

Handout 5 (9)

Proof. For any Turing machine M let M' be the machine whose description is: "on input x, if $x = \mathtt{a}$ then halt. Else behave like M on input Λ". The function f defined by $f(M) = M'$ is total computable. M' always halts on input $\mathtt{a}$, and halts on input $\mathtt{b}$ iff M halts on Λ. So

$$M \in \overline{K_2} \text{ iff } M' \in A\,,$$

and $\overline{K_2} \leq_{\mathrm{m}} A$ via f. ■

Since non-r.e. and non-co-r.e. inherit up, the claims imply that A is neither r.e. nor co-r.e.

Glossary and notation

$d(M) \in \{\mathtt{a}, \mathtt{b}\}^*$ is the code of the Turing machine M.

$\mathbf{N} = \{0, 1, 2, ...\}$ is the set of natural numbers.

$K_P = \{M \mid P(\mathrm{domain}(M))\}$ where P is a property of r.e. sets.

A set A is $\leq_{\mathrm{m}}$-complete for r.e. sets if and only if

(i) A is an r.e. set

(ii) $B \leq_{\mathrm{m}} A$ for any r.e. set B.

Δ is the blank symbol for Turing machines.

$x \not\rightarrow_S^* y$ means that y is not derivable from x for strings x, y over the alphabet Σ_S of the Thue system S.

Handout 5 Review Material of last year (10)

6.044J/18.423J: Computability, Programming, and Logic Handout 4
Massachusetts Institute of Technology 25 September 1987 (due 01 October)

Problem Set 2

Problem 1. Show that a language $L \subseteq \Sigma^*$ is r.e. iff L is empty or equal to the range of a total recursive function $f : \{a, b\}^* \to \Sigma^*$. (*Hint*: Let $f(pair(x, y)) = y$ if an appropriate Turing machine halts on input y in $|x|$ steps, where *pair* is a pairing function on $\{a, b\}^*$. That is, *pair* : $\{a, b\}^* \times \Sigma^* \to \{a, b\}^*$ is a recursive function which codes pairs of strings into strings over $\{a, b\}$.)

Problem 2. Let D be a decidable language over the alphabet $\{a, b\}$, and let E be an r.e. language over $\{a, b\}$. Let f be a total recursive function from $\{a, b\}^*$ to $\{a, b\}^*$, and let ψ be a partial recursive function from $\{a, b\}^*$ to $\{a, b\}^*$.

(a) Show that $f^{-1}(D) = \{x \in \{a, b\}^* \mid f(x) \in D\}$ is recursive.

(b) Show that $\psi^{-1}(E) = \{x \in \{a, b\}^* \mid \psi(x) \in E\}$ is r.e.

(c) Show that $f(D) = \{f(x) \in \{a, b\}^* \mid x \in D\}$ is r.e.

(d) Give examples to show that "r.e." cannot be replaced by "recursive" in (b) and (c).

Problem 3. For each of the sets below, state whether it is recursive,r.e. but not co-r.e., or co-r.e. but not r.e., and briefly explain why.

(a) The set of TMs that halt on no inputs.

(b) The set of TMs that halt on blank tape in $\leq 10^9$ steps.

(c) The set of TMs that halt on blank tape in $\geq 10^9$ steps.

(d) $\{M \mid 10^9 < |d(M)|\}$.

(e) The set of integers n such that there are at least n occurrences of the digit '8' after the n^{th} place in the decimal expansion of $(2.7)^{\ln \pi}$. (*Hint*: Case 1: there are infinitely many '8's in the decimal expansion of $(2.7)^{\ln \pi}$, Case 2: there are finitely many '8's in the decimal expansion of $(2.7)^{\ln \pi}$.)

Handout 5 (11)

Problem 4. Define

$$K_{\text{tot}} = \{M \mid M \text{ is a Turing machine which halts on every input } x \in \{\mathtt{a},\mathtt{b}\}^*\} .$$

(a) Show that $K_2 \leq_{\text{m}} K_{\text{tot}}$, where K_2 is the blank-tape halting problem.

(b) Use a diagonalization argument to show that K_{tot} is not the range of a total recursive function from $\{\mathtt{a},\mathtt{b}\}^*$ to $\{\mathtt{a},\mathtt{b}\}^*$. (*Hint*: Let $f : \{\mathtt{a},\mathtt{b}\}^* \to K_{\text{tot}}$ be onto and recursive. Define the "diagonal function" $\rho : \{\mathtt{a},\mathtt{b}\}^* \to \{\mathtt{a},\mathtt{b}\}$ by

$$\rho(x) = \begin{cases} \mathtt{a} & \text{if } d(M) = f(x) \text{ and } M \text{ halts on input } x \text{ with output } \mathtt{b}, \\ \mathtt{b} & \text{otherwise.} \end{cases}$$

Show that a contradiction results from the assumption that f is computable by some machine M_0.)

(c) Conclude from (a) and (b) that K_{tot} is neither r.e. nor co-r.e.

Handout 5. Review Material of last year (12)

6.044J/18.423J: Computability, Programming, and Logic
Massachusetts Institute of Technology

Handout 7
07 October 1987

Problem Set 2 Solutions

Announcement: The Quiz is on Wednesday 07 October, 7-9:00PM, in 34-302. It is open book.

Problem 1. Show that a language $L \subseteq \Sigma^*$ is r.e. iff L is empty or equal to the range of a total recursive function $f : \{\mathtt{a}, \mathtt{b}\}^* \to \Sigma^*$. (*Hint*: Let $f(\mathit{pair}(x, y)) = y$ if an appropriate Turing machine halts on input y in $|x|$ steps, where *pair* is a pairing function on $\{\mathtt{a}, \mathtt{b}\}^*$. That is, $\mathit{pair} : \{\mathtt{a}, \mathtt{b}\}^* \times \Sigma^* \to \{\mathtt{a}, \mathtt{b}\}^*$ is a recursive function which codes pairs of strings into strings over $\{\mathtt{a}, \mathtt{b}\}$.)

We proved in class that $L \subseteq \Sigma^*$ is r.e. iff it is the range of a partial recursive function $\psi : \{\mathtt{a}, \mathtt{b}\}^* \to \Sigma^*$, a fact we will use here.

Suppose $L \subseteq \Sigma^*$ is empty or the range of a total recursive function $f : \{\mathtt{a}, \mathtt{b}\}^* \to \Sigma^*$. In either case it is the range of a partial recursive function $\psi : \{\mathtt{a}, \mathtt{b}\}^* \to \Sigma^*$, for if $L = \emptyset$ we can let ψ be the function which is undefined for all inputs, and otherwise we can let ψ be f. So L is r.e.

Conversely suppose L is recursively enumerable. So L is the range of a partial recursive function $\psi : \{\mathtt{a}, \mathtt{b}\}^* \to \Sigma^*$. If $L = \emptyset$ then it is certainly r.e., so suppose L is non-empty and let z be an element of L. Let M be a machine computing ψ. Define $f : \{\mathtt{a}, \mathtt{b}\}^* \to \Sigma^*$ by

$$f(w) = \begin{cases} y & \text{if } \mathtt{w} = \mathit{pair}(x, y) \text{ and } M \text{ halts on input } y \text{ in} \leq |x| \text{ steps} \\ z & \text{otherwise} \end{cases}$$

The function f is certainly total, and since *pair* is a good coding function it is also computable. M halts on input y iff it does so in $|x|$ steps for some x. So the range of f is the set of outputs of M: $\mathrm{range}(f) = \mathrm{range}(\psi) = L$.

Problem 2. Let D be a decidable language over the alphabet $\{\mathtt{a}, \mathtt{b}\}$, and let E be an r.e. language over $\{\mathtt{a}, \mathtt{b}\}$. Let f be a total recursive function from $\{\mathtt{a}, \mathtt{b}\}^*$ to $\{\mathtt{a}, \mathtt{b}\}^*$, and let ψ be a partial recursive function from $\{\mathtt{a}, \mathtt{b}\}^*$ to $\{\mathtt{a}, \mathtt{b}\}^*$.

(a) Show that $f^{-1}(D) = \{x \in \{\mathtt{a}, \mathtt{b}\} \mid f(x) \in D\}$ is recursive.

(b) Show that $\psi^{-1}(E) = \{x \in \{\mathtt{a}, \mathtt{b}\}^* \mid \psi(x) \in E\}$ is r.e.

(c) Show that $f(D) = \{f(x) \in \{\mathtt{a}, \mathtt{b}\}^* \mid x \in D\}$ is r.e.

(d) Give examples to show that "r.e." cannot be replaced by "recursive" in (b) and (c).

Handout 5 (12)

Let

- M_D be a machine deciding D.
- M_E be a machine accepting E.
- M_f be a machine computing f.
- M_ψ be a machine computing ψ.

(a) A decider M for $f^{-1}(D)$ works as follows. On input $x \in \{a, b\}^*$,

- Compute $y = f(x)$ (using M_f).
- Accept x if M_D accepts y, and reject x if M_D rejects y.

(b) An acceptor M for $\psi^{-1}(E)$ works as follows. On input $x \in \{a, b\}^*$,

- Run M_ψ on input x.
- If M_ψ halts then let y denote its output, and run M_E on input y.
- If M_E halts on y then halt.

(c) Define a partial function $\psi : \{a, b\}^* \to \{a, b\}^*$ by

$$\psi(x) = \begin{cases} f(x) & \text{if } x \in D \\ \uparrow & \text{otherwise} \end{cases}$$

(where $\uparrow$ denotes "undefined"). Since f is total recursive and D is decidable, ψ is partial recursive. The range of ψ is $f(D)$. So $f(D)$ is r.e.

(d) Let E be any r.e. non-recursive set, and let ψ be the identity mapping on $\{a, b\}^*$. Then $\psi^{-1}(E) = E$ is not recursive. This provides the counterexample for (b).

For (c) let $D = \{a, b\}^*$ and let f be a total recursive function whose range is K_0. We know such an f exists by problem 1.

Problem 3. For each of the sets below, state whether it is recursive,r.e. but not co-r.e.. or co-r.e. but not r.e., and briefly explain why.

(a) The set of TMs that halt on no inputs.

(b) The set of TMs that halt on blank tape in $\leq 10^9$ steps.

(c) The set of TMs that halt on blank tape in $\geq 10^9$ steps.

(d) $\{M \mid 10^9 < |d(M)|\}$.

Handouts (18)

(e) The set of integers n such that there are at least n occurrences of the digit '8' after the n^{th} place in the decimal expansion of $(2.7)^{\ln \pi}$. (*Hint*: Case 1: there are infinitely many '8's in the decimal expansion of $(2.7)^{\ln \pi}$, Case 2: there are finitely many '8's in the decimal expansion of $(2.7)^{\ln \pi}$.)

(a) The set $A = \{M |\ M$ does not halt on x for all $x \in \{\mathtt{a}, \mathtt{b}\}^*\}$ is co-r.e. because it is easy to accept those M which halt on some input. To show that A is not r.e. either show that $K_2 \leq_m A$ as done in class for similar examples, or appeal to Rice's theorem.

(b) $\{M \mid M$ halts on input Λ in $\leq 10^9$ steps$\}$ is recursive. Given M, just run it for 10^9 steps on input Λ and see whether or not it halts.

(c) We omit the easy argument that the set $A = \{M \mid M$ halts on input Λ in $\geq 10^9$ steps$\}$ is recursively enumerable. A is not co-r.e. because $K_2 \leq_m A$; the reduction maps an input machine M into a machine M' which on any input wastes 10^9 steps and then acts like M on that input.

(d) $A = \{M \mid 10^9 < d(M)\}$ is recursive. It is easy to write a program which accepts precisley those strings $x \in \{\mathtt{a}, \mathtt{b}\}^*$ of length $> 10^9$ which are well-formed Turing machine codes. In fact, given our convention of regarding any string in $\{\mathtt{a}, \mathtt{b}\}^*$ as the code of some Turing machine, A is just the set of all strings of length $> 10^9$.

(e) This is recursive. There are either an infinite or a finite number of '8's in the decimal expansion of $(2.7)^{\ln \pi}$. In the first case our set is the set of all integers. In the second case it is a finite set. In either case it is recursive.

Problem 4. Define

$$K_{\text{tot}} = \{M \mid M \text{ is a Turing machine which halts on every input } x \in \{\mathtt{a}, \mathtt{b}\}^*\}\ .$$

(a) Show that $K_2 \leq_m K_{\text{tot}}$, where K_2 is the blank-tape halting problem.

(b) Use a diagonalization argument to show that K_{tot} is not the range of a total recursive function from $\{\mathtt{a}, \mathtt{b}\}^*$ to $\{\mathtt{a}, \mathtt{b}\}^*$. (*Hint*: Let $f : \{\mathtt{a}, \mathtt{b}\}^* \to K_{\text{tot}}$ be onto and recursive. Define the "diagonal function" $\rho : \{\mathtt{a}, \mathtt{b}\}^* \to \{\mathtt{a}, \mathtt{b}\}$ by

$$\rho(x) = \begin{cases} \mathtt{a} & \text{if } M \text{ halts on input } x \text{ with output } \mathtt{b}, \text{ where } d(M) = f(x) \\ \mathtt{b} & \text{otherwise.} \end{cases}$$

Show that a contradiction results from the assumption that f is computable by some machine M_0.)

(c) Conclude from (a) and (b) that K_{tot} is neither r.e. nor co-r.e.

Handout 14

(a) Let M be a Turing machine. Let M' be the machine which on input x ignores x and behaves exactly like M would on input Λ. So for any x,

$$M' \text{ halts on } x \text{ iff } M \text{ halts on } \Lambda.$$

So

$$M' \in K_{\text{tot}} \text{ iff } M \in K_2.$$

The translator function f taking M to M' is total recursive. So $K_2 \leq_m K_{\text{tot}}$.

(b) Assume $f : \{a, b\}^* \to K_{\text{tot}}$ is onto and recursive.

Claim 1. The function ρ is computable.

Proof. Consider a machine M_ρ operating as follows: " On input $x \in \{a, b\}^*$, compute $f(x)$. Let M be the machine for which $d(M) = f(x)$. Run M on input x. By definition $M \in K_{\text{tot}}$, so M will halt on input x. If the output of M is b, output a and halt. Otherwise output b and halt." This machine M_ρ clearly computes ρ. ■

The function ρ is by definition total. So $M_\rho \in K_{\text{tot}}$ by Claim 1. Since f is onto, there is an $x_0 \in \{a, b\}^*$ such that $f(x_0) = d(M_\rho)$. Now

$$\begin{aligned} \rho(x_0) = a \quad &\text{iff} \quad M \text{ halts on input } x_0 \text{ with output b, where } d(M) = f(x_0) \\ &\text{iff} \quad M_\rho \text{ halts on input } x_0 \text{ with output b} \\ &\text{iff} \quad \rho(x_0) = b \,, \end{aligned}$$

a contradiction (the first iff follows from the definition of ρ, the second because $d(M_\rho) = f(x_0)$, and the last because M_ρ computes ρ). So f could not have been both onto and computable.

(c) K_{tot} is not co-r.e. by (a) since non co-r.e. inherits up. It is not r.e. because (b) says it is not the range of a total recursive function.

6.044 Problem set #1 Solutions

Problem 1:

Note that pair is acting on couples, so that pair is a total function iff $Dom(pair) = \{a,b\}^* \times \{a,b\}^*$.

Some of you wrote $Dom(pair) = \{a,b\}^*$ which is wrong!

$1^a \circ left_b$ is a function that can be computed by the following machine:

"On input $x \in \{a,b\}^*$, read in from left to right the characters from x in pairs. If the two characters of the pair are the same, then add that character on the right of the string representing the output we are computing. If the two characters are "ab" or "ba" or "aΛ" or "bΛ" or "$\Lambda\Lambda$" halt immediately and return the current answer string."

With this definition $left_b$ is a total function on $\{a,b\}^*$. Note however, that on $range(pair_b)$ the cases ba, aΛ, bΛ, or $\Lambda\Lambda$ cannot occur.

- $right_b$ is described by putting "right to left" instead of "left to right" and "left" — "right" ... in the above paragraph

- $pair_b$ is computable: the only operations involved are duplication of letters and substitution of a coma into string ab.

1.[b] By assumption pair is a total computable function on $\{a,b\}^* \times \{a,b\}^*$ and left, right ——— $\{a,b\}^*$.

Using this we can define a TM M' that will return a on input x if $x \in$ range(pair), and returning b on input x if $x \notin$ range(pair):

" Apply left and right to x, apply pair to the result of the two, and then check to see whether the output from pair equals the original input x. If it does return a. If not return b."

1c. DEFINE LEFT(X) AS FOLLOWS:

IF $X \notin$ RANGE(P), THEN Λ

OTHERWISE, EXHAUSTIVELY SEARCH THE SPACE OF PAIRS OF STRINGS UNTIL FINDING A PAIR (Y,Z) SUCH THAT PAIR$(Y,Z) = X$

OUTPUT Y AND HALT.

THE PRECEDING DESCRIPTION IS CLEARLY COMPUTABLE GIVEN THAT RANGE(P) IS RECURSIVE; IT IS ALSO TOTAL.

FROM THE DEFINITION, IT IS CLEAR THAT LEFT(PAIR$(Y,Z)) = Y$

RIGHT IS DEFINED SIMILARLY.

(14) The following coding scheme for computing pair (x,y):

y \ x	Λ	a	b	aa	ab	ba
Λ	Λ	b	ba	aba		
a	a	ab	aab			
b	aa	aaa				
aa	bb					
ab	:					
ba						

The coding and decoding of pairs is done by looking at the table. (Table entries are filled by a canonical listing of words in $\{a,b\}^*$.

NOTE: $\lceil a \rceil$ IS THE CEILING FUNCTION:
THE INTEGER n SUCH THAT $n-1 < a \leq n$

(1e) Let the first string, x, have length n characters. Let the second string, y, have length m characters.

Encode the length of x in binary, with $a=0$ and $b=1$. This binary length description will take $\lceil \log_2(n) \rceil$ bits. Using a left-side Most Significant Bit, the leftmost bit of this code must be a 1 ("b"). Thus, we can describe the dividing point between x and y with $\log_2 |x|$ bits. We must, however, find a way to know when the "header" giving the length ends, and x itself begins. We do this by adding $\lceil \log_2 n \rceil$ additional "a"s to the front... we can tell where the length descriptor starts because it will be the first "b" encountered (the MSB), and we then know the size of the length description and the dividing point of x and y.

$$\text{Total length: } |x| + |y| + 2\lceil \log_2 |x| \rceil$$

TO CONCLUDE, NOTE THAT $\lim_{|n| \to \infty} \frac{\log_2 |n|}{|n|} = 0$

SO $\exists N(\epsilon) : |n| \geq N \Rightarrow 2 \log |x| \leq \epsilon |n|$

2. a-c. SEE THE SOLUTIONS TO THE ANALOGOUS PROBLEMS IN HANDOUT 5.

d. PAD(M) IS DECIDABLE BECAUSE THERE IS A FUNCTION WHICH DETERMINES MEMBERSHIP IN IT: RUN M ON x FOR n STEPS, AND IF ON THE n^{TH} STEP IT HALTS, THEN PAIR(x, a^n) IS IN THE SET.

CONSIDER f = LEFT AND D = PAD(M), WHERE THE DOMAIN OF M IS AN R.E. BUT NOT CO-R.E. SET.
WE JUST SHOWED THAT PAD(M) IS DECIDABLE.
LEFT(PAD(M)) IS THE DOMAIN OF M – THE VALUES ON WHICH M HALTS.
THIS IS A NONRECURSIVE SET.

3a WE SHOULDN'T HAVE ASSIGNED THIS PROBLEM YET SINCE THE SET, WHICH WAS INTRODUCED IN LECTURE AS K_{TOT}, IS NEITHER R.E. NOR CO-R.E. THE PROOF WAS GIVEN IN LECTURE

3b. The set of all TM's that halt on a blank tape in a number ~~of~~ steps between 20073 and 997631.

This set is recursive because we can simply run the machine on a blank tape for 997632 times or halt, whichever comes first. If it halts before 20073 or doesn't halt, then it's not in the set; otherwise it is.

3 c) Call it $= \{ M \mid M \text{ halts on "aba" in} \geq 749 \text{ steps} \}$.

- A is obviously re:

For any machine M run M on aba (For the very formal of you this can be done through Thm 1 of your notes, page 3). If it halts before 749 steps it is not in the set. Otherwise if it halts it's in the set.

(But we cannot say anything about its membership in the set until it does halt.)

this membership test conforms to the definition of a Re. set.

- A is not co-RE:

This is proved by showing that $K_0 \leq_m A$ and using Corollary 4 pt of handout 2. For this purpose construct ~~a machine N~~ for any Turing machine M, a machine N that acts as follows on input x:

"Do nothing for 748 steps.
Erase x.
Act like M on input Λ"

then it is clear that:

M halts on Λ iff N halts on aba in ≥ 749 steps.

(ie) $M \in K_0$ iff $N \notin A$

It is obvious that $M \xrightarrow{f} N$ is computable

Hence $K_0 \leq_m A$ as claimed.

d. THE SET IS RECURSIVE.
THE FUNCTION WHICH CALCULATES $|d(M)|$ AND COMPARES IT TO 10^9 IS DECIDABLE.

e. CASE 1 THE SET IS THE SET OF ALL NATURAL NUMBERS. THIS IS A RECURSIVE SET.
CASE 2 THE SET IS FINITE. ALL FINITE SETS ARE RECURSIVE.
SO THE SET IS RECURSIVE.

Quiz 1

Instructions. Do all four (4) problems; a total of 100 points is allocated as shown on each problem. This exam is *open book*. There is a short glossary and summary of notation on the last page. You have two hours. Good luck.

Problem 1 [25 points]. For each of the sets below, indicate with a single capital letter whether it is (D) decidable, (R) r.e. but not co-r.e., (C) co-r.e. but not r.e., or (N) neither. No explanation is required and there is no penalty for guessing.

(a) $\{(M_1, M_2) \mid \text{domain}(M_1) \cap \text{domain}(M_2) = \emptyset\}$.

(b) $K_1 \times K_0$

(c) $\{M \mid M$ is a Turing machine and $\exists x \in \{\mathtt{a}, \mathtt{b}\}^*$ such that M accepts x in at most $|x|^2$ steps$\}$.

(d) $\{x \in \{\mathtt{a}, \mathtt{b}\}^* \mid \exists$ Turing machine M accepting x in at most $|x|^2$ steps$\}$.

(e) $\{p \mid p$ is a polynomial in n variables with integer coefficients, $n > 0$, and for all integers $x_1, \ldots, x_n$, it is the case that $p(x_1, ..., x_n) \neq 9\}$.

Problem 2 [25 points]. Show that a set A is r.e. iff A is finite or is the range of a total one-to-one recursive function. (*Hint* : You may find it helpful to use the fact stated in Handout 2, Theorem 7, page 5 that every nonempty r.e. set is the range of a total recursive function.)

Problem 3 [25 points]. *Note*: Do *either* part (a) *or* part (b):

(a) Prove the statement of Handout 2, page 7:

> If A is r.e. but not recursive, then A and $\overline{A}$ are $\leq_m$-incomparable.

(*Hint*: Assume to the contrary that, *e.g.*, $\overline{A} \leq_m A$.)

(b) Prove that $\{M \mid M$ halts on input $d(M)$ in a number of steps divisible by $3\} \equiv_m K_1$.

Problem 4 [25 points]. A Turing machine is said to be *n-bounded* iff it stops in at most n steps on every input. Let

$$A_n = \{M \mid M \text{ is } n\text{-bounded}\} \text{ and } A = \bigcup_n A_n.$$

(a) Explain why A_{29} is decidable. (*Hint*: Use the fact that it takes one step to read an input).

(b) Explain why A is r.e.

(c) Explain why Rice's theorem does not apply to A. More precisely show that $A \neq K_P$ for any property P of r.e. sets.

(d) Prove that A is not co-r.e.

Glossary and notation

$d(M) \in \{\mathtt{a}, \mathtt{b}\}^*$ is the code of the Turing machine M.

$K_P = \{M \mid P(\text{domain}(M))\}$ where P is a property of r.e. sets.

$K_0 = \{(M, x) \mid M \text{ is a Turing machine that halts on input } x\}$.

$K_1 = \{M \mid M \text{ is a Turing machine that halts on input } d(M)\}$.

Problem Set 2

Reading assignment. (1) Manna text on Post Machines and Post Corresondence Problem. (2) Excerpts distributed in class from the Lewis and Papadimitriou text.

Problem 1.

(a) Prove that if A and B are r.e. sets then so is $A \cup B$.

(b) Show that every set A contained in $\{\mathbf{a}, \mathbf{b}\}^*$ is equal to a countable union $A_1 \cup A_2 \cup A_3 \ldots$ such that for $i \geq 1$, the set A_i is decidable. Conclude that the family of r.e. sets is not closed under countable unions.

Problem 2. Define the Initialized Post Correspondance Problem (IPCP) as follows:

- An instance of the IPCP is the same as an instance of the PCP, namely a finite ordered sequence $((\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_n, \beta_n))$ of pairs of words.
- A solution for an instance of the IPCP is one for the PCP, except that an IPCP solution *must start* with the pair (α_1, β_1).

(a) For any PCP-instance S as above, let c be a letter not in the alphabet of S. Define an IPCP-instance, S' to consist of the pairs (c, cc), the pair in S, and also all pairs of the form $(c\alpha_i, \beta_i)$ for $(\alpha_i, \beta_i) \in S$, with (c, cc) as the initial pair.

Carefully prove that S has a solution iff S' has an initialized solution. Conclude that PCP $\leq_{\mathrm{m}}$ IPCP.

(b) Prove that in fact, IPCP $\equiv_{\mathrm{m}}$ PCP. (*Hint:* Transform an IPCP-instance T into a PCP-instance T' by padding the pairs of T with alternating * as in Manna's proof of Theorem 1.9, page 62.)

Problem 3.

(a) Describe the successive boxes of its flow-chart entered by the Post Machine below, and the successive contents of x, in the computation starting with $x = \Lambda$, the empty string.

(b) Exhibit the PCP constructed from this flow-chart according to Manna's proof of page 62.

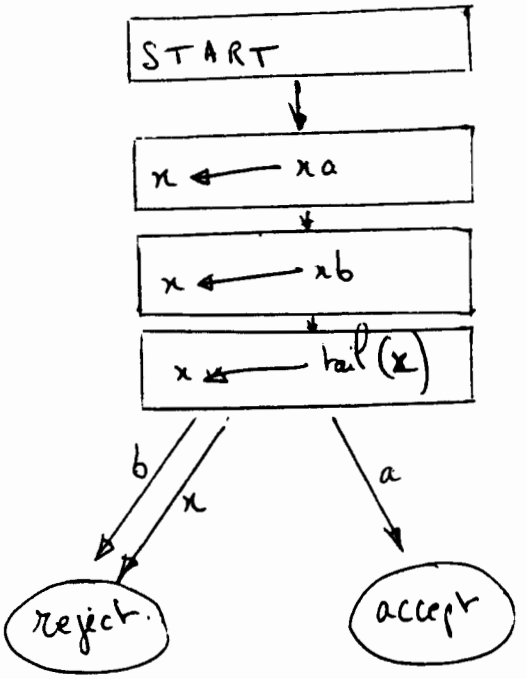
(c) Exhibit the (necessarily unique!) shortest solution of the PCP of the the previous part.

The next two problems are taken from Lewis' and Papadimitriou's book, "Elements of the Theory of Computation".

Problem 4. Do problem 6.5.1 of L.& P.

Problem 5. Do problem 6.5.7 of L.& P.

flow chart of Problem 3 :



Handout 9.

Attachment to handout 8 : Problem Set 2.

Copies of "Elements of the theory of computation"
By Lewis and Papadimitriou.

pages 168 to 179
pages 296 to 301
pages 308 to 309. (problems for Problem Set 2.)

PRENTICE-HALL SOFTWARE SERIES

Brian W. Kernighan, advisor

ELEMENTS OF THE THEORY OF COMPUTATION

Harry R. Lewis
Harvard University

Christos H. Papadimitriou
Massachusetts Institute of Technology

Prentice-Hall, Inc.
Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

LEWIS, HARRY R
Elements of the theory of computation.

(Prentice-Hall software series)
Bibliography: p.
Includes index.
1. Machine theory. 2. Formal languages. 3. Computational complexity. 4. Logic, Symbolic and mathematical. I. Papadimitriou, Christos H., joint author.
II. Title. III. Series.
QA267.L49 511 80-21293
ISBN 0-13-273417-6

Editorial/Production Supervision
and Interior Design by Kathryn Gollin Marshak
Cover Design by Jayne Conte
Manufacturing Buyer: Joyce Levatino

Printed in the United States of America

10 9 8 7

Prentice-Hall International, Inc., *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall of Canada, Ltd., *Toronto*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Prentice-Hall of Southeast Asia Pte. Ltd., *Singapore*
Whitehall Books Limited, *Wellington, New Zealand*

For Marlyn and Xanthippi

CHAPTER 4

TURING MACHINES

4.1 THE DEFINITION OF A TURING MACHINE

We have seen in the last two chapters that neither finite automata nor pushdown automata can be regarded as truly general models for computers, since they are not capable of recognizing even such simple languages as $\{a^n b^n c^n : n \geq 0\}$. In this chapter we take up the study of devices that can recognize this and many more complicated languages. These devices will also be used in ways other than as language recognizers: as computers of functions from strings to strings, for example, and as machines for systematically enumerating the strings in a language.

Although these devices, called **Turing machines** after their inventor Alan Turing (1912–1954), are more general than the automata previously studied, their basic appearance is similar to those automata. A Turing machine consists of a finite control, a tape, and a head that can be used for reading or writing on that tape. The formal definitions of Turing machines and their operation are in the same mathematical style as those used for finite and pushdown automata. So in order to gain the additional computational power and generality of function that Turing machines possess, we shall not move to an entirely new sort of model for a computer.

Nevertheless, Turing machines are not simply one more class of automata, to be replaced later on by a yet more powerful type. We shall see in this chapter that, as primitive as Turing machines seem to be, attempts to strengthen them seem not to have any effect. For example, the addition of extra tapes or extra heads on those tapes results in automata that turn out to

be no stronger in terms of computing power than basic Turing machines. We show results of this kind by simulation methods: We can convert any "augmented" Turing machine into a standard machine which functions in an analogous way. Thus any computation that can be carried out on the fancier type of machine can actually be carried out on a Turing machine of the standard variety.

So the Turing machines seem to form a stable and maximal class of automata, in terms of the computations they can perform. In the next chapter we take this idea one step further: we show that Turing machines are equivalent in power to entirely different ways of carrying out computations, not based on the idea of an automaton with states and tapes. Indeed, we shall argue that *any* way of formalizing the idea of a "computational procedure" or an "algorithm" is equivalent to the idea of a Turing machine.

But this is getting ahead of our story. The important points to remember by way of introduction are that Turing machines are designed to satisfy simultaneously these three criteria.

(a) They should be automata; that is, their construction and function should be in the same general spirit as the devices previously studied.

(b) They should be as simple as possible to describe, define formally, and reason about.

(c) They should be as general as possible in terms of the computations they can carry out.

Now let us look more closely at these machines. In essence, a Turing machine consists of a finite-state control unit and a tape (see Figure 4–1).

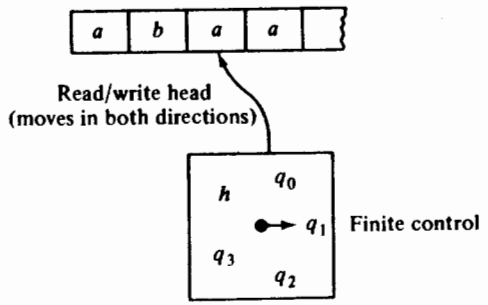


Figure 4–1

Communication between the two is provided by a single head, which reads symbols from the tape and is also used to change the symbols on the tape. The control unit operates in discrete steps; at each step it performs two func-

tions in a way dependent on its current state and the tape symbol currently scanned by the read/write head.

1. Put the control unit in a new state.
2. *Either:*
   (a) Write a symbol in the tape square currently scanned, replacing the one already there; *or*
   (b) Move the read/write head one tape square to the left or right.

The tape has a left end, but it extends indefinitely to the right. Since the machine can move its head only one square at a time, after any finite computation only finitely many tape squares will have been visited. (If the machine attempts to move its head to the left off the end of the tape, it ceases to operate.)

A Turing machine is supplied with input by inscribing that input string on tape squares at the left end of the tape. The rest of the tape initially contains blank symbols. The machine is free to alter its input in any way it sees fit, as well as to write on the unlimited blank portion of the tape to the right.

Because a Turing machine can write on its tape, it can leave an answer on the tape at the end of a computation. Therefore we do not need to provide special "final" states, as we have done for finite and pushdown automata. There is a special **halt state** that is used to signal the end of computation, but no favorable or unfavorable connotation should be inferred. Since this halt state will be the same for all Turing machines, we denote it by h; this symbol will not be used hereafter for any other purpose. Similarly, the blank symbol will be fixed as #. Also, we shall use the distinct symbols L and R to denote movement of the head to the left or right; we assume that these two symbols are not members of any alphabet we consider.

We can now present the formal definition of a Turing machine.

Definition 4.1.1

A **Turing machine** is a quadruple (K, Σ, δ, s), where

K is a finite set of **states**, *not* containing the halt state h;

Σ is an alphabet, containing the blank symbol #, but not containing the symbols L and R;

$s \in K$ is the **initial state**;

δ is a function from $K \times \Sigma$ to $(K \cup \{h\}) \times (\Sigma \cup \{L, R\})$.

If $q \in K$, $a \in \Sigma$, and $\delta(q, a) = (p, b)$, then M, when in state q and scanning symbol a, will enter state p, and (1) if b is a symbol in Σ, rewrite the a as b, or (2) if b is L or R, move its head in direction b. Since δ is a function,

the operation of M is deterministic and will stop only when M enters the halt state or attempts to move left off the end of the tape.

Example 4.1.1

Consider the Turing machine $M = (K, \Sigma, \delta, s)$, where

$$K = \{q_0, q_1\}$$
$$\Sigma = \{a, \#\}$$
$$s = q_0$$

and δ is given by the following table.

q	σ	$\delta(q, \sigma)$
q_0	a	$(q_1, \#)$
q_0	#	$(h, \#)$
q_1	a	(q_0, a)
q_1	#	(q_0, R)

When M is started in its initial state q_0, it scans its head to the right, changing all a's to #'s as it goes, until it finds a tape square already containing #; then it halts. (Changing a nonblank symbol to the blank symbol will be called **erasing** the nonblank symbol.) To be specific, suppose that M is started with its head scanning the first of four a's, the last of which is followed by a #. Then M will go back and forth between states q_0 and q_1 four times, alternately changing an a to a # and moving the head right; the first and last lines of the table for δ are the relevant ones during this sequence of moves. At this point, M will find itself in state q_0 scanning # and, according to the second line of the table, will halt. Note that the third line of the table, that is, the value of $\delta(q_1, a)$, is irrelevant, since M can never be in state q_1 scanning an a if it is started in state q_0. Nevertheless, some value must be associated with $\delta(q_1, a)$, since δ is required to be a function with domain $K \times \Sigma$.

Example 4.1.2

Consider the Turing machine $M = (K, \Sigma, \delta, s)$, where

$$K = \{q_0\}$$
$$\Sigma = \{a, \#\}$$
$$s = q_0$$

and δ is given by the following table.

q	σ	$\delta(q, \sigma)$
q_0	a	(q_0, L)
q_0	#	$(h, \#)$

This machine scans to the left until it finds a # and then halts. If every tape square from the head position back to the leftmost tape square contains an a, then M will run its tape head off the left end of the tape and cease operating without halting. (In this case we shall say that M **hangs**.)

We now formalize the operation of a Turing machine.

To specify the status of a Turing machine computation, we need to specify the state, the contents of the tape, and the position of the head. Since all but a finite initial portion of the tape will be blank, the contents of the tape can be specified by a string. We choose to break that string into three pieces: the part, possibly empty, to the left of the scanned square; the single symbol in the scanned square; and the part, possibly empty, to the right of the scanned square. Moreover, so that no two of these (string, symbol, string) triples will correspond to the same combination of head position and tape contents, we insist that the last string not end with a blank (all tape squares to the right of the last one explicitly represented are assumed to contain blanks anyway). These considerations lead us to the following definitions.

Definition 4.1.2

A **configuration** of a Turing machine $M = (K, \Sigma, \delta, s)$ is a member of

$$(K \cup \{h\}) \times \Sigma^* \times \Sigma \times (\Sigma^*(\Sigma - \{\#\}) \cup \{e\}).$$

Thus (q, e, a, aba), $(h, \#\#, \#, \#a)$, and $(q, \#a\#, \#, e)$ are configurations (see Figure 4–2), but $(q, baa, a, bc\#)$ is not. A configuration whose state component is h will be called a **halted configuration**.

Sometimes we wish to depict the appearance of the tape and the position of the head without indicating the state of the finite control. In this case we write $w\underline{a}u$ instead of the configuration (q, w, a, u): The underlined symbol indicates the head position. For the three configurations illustrated in Figure 4–2, the tapes and head positions would be represented as $\underline{a}aba$, $\#\#\underline{\#}\#a$, and $\#a\#\underline{\#}$. Also, we can write configurations by including the state together with the notation for the tape and head position. That is, we can write (q, w, a, u) as $(q, w\underline{a}u)$. Using this convention, we would write the three configurations shown in Figure 4–2 as $(q, \underline{a}aba)$, $(h, \#\#\underline{\#}\#a)$, and $(q, \#a\#\underline{\#})$. We call this the **abbreviated** notation for configurations.

For Turing machine configurations, **yields in one step** is defined as follows.

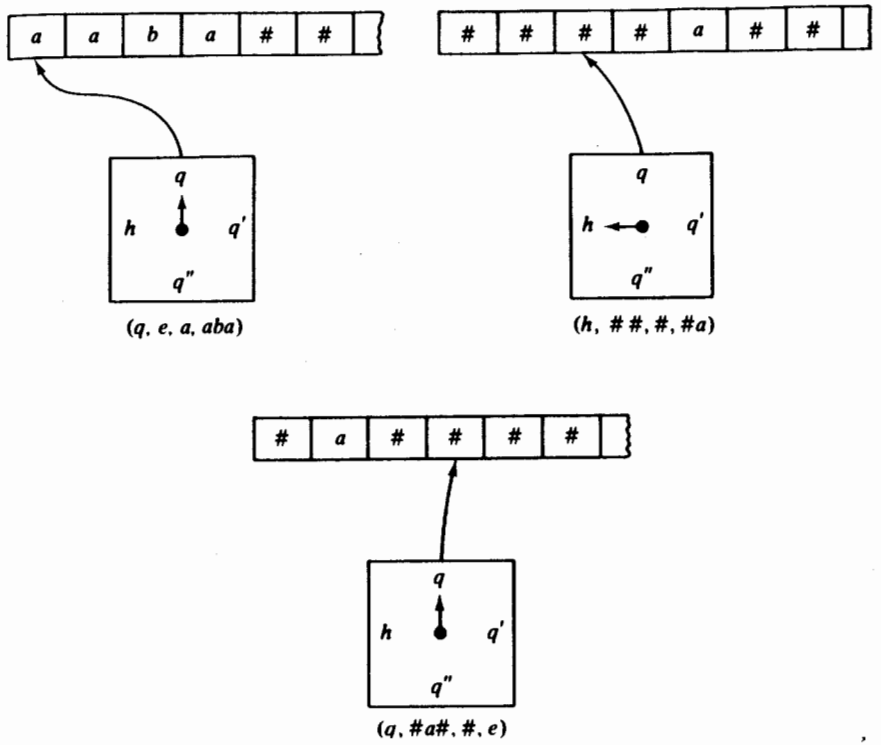


Figure 4–2

Definition 4.1.3

Let $M = (K, \Sigma, \delta, s)$ be a Turing machine and let (q_1, w_1, a_1, u_1) and (q_2, w_2, a_2, u_2) be configurations of M. Then

$$(q_1, w_1, a_1, u_1) \vdash_M (q_2, w_2, a_2, u_2)$$

if and only if, for some $b \in \Sigma \cup \{L, R\}$, $\delta(q_1, a_1) = (q_2, b)$ and either

1. $b \in \Sigma$, $w_1 = w_2$, $u_1 = u_2$, and $a_2 = b$;

or

2. $b = L$, $w_1 = w_2a_2$, and either
   (a) $u_2 = a_1u_1$, if $a_1 \neq \#$ or $u_1 \neq e$, or
   (b) $u_2 = e$, if $a_1 = \#$ and $u_1 = e$;

or

3. $b = R$, $w_2 = w_1a_1$, and either
   (a) $u_1 = a_2u_2$, or
   (b) $u_1 = u_2 = e$ and $a_2 = \#$.

In Case 1, M rewrites a symbol without moving its head. In Case 2, M moves its head one square to the left; if it is moving to the left off blank tape, the blank symbol on the square just scanned disappears from the configuration. In Case 3, M moves its head one square to the right; if it is moving onto blank tape, a new blank symbol appears in the configuration as the new scanned symbol.

Example 4.1.3

To illustrate these cases, let $w, u \in \Sigma^*$, where u does not end with #, and let $a, b \in \Sigma$.

Case 1. $\delta(q_1, a) = (q_2, b)$
Example: $(q_1, w\underline{a}u) \vdash_M (q_2, w\underline{b}u)$

Case 2. $\delta(q_1, a) = (q_2, L)$
Example for (a): $(q_1, wb\underline{a}u) \vdash_M (q_2, w\underline{b}au)$
Example for (b): $(q_1, wb\#) \vdash_M (q_2, w\underline{b})$

Case 3. $\delta(q_1, a) = (q_2, R)$
Example for (a): $(q_1, w\underline{a}bu) \vdash_M (q_2, wa\underline{b}u)$
Example for (b): $(q_1, w\underline{a}) \vdash_M (q_2, wa\#)$

Note that if $b = L$ and $w_1 = e$, then (q_1, w_1, a_1, u_1) yields no configuration, since there is no $w_2 \in \Sigma^*$ and $a_2 \in \Sigma$ such that $w_1 = w_2a_2$. Such a configuration will be called a **hanging configuration**. On the other hand, every configuration that is not a halted or a hanging configuration yields exactly one configuration in one step.

Definition 4.1.4

For any Turing machine M, $\vdash_M^*$ is the reflexive, transitive closure of $\vdash_M$; we say that configuration C_1 **yields** configuration C_2 if $C_1 \vdash_M^* C_2$. A **computation** by M is a sequence of configurations $C_0, C_1, \ldots, C_n$ for some $n \geq 0$ such that

$$C_0 \vdash_M C_1 \vdash_M C_2 \vdash_M \cdots \vdash_M C_n.$$

We say that the computation is of **length** n or has n **steps**.

Example 4.1.4

Consider the Turing machine M described in Example 4.1.1. If M is started in configuration $(q_0, \underline{a}aaa)$, its computation would be represented formally as follows.

$$\begin{aligned}(q_0, \underline{a}aaa) &\vdash_M (q_1, \#\underline{a}aa)\\ &\vdash_M (q_0, \#\underline{a}aa)\\ &\vdash_M (q_1, \#\#aa)\\ &\vdash_M (q_0, \#\#\underline{a}a)\\ &\vdash_M (q_1, \#\#\#a)\\ &\vdash_M (q_0, \#\#\#\underline{a})\\ &\vdash_M (q_1, \#\#\#\#)\\ &\vdash_M (q_0, \#\#\#\#\#)\\ &\vdash_M (h, \#\#\#\#\#)\end{aligned}$$

This computation has nine steps.

Example 4.1.5

Consider again the Turing machine of Example 4.1.2. From the configuration $(q_0, aa\underline{a})$ we have

$$\begin{aligned}(q_0, aa\underline{a}) &\vdash_M (q_0, a\underline{a}a)\\ &\vdash_M (q_0, \underline{a}aa).\end{aligned}$$

This last configuration (which would be written in full as (q_0, e, a, aa)) is a hanging configuration.

4.2 COMPUTING WITH TURING MACHINES

Turing machines, as we have said, will be used not just as language recognizers but in a number of other ways as well. So far, however, we have merely presented the bare machines without any indication of how they are to be used systematically. It is as though a computer had been delivered with only a wiring diagram but with no advice about the usual procedures for getting information into and out of it. It is time, therefore, to fix some conventions for the use of Turing machines.

First, we adopt the following policy for presenting input to Turing machines: The input string is surrounded by a blank on each side and is written on the leftmost squares of the tape; the head is positioned at the tape square containing the blank that marks the right end of the input; and the machine starts operating in its initial state. Formally, if $M = (K, \Sigma, \delta, s)$ is a Turing machine and $w \in \Sigma^*$, then M is said to **halt on input** w if and only if $(s, \#w\#)$ yields some halted configuration. Similarly, M is said to **hang on input** w if $(s, \#w\#)$ yields some hanging configuration.

We start by viewing Turing machines as computers of functions from strings to strings.

Definition 4.2.1

Let Σ_0 and Σ_1 be alphabets *not* containing the blank symbol #. Let f be a function from Σ_0^* to Σ_1^*. A Turing machine $M = (K, \Sigma, \delta, s)$ is said to **compute** f if $\Sigma_0, \Sigma_1 \subseteq \Sigma$ and for any $w \in \Sigma_0^*$, if $f(w) = u$ then

$$(s, \#w\underline{\#}) \vdash_M^* (h, \#u\underline{\#}).$$

If some such Turing machine M exists, then f is said to be a **Turing-computable function.**

Thus when M is presented with input w in the way we have standardized above, M is required to halt eventually with $f(w)$ on the tape, preceded by a blank, and with the input head just to the right of the computed value. The rest of the tape must be blank. Since this is the same as the form in which we required the input to be presented, this convention will facilitate the composition of machines to form more powerful machines. Note, however, that no assurances are given about the behavior of M if it is started in any but the specified way, for example, if the input string contains symbols in $\Sigma - \Sigma_0$.

Note also that a Turing machine that computes a function may not hang on any input in the domain of that function. This is obvious, since a Turing machine that hangs on an input does not halt on that input. It is also critical, however, since it will enable us to assume that a Turing machine that computes a function never attempts to consult any tape square to the left of the blank square that marks the left end of that input.

Example 4.2.1

Let $\Sigma_0 = \Sigma_1 = \{a, b\}$, and let $f: \Sigma_0^* \to \Sigma_1^*$ be defined as follows: For any $w \in \Sigma_0^*$, $f(w) = \bar{w}$, where $\bar{w}$ is the result of replacing each occurrence of a in w by b, and vice versa. Then f is computed by a Turing machine M which scans backwards through its input, changing a's to b's and vice versa, until the blank marking the leftmost square is found. In order to leave the tape in the proper format, M must then return its head to the blank marking the right end of the input. Formally, $M = (K, \Sigma, \delta, s)$, where $K = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b, \#\}$, $s = q_0$, and δ is given by the following table.

q	σ	$\delta(q, \sigma)$
q_0	a	(q_1, L)
q_0	b	(q_1, L)
q_0	#	(q_1, L)
q_1	a	(q_0, b)
q_1	b	(q_0, a)
q_1	#	(q_2, R)
q_2	a	(q_2, R)
q_2	b	(q_2, R)
q_2	#	$(h, \#)$

For example, consider the following.

$$\begin{aligned}
(q_0, \#aab\underline{\#}) &\vdash_M (q_1, \#aa\underline{b}) \\
&\vdash_M (q_0, \#aa\underline{a}) \\
&\vdash_M (q_1, \#a\underline{a}a) \\
&\vdash_M (q_0, \#a\underline{b}ba) \\
&\vdash_M (q_1, \#\underline{a}aba) \\
&\vdash_M (q_0, \#\underline{b}bba) \\
&\vdash_M (q_1, \#\underline{\#}bba) \\
&\vdash_M (q_2, \#\underline{b}bba) \\
&\vdash_M (q_2, \#b\underline{b}ba) \\
&\vdash_M (q_2, \#bb\underline{a}) \\
&\vdash_M (q_2, \#bba\underline{\#}) \\
&\vdash_M (h, \#bba\underline{\#})
\end{aligned}$$

Note also that

$$\begin{aligned}
(q_0, \#\underline{\#}) &\vdash_M (q_1, \underline{\#}\#) \\
&\vdash_M (q_2, \#\underline{\#}) \\
&\vdash_M (h, \#\underline{\#})
\end{aligned}$$

so that M correctly computes the value $f(e) = e$. Since $(q_0, \#w\underline{\#}) \vdash_M^* (h, \#\bar{w}\underline{\#})$ for all $w \in \Sigma_0^*$, M computes f and f is Turing-computable.

Note also that the behavior of M, when started from a configuration such as $(q_0, \#\#ab\underline{\#}\#ab)$, is totally irrelevant to the question of whether M computes f.

The notion of a Turing-computable function from strings to strings can be extended in several ways. For one, we may consider functions of any number of arguments, including zero. Thus let f be a function from $(\Sigma_0^*)^k$ to Σ_1^*, where $k \geq 0$, and let M be a Turing machine (K, Σ, δ, s), where Σ_0, $\Sigma_1 \subseteq \Sigma$. Suppose that, for any $w_1, \ldots, w_k \in \Sigma_0^*$, if $f(w_1, \ldots, w_k) = u$, then $(s, \#w_1\#w_2\# \ldots \#w_k\underline{\#}) \vdash_M^* (h, \#u\underline{\#})$. Then we again say that M **computes** f and that f is **Turing-computable.**

We shall also wish to discuss the Turing-computable functions from natural numbers to natural numbers. Let I be some fixed symbol other than the blank symbol; then the natural number n may be represented (in **unary notation**, as we shall say) by the string I^n. (Thus zero is represented by the empty string.) Now a function $f: \mathbb{N} \to \mathbb{N}$ is said to be **computed** by a Turing machine M if M computes the function $f': \{I\}^* \to \{I\}^*$, where $f'(I^n) = I^{f(n)}$ for each $n \in \mathbb{N}$. That is, a Turing machine computes a function from numbers to numbers by computing the corresponding function from unary notations to unary notations. More generally, a Turing machine M computes a

function $f: \mathbf{N}^k \to \mathbf{N}$ if it computes the function $f': (\{I\}^*)^k \to \{I\}^*$, where $f'(I^{n_1}, \ldots, I^{n_k}) = I^{f(n_1,\ldots,n_k)}$ for any $n_1, \ldots, n_k \geq 0$. A function from numbers to numbers, or from k-tuples of numbers to numbers, is **Turing-computable** if there is a Turing machine that computes it.

Example 4.2.2

Let f be the successor function: $f(n) = n + 1$ for each $n \in \mathbf{N}$.

We design a Turing machine M which computes f by writing an I in the tape square in which its head is initially located, moving its head one square to the right, and halting. Formally, $M = (K, \Sigma, \delta, s)$, where $K = \{q_0\}$, $\Sigma = \{I, \#\}$, $s = q_0$, and δ is as follows.

q	σ	$\delta(q, \sigma)$
q_0	I	(h, R)
q_0	$\#$	(q_0, I)

For example,

$$(q_0, \#II\#) \vdash_M (q_0, \#III\underline{\#}) \vdash_M (h, \#III\#)$$

and in general, $(q_0, \#I^n\#) \vdash_M^* (h, \#I^{n+1}\#)$. (As in Example 4.2.1, M will behave differently if started in a configuration not of the form $(q_0, \#I^n\#)$ for some n, but this is of no concern.) Note in particular the case $n = 0$.

$$(q_0, \#\#) \vdash_M (q_0, \#\underline{I}) \vdash_M (h, \#I\#)$$

Thus M computes the function $f': \{I\}^* \to \{I\}^*$ such that $f'(I^n) = I^{n+1}$ for each n; and therefore M computes the successor function $f: \mathbf{N} \to \mathbf{N}$.

Another important derivative of the notion of a Turing-computable function is that of a Turing-decidable language. Let us again fix an alphabet Σ_0 not containing the blank symbol. Let Ⓨ and Ⓝ be two fixed symbols not in Σ_0. Then a language $L \subseteq \Sigma_0^*$ is **Turing-decidable** if and only if the function $\chi_L: \Sigma_0^* \to \{Ⓨ, Ⓝ\}$ is Turing-computable, where for each $w \in \Sigma_0^*$,

$$\chi_L(w) = \begin{cases} Ⓨ & \text{if } w \in L \\ Ⓝ & \text{if } w \notin L. \end{cases}$$

If χ_L is computed by a Turing machine M, then M is said to **decide** L, or to be a **decision procedure** for L.

Example 4.2.3

Let $\Sigma_0 = \{a\}$, and let $L = \{w \in \Sigma_0^* : |w| \text{ is even}\}$. Then the following Turing machine $M = (K, \Sigma, \delta, s)$ is a decision procedure for L: $K = \{q_0, \ldots, q_6\}$, $\Sigma = \{a, Ⓨ, Ⓝ, \#\}$, $s = q_0$, and δ is given by the following table.

q	σ	$\delta(q, \sigma)$
q_0	$\#$	(q_1, L)
q_1	a	$(q_2, \#)$
q_1	$\#$	(q_4, R)
q_2	$\#$	(q_3, L)
q_3	a	$(q_0, \#)$
q_3	$\#$	(q_6, R)
q_4	$\#$	$(q_5, Ⓨ)$
q_5	Ⓨ	(h, R)
q_5	Ⓝ	(h, R)
q_6	$\#$	$(q_5, Ⓝ)$

(Omitted entries are irrelevant and may be defined arbitrarily.) Here M erases its input string from right to left, keeping track in its finite control of the parity (odd or even) of the number of symbols it has erased. States q_0, q_1, q_2, and q_3 are used for this purpose. On discovering the blank that marked the left end of the input, M passes from state q_1 to q_4 or from q_3 to q_6, writes Ⓨ or Ⓝ accordingly, moves its head one square to the right, and halts. Thus

$$(q_0, \#a^n\#) \vdash_M^* (h, \#Ⓨ\#) \qquad \text{if } n \text{ is even;}$$
$$(q_0, \#a^n\#) \vdash_M^* (h, \#Ⓝ\#) \qquad \text{if } n \text{ is odd;}$$

and so M computes χ_L. Therefore L is Turing-decidable.

Yet another way Turing machines can be used is as language *acceptors*. Again, fix some alphabet Σ_0 not containing $\#$. We say that a Turing machine M **accepts** a string $w \in \Sigma_0^*$ if M halts on input w. Thus M accepts a language $L \subseteq \Sigma_0^*$ if and only if $L = \{w \in \Sigma_0^* : M \text{ accepts } w\}$, and a language is said to be **Turing-acceptable** if there is some Turing machine that accepts it. We shall see that every Turing-decidable language is Turing-acceptable, but not vice versa; this distinction will be a major concern of Chapter 6.

Example 4.2.4

Let $\Sigma_0 = \{a, b\}$ and let $L = \{w \in \Sigma_0^* : w \text{ contains at least one } a\}$. Then L is accepted by the Turing machine $M = (K, \Sigma, \delta, s)$, where $K = \{q_0\}$, $\Sigma = \{a, b, \#\}$, $s = q_0$, and δ is given by the following table.

q	σ	$\delta(q, \sigma)$
q_0	a	(h, a)
q_0	b	(q_0, L)
q_0	$\#$	(q_0, L)

Comment about the Tiling problem.

" For this problem the T. M. considered are said to halt if they ever go to the left of the square they start on, as well as if they really halt"

(ie: we think of the tape to be one-way infinite and we halt if we "fall" from the left).

there can be none for determining whether a context-free grammar is ambiguous.

If P has a match $u_{i_1} \dots u_{i_n} = v_{i_1} \dots v_{i_n}$, then clearly the string $a_{i_n} \dots a_{i_1} u_{i_1} \dots u_{i_n} = a_{i_n} \dots a_{i_1} v_{i_1} \dots v_{i_n}$ has two different parse trees, one arising from G_1 and one from G_2. Conversely, if w is a string in $L(G)$ with two different parse trees, then $w \in L(G_1)$ and $w \in L(G_2)$, since G_1 and G_2 are both unambiguous separately. But then $w = a_{i_n} \dots a_{i_1} u_{i_1} \dots u_{i_n} = a_{i_n} \dots a_{i_1} v_{i_1} \dots v_{i_n}$ for some $i_1, \dots, i_n$, $n \geq 1$, and so w is a match. ■

6.5 AN UNSOLVABLE TILING PROBLEM

We are given a finite set of tiles, each one unit square. We are asked to tile the first quadrant of the plane with copies of these tiles, placing one tile in each square region formed when lines are drawn one unit apart parallel to the horizontal and vertical axes, as shown in Figure 6–4. We have an infinite supply of copies of each tile.

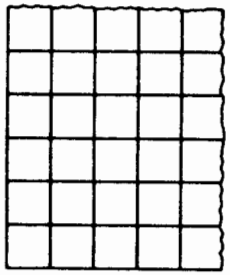


Figure 6–4

The only restrictions are that a special "origin" tile must be placed in the lower left-hand corner; that only certain pairs of tiles may abut each other horizontally; and that only certain pairs of tiles may abut each other vertically. (Tiles may not be rotated or turned over.) Is there an algorithm for determining whether the first quadrant can be tiled, given a finite set of tiles, the origin tile, and the adjacency rules?

This problem can be formalized as follows. A **tiling system** is a quadruple $\mathfrak{D} = (D, d_0, H, V)$, where D is a finite set, $d_0 \in D$, and $H, V \subseteq D \times D$. A **tiling** by $\mathfrak{D}$ is a function $f: \mathbf{N} \times \mathbf{N} \to D$ such that the following hold.

$$f(0, 0) = d_0$$
$$(f(n, m), f(n + 1, m)) \in H \qquad \text{for all } n, m \in \mathbf{N}$$
$$(f(n, m), f(n, m + 1)) \in V \qquad \text{for all } n, m \in \mathbf{N}$$

Theorem 6.5.1. *The problem of determining, given a tiling system, whether there is a tiling by that system is unsolvable.*

Proof. We reduce to this tiling problem the problem of determining, given a Turing machine M, whether M eventually halts when started on the leftmost square of a blank tape. This is a trivial variation on the problem of Theorem 6.3.1(c); there M starts on the *second* square of a blank tape. Clearly this slightly different problem is unsolvable as well. Thus when we have shown that this version of the halting problem is reducible to the tiling problem, we shall have shown that the tiling problem is unsolvable.

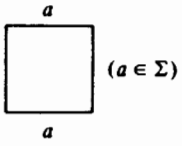
The basic idea is to construct from any Turing machine M a tiling system $\mathfrak{D}$ such that a tiling by $\mathfrak{D}$, if one exists, represents an infinite computation by M starting from the blank tape. Configurations of M are represented horizontally in a tiling; successive configurations appear one above the next. If M operates forever, successive rows can be tiled ad infinitum; but if M halts after k steps, it is possible to tile only k rows.

We can think of the edges of the tiles as being marked with certain information; we allow tiles to abut each other horizontally or vertically only if the markings on the adjacent edges are identical. On the horizontal edges, these markings are either a symbol from the alphabet of M or a state-symbol combination. The tiling system is arranged so that if a tiling is possible, then by looking at the markings on the horizontal edges between the kth and $(k + 1)$st rows of tiles, we can read off the configuration of M after $k - 1$ steps of its computation. Thus only one edge along such a border is marked with a state-symbol pair; the other edges are marked with single symbols.

The marking on a vertical edge of a tile is either absent or consists of a state of M, together with a "directional" indicator, which we indicate by an arrowhead. (Two exceptions are given under (e) below.) These markings on the vertical edges are used to communicate a left- or right-hand movement of the head from one tile to the next.

To be specific, let $M = (K, \Sigma, \delta, s)$. Then $\mathfrak{D} = (D, d_0, H, V)$, where D contains the following tiles.

(a) For each $a \in \Sigma$, the tiles illustrated in Figure 6–5, which simply communicate any unchanged symbols upwards from configuration to configuration.



$(a \in \Sigma)$

Figure 6–5

(b) For each $a \in \Sigma$ and $q \in K$ such that $\delta(q, a) = (p, b)$, where $p \in K$ and $b \in \Sigma$, the tile shown in Figure 6–6. This tile communicates the head position upwards and also changes the state and scanned symbol appropriately.

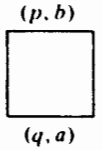


Figure 6–6

(c) For each $q \in K$ and $a \in \Sigma$ such that $\delta(q, a) = (p, R)$ for some $p \in K$, the tile shown in Figure 6–7(a) and also, for each $b \in \Sigma$, the tile shown in Figure 6–7(b). These tiles communicate head movement one square from left to right, while changing the state appropriately.

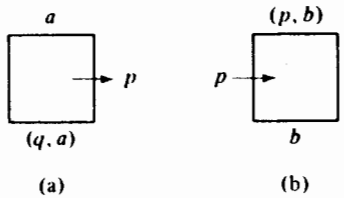


Figure 6–7

(d) Tiles similar to those of (c) for the case in which $\delta(q, a) = (p, L)$ are illustrated in Figure 6–8.

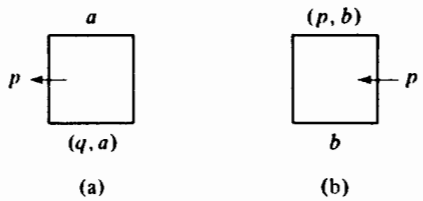


Figure 6–8

These tiles do the bulk of the simulation of M by $\mathfrak{D}$. It remains only to specify some tiles to initiate the computation and ensure that the bottom row is tiled correctly.

(e) The origin tile d_0 is illustrated in Figure 6–9(a). It specifies on its vertical edge the initial state of M and the blank symbol. Its right edge is

marked with the blank symbol; this edge can be matched only by the left edge of the final tile, that shown in Figure 6–9(b), which in turn propagates to the right the information that the top edge of every tile in the bottom row is marked with the blank.

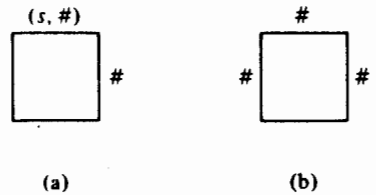


Figure 6–9

This completes the construction of $\mathfrak{D}$. The last set of tiles ensures that the border between the first two rows is marked $(s, \#)\#\#\#\ldots$; the other tiles force each subsequent border to be marked correctly. Note that no tile mentions the halt state, so that if M halts after k steps only k rows can be tiled. ■

Example 6.5.1

Consider the Turing machine (K, Σ, δ, s), where $\Sigma = \{\#\}$, $K = \{s, q\}$, and δ is given by

$$\delta(s, \#) = (q, R)$$
$$\delta(q, \#) = (s, L).$$

This machine simply oscillates its head from left to right and back again, never moving beyond the first tape square. The tiling of the plane associated with the infinite computation of M is shown in Figure 6–10.

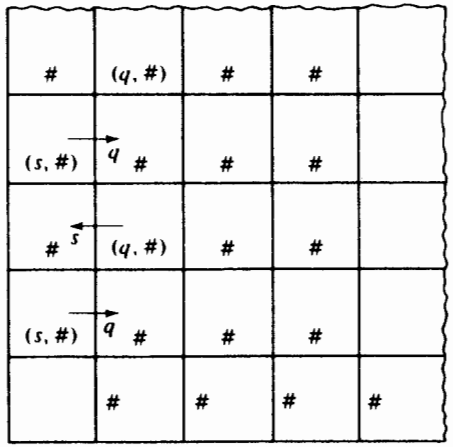


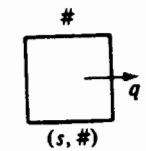
Figure 6–10

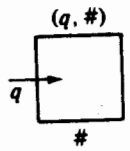
The complete set of tiles associated with M is shown in Figure 6–11.

None

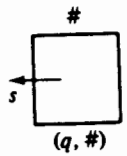
(a)

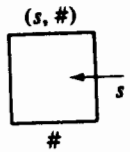
(b)



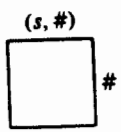


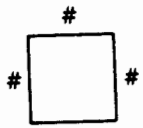
(c)





(d)





(e)

Figure 6–11

PROBLEMS

6.1.1. What is R_u, if u is not the encoding of any Turing machine?

6.1.2. In this problem you will find a particular example of an uncomputable function without using a diagonal argument.

(a) Show that any Turing-computable function from numbers to numbers is computed by a Turing machine with alphabet $\{I, \#\}$.

The **busy-beaver function** $\beta: \mathbf{N} \to \mathbf{N}$ is defined as follows: for each $n \geq 0$, $\beta(n)$ is the largest number m such that, for some Turing machine M with alphabet $\{I, \#\}$ and with exactly n states, $(s, \# \sharp) \vdash_M^* (h, \#I^m \sharp)$, where s is the initial state of M.

(b) Show that for any n and m, $\beta(n) \geq \beta(m)$ if and only if $n \geq m$. (That is, if more states are available a larger number of I's can be written, starting from the empty tape.)

(c) Show that if $f: \mathbf{N} \to \mathbf{N}$ is computed by a Turing machine with alphabet $\{I, \#\}$ and k states, then $\beta(n + k) \geq f(n)$ for all n. In other words, there is a Turing machine M_n with $n + k$ states which writes at least $f(n)$ I's on a blank tape before halting in the conventional way. (*Hint:* It is easy enough to find machines N_n with n states that write n I's. Combine these machines with the fixed machine F which computes f; that is, consider the machines $N_n F$.)

(d) Show that β is not Turing-computable. (Suppose it were. Then the function γ such that $\gamma(n) = \beta(2n)$ would also be Turing-computable. Using (a), let f in (c) be γ and apply (b) to the resulting inequality.)

(e) Why do we insist throughout this problem that only Turing machines with alphabet $\{I, \#\}$ be considered? (Any other fixed alphabet containing I and # would do as well.)

6.1.3. According to Theorem 6.1.4, the class of Turing-acceptable languages is not closed under complementation. But show that it is closed under union and intersection.

6.1.4. Show that the class of Turing-decidable languages is closed under union, complementation, intersection, concatenation, and Kleene star.

6.1.5. The universal Turing machine described in Section 5.7 might not accept K_0 because it might accept some strings not of the form $\rho(M)\rho(w)$ for any Turing machine M and string w. Describe carefully the changes that have to be made to the machine of Section 5.7 to obtain one that accepts K_0.

6.1.6. Show that any finite set is Turing-decidable.

6.2.1. Prove that a language is Turing-decidable if and only if it is enumerated *in lexicographic order* by some Turing machine.

6.2.2. Suppose we modify Definition 6.2.2 as follows: M enumerates L if and only if, for some fixed state q of M,

$$L = \{w: (s, \sharp) \vdash_M^* (q, \#w\sharp)\},$$

6.4.10. Suppose that in Problem 6.4.9 the definition of "weak match" is strengthened so that $j_1, \ldots, j_n$ is required to be a *permutation* of $i_1, \ldots, i_n$. Is it still solvable whether a correspondence system has a weak match?

6.4.11. Prove the second sentence in the statement of Theorem 6.4.2.

6.5.1. Let $M = (\{s\}, \{a, \#\}, \delta, s)$, where δ is as follows.

$$\delta(s, \#) = (s, a)$$
$$\delta(s, a) = (s, R)$$

Find the set of tiles associated with M via the construction in Section 6.5, and illustrate the first four rows of a tiling of the plane by means of these tiles.

6.5.2. What will be the appearance of a tiling by $\mathfrak{D}$ if M hangs?

6.5.3. Show that there is some fixed set of tiles D and adjacency rules H and V such that the following problem is unsolvable: Given a **partial tiling**, that is, a mapping $f: S \to D$ for some finite subset $S \subseteq \mathbf{N} \times \mathbf{N}$ such that f obeys the adjacency rules, can f be extended to a tiling of the whole plane?

6.5.4. Suppose the rules of the tiling game are changed so that instead of fixing a particular tile to be placed at the origin, we fix instead a particular set of tiles and stipulate that only these tiles may be used in tiling the first row. Show that the tiling problem remains unsolvable.

6.5.5. Prove formally, by induction, the correctness of the construction given in Section 6.5.

6.5.6. Suppose the rules of the tiling game are changed as follows: The tiles are not perfectly square, but may have various bumps and notches along their edges. Two tiles may be laid down next to each other only if their edges fit together perfectly, like pieces of a jigsaw puzzle. Show that the tiling problem remains unsolvable, even if we are now allowed to rotate tiles or turn them over. (There is still a specified "origin tile" which must be put down at position (0, 0) in a fixed orientation.)

6.5.7. Suppose that we think of square tiles as being determined by the *colors* of their four edges, and that two edges may abut provided that they are similarly colored. Show that if we are allowed to rotate tiles and turn them over, then *any* nonempty set of tiles can be used to tile the entire first quadrant (even when we continue to require that one special tile be placed at the origin).

REFERENCES

The unsolvability of the halting problem for Turing machines is shown in Turing's 1936 paper cited at the end of Chapter 4. Church's paper cited at the end of Chapter 5 establishes an unsolvability result for his formal computational system.

Thue systems were introduced in

A. THUE, "Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln," *Skrifter utgit av Videnskapsselskapet i Kristiana*, I. Matematisk-naturvidenskabelig klasse 1914, 10 (1914), 34 pp.

Their unsolvability was first demonstrated in

E. L. POST, "Recursive Unsolvability of a Problem of Thue," *Journal of Symbolic Logic*, 12 (1947), 1–11.

Correspondence systems and the original proof of their unsolvability can be found in

E. L. POST, "A Variant of a Recursively Unsolvable Problem," *Bulletin of the American Mathematical Society*, 52 (1946), 264–268.

Theorem 6.4.5 and Problem 6.4.4 (unsolvable problems for context-free languages) are from the paper by Bar-Hillel, Perles, and Shamir cited at the end of Chapter 3.

Wang's domino problem is from

H. WANG, "Proving Theorems by Pattern Recognition II," *Bell System Technical Journal*, 40 (1961), 1–141.

Closely related is

J. R. BÜCHI, "Turing-machines and the Entscheidungsproblem," *Mathematische Annalen*, 148 (1962), 201–213.

It turns out that the domino problem remains unsolvable even when the origin constraint is omitted and only the adjacency constraints are imposed. This was first proved by

R. BERGER, "The Undecidability of the Domino Problem," *Memoirs of the American Mathematical Society*, 66 (1966).

A simpler and more readable version may be found in

R. M. ROBINSON, "Undecidability and Nonperiodicity for Tilings of the Plane," *Inventiones Mathematicae*, 12 (1971), 177–209.

Problem 6.4.5 on two-headed finite automata is from the paper by Rabin and Scott cited at the end of Chapter 2.

A celebrated problem, known as "Hilbert's Tenth Problem," asks whether there is a method for finding integer roots of arbitrary multivariate polynomials. The problem was proposed in

Quiz 1 Solutions

We will use the following fact and notation: the set of strings in $\{\mathtt{a},\mathtt{b}\}^*$ can be ordered in a sequence $(x_i)_{i\in\mathbf{N}}$.

Problem 1 [25 points]. For each of the sets below, indicate with a single capital letter whether it is (D) decidable, (R) r.e. but not co-r.e., (C) co-r.e. but not r.e., or (N) neither. No explanation is required and there is no penalty for guessing.

In the solutions of all of the five questions of this problem we will denote by A the set to be studied.

(a) $A = \{(M_1, M_2) \mid \text{domain}(M_1) \cap \text{domain}(M_2) = \emptyset\}$.Answer:(C).

Then $\overline{A} = \{(M_1, M_2) \mid \text{domain}(M_1) \cap \text{domain}(M_2) \neq \emptyset\}$. Consider the machine N that given input (M_1, M_2) acts as follows: "Run M_1 and M_2 in parallel on all inputs, and halt as soon as it is checked that M_1 and M_2 have stopped on a same input." This procedure is total and computable, and $\overline{A}$ =domain(N); hence $\overline{A}$ is r.e. Denote $B = \{M_1 \mid \text{domain}(M_1) = \emptyset\}$.

Claim: $B \leq_{\mathrm{m}} A$

Proof Consider some given machine M_2 for which domain(M_2)= Σ^*. Define the function $f : M_1 \longmapsto (M_1, M_2)$. f is total and clearly computable (*i.e.*programmable) and $M_1 \in B$ iff $f(M_1) \in A$. QED-claim.

But $B = K_P$, where a set S has property P iff $S = \emptyset$. P being not trivial, Rice's theorem implies that B is not decidable. Hence A is not decidable, and being co-r.e. is not r.e.

(b) $A = K_1 \times K_0$.Answer:(R).

Claim: The product $A \times B$ of two r.e. sets is r.e.

Proof: Let $M_A \times M_B$ be the machine working as follows on any input $(x, y) \in {\Sigma_A}^* \times {\Sigma_B}^*$: "Have M_A working on input x and M_B on input y. Halt as soon as(and only if) M_A and M_B have stopped on their respective input."

We see that Domain($M_A \times M_B$)= Domain(M_A)$\times$Domain(M_B)= $A \times B$. QED-claim.

As an application we find that $K_1 \times K_0$ is r.e.

Claim: $K_1 \leq_{\mathrm{m}} K_1 \times K_0$.

Proof: Let $y \in K_0$. Consider $f : x \longmapsto (x, y)$. Then $x \in K_1$ iff $(x, y) \in K_1 \times K_0$.$f$

being total and computable, the result follows.

QED-claim.

We know that K_1 is not co-r.e. By inheritance property, we deduce that $K_1 \times K_0$ is not co-r.e. either.

(c) $A = \{M \mid M$ is a Turing machine and $\exists x \in \{\mathtt{a}, \mathtt{b}\}^*$ such that M accepts x in at most $|x|^2$ steps$\}$.Answer:(R).

Construct the following Turing machine N that have Turing machines as inputs and operates in the following way: "On input M, have M work exhaustively on all possible inputs x for $|x|^2$ steps. Halt as soon as and only if, in this process, M halts at some point". We see that $M \in A$ iff N halts on input M; this means that A =domain(N) so that A is an r.e. set.

Claim: $K_2 \leq_m A$.

Consequence: By up-inheritance of non r.e.-ness, A is not co-r.e.

Proof of the claim: For any Turing machine M we can define a new Turing machine N that operates as follows: "On every input x, act like M on input Λ ". By definition the behavior of M does not depend on its input: it always halt or it never halts.Then,

$M \in K_2$ iff $\exists n \in \mathbf{N}$ for which N halts on Λ in n steps,

iff $\exists n \in \mathbf{N}$ for which N halts on "$\underbrace{a, \ldots, a}_{n \text{ of them}}$" in n steps,

iff $N \in A$.

The function f defined by $N = f(M)$ is total and recursive. Hence $K_2 \leq_m A$.

QED-claim.

(d) $A = \{x \in \{\mathtt{a}, \mathtt{b}\}^* \mid \exists$ Turing machine M accepting x in at most $|x|^2$ steps$\}$.Answer:(D).

Consider the Turing machine that accepts immediatly any input *i.e.*$\forall x \in \{\mathtt{a}, \mathtt{b}\}^*$, M accepts x in 0 steps. The existence of this machine shows that $A = \{\mathtt{a}, \mathtt{b}\}^*$ which is a decidable set.

(e) A=$\{p \mid p$ is a polynomial in n variables with integer coefficients, $n > 0$, and for all integers $x_1, \ldots, x_n$, it is the case that $p(x_1, \ldots, x_n) \neq 9\}$.Answer:(C).

$\overline{A} = \{p \mid p$ is a polynomial in n variables with integer coefficients, $n > 0$, and there exists integers $x_1, \ldots, x_n$, such that $p(x_1, \ldots, x_n) - 9 = 0\}$. Recall the instance of Hilbert's 10th problem: n being any integer bigger than 2, denote "$B_n = \{q \mid q$ is a polynomial in n variables with integer coefficients, and there exists integers $x_1, \ldots, x_n$ such that $q(x_1, \ldots, x_n) = 0\}$". We saw in class that $\forall n \geq 2$, B_n is r.e. but not co-r.e. For the sake of this problem,consider also $B = \{q \mid q$ is a polynomial in n variables with integer coefficients for *some* $n > 0$, and there exists integers $x_1, \ldots, x_n$ such that $q(x_1, \ldots, x_n) = 0\}$. Now it is clear that for any polynomial q with integer coefficients and of given degree n, $q \in B_n$ iff $q \in B$, so that $B_n \leq_m B$. Using the up-inheritance of non co-r.e.-ness we deduce that B is not co-r.e. To establish that B is r.e. consider the

machine M that operates as follows on any input q, where q is an integer polynomial: "compute the degree n of q. Then go in some predetermined way through all the n-uples of integers $(x_1, \cdots, x_n)$, and for each of them compute $q(x_1, \cdots, x_n)$. Halt as soon as(and only if) one of these $q(x_1, \cdots, x_n)$ is found to be zero". It is clear that B =domain(M).
Claim: $\overline{A} \equiv_m B$
Proof: Define $f(p) = p - 9$, which is obviously total and recursive. Then $p \in \overline{A}$ iff $f(p) \in B$ *i.e.* $\overline{A} \leq_m B$. Define $g(p) = q + 9$, which is obviously total and recursive. Then $q \in B$ iff $g(q) \in \overline{A}$ *i.e.*$B \leq_m \overline{A}$.

QED-claim.

Hence $\overline{A}$ is r.e. and not co-r.e., so that A is co-r.e. and not r.e.

Problem 2 [25 points]. Show that à set A is r.e. iff A is finite or is the range of a total one-to-one recursive function. (*Hint* : You may find it helpful to use the fact stated in Handout 2, Theorem 7, page 5 that every nonempty r.e. set is the range of a total recursive function.)

There are two directions to establish. Let's begin with the easy one.
$\Longleftarrow$:) If A is finite, it is actually decidable and hence r.e. If A is the range of a total one-to-one recursive function, then from theorem 7, page 5 of handout 2, A is r.e.

The main difficulty is to establish the converse,*i.e.*to refine the above quoted theorem to get a *one-to-one* function.
$\Longrightarrow$:) Assume A is r.e. If A is finite the implication is trivial. Hence we can restrict ourselves to the case where A is r.e. and infinite. Consider then a Turing machine (whose existence is asserted by theorem 7) computing a function f whose range is A. Define a new Turing machine N that operates as follows:
"On input x_i (see the note at the head of these solutions), have "M operate in parallel" on all inputs (according to some predetermined procedure as the one seen in class). Halt as soon as "M operating in parallel" has halted with its ith <u>new</u> output y_i, and display y_i".
Note:

- By definition of M, y_i being an output of M, we have that $y_i \in A$.
- We cannot ran short of y_i's: as A is infinite, we have an infinite supply of them.
- The range of N is the same as the range of M, *i.e.* the whole set A.
- By construction, the function that N computes is *one-to-one*:
  Consider two different inputs x and x'. Then $x = x_i$ and $x' = x_j$ for two different indices i and j. Assume for instance that $j > i$. Then on input x_j, N will halt when "M on parallel" will have produced $j - i$ *new* outputs after the one corresponding to x_i. In particular these outputs are different.

- Call g the function computed by N. g is clealy total (for every input x_i, N produces some output). On the other hand, the correspondance $M \longrightarrow N$ is clearly programmable so that the function g is recursive.

To summarize: If A is an infinite r.e. set, A is the range of g, where g is some total recursive function.

Problem 3 [25 points]. Prove the statement of Handout 2, page 7: If A is r.e. but not recursive, then A and $\overline{A}$ are $\leq_m$-incomparable.
(*Hint*: Assume to the contrary that, *e.g.*, $\overline{A} \leq_m A$.)

We will work by contradiction. Assume first to the contrary that $\overline{A} \leq_m A$. Then $\overline{A}$ is r.e. since r.e. inherits down. But then A is decidable since r.e. and co-r.e. implies decidable. This contradicts the hypothesis. Alternatively, suppose $A \leq_m \overline{A}$. Then $\overline{A} \leq_m \overline{\overline{A}} = A$, by complement property of $\leq_m$, which leads to the above contradictions. Hence neither $A \leq_m \overline{A}$ or $\overline{A} \leq_m A$ holds.

Problem 4 [25 points]. A Turing machine is said to be *n-bounded* iff it stops in at most n steps on every input. Let

$$A_n = \{M \mid M \text{ is } n\text{-bounded}\} \text{ and } A = \bigcup_n A_n.$$

(a) Explain why A_{29} is decidable.(*Hint*: Use the fact that it takes one step to read an input).

$M \in A_{29}$ iff on *every* input x, M halts in at most 29 steps. Call $\Pi_{29}(x)$ the input obtained by keeping only the first 29 left characters of x (and padding with blanks on the left if $|x| < 29$). As, by hypothesis, it takes one step to read an input character, we see that,

$\forall x \in \Sigma^*$ M halts on input x in at most 29 steps
iff M halts on input $\Pi_{29}(x)$ in at most 29 steps .

Hence the test for M involves only inputs of a *given* length: $M \in A_{29}$ iff $\forall x \in \Sigma^*$ with $|x| \leq 29$, M halts in at most 29 steps. This being understood, we construct the machine N that, on input a machine M, acts as follows:
"Have M work successively on all inputs x of length less then 29 for 29 steps. If in this process M halts, then halt immediatly everything and display a. If not, then display b when this (finite) process is over". Then it is clear that :
$M \in A_{29}$ iff N outputs a on input M,
$M \notin A_{29}$ iff N outputs b on input M.
Hence A_{29} is decidable.

(b) Explain why A is r.e.

Attention: It is not true that an *infinite* union of r.e.(decidable) sets is still r.e.(decidable): see homeworkset 2 for more details.
It should be clear that in the previous question we could have used any given number in place of 29. Call N_{29} the machine checking for membership in A_{29} that we constructed in (a). In the same way, for any k, call N_k the machine checking for membership in A_k.
Construct the following machine T:
"On input a machine M, apply N_1 to M, then N_2 to M, then N_3 to M,... and so on. Halt as soon as and only if some N_k halts on M". Then,

$M \in A$ iff $M \in \bigcup_n A_n$,
iff $\exists n$ s.t. $M \notin A_1 \dots M \notin A_{n-1}, M \in A_n$,
iff $\exists n$ s.t. T halts on M in its n-th subroutine,
iff T halts on M.

Hence A =domain(T) and A is r.e.

(c) Explain why Rice's theorem does not apply to A. More precisely show that $A \neq K_P$ for any property P of r.e. sets.

Rice's theorem asserts that:

"[There is a non trivial property P of r.e. sets such that $A = K_P$]

implies

[A is not decidable]".

Rice's theorem will not apply to A
iff we cannot find a non trivial property P of r.e. sets such that $A = K_P$,
iff we cannot find a non trivial property P of r.e. sets such that $M \in A \iff P[\text{domain}(M)]$,
iff we can find two machines M and M' such that domain(M)=domain(M') but $M \in A$ and $M' \notin A$.

For this purpose consider the two following machines M and M':

- "On every input x, M halts immediatly without doing anything",
- "On every input x, M' reads x and then halts".

It is clear that,

- domain(M)=domain(M')=Σ^*,
- $M \in A_1 \subseteq A$,

Claim: $M' \notin A$.
Proof: $\forall n$ consider some input x of length $n+1$. Then, on this input, M' works during $n+1$ steps before halting. Hence $M' \notin A_n$ for any arbitrary n *i.e.*$M' \notin A$.
QED-claim.

(d) Prove that A is not co-r.e.

Claim: $K_2 \leq_m A$
Proof: Let M be a Turing machine. Consider the usual Turing machine N derived from M that operates as follows: "On every input x, act like M on input Λ".
Then $M \in K_2$ iff N halts on Λ, say in n steps,
iff $N \in A_n$ for some n,
iff $N \in A$
QED-claim.

Corollary: A is not co-r.e.
Proof: K_2 is not co-r.e., and non-co-r.e. inherits up.

Glossary and notation

$d(M) \in \{\mathtt{a}, \mathtt{b}\}^*$ is the code of the Turing machine M.

$K_P = \{M \mid P(\text{domain}(M))\}$ where P is a property of r.e. sets.

$K_0 = \{(M, x) \mid M$ is a Turing machine that halts on input $x\}$.

$K_1 = \{M \mid M$ is a Turing machine that halts on input $d(M)\}$.

Notes on Predicate Calculus

1 Syntax

The syntax of the predicate calculus is constructed out of two kinds of symbols: the *logical* symbols and the *nonlogical* symbols. The logical symbols are

- Delimiters: ")", "(" (parentheses), "," (comma).
- Truth Constants: `true`, `false`.
- Propositional Connectives: $\neg$ (not), $\supset$ (implies), $\wedge$ (and), $\vee$ (or).
- The equality operator: $=$.
- Quantifiers: $\forall$ (universal quantifier), $\exists$ (existential quantifier).
- Variables: for $i, n \geq 0$,
  - (1) symbols F_i^n called *n-ary function variables*; the zero-ary function variables F_i^0 are also called *individual variables* and are also written as "x_i".
  - (2) symbols P_i^n called *n-ary predicate variables*; the zeroary predicate variables P_i^0 are also called *propositional variables.*

The nonlogical symbols are

- symbols f_i^n called *n-ary function constants*; the zero-ary function constants f_i^0 are also called *individual constants* and are also written "c_i".
- symbols p_i^n called *n-ary predicate constants*; the p_i^0 are also called *propositional constants.*

Definition 1. A *signature* is a set of nonlogical symbols.

Terms and well-formed formulas (wff's) over a given signature, $\mathcal{S}$, are defined by induction.

Definition 2. A *term* over $\mathcal{S}$ is defined as follows:

(1) Each individual variable x_i is a term over $\mathcal{S}$.

(2) Each individual constant $c_i \in \mathcal{S}$ is a term over $\mathcal{S}$.

(3) If $t_1, \ldots, t_n$ are terms over $\mathcal{S}$ and f_i^n is a function variable then $f_i^n(t_1, \ldots, t_n)$ is a term over $\mathcal{S}$.

(4) If $t_1, \ldots, t_n$ are terms over $\mathcal{S}$ and $F_i^n \in \mathcal{S}$ is a function constant then $F_i^n(t_1, \ldots, t_n)$ is a term over $\mathcal{S}$.

Note that the parentheses and commas used in the constuction of terms are the logical delimiter symbols.

Definition 3. *Atomic formulas* over $\mathcal{S}$ are defined as follows:

(1) Each propositional variable P_i^0 is an atomic formula over $\mathcal{S}$.

(2) Each propositional constant $p_i^0 \in \mathcal{S}$ is an atomic formula over $\mathcal{S}$.

(3) If t_1 and t_2 are terms over $\mathcal{S}$ then $(t_1 = t_2)$ is an atomic formula over $\mathcal{S}$.

(4) If $t_1, \ldots t_n$ are terms over $\mathcal{S}$ and P_i^n is a predicate variable then $P_i^n(t_1, \ldots, t_n)$ is an atomic formula over $\mathcal{S}$.

(5) If $t_1, \ldots t_n$ are terms over $\mathcal{S}$ and $p_i^n \in \mathcal{S}$ is a predicate constant then $p_i^n(t_1, \ldots, t_n)$ is an atomic formula over $\mathcal{S}$.

Definition 4. *Well-formed formulas* (wff's) over $\mathcal{S}$ are defined as follows:

(1) An atomic formula over $\mathcal{S}$ is a wff over $\mathcal{S}$.

(2) If A and B are wff's over $\mathcal{S}$ then so are $(\neg A)$, $(A \supset B)$, $(A \wedge B)$, and $(A \vee B)$.

(3) If x_i is an individual variable and A is a wff over $\mathcal{S}$ then $(\forall x_i A)$ and $(\exists x_i A)$ are wff's over $\mathcal{S}$.

We emphasize that terms and wff's over $\mathcal{S}$ are *syntactic* entities constructed out of the symbols in $\mathcal{S}$ and the logical symbols.

Parentheses will be dropped in writing wff's where there is no ambiguity. We will sometimes use symbols different from the above, such as x, y, z for variables and f, g for function constants, following Manna's notational conventions.

We omit here the important technical definitions of *free and bound variables* in a wff. See Manna for this. We can now define closed terms and wff's:

Definition 5. A *closed* term (wff) is a term (wff) with no free variables.

We write $[A]^{t_1,\ldots,t_n}_{x_1,\ldots,x_n}$ for the wff obtained by replacing each free occurence of x_i with t_i in A. Bound variables in A are renamed if necessary so that the quantifiers of A do not capture the free variables in the t_i. We omit the details of renaming.

It is a small technical convenience to distinguish constants from variables. Clearly, they behave syntactically much the same–the only difference is that variables can be bound by quantifiers, while constants cannot.

2 Semantics

Let $\mathcal{S}$ be a signature. We proceed to define the meaning of terms and wff's over $\mathcal{S}$.

Definition 6. A *model* of $\mathcal{S}$ is a pair $\mathcal{M} = (D, I_c)$ where D is a non-empty set, called the *domain* of $\mathcal{M}$, and I_c is a function with domain $\mathcal{S}$ which assigns values to the elements of $\mathcal{S}$ as follows:

(1) $I_c(c_i)$ is an element of D for all $c_i \in \mathcal{S}$.

(2) $I_c(p_i^0) \in \{\texttt{true}, \texttt{false}\}$ for all $p_i^0 \in \mathcal{S}$.

(3) $I_c(f_i^n) : D^n \to D$ is a *total* n-ary function over D for all $n \geq 1$ and $f_i^n \in \mathcal{S}$.

(4) $I_c(p_i^n) : D^n \to \{\texttt{true}, \texttt{false}\}$ is an n-ary predicate over D for all $n \geq 1$ and $p_i^n \in \mathcal{S}$.

Definition 7. A *valuation* for a model $\mathcal{M} = (D, I_c)$ of $\mathcal{S}$ is a function I_v which assigns values to the variables as follows:

(1) $I_v(x_i)$ is an element of D for all individual variables x_i.

(2) $I_v(P_i^0) \in \{\texttt{true}, \texttt{false}\}$ for all P_i^0.

(3) $I_v(F_i^n) : D^n \to D$ is a n-ary function over D for all $n \geq 1$ and all function variables F_i^n.

(4) $I_v(P_i^n) : D^n \to \{\texttt{true}, \texttt{false}\}$ is a n-ary predicate over D for all $n \geq 1$ and all predicate variables P_i^n.

Definition 8. An *interpretation* of $\mathcal{S}$ is pair $\mathcal{I} = (\mathcal{M}, I_v)$ where $\mathcal{M}$ is a model for $\mathcal{S}$ and I_v is a valuation for $\mathcal{M}$.

We define the meaning of a term by induction over the definition of a term.

Definition 9. The *meaning* of a term t over $\mathcal{S}$ in an interpretation $\mathcal{I} = ((D, I_c), I_v)$ of $\mathcal{S}$ is an element $\mathcal{I}(t)$ of D defined as follows:

(1) $\mathcal{I}(x_i) = I_v(x_i)$ for each individual variable x_i.

(2) $\mathcal{I}(c_i) = I_c(c_i)$ for each individual constant $c_i \in \mathcal{S}$.

(3) If $t_1, \ldots, t_n$ are terms over $\mathcal{S}$ and f_i^n is a function variable then

$$\mathcal{I}(f_i^n(t_1, \ldots, t_n)) = I_v(f_i^n)(\mathcal{I}(t_i), \ldots, \mathcal{I}(t_n)) .$$

(4) If $t_1, \ldots, t_n$ are terms over $\mathcal{S}$ and $F_i^n \in \mathcal{S}$ is a function constant then

$$\mathcal{I}(F_i^n(t_1, \ldots, t_n)) = I_c(F_i^n)(\mathcal{I}(t_i), \ldots, \mathcal{I}(t_n)) .$$

Note that the symbol $\mathcal{I}$ is now being *overloaded*, *viz.*, used two ways: to denote an interpretation, and also to denote a function from terms to mathematical meanings. The intended use is usally clear from context, so the ambiguity causes no problems. In fact, we'll go further and let $\mathcal{I}$ also denote a function from wff's to truth values:

Definition 10. (Satisfaction of atomic formulas) Let A be an atomic formula over $\mathcal{S}$ and let $\mathcal{I} = ((D, I_c), I_v)$ be an interpretaion of $\mathcal{S}$. The truth value, $\mathcal{I}(A) \in \{\texttt{true}, \texttt{false}\}$, of A under Iis defined inductively as follows:

(1) $\mathcal{I}(P_i^0) = I_v(P_i^0)$ for each propositional variable P_i^0.

(2) $\mathcal{I}(p_i^0) = I_c(p_i^0)$ for each propositional constant $p_i^0 \in \mathcal{S}$.

(3) If t_1 and t_2 are terms over $\mathcal{S}$, then

$$\mathcal{I}((t_1 = t_2)) = \texttt{true} \quad \text{iff} \quad \mathcal{I}(t_1) \text{ is equal to } \mathcal{I}(t_2) .$$

(4) If $t_1, \ldots t_n$ are terms over $\mathcal{S}$ and P_i^n is a predicate variable, then

$$\mathcal{I}(P_i^n(t_1, \ldots, t_n)) = I_v(P_i^n)(\mathcal{I}(t_i), \ldots, \mathcal{I}(t_n)) .$$

(5) If $t_1, \ldots t_n$ are terms over $\mathcal{S}$ and $p_i^n \in \mathcal{S}$ is a predicate constant then

$$\mathcal{I}(p_i^n(t_1, \ldots, t_n)) = I_c(p_i^n)(\mathcal{I}(t_i), \ldots, \mathcal{I}(t_n)) .$$

We say $\mathcal{I}$ *satisfies* A, symbolically $\mathcal{I} \models A$, iff $\mathcal{I}(A) = \texttt{true}$.

We need some preliminary notation before proceeding to define satisfaction of wff's. The following describes the patching operation on functions and interpretations.

Notation 1. If $f : S \to T$ is a function and $s \in S, t \in T$ then $f[s \mapsto t]$ denotes the function from S to T defined by

$$f[s \mapsto t](s') = \begin{cases} f(s') & \text{if } s' \neq s, \\ t & \text{if } s' = s. \end{cases}$$

Notation 2. If $\mathcal{I} = ((D, I_c), I_v)$ is an interpretation of $\mathcal{S}$, $d \in D$, and x_i is an individual variable, then $\mathcal{I}[x_i \mapsto d]$ denotes the interpretation $((D, I_c), I_v[x_i \mapsto d])$.

Definition 11. (Satisfaction of wff's) Let A be a wff over $\mathcal{S}$ and let $\mathcal{I} = ((D, I_c), I_v)$ be an interpretaion of $\mathcal{S}$. $\mathcal{I}(A) \in \{\texttt{true}, \texttt{false}\}$ is defined inductively as follows:

(1) if A is atomic, then $\mathcal{I}(A)$ is given by Definition 10.

(2) If A and B are wff's over $\mathcal{S}$ then

- $\mathcal{I}(\neg A) =$ the Boolean complement of $\mathcal{I}(A)$.
- $\mathcal{I}(A \supset B) =$ the disjunction (or) of $\mathcal{I}(B)$ and the complement of $\mathcal{I}(A)$
- $\mathcal{I}(A \wedge B) =$ the conjunction (and) of $\mathcal{I}(A)$ and $\mathcal{I}(B)$.
- $\mathcal{I}(A \vee B) =$ the disjunction of $\mathcal{I}(A)$ and $\mathcal{I}(B)$.

(3) If x_i is an individual variable and A is a wff over $\mathcal{S}$, then

$$\mathcal{I}(\forall x_i A) = \bigwedge_{d \in D} \mathcal{I}[x_i \mapsto d](A) .$$

(4) If x_i is an individual variable and A is a wff over $\mathcal{S}$, then

$$\mathcal{I}(\exists x_i A) = \bigvee_{d \in D} \mathcal{I}[x_i \mapsto d](A) .$$

We will often talk of satisfaction over models rather than over interpretations. This is defined as follows.

Definition 12. Let A be a wff over $\mathcal{S}$ and let $\mathcal{M}$ be a model of $\mathcal{S}$. We say that $\mathcal{M}$ *satisfies* A, written $\mathcal{M} \models A$, iff for all valuations I_v for $\mathcal{M}$ it is the case that $(\mathcal{M}, I_v) \models A$. Another way of saying that $\mathcal{M}$ satisfies A, is to say that A is *valid in* $\mathcal{M}$. The *second-order theory* of $\mathcal{M}$, $\mathrm{Th}_2(\mathcal{M})$, is $\{A \mid \mathcal{M} \models A\}$. The *(first-order) theory* of $\mathcal{M}$, $\mathrm{Th}(\mathcal{M})$, is $\{A \mid A$ is a first-order wff and $\mathcal{M} \models A\}$.

It is a familiar fact that the interpretation of variables which are not free in a wff doesn't effect the truth value of the wff. In particular,

Lemma 1. Let A be a closed wff (Definition 5) and suppose $\mathcal{I} = (\mathcal{M}, I_v) \models A$ for some valuation $\mathcal{I}_v$. Then $\mathcal{M} \models A$.

Proof Sketch: By induction on the definition of $\mathcal{I}(A)$, using the induction hypothesis that if $\mathcal{I}^1$ and $\mathcal{I}^2$ are interpretations over the same model such that $\mathcal{I}^1(x) = \mathcal{I}^2(x)$ for all $x \in$ free-variables(A), then $\mathcal{I}^1(A) = \mathcal{I}^2(A)$. ■

There is another notation we sometimes use for models. A model $\mathcal{M} = (D, I_c)$ of $\mathcal{S}$ is written by listing the meanings of its symbols, as

$$\langle D, I_c(c_i), I_c(f_i^n), I_c(p_i^n) \rangle_{c_i, f_i^n, p_i^n \in \mathcal{S}} .$$

For example, the model whose domain is the set on "binary" strings $\{\mathtt{a}, \mathtt{b}\}^*$ with constants $\mathtt{a}$ and $\mathtt{b}$, the concatenation function $\cdot$, and the "equal length" predicate, eqlngth, where eqlngth(x, y) iff $|x| = |y|$, would be indicated by $\langle \{\mathtt{a}, \mathtt{b}\}^*, \mathtt{a}, \mathtt{b}, \cdot, \mathrm{eqlngth} \rangle$. We'll call this model *binary strings under concatenation* for short. Similarly, the model of nonnegative integers under addition and multiplication, $\langle N, +, * \rangle$, is sometimes called *arithmetic*.

3 Subsets of the Predicate Calculus

The predicate calculus as defined above is usually called the *second-order predicate calculus*. The terms and wff's over certain restricted subsets of the second-order predicate calculus symbols are of special interest.

The *first-order predicate calculus* is obtained by restricting the allowable symbols so that the only variables allowed are the individual variables x_i. The first-order terms and wff's over a signauture $\mathcal{S}$ are thus those second-order terms and wff's which contain no predicate

variables at all, and no n-ary funtion variables with $n \geq 1$. Interpretations are modified appropriately to assign meaning to only the relevant symbols.

Another interesting subclass is obtained by removing the symbol for equality. We will talk, for example, of the first-order predicate calculus without equality.

4 Validity and Satisfiability

Definition 13. A wff A over a signature $\mathcal{S}$ is *valid* iff A is valid for all models of $\mathcal{S}$.

We often talk of valid wff's without specific reference to a signature. In this case we are referring to the signature consisting of all allowable nonlogical symbols. Thus the phrase *validities of the first-order predicate calculus* refers to all the valid first-order wff's.

Vocabulary: Valid wff's are also called *tautologies.*

Definition 14. A wff A is *satisfiable* iff there is an interpretation $\mathcal{I}$ such that $\mathcal{I} \models A$.

Definition 15. A wff A is *unsatisfiable* iff it is not satisfiable.

The following remark will be of importance when we describe a procedure for enumerating the validities of the first-order predicate calculus.

Remark 1. A is valid iff $\neg A$ is unsatisfiable.

5 A Summary of theorems

Theorem 1. (essentially Gödel's Incompleteness Theorem) The first order theory of binary strings under the operation of concatenation is neither r.e. nor co-r.e.

Let $\text{Taut}_2(\mathcal{S})$ be the set of valid wff's with signature $\mathcal{S}$, and likewise $\text{Taut}(\mathcal{S})$ be the set of valid first-order wff's.

Lemma 2. $\text{Th}(\langle \{\mathtt{a}, \mathtt{b}\}^*, \cdot, \text{eqlngth} >) \leq_m \text{Taut}_2()$

Corollary 1. The set of valid wff's of second-order predicate calculus, Taut(), is neither r.e. nor co-r.e.

Theorem 2. The valid wff's of the first-order predicate calculus are r.e.

Theorem 3. The valid wff's of the first-order predicate calculus are not recursive.

See Manna section 2-1.6 for a proof of this.

See the next sections.

6 Special Classes of Formulas

Definition 16. A wff A is a *universal* wff if it is of the form $\forall x_1 \ldots \forall x_n B$ where B is quantifier free. A is an *existential* wff if it is of the form $\exists x_1 \ldots \exists x_n B$ where B is quantifier free. B is called the *matrix* of the wff.

Prenex normal form, which we will not define here, is another important form for wff's. See Manna section 2-1.5 for a definition.

7 Relations Between Formulas

Let A and B be wff's (not necessarily over the same signature).

Definition 17. A and B are *equisatisfiable* iff

$$A \text{ is satisfiable iff } B \text{ is satisfiable.}$$

Definition 18. A and B are *equivalent* iff for every interpretation $\mathcal{I}$ which assigns meaning to the symbols in both A and B,

$$\mathcal{I} \models A \quad \text{iff} \quad \mathcal{I} \models B\,.$$

We will be interested in various effective transformations between wff's which have the property of yielding a wff either equivalent to the original one or at least equisatisfiable with it. The first lemma in this vein is about the reduction to Prenex normal form.

Lemma 3. There is an effective procedure to transform a given wff A into an equivalent wff B which is in Prenex normal form.

Further simplifications of the structure of a wff are possible if one asks only to preserve satisfiability.

Lemma 4. There is an effective procedure to transform a given first-order wff A into an equisatisfiable first-order wff B which is a universal wff.

The technique used to reduce a wff A in Prenex normal form to an equisatisfiable universal wff B is called *Skolemization*. For this reason, B is sometimes called the *Skolemized* form of A.

8 Herbrand's Theorem

Vocabulary: A closed term is also referred to as a *ground term*.

Let $A = \forall x_1 \ldots \forall x_n B$ be a first-order universal wff, and let $\mathcal{S}$ be the signature which consists of all the nonlogical symbols occuring in A. If A has no individual constants, add a new individual constant c to $\mathcal{S}$.

Definition 19. A *ground instance* of A is a wff obtained b
of a variable in B by a ground term over $\mathcal{S}$. That is, a
the form $[B]^{t_1,\ldots,t_n}_{x_1,\ldots,x_n}$ where $t_1,\ldots,t_n$ are ground terms of A.

We consider next the notion of propositional satisfiability of a co
instances of A. Let $G_1 \wedge \ldots \wedge G_n$ be a finite conjunction of ground insta
each distinct atomic subwff of the wff's $G_1,\ldots,G_n$ as a propositional variable a
the value `true` or `false`. The wff $G_1 \wedge \ldots \wedge G_n$ now has a `true` or `false` value dete
by the assignments to the atomic subwff's and the rules of propositional logic.

Definition 20. $G_1 \wedge \ldots \wedge G_n$ is *propositionally satisfiable* iff there is an assignment of truth values to the atomic subwff's under which $G_1 \wedge \ldots \wedge G_n$ has the value `true`. $G_1 \wedge \ldots \wedge G_n$ is *propositionally unsatisfiable* iff it is not propositionally satisfiable.

Theorem 4. (Herbrand's theorem) A first-order universal wff is unsatisfiable iff there is a finite conjunction of ground instances of the wff which is propositionally unsatisfiable.

Theorem 4 provides a semi-decision procedure for the validity problem of the first order predicate calculus. Lemmas 3 and 4 and Theorem 4 provide the proof of Theorem 2.

Notes on Predicate Calculus

1 Syntax

The syntax of the predicate calculus is constructed out of two kinds of symbols: the *logical* symbols and the *nonlogical* symbols. The logical symbols are

- Delimiters: ")", "(" (parentheses), "," (comma).
- Truth Constants: **true**, **false**.
- Propositional Connectives: $\neg$ (not), $\supset$ (implies), $\wedge$ (and), $\vee$ (or).
- The equality operator: $=$.
- Quantifiers: $\forall$ (universal quantifier), $\exists$ (existential quantifier).
- Variables: for $i, n \geq 0$,
  - (1) symbols F_i^n called *n-ary function variables*; the zero-ary function variables F_i^0 are also called *individual variables* and are also written as "x_i".
  - (2) symbols P_i^n called *n-ary predicate variables*; the zeroary predicate variables P_i^0 are also called *propositional variables*.

The nonlogical symbols are

- symbols f_i^n called *n-ary function constants*; the zero-ary function constants f_i^0 are also called *individual constants* and are also written "c_i".
- symbols p_i^n called *n-ary predicate constants*; the p_i^0 are also called *propositional constants*.

Definition 1. A *signature* is a set of nonlogical symbols.

Terms and well-formed formulas (wff's) over a given signature, $\mathcal{S}$, are defined by induction.

Definition 2. A *term* over $\mathcal{S}$ is defined as follows:

(1) Each individual variable x_i is a term over $\mathcal{S}$.

(2) Each individual constant $c_i \in \mathcal{S}$ is a term over $\mathcal{S}$.

(3) If $t_1, \ldots, t_n$ are terms over $\mathcal{S}$ and f_i^n is a function variable then $f_i^n(t_1, \ldots, t_n)$ is a term over $\mathcal{S}$.

(4) If $t_1, \ldots, t_n$ are terms over $\mathcal{S}$ and $F_i^n \in \mathcal{S}$ is a function constant then $F_i^n(t_1, \ldots, t_n)$ is a term over $\mathcal{S}$.

Note that the parentheses and commas used in the constuction of terms are the logical delimiter symbols.

Definition 3. *Atomic formulas* over $\mathcal{S}$ are defined as follows:

(1) Each propositional variable P_i^0 is an atomic formula over $\mathcal{S}$.

(2) Each propositional constant $p_i^0 \in \mathcal{S}$ is an atomic formula over $\mathcal{S}$.

(3) If t_1 and t_2 are terms over $\mathcal{S}$ then $(t_1 = t_2)$ is an atomic formula over $\mathcal{S}$.

(4) If $t_1, \ldots t_n$ are terms over $\mathcal{S}$ and P_i^n is a predicate variable then $P_i^n(t_1, \ldots, t_n)$ is an atomic formula over $\mathcal{S}$.

(5) If $t_1, \ldots t_n$ are terms over $\mathcal{S}$ and $p_i^n \in \mathcal{S}$ is a predicate constant then $p_i^n(t_1, \ldots, t_n)$ is an atomic formula over $\mathcal{S}$.

Definition 4. *Well-formed formulas* (wff's) over $\mathcal{S}$ are defined as follows:

(1) An atomic formula over $\mathcal{S}$ is a wff over $\mathcal{S}$.

(2) If A and B are wff's over $\mathcal{S}$ then so are $(\neg A)$, $(A \supset B)$, $(A \wedge B)$, and $(A \vee B)$.

(3) If x_i is an individual variable and A is a wff over $\mathcal{S}$ then $(\forall x_i A)$ and $(\exists x_i A)$ are wff's over $\mathcal{S}$.

We emphasize that terms and wff's over $\mathcal{S}$ are *syntactic* entities constructed out of the symbols in $\mathcal{S}$ and the logical symbols.

Parentheses will be dropped in writing wff's where there is no ambiguity. We will sometimes use symbols different from the above, such as x, y, z for variables and f, g for function constants, following Manna's notational conventions.

We omit here the important technical definitions of *free and bound variables* in a wff. See Manna for this. We can now define closed terms and wff's:

Definition 5. A *closed* term (wff) is a term (wff) with no free variables.

We write $[A]_{x_1,\ldots,x_n}^{t_1,\ldots,t_n}$ for the wff obtained by replacing each free occurence of x_i with t_i in A. Bound variables in A are renamed if necessary so that the quantifiers of A do not capture the free variables in the t_i. We omit the details of renaming.

It is a small technical convenience to distinguish constants from variables. Clearly, they behave syntactically much the same—the only difference is that variables can be bound by quantifiers, while constants cannot.

2 Semantics

Let $\mathcal{S}$ be a signature. We proceed to define the meaning of terms and wff's over $\mathcal{S}$.

Definition 6. A *model* of $\mathcal{S}$ is a pair $\mathcal{M} = (D, I_c)$ where D is a non-empty set, called the *domain* of $\mathcal{M}$, and I_c is a function with domain $\mathcal{S}$ which assigns values to the elements of $\mathcal{S}$ as follows:

(1) $I_c(c_i)$ is an element of D for all $c_i \in \mathcal{S}$.

(2) $I_c(p_i^0) \in \{\texttt{true}, \texttt{false}\}$ for all $p_i^0 \in \mathcal{S}$.

(3) $I_c(f_i^n) : D^n \to D$ is a *total* n-ary function over D for all $n \geq 1$ and $f_i^n \in \mathcal{S}$.

(4) $I_c(p_i^n) : D^n \to \{\texttt{true}, \texttt{false}\}$ is an n-ary predicate over D for all $n \geq 1$ and $p_i^n \in \mathcal{S}$.

Definition 7. A *valuation* for a model $\mathcal{M} = (D, I_c)$ of $\mathcal{S}$ is a function I_v which assigns values to the variables as follows:

(1) $I_v(x_i)$ is an element of D for all individual variables x_i.

(2) $I_v(P_i^0) \in \{\texttt{true}, \texttt{false}\}$ for all P_i^0.

(3) $I_v(F_i^n) : D^n \to D$ is a n-ary function over D for all $n \geq 1$ and all function variables F_i^n.

(4) $I_v(P_i^n) : D^n \to \{\texttt{true}, \texttt{false}\}$ is a n-ary predicate over D for all $n \geq 1$ and all predicate variables P_i^n.

Definition 8. An *interpretation* of $\mathcal{S}$ is pair $\mathcal{I} = (\mathcal{M}, I_v)$ where $\mathcal{M}$ is a model for $\mathcal{S}$ and I_v is a valuation for $\mathcal{M}$.

We define the meaning of a term by induction over the definition of a term.

Definition 9. The *meaning* of a term t over $\mathcal{S}$ in an interpretation $\mathcal{I} = ((D, I_c), I_v)$ of $\mathcal{S}$ is an element $\mathcal{I}(t)$ of D defined as follows:

(1) $\mathcal{I}(x_i) = I_v(x_i)$ for each individual variable x_i.

(2) $\mathcal{I}(c_i) = I_c(c_i)$ for each individual constant $c_i \in \mathcal{S}$.

(3) If $t_1, \ldots, t_n$ are terms over $\mathcal{S}$ and f_i^n is a function variable then

$$\mathcal{I}(f_i^n(t_1, \ldots, t_n)) = I_v(f_i^n)(\mathcal{I}(t_i), \ldots, \mathcal{I}(t_n)) .$$

(4) If $t_1, \ldots, t_n$ are terms over $\mathcal{S}$ and $F_i^n \in \mathcal{S}$ is a function constant then

$$\mathcal{I}(F_i^n(t_1, \ldots, t_n)) = I_c(F_i^n)(\mathcal{I}(t_i), \ldots, \mathcal{I}(t_n)) .$$

Note that the symbol $\mathcal{I}$ is now being *overloaded*, *viz.*, used two ways: to denote an interpretation, and also to denote a function from terms to mathematical meanings. The intended use is usually clear from context, so the ambiguity causes no problems. In fact, we'll go further and let $\mathcal{I}$ also denote a function from wff's to truth values:

Definition 10. (Satisfaction of atomic formulas) Let A be an atomic formula over $\mathcal{S}$ and let $\mathcal{I} = ((D, I_c), I_v)$ be an interpretaion of $\mathcal{S}$. The truth value, $\mathcal{I}(A) \in \{\texttt{true}, \texttt{false}\}$, of A under $\mathcal{I}$ is defined inductively as follows:

(1) $\mathcal{I}(P_i^0) = I_v(P_i^0)$ for each propositional variable P_i^0.

(2) $\mathcal{I}(p_i^0) = I_c(p_i^0)$ for each propositional constant $p_i^0 \in \mathcal{S}$.

(3) If t_1 and t_2 are terms over $\mathcal{S}$, then

$$\mathcal{I}((t_1 = t_2)) = \texttt{true} \quad \text{iff} \quad \mathcal{I}(t_1) \text{ is equal to } \mathcal{I}(t_2)\,.$$

(4) If $t_1, \ldots t_n$ are terms over $\mathcal{S}$ and P_i^n is a predicate variable, then

$$\mathcal{I}(P_i^n(t_1, \ldots, t_n)) = I_v(P_i^n)(\mathcal{I}(t_i), \ldots, \mathcal{I}(t_n))\,.$$

(5) If $t_1, \ldots t_n$ are terms over $\mathcal{S}$ and $p_i^n \in \mathcal{S}$ is a predicate constant then

$$\mathcal{I}(p_i^n(t_1, \ldots, t_n)) = I_c(p_i^n)(\mathcal{I}(t_i), \ldots, \mathcal{I}(t_n))\,.$$

We need some preliminary notation before proceeding to define satisfaction of wff's. The following describes the patching operation on functions and interpretations.

Notation 1. If $f : S \to T$ is a function and $s \in S, t \in T$ then $f[s \mapsto t]$ denotes the function from S to T defined by

$$f[s \mapsto t](s') = \begin{cases} f(s') & \text{if } s' \neq s, \\ t & \text{if } s' = s. \end{cases}$$

Notation 2. If $\mathcal{I} = ((D, I_c), I_v)$ is an interpretation of $\mathcal{S}$, $d \in D$, and x_i is an individual variable, then $\mathcal{I}[x_i \mapsto d]$ denotes the interpretation $((D, I_c), I_v[x_i \mapsto d])$.

Definition 11. (Satisfaction of wff's) For wff's A over $\mathcal{S}$ and interpretations $\mathcal{I} = ((D, I_c), I_v)$ of $\mathcal{S}$, the truth value $\mathcal{I}(A) \in \{\texttt{true}, \texttt{false}\}$ is defined inductively as follows:

(1) if A is atomic, then $\mathcal{I}(A)$ is given by Definition 10.

(2) If A and B are wff's over $\mathcal{S}$ then

- $\mathcal{I}(\neg A)$ = the Boolean complement (not) of $\mathcal{I}(A)$, *i.e.*, $\neg\mathcal{I}(A)$.

- $\mathcal{I}(A \supset B)$ = the disjunction (or) of $\mathcal{I}(B)$ and the complement of $\mathcal{I}(A)$, *i.e.*, $\mathcal{I}(B) \vee \neg\mathcal{I}(A)$. (This of course is equal to $\mathcal{I}(A) \supset \mathcal{I}(B)$.)
- $\mathcal{I}(A \wedge B) = \mathcal{I}(A) \wedge \mathcal{I}(B)$.
- $\mathcal{I}(A \vee B) = \mathcal{I}(A) \vee \mathcal{I}(B)$.

(3) If x_i is an individual variable and A is a wff over $\mathcal{S}$, then

$$\mathcal{I}(\forall x_i A) = \bigwedge_{d \in D} \mathcal{I}[x_i \mapsto d](A) .$$

(Equivalently, $\mathcal{I}(\forall x_i A) = \texttt{true}$ iff $\mathcal{I}[x_i \mapsto d](A) = \texttt{true}$ for all $d \in D$.)

(4) If x_i is an individual variable and A is a wff over $\mathcal{S}$, then

$$\mathcal{I}(\exists x_i A) = \bigvee_{d \in D} \mathcal{I}[x_i \mapsto d](A) .$$

Definition 12. We say $\mathcal{I}$ *satisfies* A, symbolically $\mathcal{I} \models A$, iff $\mathcal{I}(A) = \texttt{true}$. We will also talk of satisfaction over *models* rather than over interpretations. We say that $\mathcal{M}$ *satisfies* A, written $\mathcal{M} \models A$, iff for all valuations I_v for $\mathcal{M}$ it is the case that $(\mathcal{M}, I_v) \models A$. We can also read $\mathcal{M} \models A$ as "A is *valid in* $\mathcal{M}$".

The *second-order theory* of $\mathcal{M}$, $\mathrm{Th}_2(\mathcal{M})$, is $\{A \mid \mathcal{M} \models A\}$. The *(first-order) theory* of $\mathcal{M}$, $\mathrm{Th}(\mathcal{M})$, is $\{A \mid A$ is a first-order wff and $\mathcal{M} \models A\}$.

It is a familiar fact that the interpretation of variables which are not free in a wff doesn't effect the truth value of the wff. In particular,

Lemma 1. Let A be a closed wff (Definition 5) and suppose $\mathcal{I} = (\mathcal{M}, I_v) \models A$ for some valuation $\mathcal{I}_v$. Then $\mathcal{M} \models A$.

Proof Sketch: By induction on the definition of A, using the induction hypothesis that if $\mathcal{I}^1$ and $\mathcal{I}^2$ are interpretations over the same model such that $\mathcal{I}^1(x) = \mathcal{I}^2(x)$ for all $x \in$ free-variables(A), then $\mathcal{I}^1(A) = \mathcal{I}^2(A)$. ■

There is another notation we sometimes use for models. A model $\mathcal{M} = (D, I_c)$ of $\mathcal{S}$ is written by listing the meanings of its nonlogical symbols, as

$$\langle D, I_c(c_i), I_c(f_i^n), I_c(p_i^n) \rangle_{c_i, f_i^n, p_i^n \in \mathcal{S}} .$$

For example, the model whose domain is the set of "binary" strings $\{\mathtt{a}, \mathtt{b}\}^*$ with constants $\mathtt{a}$ and $\mathtt{b}$, the concatenation function $\cdot$, and the "equal length" predicate, eqlngth, where eqlngth(x, y) iff $|x| = |y|$, would be indicated by $\langle \{\mathtt{a}, \mathtt{b}\}^*, \mathtt{a}, \mathtt{b}, \cdot, \mathrm{eqlngth} \rangle$. We'll call this model *binary strings under concatenation* for short. Similarly, the model of nonnegative integers under addition and multiplication, $\langle N, +, * \rangle$, is sometimes called *arithmetic*.

3 Subsets of the Predicate Calculus

The predicate calculus as defined above is usually called the *second-order predicate calculus*. The terms and wff's over certain restricted subsets of the second-order predicate calculus symbols are of special interest.

The *first-order predicate calculus* is obtained by restricting the allowable symbols so that the only variables allowed are the individual variables x_i. The first-order terms and wff's over a signauture $\mathcal{S}$ are thus those second-order terms and wff's which contain no predicate variables at all, and no n-ary function variables with $n \geq 1$. Interpretations are modified appropriately to assign meaning to only the relevant symbols.

Another interesting subclass is obtained by removing the symbol for equality. We will talk, for example, of the first-order predicate calculus without equality.

4 Validity and Satisfiability

Definition 13. A wff A over a signature $\mathcal{S}$ is *valid* iff A is valid for all models of $\mathcal{S}$.

We often talk of valid wff's without specific reference to a signature. In this case we are referring to the signature consisting of all allowable nonlogical symbols. Thus the phrase *validities of the first-order predicate calculus* refers to all the valid first-order wff's.

Definition 14. A wff A is *satisfiable* iff there is an interpretation $\mathcal{I}$ such that $\mathcal{I} \models A$.

Definition 15. A wff A is *unsatisfiable* iff it is not satisfiable.

The following remark will be of importance when we describe a procedure for enumerating the validities of the first-order predicate calculus.

Remark 1. A is valid iff $\neg A$ is unsatisfiable.

Vocabulary: Valid wff's are also called *tautologies.*

Definition 16. Let $\text{Taut}_2(\mathcal{S})$ be the set of valid wff's with signature $\mathcal{S}$, and likewise $\text{Taut}(\mathcal{S})$ be the set of valid first-order wff's.

5 A Summary of theorems

Theorem 1. (essentially Gödel's Incompleteness Theorem) The first order theory of binary strings, $\mathrm{Th}(\langle\{\mathtt{a},\mathtt{b}\}^*, \mathtt{a}, \mathtt{b}, \cdot, \mathrm{eqlngth}\rangle)$, is neither r.e. nor co-r.e.

Lemma 2. $\mathrm{Th}(\langle\{\mathtt{a},\mathtt{b}\}^*, \mathtt{a}, \mathtt{b}, \cdot, \mathrm{eqlngth}\rangle) \leq_m \mathrm{Taut}_2(\emptyset)$

Corollary 1. The set $\mathrm{Taut}_2(\emptyset)$ of valid wff's of second-order predicate calculus without nonlogical symbols is neither r.e. nor co-r.e.

Theorem 2. The set of valid wff's of the first-order predicate calculus is not recursive.

See Manna section 2-1.6 for a proof of this.

We will demonstrate in the sections below that the set of valid wff's of the first-order predicate calculus *is* r.e.

6 Special Classes of Formulas

Definition 17. A wff A is a *universal* wff if it is of the form $\forall x_1 \ldots \forall x_n B$ where B is quantifier free. A is an *existential* wff if it is of the form $\exists x_1 \ldots \exists x_n B$ where B is quantifier free. B is called the *matrix* of the wff.

Prenex normal form, which we will not define here, is another important form for wff's. See Manna section 2-1.5 for a definition.

7 Relations Between Formulas

Let A and B be wff's (not necessarily over the same signature).

Definition 18. A and B are *equivalent* iff $\mathcal{I}(A) = \mathcal{I}(B)$ for every interpretation $\mathcal{I}$ which assigns meaning to the symbols in both A and B.

Remark. A is equivalent to B iff $\models (A \supset B) \wedge (B \supset A)$.

Definition 19. A and B are *equisatisfiable* iff

A is satisfiable iff B is satisfiable.

We will be interested in various effective transformations between wff's which have the property of yielding a wff either equivalent to the original one or at least equisatisfiable with it. The first lemma in this vein is about the reduction to prenex normal form.

Lemma 3. There is an effective procedure to transform a given wff A into an equivalent wff B which is in prenex normal form.

Further simplifications of the structure of a wff are possible if one asks only to preserve satisfiability.

Lemma 4. There is an effective procedure to transform a given first-order wff A into an equisatisfiable first-order wff B which is a universal wff.

The technique used to reduce a wff A in prenex normal form to an equisatisfiable universal wff B is called *Skolemization*. For this reason, B is sometimes called the *Skolemized* form of A.

8 Herbrand's Theorem

Vocabulary: A closed term is also referred to as a *ground term*.

Let $A = \forall x_1 \ldots \forall x_n B$ be a first-order universal wff with matrix B, and let $\mathcal{S}$ be the signature which consists of all the nonlogical symbols occuring in A. If A has no individual constants, add a new individual constant c to $\mathcal{S}$.

Definition 20. A *ground instance* of A is a wff obtained by replacing each free occurrence of a variable in B by a ground term over $\mathcal{S}$. That is, a ground instance of A is a wff of the form $[B]^{t_1,\ldots,t_n}_{x_1,\ldots,x_n}$ where $t_1, \ldots, t_n$ are ground terms over $\mathcal{S}$.

We consider next the notion of propositional satisfiability of a conjunction of ground instances of A. Let $G_1 \wedge \ldots \wedge G_n$ be a finite conjunction of ground instances of A. Treat each distinct atomic subwff of the wff's $G_1, \ldots, G_n$ as a propositional variable and assign it the value `true` or `false`. The wff $G_1 \wedge \ldots \wedge G_n$ now has a `true` or `false` value determined by the assignments to the atomic subwff's and the rules of propositional logic.

Definition 21. $G_1 \wedge \ldots \wedge G_n$ is *propositionally satisfiable* iff there is an assignment of truth values to the atomic subwff's under which $G_1 \wedge \ldots \wedge G_n$ has the value `true`. $G_1 \wedge \ldots \wedge G_n$ is *propositionally unsatisfiable* iff it is not propositionally satisfiable.

Theorem 3. (Herbrand's theorem) A first-order universal wff is unsatisfiable iff there is a finite conjunction of ground instances of the wff which is propositionally unsatisfiable.

Theorem 3 provides a semi-decision procedure for the validity problem of the first order predicate calculus. Namely, to see if A is valid, transform $\neg A$ into universal form according to Lemmas 3 and 4. Then start checking, in parallel, all finite subsets of the ground instances of the resulting universal formula for propositional unsatisfiability. In particular, Theorem 3 immdiately implies the following

Theorem 4. The set of valid wff's of the first-order predicate calculus is r.e.

Review problems for Quiz 2

These problems were assigned last year on homeworks and quizzes about materials that will be covered on next quiz.

Problem 1. Explain why the following Post Correspondence Problem has no solution: (This is problem 1-20(a) from Manna's text)

$$\{(\mathtt{ba}, \mathtt{bab}), (\mathtt{abb}, \mathtt{bb}), (\mathtt{bab}, \mathtt{abb})\} .$$

A solution would be forced to begin with the pair (ba, bab) because the other pairs are mismatched in the first charecter. At this point there is one more b at the bottom than at the top. Since all three pairs have the property that the number of bs in the second half of the pair is $\geq$ the number of bs in the first half, further use of any of the pairs will not reduce the diference in the number of bs between bottom and top. So no solution is possible.

Problem 2. Describe a countermodel to show that the following formula is not valid:

$$\{\forall x \forall y \forall z\, [p(x,y) \wedge p(y,z) \supset p(x,z)] \wedge \forall x \forall y\, [p(x,y) \vee p(y,x)]\} \supset \exists x \forall y\, p(x,y) \ .$$

Let the domain of the model $\mathcal{M}$ be the integers $\mathbf{Z}$, and let the interpretaion of p be the usual ordering on the integers,

$$I_c(p)(m,n) \quad \text{iff} \quad m \geq n$$

for all $m, n \in \mathbf{Z}$. Since $\geq$ is a transitive and total order on $\mathbf{Z}$ the antecedents of the implication in the formula are true. But since there is not largest integer, the consequent of the implication is false. So $\mathcal{M}$ does not satisfy the formula.

Problem 3. Let $\mathcal{S} = \{p_1, p_2\}$ be a signature consisting of two unary predicate constants. Write down a satisfiable first-order formula, A, *without equality* over the signature $\mathcal{S}$ such that the domain of any model of A has at least 4 elements.

Elements with different "truth patterns" (cf. Problem Set 5, problem 2) are distinct. Hence the formula

$$\begin{aligned} \exists x_1 \exists x_2 \exists x_3 \exists x_4 [\quad & (p_1(x_1) \ \wedge \ p_2(x_1)) \\ & \wedge (\neg p_1(x_2) \wedge \ p_2(x_2)) \\ & \wedge (p_1(x_3) \ \wedge \neg p_2(x_3)) \\ & \wedge (\neg p_1(x_4) \wedge \neg p_2(x_4))] \end{aligned}$$

does the job.

Problem 4. A $\exists\forall$-wff is a first-order wff of the form

$$\exists x_1 \ldots \exists x_n \forall y_1 \ldots \forall y_m \, B$$

where B is a quantifier free wff. Similarly, a $\forall\exists$-wff is of the form

$$\forall x_1 \ldots \forall x_n \exists y_1 \ldots \exists y_m \, B \, .$$

Show that the validity problem for $\forall\exists$ wffs is many-one reducible to the unsatisfiability problem for $\exists\forall$ wffs.

A wff is valid iff its negation is unsatisfiable, and the negation of a $\forall\exists$-wff is (equivalent to) $\exists\forall$-wff. The function mapping a $\forall\exists$-wff A to a $\exists\forall$-wff equivalent to $\neg A$ is computable. Our many-one reduction consists of this function.

Problem 5. Let $\mathcal{S}$ be a first order predicate calculus signature consisting of n unary predicate constants,

$$\mathcal{S} = \{P_1, \ldots, P_n\} \, .$$

Let $\mathcal{I} = (D, I_c, I_v)$ be an interpretation of $\mathcal{S}$. We say that two elements d and d' of D *have the same truth pattern* iff

$$I_c(P_i)(d) \text{ iff } I_c(P_i)(d') \text{ for all } i = 1, \ldots n \, .$$

It is easy to see that the property of having the same truth pattern defines an equivalence relation over D. The equivalence class of an element d of D is

$$[d] = \{d' \in D \mid d \text{ and } d' \text{ have the same truth pattern}\} \, .$$

The *collapse* of $\mathcal{I}$ is the interpretation $\overline{\mathcal{I}} = (\overline{D}, \overline{I}_c, \overline{I}_v)$ of $\mathcal{S}$ defined as follows:

- the domain $\overline{D}$ of $\overline{\mathcal{I}}$ is the set of equivalence classes,

$$\overline{D} = \{[d] \mid d \in D\} \, .$$

- For each unary predicate symbol P_i of $\mathcal{S}$ and each $d \in D$ we define

$$\overline{I}_c(P_i)([d]) \text{ iff } I_c(P_i)(d) \, .$$

  ($\overline{I}_c(P_i)$ is well defined because if $[d] = [d']$ then $I_c(P_i)(d)$ iff $I_c(P_i)(d')$).

- For each individual variable x we let

$$\overline{I}_v(x) = [I_v(x)] \, .$$

(a) Prove by induction on the definition of a first order wff A *without equality* over the signature $\mathcal{S}$ that

$$\mathcal{I} \models A \text{ iff } \overline{\mathcal{I}} \models A\,.$$

We need a technical remark about the relationship between the patching and the collapsing operations.

Claim 1. $\overline{\mathcal{I}[x \mapsto d]} = \overline{\mathcal{I}}[x \mapsto [d]]$ for any $d \in D$ and any individual variable x.

Proof: By the definition of $\overline{I}_v$. ■

We now induct on the definition of wffs without equality over $\mathcal{S}$ to show that

$$\mathcal{I} \models A \text{ iff } \overline{\mathcal{I}} \models A\,.$$

The only atomic wff is of the form $P_i(x)$. (Note that the only terms over $\mathcal{S}$ are individual variables x). We have

$$\overline{\mathcal{I}} \models P_i(x) \quad \text{iff} \quad \overline{I}_c(P_i)(\overline{I}_v(x)) \quad \text{iff} \quad \overline{\mathcal{I}}_c(P_i)([\mathcal{I}_v(x)]) \quad \text{iff} \quad I_c(P_i)(I_v(x))$$

using the definitions of $\overline{I}_c$ and $\overline{I}_v$. So

$$\overline{\mathcal{I}} \models P_i(x) \quad \text{iff} \quad \mathcal{I} \models P_i(x)\,.$$

If A is of the form $\neg B$ or $B \wedge C$ the result follows easily by induction. Finally suppose A is $\forall x\, B$. Then

$$\begin{aligned} \mathcal{I} \models A \quad &\text{iff} \quad \mathcal{I}[x \mapsto d] \models B \text{ for all } d \in D \\ &\text{iff} \quad \overline{\mathcal{I}[x \mapsto d]} \models B \text{ for all } d \in D \\ &\text{iff} \quad \overline{\mathcal{I}}[x \mapsto [d]] \models B \text{ for all } d \in D \\ &\text{iff} \quad \overline{\mathcal{I}} \models A\,. \end{aligned}$$

The first iff here is by the definition of the meaning of $\forall$. The second iff is by the induction hypothesis. The third iff is by Claim 1. The last iff is again by the definition of the meaning of $\forall$.

(b) Use part (a) to show that the validity problem for $\mathcal{S}$ is decidable for formulas without equality. That is, show that there is a program which given any first order formula A without equality over the signature $\mathcal{S}$ outputs "yes" if A is valid and "no" otherwise.

The first necessary remark is

Lemma 1. There is an algorithm which when given a *finite* model $\mathcal{M}$ and a formula A decides whether $\mathcal{M} \models A$.

To any subset D of $\{\mathtt{true},\mathtt{false}\}^n$ we associate a model $\mathcal{M}^D = (D, I_c^D)$ of $\mathcal{S}$ with domain D and with I_c^D defined as follows: for each $i = 1,\ldots n$ and each $(v_1,\ldots,v_n) \in \{\mathtt{true},\mathtt{false}\}^n$,

$$I_c^D(P_i)((v_1,\ldots,v_n)) \quad \text{iff} \quad v_i = \mathtt{true}\,.$$

Part (a) of this problem implies that for *any* model $\mathcal{M}$ of $\mathcal{S}$ there is some D such that for any wff without equality,

$$\mathcal{M} \models A \quad \text{iff} \quad \mathcal{M}^D \models A\,.$$

This implies that to check whether A is valid it suffices to check that A is true in all the models $\mathcal{M}^D$. Now note that the number of models $\mathcal{M}^D$ is finite (in fact, $|\{\mathcal{M}^D \mid D \subseteq \{\mathtt{true},\mathtt{false}\}^n\}| = 2^{n2^n}$) and each $\mathcal{M}^D$ is finite. Hence we may use Lemma 1 to check whether it is the case that $\mathcal{M}^D \models A$ for all $D \subseteq \{\mathtt{true},\mathtt{false}\}^n$.

Problem 6.

(a) Let A range over second-order wffs with signature $\{+,*,=\}$. Show that it is decidable whether $\mathcal{Z} = \langle Z_3, +, *\rangle \models A$, where Z_3 denotes the integers modulo 3.

> The structure $\mathcal{Z}$ is finite. Given any formula A we can decide whether or not $\mathcal{Z} \models A$ by an exhaustive search through all possibilities for the quantified variables. A more formal argument would specify a recursive algorithm based on the inductive definition of the formula.

(b) Generalize part (a) to conclude that

$$\{(B,n) \mid B \text{ is a second-order wff (with any signature), } n > 0, \text{ and } B \text{ has a model whose domain is of size } n\ \}$$

is decidable.

> Given (B,n) let $\mathcal{S}$ be the signature consisting of all the function and predicate constants occuring in B. There are only finitely many (upto isomorphism) different models over this finite signature which have a domain of size n, and we can systematically genereate these models (i.e. generate a model from each isomorphism class) and test for each, using the method of part (a), whether or not they satisfy A.

Problem 7. Manna exercise 2-9 (a).
See the attached solutions

Problem 8. Manna exercise 2-10 (a),(b),(c),(d),(n).
See the attached solutions

Problem 7. (Manna 2-9a)

Show $\exists x \forall y [(p(x,y) \wedge \neg p(y,x)) \supset (p(x,x) \equiv p(y,y))]$ to be invalid

A Counterexample:

Domain = D = Integers

$$p(x,y) = \begin{cases} T, \text{ if } x > y \\ F, \text{ if } x \leq y \end{cases}$$

$$p(x,y) = \begin{cases} T \text{ if } x = y \text{ and } x \text{ is even} \\ F \text{ if } x = y \text{ and } x \text{ is odd} \end{cases}$$

The formula claims that there is an x such that for any y the implication holds.

But for any integer x the implication is false for $y = x-1$ because

$p(x,y) = p(x, x-1) = T$

$p(y,x) = p(x-1, x) = F$

$(p(x,y) \wedge \neg p(y,x)) = (T \wedge \neg F) = T$ so the antecedent holds

but $p(x,x) = T$ iff x is even, and

$p(y,y) = T$ iff x is odd, so

$p(x,x) \not\equiv p(y,y)$ Thus the $\supset$ is false, and so is the whole formula

Problem 8 :
(problems 2-10 (a)(b)(c)(d)(n) from Manna)

(a) $p(x) \supset \exists x\, p(x)$ is VALID. $(A \supset B)$ is not valid when, under some interpretation, A is True and B is False. But, clearly, under an interpretation where $p(x)$ holds, there exists an x such that $p(x)$ holds. So this wff is VALID.

(b) $\exists x\, p(x) \supset p(x)$ is NOT VALID. Choose $D = \{a, b\}$ and
$p(x) = \begin{cases} \text{True} & x = a \\ \text{False} & x = b \end{cases}$.

Clearly $\exists x\, p(x)$ holds for any interpretation using this D and $p(x)$. But if an interpretation assigned the free variable x (from the r.h.s. of the implication) a value of b, the equation would not hold. So the wff is NOT VALID.

(c) $p(x) \supset \forall x\, p(x)$ is NOT VALID. Again, choose $D = \{a, b\}$ Use $p(x)$ as defined for part (b). Allowing the free variable x to take the value a makes $p(x)$ true. But $\forall x\, p(x)$ is clearly not true. So we get $T \supset F$, which tells us the wff is NOT VALID.

(d) $\forall x\, p(x) \supset p(x)$ is VALID. Clearly, under any interpretation where $\forall x\, p(x)$ holds, $p(x)$ will hold for any value that an interpretation may assign to x.

(n) $\exists P\, [\exists x\, P(x) \supset \forall x\, P(x)]$ is VALID.

For any interpretation $\exists$ a "constant" predicate which is always True. Choose P to be that predicate.

problem Set 3

Problem 1. Manna, page 153, exercise 2-9 (b),(c).

Problem 2. Manna, page 153, exercise 2-10 (e),(f),(g),(h),(o).

Problem 3. Prove lemma 1 page 5 of handout 11.
(*Hint:* work the induction in detail.)

Problem 4. Let $\mathcal{I} = ((D, \mathcal{I}_c), \mathcal{I}_v)$ and $\mathcal{I}' = ((D', \mathcal{I}'_c), \mathcal{I}'_v)$ be interpretations over a common second-order predicate calculus signature $\mathcal{S}$.

Definition 1. A function $h : D \to D'$ is a *homomorphism between* $\mathcal{I}$ *and* $\mathcal{I}'$, written $h : \mathcal{I} \to \mathcal{I}'$, iff

(1) for all n-ary function constants f in $\mathcal{S}$ and for all $d_1, \ldots, d_n \in D$,

$$h(\mathcal{I}_c(f)(d_1, \ldots, d_n)) = \mathcal{I}'_c(f)(h(d_1), \ldots, h(d_n)) ,$$

and similarly for all n-ary function variables F.

(2) for all n-ary predicate constants p in $\mathcal{S}$ and for all $d_1, \ldots, d_n \in D$,

$$\mathcal{I}_c(p)(d_1, \ldots, d_n) \quad \text{iff} \quad \mathcal{I}'_c(p)(h(d_1), \ldots h(d_n)) ,$$

and similarly for all n-ary predicate variables P.

The homomorhism is *onto* iff $h : D \to D'$ is onto, and in this case $\mathcal{I}'$ is said to be a *homomorphic image* of $\mathcal{I}$.

We will need the notion of isomorphism of models which we proceed to define here.

Definition 2. A homomorphism $h : \mathcal{I} \to \mathcal{I}'$ is an *isomorphism between* $\mathcal{I}$ *and* $\mathcal{I}'$ iff the function $h : D \to D'$ is a bijection. $\mathcal{I}$ and $\mathcal{I}'$ are *isomorphic*, written $\mathcal{I} \cong \mathcal{I}'$, if there is an isomorphism between $\mathcal{I}$ and $\mathcal{I}'$.

Definition 3. Two models $\mathcal{M}$ and $\mathcal{M}'$ over a common signature $\mathcal{S}$ are isomorphic iff there is a pair of interpretations $\mathcal{I}_v$ and $\mathcal{I}'_v$ of the free variables such that $(\mathcal{M}, \mathcal{I}_v) \cong (\mathcal{M}', \mathcal{I}'_v)$.

Let $\mathcal{S}$ be any signature. Let $\mathcal{M}_1, \mathcal{M}_2$ be models of $\mathcal{S}$. Prove that if $\mathcal{M}_1$ is isomorphic to $\mathcal{M}_2$ then $\mathrm{Th}_2(\mathcal{M}_1) = \mathrm{Th}_2(\mathcal{M}_2)$.
(*Hint:* by induction on the definition of wffs, show that for any wff A:

$$(\mathcal{M}_1, I_v) \models A \text{ iff } (\mathcal{M}_2, h \circ I_v) \models A ,$$

... where $h : D_1 \longrightarrow D_2$ is the isomorphism from M_1 to M_2, and $h \circ I_v(x_i) = h(I_v(x_i))$ for all variables x_i.)

Problem Set 2 Solutions

Mary Vogt
6.044J/18.423J
Problem Set 2
October 24, 1988

Problem 1.

a) If A is a recursively enumerable set, there exists a Turing machine M_A which accepts every element of A and loops on or rejects every element of $\overline{A}$. There is an equivalent Turing machine M_B for the set B. Construct a two-tape Turing machine, M_0, which copies one of its input tapes, x, to its other tape, then simulates the behavior of M_A on one of its input tapes and the behavior of M_B on its other input tape. If one of these simulations halts and accepts its input, M_0 also halts and accepts its input. If both of the simulations halt and reject their inputs, M_0 also halts and rejects its input. If both loop, M_0 loops. M_0 can be shown to be equivalent to a one-tape Turing machine, M_1. Since there is a Turing machine which halts and accepts its input if and only if the input is in $A \cup B$, and either loops or rejects if the input is not in $A \cup B$, $A \cup B$ is a recursively enumerable set.

b) Any set $A = \{a_1, a_2, ...\}$ in $\{\mathtt{a},\mathtt{b}\}^*$ is countable, so it can be expressed as a countable union of sets $A_1, A_2, ...$ where $A_j = \{a_j\}$. Each of these sets is decidable, thus recursively enumerable. However, A is not necessarily recursively enumerable.

This problem was discussed with Aaron Goodisman.

Mary Vogt
6.044J/18.423J
Problem Set 2
October 24, 1988

Problem 2.

a)

i. Prove: If S has a solution, then S' has an initialized solution.
Let $S = \{(\alpha_1, \beta_1), \ldots, (\alpha_k, \beta_k)\}$
$\exists \{i_1, \ldots, i_m\}$ such that
$\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$
$(cc)(\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m}) = (cc)(\beta_{i_1}\beta_{i_2}\ldots\beta_{i_m})$
$(c)(c\alpha_{i_1})(\alpha_{i_2}\ldots\alpha_{i_m}) = (cc)(\beta_{i_1}\beta_{i_2}\ldots\beta_{i_m})$
In S', c corresponds to cc, $c\alpha_{i_1}$ corresponds to β_{i_1}, and α_{i_j} corresponds to β_{i_j}, so that the above equation indicates that there is a solution to S'.

ii. Prove: If S' has an initialized solution, then S has a solution.
Let $S' = \{(\gamma_0, \delta_0), \ldots, (\gamma_2 k, \delta_n)\}$
$(\gamma_0, \delta_0) = (c, cc)$
$(\gamma_j, \delta_j) = (\alpha_j, \beta_j)$ for $1 \le j \le k$
$(\gamma_j, \delta_j) = (c\alpha_j, \beta_j)$ for $k < j \le 2k$
$\exists \{i_0, \ldots, i_m\}$ such that
$\gamma_{(i_0)}\gamma_{(i_1)}\ldots\gamma_{(i_m)} = \delta(i_0)\delta(i_1)\ldots\delta(i_m)$

... where by assumption $i_0 = 0$ (ie) the first word of the solution to S' is (c, cc).

then $(\gamma_{i_1}, \delta_{i_1})$ has to be of one of the two forms:

(c, cc) or $(c\alpha_i, \beta_i)$.

- Assume $(\gamma_{i_1}, \delta_{i_1}) = (c, cc)$. Then:

$\gamma_{i_0} \cdot \gamma_{i_1}$ would be cc
$\delta_{i_0} \cdot \delta_{i_1}$ would be $cc\,cc$

... so that the number of c's downstairs would exceed the number of c's upstairs by 2 at this point.

For any string s let us denote $|s|_c$ = number of c's in s.

With this notation:

$$|\gamma_{i_0}\cdot\gamma_{i_1}|_c = 2 \quad \text{whereas} \quad |\delta_{i_0}\cdot\delta_{i_1}|_c = 4.$$

$$\text{and} \quad |\gamma_{i_0}\cdot\gamma_{i_1}|_c = |\delta_{i_0}\cdot\delta_{i_1}|_c - 2. \tag{1}$$

As $\gamma_{i_0}\cdots\gamma_{i_m} = \delta_{i_0}\cdots\delta_{i_m}$ we have:

$$|\gamma_{i_0}\cdots\gamma_{i_m}|_c = |\delta_{i_0}\cdots\delta_{i_m}|_c. \tag{2}$$

Consider the first step r where:

$$|\gamma_{i_0}\cdots\gamma_{i_r}|_c \geq |\delta_{i_0}\cdots\delta_{i_r}|_c - 1$$

(ie)
$$\begin{cases} |\gamma_{i_0}\cdots\gamma_{i_r}|_c \geq |\delta_{i_0}\cdots\delta_{i_r}|_c - 1 & (3) \\ |\gamma_{i_0}\cdots\gamma_{i_{r-1}}|_c \leq |\delta_{i_0}\cdots\delta_{i_{r-1}}|_c - 2 & (4) \end{cases}$$

(Note that there is such an r verifying (3) and (4) because (1) and (2) are verified)

But this implies that $|\gamma_{i_r}|_c > |\delta_{i_r}|_c$ which is impossible.

Hence $\boxed{(\gamma_{i_1}, \delta_{i_1}) = (c\alpha_i, \beta_i)}$ for some i.

- So we have:

$$\gamma_{i_0}\cdot\gamma_{i_1} = c\,|\,c\alpha_i$$

$$\delta_{i_0}\cdot\delta_{i_1} = cc\,|\,\beta_i$$

* Assume that $|\gamma_{i_0}\cdot\gamma_{i_1}\cdots\gamma_{i_m}|_c = 2 = |\delta_{i_0}\cdots\delta_{i_m}|$.

then $\forall r \in \{2..m\}$ $(\gamma_{i_r}, \delta_{i_r})$ is of the form

(α_i, β_j) for some j and:

$$\gamma_{i_0} \cdots \gamma_{i_m} = c\,|c\alpha_{i_1}|\,\alpha_{i_2}|\cdots\alpha_{i_m}$$
$$= \delta_{i_0} \cdots \delta_{i_m} = c\,c\,|\beta_{i_1}|\,\beta_{i_2}|\cdots\beta_{i_m}.$$

So that we get a solution to S:

$$\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_m} = \beta_{i_1}\cdots\beta_{i_m}.$$

*) Assume then that $|\gamma_{i_0}\cdots\gamma_{i_m}|_c = |\delta_{i_0}\cdots\delta_{i_m}|_c > 2$.

then call r the first step at which:

$$|\gamma_{i_0}\cdots\gamma_{i_r}|_c > 2.$$

We must have (think about it!):

$\gamma_{i_0}\cdots\gamma_{i_{r-1}} = \delta_{i_0}\cdots\delta_{i_{r-1}}$ and we are back in the case above studied!

DONE!

Problem 2

b) Consider an instance of the IPCP:

$S' = \left((\alpha_1, \beta_1), (\alpha_2, \beta_2) \dots (\alpha_n, \beta_n) \right)$.

For any string α_i or β_i denote $\widetilde{\alpha_i}$ and $\widetilde{\beta_i}$ the string obtained by intertwining a star in between each of the alphabet letters of α_i or β_i.

Ex: if $\alpha = aaba$ then $\widetilde{\alpha} = a*a*b*a$.

Construct a PCP instance S consisting of the pair:

$(***\widetilde{\alpha_1}, ***\widetilde{\beta_1}*)$, the pairs $(*\widetilde{\alpha_i}, \widetilde{\beta_i}*)$ for $i = 1 \dots n$

and the pairs $(*\widetilde{\alpha_i}*, \widetilde{\beta_i}*)$ for $i = 1 \dots n$.

Claim: S' has an initialized solution iff S has a solution.

proof: $\Rightarrow$ if $\alpha_1 \alpha_{i_2} \dots \alpha_{i_m} = \beta_1 \beta_{i_2} \dots \beta_{i_m}$

then $***\widetilde{\alpha_1} | *\widetilde{\alpha_{i_2}} | *\widetilde{\alpha_{i_3}} | \dots | *\widetilde{\alpha_{i_{m-1}}} | *\widetilde{\alpha_{i_m}}* =$

$***\widetilde{\beta_1} * | \widetilde{\beta_{i_2}} * | \widetilde{\beta_{i_3}} * | \dots | \widetilde{\beta_{i_{m-1}}} * | \widetilde{\beta_{i_m}} *$

$\Leftarrow$ Consider a solution to S and (x, y) its first term.

By looking at the three different types of words in S we see that the only possibility is $(x, y) = (***\widetilde{\alpha_1}, ***\widetilde{\beta_1}*)$

Hence if we remove **all** the additional stars from the solution of S we get a solution for S'.

Note: In every solution of S: $(x_1, y_1) \dots (x_m, y_m)$,

x_m is of the type $*\widetilde{\alpha_{i_m}}*$. Conversely, an element $*\widetilde{\alpha}*$ can only be at the end of a sequence $x_1 \dots x_m$.

Corollary $IPCP \leq_m PCP$.

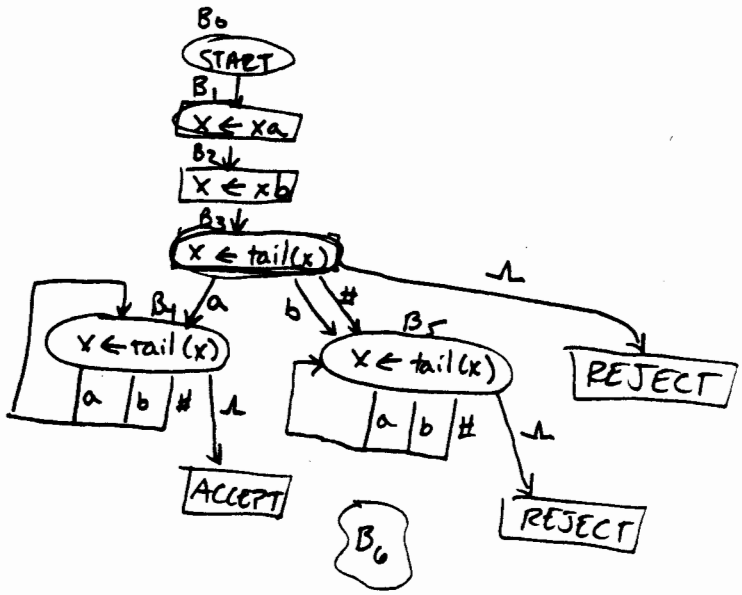
Mary Vogt
6.044J/18.423J
Problem Set 2
October 24, 1988

Problem 3.

a) Start : $x = \Lambda$
$x \leftarrow xa$: $x = a$
$x \leftarrow xb$: $x = ab$
$x \leftarrow \text{tail}(x)$: $x = b$
$\text{head}(x) = a$, so ACCEPT.

b) The pairs of S are:

1 $(B_0, B_0 * B_1 *)$
2 $(*B_1, a * B_2 *)$
3 $(*B_2, b * B_3 *)$
4 $(*B_3 * a, B_4 *)$
5 $(*B_3 * b, B_5 *)$
6 $(*B_3 * \#, B_5 *)$
7 $(*B_3 * B_6, B_6)$
8 $(*B_4 * a, B_4 *)$
9 $(*B_4 * b, B_4 *)$
10 $(*B_4 * \#, B_4 *)$
11 $(*B_4 * B_6, B_6)$
12 $(*B_5 * a, B_5 *)$
13 $(*B_5 * b, B_5 *)$
14 $(*B_5 * \#, B_5 *)$
15 $(*B_5 * B_6, B_6)$
16 $(*a, a *)$
17 $(*b, b *)$
18 $(*\#, \# *)$



Aaron Goodisman 6.044J/18.423J Problem Set 2 Problem 3 24 October 1988

3(c).

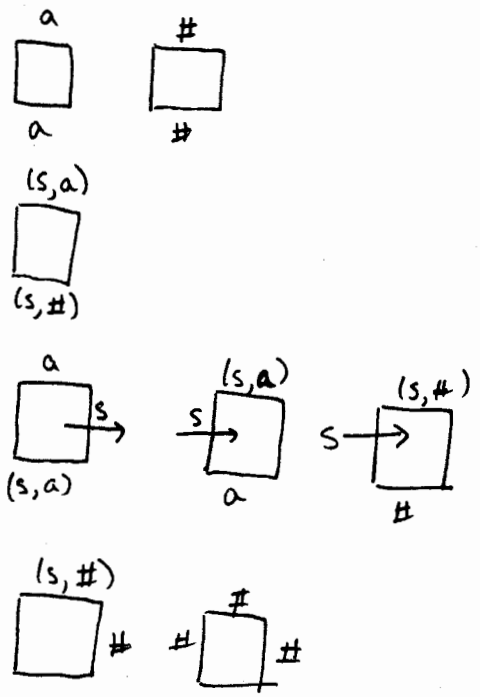
$\{1, 2, 16, 3, 16, 17, 4, 17, 9, 11\}$

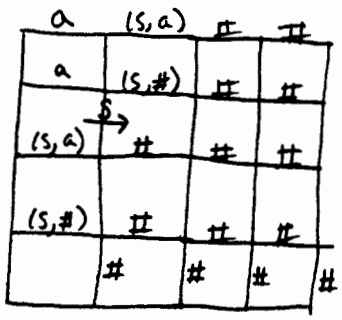
$B_0 * B_1 * a * B_2 * a * b * B_3 * a * b * B_4 * b * B_4 * B_6$

$B_0 * B_1 * a * B_2 * a * b * B_3 * a * b * B_4 * b * B_4 * B_6$

Mary Vogt
6.044J/18.423J
Problem Set 2
October 24, 1988

Problem 4. The Turing machine $M = (\{s\}, \{a, \#\}, \delta, s), \delta(s, \#) = (s, a), \delta(s, a) = (s, R)$ is associated with the following set of tiles:





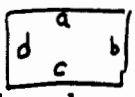
This machine writes an a onto any blank square of its input tape and leaves alone any square which starts with an a in it. The machine never halts, so the tiling it generates never halts. The first four rows of this tiling are above.

This problem was discussed with Aaron Goodisman

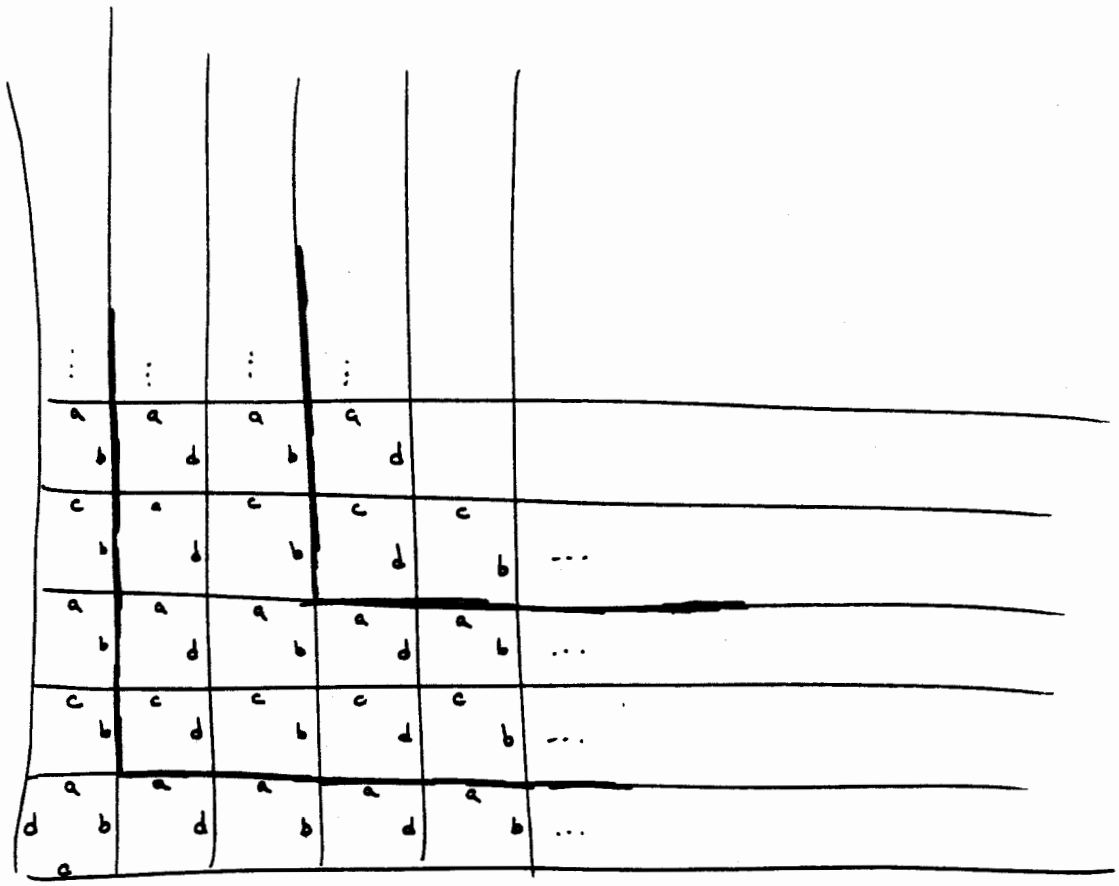
This problem was discussed with Mary Vogt.

Problem 5:

5(6.5.7).

Given any tile 

, the ability to rotate and flip gives all possible arrangements of the colors a, b, c, d in which a is opposite c, and b is opposite d. Any single tile can be used to tile the entire first quadrant no matter in what orientation it is put at the origin. Note that the section between the darkened lines can be repeated to fill the quadrant. Since this holds for any one tile, it certainly holds for any non-empty set of tiles.



Supplement for Problem Set 3

The following **practice problems** and solutions will be useful as a guide to solving the isomorphism problem (Problem 3) in the current Problem Set 3, and also helpful to study for the second **quiz next Wednesday, Nov. 9.**

You may notice a minor difference in the definitions of homomorphism given here and in Problem Set 3. Both are correct, but the one below is a bit more explicit about handling individual constants. Finally, at the end of this supplement, note an addition to the hint which accompanied problem 3 on Problem Set 3 (handout 13).

Definition 1. Let $\mathcal{I} = ((D, \mathcal{I}_c), \mathcal{I}_v)$ and $\mathcal{I}' = ((D', \mathcal{I}'_c), \mathcal{I}'_v)$ be interpretations over a common second-order predicate calculus signature $\mathcal{S}$.

A function $\eta : D \to D'$ is a *homomorphism between $\mathcal{I}$ and $\mathcal{I}'$*, written $\eta : \mathcal{I} \to \mathcal{I}'$, iff

(1) for all individual constants c in $\mathcal{S}$,

$$\eta(\mathcal{I}_c(c)) = \mathcal{I}'_c(c) \ ,$$

and similarly for all first-order variables x (replacing "$c, \mathcal{I}_c, \mathcal{I}'_c$" by "$x, \mathcal{I}_v, \mathcal{I}'_v$", respectively),

(2) for $n > 0$ and all n-ary function constants f in $\mathcal{S}$ and for all $d_1, \ldots, d_n \in D$,

$$\eta(\mathcal{I}_c(f)(d_1, \ldots, d_n)) = \mathcal{I}'_c(f)(\eta(d_1), \ldots, \eta(d_n)) \ ,$$

and similarly for all n-ary function variables F,

(3) for $n > 0$ and all n-ary predicate constants p in $\mathcal{S}$ and for all $d_1, \ldots, d_n \in D$,

$$\mathcal{I}_c(p)(d_1, \ldots, d_n) = \mathcal{I}'_c(p)(\eta(d_1), \ldots \eta(d_n)) \ ,$$

and similarly for all n-ary predicate variables P.

The homomorphism is *onto* iff $\eta : D \to D'$ is onto, and in this case $\mathcal{I}'$ is said to be a *homomorphic image* of $\mathcal{I}$.

Proof:

Claim 1. $\eta(\mathcal{I}(t)) = \mathcal{I}'(t)$ for all terms t over $\mathcal{S}$.

By induction on the definition of terms:

(1) If t is an individual constant $c \in \mathcal{S}$, we have

$$\begin{aligned}\eta(\mathcal{I}(c)) =& \eta(\mathcal{I}_c(c)) \quad \text{(by the definition of } \mathcal{I}(c)\text{)}\\ =& \mathcal{I}'_c(c) \quad \text{(by part 1 of Definition 1)}\\ =& \mathcal{I}'(c) \quad \text{(by the definition of } \mathcal{I}'(c)\text{)}\,,\end{aligned}$$

and likewise if t is an individual variable.

(2) If t is of the form $f(t_1, \ldots, t_n)$ where $n > 0$ and f is an n-ary function constant in $\mathcal{S}$, then

$$\begin{aligned}\eta(\mathcal{I}(f(t_1,\ldots,t_n))) =& \eta(I_c(f)(\mathcal{I}(t_1),\ldots,\mathcal{I}(t_n))) \quad \text{(by the definition of } \mathcal{I}(f(t_1,\ldots,t_n))\\ =& \mathcal{I}'_c(f)(\eta(\mathcal{I}(t_1)),\ldots,\eta(\mathcal{I}(t_n)))) \quad \text{(by part 2 of Definition 1)}\\ =& \mathcal{I}'_c(f)(\mathcal{I}'(t_1),\ldots,\mathcal{I}'(t_n))) \quad \text{(by induction)}\\ =& \mathcal{I}'(f(t_1,\ldots,t_n)) \quad \text{(by the definition of } \mathcal{I}'(f(t_1,\ldots,t_n)))\,,\end{aligned}$$

and likewise if t is of the form $F(t_1, \ldots, t_n)$.

This establishes the claim.

We can now show that, *for all $\mathcal{I}$ and homomorphic images $\mathcal{I}'$ of $\mathcal{I}$*, it is the case that $\mathcal{I}(A) = \mathcal{I}'(A)$. The proof is by induction on the definition of first-order wffs A without equality:

(1) If A is an atomic formula of the form $p(t_1, \ldots, t_n)$, we have

$$\begin{aligned}\mathcal{I}(A)) =& \mathcal{I}_c(p)(\mathcal{I}(t_1),\ldots,\mathcal{I}(t_n)) \quad \text{(by the definition of } \mathcal{I}(A)\text{)}\\ =& \mathcal{I}'_c(p)(\eta(\mathcal{I}(t_1)),\ldots,\eta(\mathcal{I}(t_n))) \quad \text{(by part 3 of Definition 1)}\\ =& \mathcal{I}'_c(p)(\mathcal{I}'(t_1),\ldots,\mathcal{I}'(t_n)) \quad \text{(by the claim)}\\ =& \mathcal{I}'(A) \quad \text{(by the definition of } \mathcal{I}'(A)\text{)}\,.\end{aligned}$$

These are the only atomic formulas, since A is first-order without equality.

(2) If A is of the form $A_1 \wedge A_2$, then

$$\begin{aligned}\mathcal{I}(A) =& \mathcal{I}(A_1) \wedge \mathcal{I}(A_2) \quad \text{(by the definition of } \mathcal{I}(A)\text{)}\\ =& \mathcal{I}'(A_1) \wedge \mathcal{I}'(A_2) \quad \text{(by induction)}\\ =& \mathcal{I}'(A) \quad \text{(by the definition of } \mathcal{I}'(A)\text{)}\,,\end{aligned}$$

and likewise if A is of the form $A_1 \vee A_2$ or $\neg A_1$.

(3) If A is of the form $\exists x B$, then, noting that $\mathcal{I}'[x \mapsto \eta(d)]$ is a homomorphic image of $\mathcal{I}[x \mapsto d]$, we have:

$$\begin{aligned}\mathcal{I}(A) &= \bigvee_{d \in D} \mathcal{I}[x \mapsto d](B) \quad \text{(by the definition of } \mathcal{I}(A)) \\ &= \bigvee_{d \in D} \mathcal{I}'[x \mapsto \eta(d)](B) \quad \text{(by induction)} \\ &= \bigvee_{d' \in D'} \mathcal{I}'[x \mapsto d'](B) \quad \text{(since } \eta \text{ is onto)} \\ &= \mathcal{I}'(A) \quad \text{(by the definition of } \mathcal{I}'(A)) \,,\end{aligned}$$

and likewise if A is of the form $\forall x B$. We needn't consider quantification over second-order variables since A is first-order.

■

Definition 2. A *model* $\mathcal{M}'$ is a homomorphic image of a model $\mathcal{M}$ iff there is a pair of valuations I_v and I'_v such that $(\mathcal{M}', \mathcal{I}'_v)$ is a homomorphic image of $(\mathcal{M}, \mathcal{I}_v)$.

A homomorphism $\eta : \mathcal{I} \to \mathcal{I}'$ is an *isomorphism between* $\mathcal{I}$ *and* $\mathcal{I}'$ iff the function $\eta : D \to D'$ is a bijection (*i.e.*, 1-1 and onto). $\mathcal{I}$ and $\mathcal{I}'$ are *isomorphic*, written $\mathcal{I} \cong \mathcal{I}'$, if there is an isomorphism between $\mathcal{I}$ and $\mathcal{I}'$. Two models $\mathcal{M}, \mathcal{M}'$ are isomorphic iff there is a pair of valuations I_v and I'_v such that $(\mathcal{M}', \mathcal{I}'_v) \cong (\mathcal{M}, \mathcal{I}_v)$.

Problem 2. Let A be the conjunction of the following first-order formulas over the signature $\mathcal{S} = \{\mathbf{0}, \boldsymbol{suc}, +\}$:

- $\forall x \forall y[(\boldsymbol{suc}(x) = \boldsymbol{suc}(y)) \supset (x = y)]$ ($\boldsymbol{suc}$ is 1-1)
- $\forall x(\boldsymbol{suc}(x) \neq \mathbf{0})$ ($\mathbf{0}$ is not a successor)
- $\forall x \forall y[(x + \mathbf{0} = x) \wedge (x + \boldsymbol{suc}(y) = \boldsymbol{suc}(x + y))]$ (inductive definition of $+$)

(a) Give an example of a model of A such that the interpretation of $+$ is *not* a commutative operation.

Proof: Let D consist of two separate copies—an "$\mathtt{a}$-copy" and a "$\mathtt{b}$-copy"—of the non-negative integers, namely, $D = \{(\mathtt{a}, n) \mid n \geq 0\} \cup \{(\mathtt{b}, n) \mid n \geq 0\}$. Define $\mathcal{I}_c(\boldsymbol{suc})$ to act like the successor function on each copy, namely, $\mathcal{I}(\boldsymbol{suc})((d, n)) = (d, n+1)$ for $d \in \{\mathtt{a}, \mathtt{b}\}, n \geq 0$. Let $\mathcal{I}_c(\mathbf{0}) = (\mathtt{a}, 0)$. Finally, let $\mathcal{I}_c(+)$ be the function $+$ on D where $(d, n) + (\mathtt{a}, m) = (d, n + m)$ and $(d, n) + (\mathtt{b}, m) = (\mathtt{b}, m)$. Then it is easy to check that $(D, \mathcal{I}_c) \models A$ but we have

$$(\mathtt{b}, 0) + (\mathtt{a}, 1) = (\mathtt{b}, 1) \neq (\mathtt{b}, 0) = (\mathtt{a}, 1) + (\mathtt{b}, 0)$$

Problem 1.

(a) Let $\mathcal{S}_0 = \{\boldsymbol{c}_1, \boldsymbol{c}_2\}$ where $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are individual constant symbols. Describe a pair of interpretations $\mathcal{I}$ and $\mathcal{I}'$ of $\mathcal{S}_0$ such that $\mathcal{I}'$ is a homomorphic image of $\mathcal{I}$ and

$$\mathcal{I} \models \neg(\boldsymbol{c}_1 = \boldsymbol{c}_2) \quad \text{but} \quad \mathcal{I}' \not\models \neg(\boldsymbol{c}_1 = \boldsymbol{c}_2)\,.$$

Proof:

Let $\mathcal{I} = ((\{1,2\}, \mathcal{I}_c), \mathcal{I}_v)$ and $\mathcal{I}' = ((\{0\}, \mathcal{I}'_c), \mathcal{I}'_v)$ where $\mathcal{I}_c(\boldsymbol{c}_1) = 1$ and $\mathcal{I}_c(\boldsymbol{c}_2) = 2$ and $\mathcal{I}'_c(\boldsymbol{c}_1) = \mathcal{I}'_c(\boldsymbol{c}_2) = 0$. Let $\mathcal{I}_v$ map each function variable F to the constant function 1 on D with arity equal to that of F, and let $\mathcal{I}_v$ map each predicate variable to the constant predicate `true` on D; likewise $\mathcal{I}'_v$ maps function symbols to the constant functions 0 and predicate symbols to the `true` predicates on D'.

The map $\eta : \{1,2\} \to \{0\}$ defined by $\eta(1) = \eta(2) = 0$ is a homomorphism from $\mathcal{I}$ onto $\mathcal{I}'$, and

$$\mathcal{I} \models \neg(\boldsymbol{c}_1 = \boldsymbol{c}_2) \quad \text{but} \quad \mathcal{I}' \models (\boldsymbol{c}_1 = \boldsymbol{c}_2)\,.$$

■

(b) Describe a pair of interpretations $\mathcal{I}$ and $\mathcal{I}'$ of the empty signature (*i.e.*, no non-logical symbols) and a wff B *without the equality symbol* such that $\mathcal{I}'$ is a homomorphic image of $\mathcal{I}$ and

$$\mathcal{I} \models B \quad \text{but} \quad \mathcal{I}' \not\models B\,.$$

Proof: Let $D = \{1,2\}$ and $D' = \{0\}$. No need to mention $\mathcal{I}_c$ or $\mathcal{I}'_c$ since the signature is empty. Let $\mathcal{I}_v, \mathcal{I}'_v$ map functions and predicates to constant functions and predicates, and as in the previous example. Then D satisfies $\exists x \exists y . x \neq y$ and D' does not. So this wff would serve for B except that it has an equality symbol in it. But $x = y$ is equivalent to $\forall P . P(x) \equiv P(y)$, so let B be

$$\exists x \exists y \neg [\forall P.\, P(x) \equiv P(y)]$$

■

(c) Suppose $\mathcal{I}'$ is a homomorphic image of $\mathcal{I}$. Prove that for any *first-order wff A without equality* (over *any* signature $\mathcal{S}$),

$$\mathcal{I}(A) = \mathcal{I}'(A)\,.$$

(*Hint*: Use induction on the definition of A. Begin by using induction on the definition of terms to show that $\eta(\mathcal{I}(t)) = \mathcal{I}'(t)$ for any term t).

so $\mathcal{I}_c(+)$ is not commutative.

■

(b) (*Optional*: Same thing when the formula

$$\forall x[(x \neq \mathbf{0}) \supset \exists y(\boldsymbol{suc}(y) = x)]$$

(every non-zero element has a successor) is conjuncted into A.

(c) Let $\mathcal{A}$ be the non-negative integers with constant 0 and the operations of successor and addition. Write down a second-order predicate calculus formula A_{arith} over the signature $\mathcal{S}$ with the property that for any model $\mathcal{M}$ of $\mathcal{S}$,

$$\mathcal{M} \models A_{\text{arith}} \wedge A \quad \text{iff} \quad \mathcal{M} \cong \mathcal{A}\,.$$

Briefly explain why your formula works. (*Hint*: This is similar to, but easier than, the construction of the formula $w_{\text{binary-strings-honestly}}$ done in class.)

Problem 3. (Added hint for Problem Set 3, problem 3.) The definition of $h \circ I_v$ does not cover the case of function variables of positive arity or the case of predicate variables. For example, if F is a unary function variable, we need to define $(h \circ \mathcal{I}_v)(F)$ to be the function $\lambda d' \in D'.h(\mathcal{I}_v(F)(h^{-1}(d')))$ from D' to D'.

Formula about Binary Strings

Handout: 16

We want a formula L with free variables x and u over the signature $\{\cdot, eqlength\}$ such that $L(x,u)$ means "$u = left_0(x)$" in the binary-strings model. Here the function $left_0 : \{a,b\}^* \to \{a,b\}^*$ was defined in Problem Set 1.

Note that $u = left_0(x)$ iff x is of the form $u'abz$ where u' is obtained from u by replacing "a" by "aa" and "b" by "bb".

Now u' is obtained in this way from u iff the following formula $D(u,u')$ is true:

$$(eqlength(u,uu) \wedge eqlength(u',u'\cdot u'))\vee \qquad \%_0(\text{means } u=u'=\lambda)$$

$$\forall v_1, v_2, c.\ (eqlength(c,a) \wedge u = v_1 \cdot c \cdot v_2) \supset \qquad \%_0(|c|=1)\ \ \%_0(v_1 \text{ is a prefix of } u)$$

$$(\exists v_1', v_2'.\ eqlength(v_1', v_1\cdot v_1) \wedge u' = v_1'\cdot c\cdot c\cdot v_2') \qquad \%_0(|v_1'| = 2*|v_1|)$$

So now L is:

$$\exists u', z.\ u = u'\cdot ab\cdot z \wedge D$$

Notes on the course of Nov 2, 1988

The theorem we want to establish is:

Thm: For all r.e set R : $R \leq_m Th(\text{binary-strings})$.

Note: actually the result could even be more precise:
$R \leq_m Th_1(\text{binary-strings without eq/length})$
(ie: 1st order theory of binary-strings of signature reduced to $\{\bullet\}$).

proof: We begin noticing that for any finite alphabet Σ:
$Th(\text{strings over } \Sigma) \leq_m Th(\text{binary-strings})$. (*)
This comes from the fact that any letter of a finite alphabet can be encoded into a certain string of $\{a,b\}^*$.

Then consider some r.e set R. Then $R = \text{dom}(M)$ for some Turing machine M. Let $\Sigma_m = \{\text{work type symbols of } M\}$
$Q_m = \{\text{flowchart boxes of } M\}$.
(Q_m is the equivalent of $\{B_1 \dots B_m\}$ of Manna's proof of Th 1-9 p61).
We will prove that:
$R \leq_m Th_1(\text{strings over } \Sigma_m \cup Q_m \cup \{\#\})$.
This result along with the fact (*) will establish our claim.

the many-one reducibility will take the form (that we are going to make precise):
$\alpha \in R$ iff M halts on input α

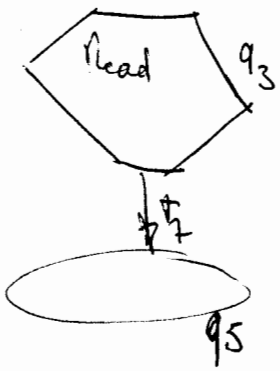
- if there is an acceptable string describing precisely the evolution of machine M on input x and that will indicate that the machine has halted.

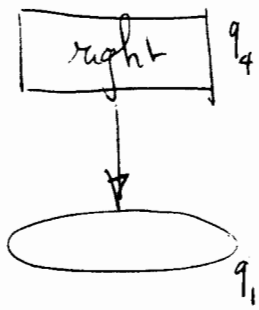
For this purpose we are going to encode over our alphabet all the possible 1-step evolution of the machine M. (A bit in the same way as we did when we reduced the halting problem to the tiling problem in homework 2).

A configuration of the machine M at a given time is described by:
$u_1 s c u_2$ where $s \in Q_m$, $c \in \Sigma_m$, u_1 and $u_2 \in \Sigma_m^*$.
This means that the non empty part of the machine tape is $u_1 c u_2$
- the head of the machine is pointing at c
and that the next move will be monitored by the flowchart box s (on input c).

Hence, in a one step move, we go from a configuration u to another configuration v.
The history of the action of M on input x will be a succession of successive configurations delimited by "#" signs.

Two such configurations u and v will be checked to be successive if the predicate $Next_m(u,v)$ is true. This means that $Next_m(\ ,\ )$ is the disjunction of all the possible successive configurations. To be a bit more precise let us consider the two following examples of possible one-step mutations.





where $q_4, q_1, q_3 \in Q_m$.

then the corresponding part of $Next(u, v)$ will be (described in metalanguage):

"

$u = u_1 s c u_2$ where $s \in Q_m$, $c \in \Sigma_m$, $u_1, u_2 \in \Sigma_m^*$

... and [if $\begin{cases} s = q_3 \\ c = \sharp_7 \end{cases}$ then $v = u_1 q_5 c u_2$.]

and [if $(s = q_4$ and $u_2 \neq \Lambda)$ then $v = u_1 c q_1 u_2$]

and [if $(s = q_4$ and $u_2 = \Lambda)$ then $v = u_1 c q_1 \Delta$]

"

↑ blank symbol

If we now want to go down from metalanguage to real predicate calculus syntax, we see that everything can be expressed as binary strings and stuff. As example:

- $(u_2 \neq \Lambda)$ can be expressed as $\neg(u_2 = u_2 \cdot u_2)$
- $(c \in \Sigma_m)$ ——— $(c = \sharp_1) \vee (c = \sharp_2) \cdots (c = \sharp_n)$
- $(u_1 \in \Sigma_m^*)$ ——— $\forall u_1', u_2', c \left(u_1 = u_1' \cdot c \cdot u_2' \wedge [eq\,length(c, \ast \ldots \Rightarrow (c \in \Sigma_m)\right.$
- (u is of the form $u_1 s c u_2$) can be expressed as $\exists u_1, s, c, u_2$ s.t. $u = u_1 s c u_2$

Having described $Next_m(\cdot,\cdot)$ let's go back to establishing that $R \leq_m Th$ (halting)

$\alpha \in R$ iff $\exists \zeta \in (\Sigma_m \cup Q_m \cup \{\sharp, \$\})^*$ { $\exists u, \zeta' [\zeta = \$ q_{start} \cdot \alpha \$ \cdot u \$ \zeta' \wedge next(q_{start}, \alpha)$

((el): "ζ starts OK.")

$$\wedge \quad (\exists u, u', \zeta')\Big[\ \zeta = \zeta' \not\# u \not\# u' \not\# \quad \wedge \ \text{next}(u, u') \quad \wedge \ (\exists u'', u''' \quad u' = u'' q_{\text{halt}} u''')\ \Big]$$

(M: "ζ ends on")

$$\wedge \quad \forall \zeta_1, \zeta_2, \zeta_3, \zeta_4 \Big[\ \zeta = \zeta_1 \not\# \zeta_2 \not\# \zeta_3 \not\# \zeta_4 \quad \wedge \ (\zeta_2, \zeta_3 \in (\Sigma_m \cup Q_m)^*) \Rightarrow \text{Next}(\zeta_2, \zeta_3)\Big] \Big\}.$$

■

Quiz 2

Instructions. Do all four (4) problems; a total of 100 points is allocated as shown on each problem. This exam is *open book*. You have two hours. Good luck.

Problem 1 [15 points]. Consider the following sequence of pairs of words:

$$S = \langle (ab, a), (b, babab), (aaab, ab), (baab, bbb) \rangle$$

(a) [10 points] Regarding S as an instance of the *Initialized* PCP (*cf.* Problem Set 2, Problem 2, Handout 8), show that it has no solution.

(b) [5 points] Regarding S as an instance of the usual PCP, show that it has a solution.

Problem 2 [25 points]. Write down a closed second-order formula which is true in precisely those models whose domains have an odd number of elements. (*Hint*: Use and/or modify any of the formulas described in class and text which mean "the domain is finite".)

Problem 3 [30 points]. Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be an alphabet, and let p be an n-ary predicate on Σ^*. The predicate p is "first-order concatenation-definable"—*definable* for short—iff there is a first-order wff w_p whose free variables are precisely $x_1, \ldots, x_n$, and whose nonlogical symbols include only $\cdot, \sigma_1, \ldots, \sigma_k$ (not eqlngth!), such that under the model of strings over Σ under concatenation, w_p means that $p(x_1, \ldots, x_n)$ holds.

For example, if p_1 is the unary predicate "is of length one," we saw in class that w_{p_1} could be

$$(x_1 = \sigma_1) \vee \ldots \vee (x_1 = \sigma_k).$$

(a) [5 points] Let $\Sigma_1 \subseteq \Sigma$ and p_{Σ_1} be the unary predicate "is a member of Σ_1^*". Exhibit a wff $w_{p_{\Sigma_1}}$ which shows that p_{Σ_1} is definable.

(b) [10 points] Let c be a symbol in $\Sigma - \Sigma_1$, and let pair : $\Sigma_1^* \times \Sigma_1^* \to (\Sigma_1 \cup \{c\})^*$ be defined by

$$\text{pair}(x, y) = xcy\ .$$

Show that the binary predicate which means "x_2 is in the range of pair and $x_1 = \text{left}(x_2)$", is definable.

(over)

(c) [15 points] The proof in class (*cf.* Handout 17) showing that $R \leq_{\mathrm{m}} \mathrm{Th}_1(\text{strings over } \Sigma_R)$ for all r.e. sets R, actually proved something stronger than was stated. First, no essential use of eqlngth was made, and second, the reduction amounted to constructing a wff which meant "$x_1 \in R$". In short, we really proved:

Lemma. For every r.e. set $R \subseteq \Sigma_2^*$, there exists $\Sigma \supseteq \Sigma_2$ such that the predicate "is in R" is definable.

Using this lemma, prove that the binary predicate "$x_1, x_2 \in \Sigma_1^*$ and $|x_1| = |x_2|$" is definable. (*Hint*: Let $\Sigma_2 = \Sigma_1 \cup \{c\}$, and $R = \{pair(x,y) \mid x, y \in \Sigma_1^* \text{ and } |x| = |y|\}$.)

Problem 4 [30 points]. Consider two models $\mathcal{M}_1 = \langle \{0,1,2,3\}, (\text{square mod } 4), 3\rangle$ and $\mathcal{M}_2 = \langle \{0,3\}, (\text{times3 mod } 6), 0\rangle$, where $(\text{square mod } 4)(n) = n^2 \bmod 4$ and $(\text{times3 mod } 6)(n) = 3n \bmod 6$.

(a) [7 points] Describe a function $\eta : \{0,1,2,3\} \to \{0,3\}$ which shows that $\mathcal{M}_2$ is a homomorphic image of $\mathcal{M}_1$. No explanation or proof is required.

(b) [8 points] Remember that our definition of homomorphism (Problem Set 3 and its Supplement—Handouts 13 and 15) is based on models *together with* valuations of the variables. So to show that $\mathcal{M}_2$ is an homomorphic image of $\mathcal{M}_1$ under η, it is necessary to show that there are valuations $\mathcal{I}_{1,v}$ and $\mathcal{I}_{2,v}$ of all the first- and second-order variables such that $(\mathcal{M}_2, \mathcal{I}_{2,v})$ is an image of $(\mathcal{M}_1, \mathcal{I}_{1,v})$ under η. Describe two such valuations $\mathcal{I}_{1,v}$ and $\mathcal{I}_{2,v}$. Again, no explanation or proof is required.

(c) [15 points] Explain why if $\mathcal{I}_{1,v}(P)$ is the "equals three" predicate on $\{0,1,2,3\}$ for some unary predicate variable P, then there is *no* valuation $\mathcal{I}_{2,v}$ such that $(\mathcal{M}_2, \mathcal{I}_{2,v})$ is an image of $(\mathcal{M}_1, \mathcal{I}_{1,v})$ under *any* homomorphic mapping.

END OF QUIZ

Solutions to Quiz 2

This version supercedes an earlier version dated 14 November.

Problem 1 [15 points]. Consider the following sequence of pairs of words:

$$S = \langle (ab, a), (b, babab), (aaab, ab), (baab, bbb) \rangle$$

(a) [10 points] Regarding S as an instance of the *Initialized* PCP (*cf.* Problem Set 2, Problem 2, Handout 8), show that it has no solution.

Answer: The first pair of words of the IPCP has to be (ab, a). Hence the second word of the second pair has to begin with a b, which happens only in the second and in the fourth word of the instance. In the first case, we get $(abb, ababa)$ which results in a conflict in the 3rd position. In the second case we get $(abbaab, bbbb)$ which results in a conflict in the fourth position.

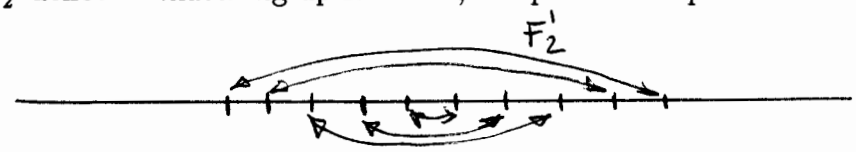
(b) [5 points] Regarding S as an instance of the usual PCP, show that it has a solution.

Answer: $\begin{matrix} bababaaab \\ bababaaab \end{matrix}$ is a solution obtained with the successive concatenation of the 2nd, 1st, 1st and 3rd words.

Problem 2 [25 points]. Write down a closed second-order formula which is true in precisely those models whose domains have an odd number of elements. (*Hint*: Use and/or modify any of the formulas described in class and text which mean "the domain is finite".)

Answer: We will begin by writing our formulas in mixed English-mathematical jargon before working down towards giving a predicate-calculus expression of them.

- Recall first, that we saw in class a wff that we will call *Finite*, meaning "the domain is finite".
- Then, a finite set D "is of even cadinality" iff "it is possible to partition it into sets of same cardinality" iff "every element can be matched up with exactly one other one".
  Letting F_2^1 denote a matching-up function, the previous expression translates into



$$\ldots \text{iff } \text{“}\exists F_2^1 \left(\left(\forall x_1 \; F_2^1(F_2^1(x_1)) = x_1 \right) \wedge \neg \left(\exists x_2 F_2^1(x_2) = x_2 \right) \right) \text{”}.$$

So a solution to our problem is to say that the domain is finite and *not* of even cardinality:

$$\mathit{Finite} \wedge \neg \left(\exists F_2^1 \ \left(\left(\forall x_1 \ F_2^1(F_2^1(x_1)) = x_1\right) \ \wedge \ \neg \left(\exists x_2 F_2^1(x_2) = x_2\right)\right)\right).$$

There are many different other wffs expressing the same fact, of course.

Problem 3 [30 points]. Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be an alphabet, and let p be an n-ary predicate on Σ^*. The predicate p is "first-order concatenation-definable"—*definable* for short—iff there is a first-order wff w_p whose free variables are precisely $x_1, \ldots, x_n$, and whose nonlogical symbols include only $\cdot, \sigma_1, \ldots, \sigma_k$ (not eqlngth!), such that under the model of strings over Σ under concatenation, w_p means that $p(x_1, \ldots, x_n)$ holds.

For example, if p_1 is the unary predicate "is of length one," we saw in class that w_{p_1} could be

$$(x_1 = \sigma_1) \vee \ldots \vee (x_1 = \sigma_k).$$

(a) [5 points] Let $\Sigma_1 \subseteq \Sigma$ and p_{Σ_1} be the unary predicate "is a member of Σ_1^*". Exhibit a wff $w_{p_{\Sigma_1}}$ which shows that p_{Σ_1} is definable.

Answer: Assume $\Sigma_1 = \{\sigma_{i_1} \ldots \sigma_{i_r}\}$ for some known $i_1 \ldots i_r \in \{1 \ldots k\}$. "$p_{\Sigma_1}$" iff (by definition) "$x_1$ is a member of Σ_1^*" iff "$\forall z_1 \forall z_2 \forall c\,(x_1 = z_1 \cdot c \cdot z_2 \ \wedge \ |c| = 1) \supset c \in \Sigma_1$" iff

$$\text{"}\forall z_1 \forall z_2 \forall c \ (x_1 = z_1 \cdot c \cdot z_2 \ \wedge \ w_{p_1}(c)) \supset ((c = \sigma_{i_1}) \vee \ldots \vee (c = \sigma_{i_r}))\text{"},$$

where, $w_{p_1}(c)$ is the result of the substitution of c for x_1 in w_{p_1}, namely, $[w_{p_1}]^c_{x_1}$ (see Handout 11, Definition 5).

(b) [10 points] Let c be a symbol in $\Sigma - \Sigma_1$, and let pair : $\Sigma_1^* \times \Sigma_1^* \to (\Sigma_1 \cup \{c\})^*$ be defined by

$$\text{pair}(x, y) = xcy \ .$$

Show that the binary predicate which means "x_2 is in the range of pair and $x_1 = \text{left}(x_2)$", is definable.

Answer:

$$[x_1 = \text{left}(x_2)] \quad \text{iff} \quad [p_{\Sigma_1}(x_1) \wedge (\exists y \ (p_{\Sigma_1}(y) \wedge x_2 = x_1 \cdot c \cdot y))].$$

(c) [15 points] The proof in class (*cf.* Handout 17) showing that $R \leq_m \text{Th}_1(\text{strings over } \Sigma_R)$ for all r.e. sets R, actually proved something stronger than was stated. First, no essential use of eqlngth was made, and second, the reduction amounted to constructing a wff which meant "$x_1 \in R$". In short, we really proved:

Lemma. For every r.e. set $R \subseteq \Sigma_2^*$, there exists $\Sigma \supseteq \Sigma_2$ such that the predicate "is in R" is definable.

Using this lemma, prove that the binary predicate "$x_1, x_2 \in \Sigma_1^*$ and $|x_1| = |x_2|$" is definable. (*Hint*: Let $\Sigma_2 = \Sigma_1 \cup \{c\}$, and $R = \{pair(x,y) \mid x, y \in \Sigma_1^*$ and $|x| = |y|\}$.)

Answer: R is obviously r.e. From the lemma we then know that "is in R" is definable. Call w_R the associated wff s.t. "x is in R" iff "$w_R(x)$". Then "$x_1, x_2 \in \Sigma_1^*$ and $|x_1| = |x_2|$" iff "$w_R(x_1 \cdot c \cdot x_2)$" which is the desired wff (Note once more that $w_R(x_1 \cdot c \cdot x_2)$ stands for what should really be $[w_R]_{x_1}^{(x_1 \cdot c \cdot x_2)}$).

Problem 4 [30 points]. Consider two models $\mathcal{M}_1 = \langle\{0,1,2,3\}, (\text{square mod } 4), 3\rangle$ and $\mathcal{M}_2 = \langle\{0,3\}, (\text{times3 mod } 6), 0\rangle$, where $(\text{square mod } 4)(n) = n^2 \bmod 4$ and $(\text{times3 mod } 6)(n) = 3n \bmod 6$.

Answer: The notation used for models in this problem is the one introduced at the end of Section 2 of Handout 11. For convenience let

$$\begin{aligned} D_1 &= \{0,1,2,3\}, \\ D_2 &= \{0,3\}, \\ f_1 &= (\text{square mod } 4) : D_1 \longrightarrow D_1, \\ f_2 &= (\text{times3 mod } 6) : D_2 \longrightarrow D_2, \\ c_1 &= 3 \in D_1, \text{ and} \\ c_2 &= 0 \in D_2. \end{aligned}$$

(a) [7 points] Describe a function $\eta : \{0,1,2,3\} \to \{0,3\}$ which shows that $\mathcal{M}_2$ is a homomorphic image of $\mathcal{M}_1$. No explanation or proof is required.

Answer: There is exactly one homomorphism η from $\mathcal{M}_1$ onto $\mathcal{M}_2$, namely, $\eta(1) = \eta(3) = 0$ and $\eta(0) = \eta(2) = 3$.

To see how this result can be established, remember that a function $\eta : D_1 \longrightarrow D_2$ will be a homomorphism iff

$$\begin{aligned} \eta(c_1) &= c_2, \text{ and} \\ \eta(f_1(d)) &= f_2(\eta(d)) \text{ forall } d \in D_1 . \end{aligned}$$

The first condition provides us with

- $\eta(3) = 0$.

The second condition translates into:

- $\eta(0) = \eta(f_1(0)) = f_2(\eta(0)) = \eta(0)$,
- $\eta(1) = \eta(f_1(1)) = f_2(\eta(1)) = \eta(1)$,
- $\eta(0) = \eta(f_1(2)) = f_2(\eta(2)) = \eta(2)$,
- $\eta(1) = \eta(f_1(3)) = f_2(\eta(3)) = \eta(3)$.

The conditions $\eta(0) = \eta(0)$ and $\eta(1) = \eta(1)$ are vacuous. Hence η will be a homomorphism from $\mathcal{M}_1$ *into* $\mathcal{M}_2$ iff

$$\begin{aligned} \eta(1) &= \eta(3) = 0, \\ \eta(0) &= \eta(2). \end{aligned}$$

Finally, since η must also map D_1 *onto* D_2, we must have $\eta(0) = \eta(2) = 3$.

(b) [8 points] Remember that our definition of homomorphism (Problem Set 3 and its Supplement—Handouts 13 and 15) is based on models *together with* valuations of the variables. So to show that $\mathcal{M}_2$ is an homomorphic image of $\mathcal{M}_1$ under η, it is necessary to show that there are valuations $\mathcal{I}_{1,v}$ and $\mathcal{I}_{2,v}$ of all the first- and second-order variables such that $(\mathcal{M}_2, \mathcal{I}_{2,v})$ is an image of $(\mathcal{M}_1, \mathcal{I}_{1,v})$ under η. Describe two such valuations $\mathcal{I}_{1,v}$ and $\mathcal{I}_{2,v}$. Again, no explanation or proof is required.

Answer: Recall that, from Definition 7, Handout 11, valuations are defined precisely by their values on individual variables x_i, predicate variables $P_i^n (n \geq 0)$ and function variables $F_i^n (n \geq 1)$. Then satisfactory valuations would be defined (among others) by letting P_i^n be the identically **true** n-ary predicate, in symbols, $\mathcal{I}_{1,v}(P_i^n) \equiv$ **true**, and likewise:

$$\begin{aligned} \mathcal{I}_{1,v}(P_i^n) &\equiv \texttt{true}, \\ \mathcal{I}_{2,v}(P_i^n) &\equiv \texttt{true}, \\ \mathcal{I}_{1,v}(F_i^n) &\equiv 2, \\ \mathcal{I}_{2,v}(F_i^n) &\equiv 3. \\ \mathcal{I}_{1,v}(x_i) &= 2, \\ \mathcal{I}_{1,v}(x_i) &= 3. \end{aligned}$$

(c) [15 points] Explain why if $\mathcal{I}_{1,v}(P)$ is the "equals three" predicate on $\{0, 1, 2, 3\}$ for some unary predicate variable P, then there is *no* valuation $\mathcal{I}_{2,v}$ such that $(\mathcal{M}_2, \mathcal{I}_{2,v})$ is an image of $(\mathcal{M}_1, \mathcal{I}_{1,v})$ under *any* homomorphic mapping.

Answer: Assume there is a valuation $\mathcal{I}_{2,v}$ such that $(\mathcal{M}_2, \mathcal{I}_{2,v})$ is an image of $(\mathcal{M}_1, \mathcal{I}_{1,v})$ under some morphism η. Since $|D_1| = 4 > 2 = |D_2|$, there must be two elements of D_1 which η maps to the same element of D_2. From the previous analysis, we know that two such elements happen to be 1 and 3, though any two will do. Suppose P under $\mathcal{I}_{1,v}$ means "equals 3." Then $P(3)$ is true and $P(1)$ is false in $\mathcal{M}_1$, so η cannot preserve both these truth values since it maps 1 and 3 to the same element of D_2.

We can reformulate this reasoning as a simple calculation:

$$\texttt{true} = \mathcal{I}_{1,v}(P)(3) = \mathcal{I}_{2,v}(P)(\eta(3)) = \mathcal{I}_{2,v}(P)(\eta(1)) = \mathcal{I}_{1,v}(P)(1) = \texttt{false},$$

which is a contradiction. Hence there is no such a valuation $\mathcal{I}_{2,v}$.

6.044J / 18.423J: Computability, Programming and Logic
16 November 1988

Statistics about quiz 2

Mean: 41.68 over a total of 100.

Low: 20

1st High: 87
2nd High: 79.

problem Set 4

Problem 1. Problems 5(b) and 6 of Handout 12 (Review Problems for Quiz 2) indicate the fact that

$$\{(\mathcal{I}, A) \mid \mathcal{I} \text{ is and interpretation with finite domain, } A \text{ is a wff, and } \mathcal{I} \models A\}$$

is decidable. Describe how you would represent a finite interpretation $\mathcal{I}$ and a wff A as SCHEME data types, and write a two or three page sketch sufficient to convince the grader that you could write a scheme procedure EVAL with two formal paramaters INTERP and WFF which would compute the truth value of the wff represent by WFF under the interpreatation represented by INTERP.

Problem 2.

(a) Write a formula F, with free variable z, and whose only nonlogical symbol is the binary function symbol for concatentation (no individual constants or predicates!), such that over the interpretation of strings over Σ, F means "z is of length one". (*Hint*: z is of length one iff z is not empty and in any parse of z into two strings, one of them is empty.)

(b) Conclude from part (a) that,

$$\mathrm{Th}(\langle \{\mathtt{a},\mathtt{b}\}^*, \mathtt{a}, \mathtt{b}, \cdot \rangle) \leq_{\mathrm{m}} \mathrm{Th}(\langle \{\mathtt{a},\mathtt{b}\}^*, \cdot \rangle).$$

(*Hint*: : Try using variables x_a, x_b in place of the constants a. b.)

(c) Show that

$$\mathrm{Th}(\langle \Sigma^*, \sigma_1, \ldots, \sigma_k, \cdot \rangle) \leq_{\mathrm{m}} \mathrm{Th}(\langle \{\mathtt{a},\mathtt{b}\}^*, \mathtt{a}.\mathtt{b}, \cdot \rangle)$$

(d) Conclude that $\mathrm{Th}(\langle \{\mathtt{a},\mathtt{b}\}^*, \cdot \rangle)$ is neither r.e. nor co-r.e.

(*Hint*: Use the results above and Quiz 2, Problem 3.)

6-044/18.423J Handout 21 - Nov 17, 1988

Solutions to Problem Set 3.

Mary Vogt
6.044
Problem Set 3
November 9, 1988
Problem 1

Problem 1. (Prob. 2-9, p.153)

(b) Let $D = \{\text{all integers}\}$ and $p(x, y)$ be $x > y$. Then $(\forall x \forall y \forall z x > y \wedge y > z) \wedge \forall x x \not> x$ is true, but $\exists x \forall y x \not> y$ is false, and so the entire wff is false.

(c) Let $D = \{\text{all integers}\}$ and $p(x, y)$ be $x \geq y$.

Theodore Ts'o

6.044
Problem Set 6
November 6, 1988

Problem 2. Manna, page 153, exercise 2-10 (e), (f), (g), (h), (o)

Which of the following wffs are valid? Construct models for those which are not valid and give informal proofs for the valid wffs.

2e. $\forall_x \forall_y p(x,y) \rightarrow \forall_y \forall_x p(x,y)$

This wff is valid.

1.	$\forall_x \forall_y p(x,y) \rightarrow \forall_x \forall_y p(x,y)$	Assumption axiom
2.	$\forall_x \forall_y p(x,y) \rightarrow \forall_y p(x,y)$	$\forall$ elimination
3.	$\forall_x \forall_y p(x,y) \rightarrow p(x,y)$	$\forall$ elimination
4.	$\forall_x \forall_y p(x,y) \rightarrow \forall_x p(x,y)$	$\forall$ introduction
5.	$\forall_x \forall_y p(x,y) \rightarrow \forall_y \forall_x p(x,y)$	$\forall$ introduction

2f. $\forall_x \forall_y p(x,y) \rightarrow \forall_x \forall_y p(y,x)$

This wff is valid.

1.	$\forall_x \forall_y p(x,y) \rightarrow \forall_x \forall_y p(x,y)$	Assumption axiom
2.	$\forall_x \forall_y p(x,y) \rightarrow \forall_y p(x,y)$	$\forall$ elimination
3.	$\forall_x \forall_y p(x,y) \rightarrow p(x,y)$	$\forall$ elimination
4.	$\forall_x \forall_y p(x,y) \forall_x \rightarrow p(x,y)$	$\forall$ introduction
5.	$\forall_x \forall_y p(x,y) \forall_y \forall_x \rightarrow p(x,y)$	$\forall$ introduction

2g. $\forall_x [p(y) \rightarrow q(x)] \rightarrow [p(y) \rightarrow \forall_x q(x)]$

This wff is valid.

1.	$\forall_x [p(y) \rightarrow q(x)] \rightarrow \forall_x [p(y) \rightarrow q(x)]$	Assumption axioms
2.	$\forall_x [p(y) \rightarrow q(x)] \rightarrow [p(y) \rightarrow q(x)]$	$\forall$ elimination
2.	$\forall_x [p(y) \rightarrow q(x)] \rightarrow [p(y) \rightarrow \forall_x q(x)]$	$\forall$ introduction

2h. $\forall_x[p(x) \rightarrow q(x)] \rightarrow [p(x) \rightarrow \forall_x q(x)]$

This wff is not valid.

The following is a countermodel: Let D be the set of integers, and let I_C maps $p(x)$ to a unary predicate which is true iff x is divisible by 4, and maps $q(x)$ to a unary predicate which is true iff x is divisible by 2.

Then, $\forall_x p(x) \rightarrow q(x)$, but $\exists_x$ such that $p(x)$ (p(4) for example), but it is not the case that $\forall_x q(x)$.

2o. $\forall_P[\exists_x P(x) \rightarrow \forall_x P(x)]$

This is wff is not valid.

The following is a countermodel: Let D be the set of integers, and let the predicate $p(x)$ which is true iff $x = 17$ is in the range of I_C. Then $\exists_{P=p}\exists_{x=17}P(x) \wedge \exists_{x=31}\neg P(x)$.

Problem 3 Prove lemma 1 page 5 of handout 11

Lemma: Let A be a closed wff and suppose $\mathcal{I} = (\mathcal{M}, \mathcal{I}_V) \models A$ for some valuation $\mathcal{I}_V$. Then $\mathcal{M} \models A$

Answer: $\mathcal{M} \models A$ means that for all valuations $\mathcal{I}_V$ we have $(\mathcal{M}, \mathcal{I}_V) \models A$.
By hypothesis, there is some valuation $\mathcal{I}_{V,1}$ such that:

$$\mathcal{I}_1 = (\mathcal{M}, \mathcal{I}_{V,1}) \models A$$

We are going to prove that for any other interpretation $\mathcal{I}_2$ we have $\mathcal{I}^1(A) = \mathcal{I}^2(A)$ which will establish our claim: $((\mathcal{M}, \mathcal{I}_V) \models A \ \forall \mathcal{I}_V)$

First we establish the following lemma:

Lemma: Assume that $\mathcal{I}^1(x_i) = \mathcal{I}^2(x_i)$ for any variable in a given set of variables S_x
_____ $\mathcal{I}^1(F_i^n) = \mathcal{I}^2(F_i^n)$ _____ function _____ variable function S_F
(_____ $\mathcal{I}^1(P_i^n) = \mathcal{I}^2(P_i^n)$ _____ predicate _____ predicate S_P)
then for any term t – constructed inductively, using at each step of this construction variables taken from S_x, S_F (or S_P) – $\mathcal{I}^1(t) = \mathcal{I}^2(t)$

proof: by induction on the definition of t:

- If $t = x_i$ then by asumption $\mathcal{I}^1(t) = \mathcal{I}^1(x_i) = \mathcal{I}^2(x_i) = \mathcal{I}^2(t)$.
- (If $t = c_i$ there is nothing to say: this is taken care of by $\mathcal{M}$ which "contains" $\mathcal{I}_c$)
- If $t = f_i^n(t_1, \dots t_n)$,

$$\mathcal{I}^1(t) = \mathcal{I}^1_c(f_i^n)(\mathcal{I}^1(t_1) \dots \mathcal{I}^1(t_n))$$

But by asumption $\mathcal{I}^1_c(f_i^n) = \mathcal{I}^2_c(f_i^n)$ (because we are considering interpretations over the same model $\mathcal{M}$),
and by hypothesis of our induction $\mathcal{I}^1(t_1) = \mathcal{I}^2(t_1)$
$\mathcal{I}^1(t_n) = \mathcal{I}^2(t_n)$.

Hence $\mathcal{J}^1(t) = \mathcal{J}^2(t)$.

- If $t = F_c^n(t_1 \dots t_n)$,
  $\mathcal{J}^1(t) = \mathcal{J}^1_\sigma(F_c^n)\left(\mathcal{J}^1(t_1) \dots \mathcal{J}^1(t_n)\right)$
  where I write $\mathcal{J}^1_\sigma(F_c^n)$ instead of the generic $\mathcal{J}^1(F_c^n)$

in order to distinguish this case from the previous one.

Using the induction hypothesis we get that $\mathcal{J}^1(t_1) = \mathcal{J}^2(t_1) \dots \mathcal{J}^1(t_n) = \mathcal{J}^2(t_n)$

Using the assumption of the lemma we can write $\mathcal{J}^1(F_c^n) = \mathcal{J}^2(F_c^n)$

Hence $\mathcal{J}^1(t) = \mathcal{J}^1(F_c^n)\left(\mathcal{J}^1(t_1) \dots \mathcal{J}^1(t_n)\right)$

$= \mathcal{J}^2(F_c^n)\left(\mathcal{J}^2(t_1) \dots \mathcal{J}^2(t_n)\right) = \mathcal{J}^2(t)$.

... which proves the lemma ∎

let's go back to the proof of our problem.

Claim: Let $\mathcal{I}^1$ and $\mathcal{I}^2$ be interpretations over the same model such that $\mathcal{I}^1(x) = \mathcal{I}^2(x)$, $\mathcal{I}^1(F_c^n) = \mathcal{I}^2(F_c^n)$ and $\mathcal{I}^1(P_c^n) = \mathcal{I}^2(P_c^n)$ for all x, F, P free-variables of A. Then $\mathcal{I}^1(A) = \mathcal{I}^2(A)$.

proof: by induction on the definition of a wff A.

- If A is an atomic formula:
  - if A is $[t_1 = t_2]$:
    From our previous lemma $\mathcal{I}^1(t_1) = \mathcal{I}^2(t_1)$ and $\mathcal{I}^1(t_2) = \mathcal{I}^2(t_2)$
    Using (3) of def 10 of Handout II we see that $\mathcal{I}^1(A) = \mathcal{I}^2(A)$
  - if A is $P_c^n(t_1 \ldots t_n)$ we saw in the previous lemma that $\mathcal{I}^1(A) = \mathcal{I}^2(A)$.
  - Similarly if A is $P_c^n(t_1 \ldots t_n)$.
- If A is of the form $A_1 \wedge A_2$
  $\mathcal{I}^1(A) = \mathcal{I}^1(A_1) \wedge \mathcal{I}^1(A_2)$ (by the definition of $\mathcal{I}^1(A)$)
  $= \mathcal{I}^2(A_1) \wedge \mathcal{I}^2(A_2)$ (by induction
  $= \mathcal{I}^2(A)$ (by the definition of $\mathcal{I}^2(A)$)
- A similar proof is obtained for A of the form $\neg A_1$, $A_1 \supset A_2$, $A \vee$
- If A is of the form $\forall x\, A_1$ we have:
  $\mathcal{I}^1(A) = \bigwedge_{d \in D} \mathcal{I}^1(x \longmapsto d)(A_1)$ (by def of $\mathcal{I}^1(A)$
  $= \bigwedge_{d \in D} \mathcal{I}^2(x \longmapsto d)(A_1)$ (by induction)
  $= \mathcal{I}^2(A)$. (by def of $\mathcal{I}^2(A)$.

and likewise for A of the form $\exists x\, A_1$

this establishes our claim and answers the question.
Note that, although A has, by assumption, no free variable, its subwffs might very well have some.
(think for instance at: $\forall x \left[\underbrace{(f_0' f_1'(x) = x)}_{A_1} \supset p_1'(x) \right]$
which is of the form $\forall x\, A_1$.
Then A has no free variable whereas A_1 has some!)

Notation 4 Let us denote $\mathcal{I}'_\sigma = h \circ \mathcal{I}_\sigma$ as a short hand notation and let us recall once more how these valuations are related:

- $\mathcal{I}'_c(f)\ (h d_1, \ldots, h d_n) = h\left(\mathcal{I}_c(f)\ (d_1 \ldots d_n)\right)$ (1)
- $(\mathcal{I}'_\sigma F)\ (h\mathcal{I}t_1, \ldots h\mathcal{I}t_n) = h\left(\mathcal{I}(F)\ (\mathcal{I}t_1, \ldots \mathcal{I}t_n)\right) = h\left(\mathcal{I}\left(F(t_1 \ldots t_n)\right)\right)$
$= \mathcal{I}'\left(F(t_1 \ldots t_n)\right)$ (2)

In order to understand the notation $\mathcal{I}' = h \circ \mathcal{I}$ you have to realize that if we interpret a term t, $\mathcal{I}(t)$ and $\mathcal{I}'(t)$ are elements of the domain D, so that the equality $\mathcal{I}'(t) = h(\mathcal{I}(t))$ is the usual equality.

- $\mathcal{I}'(P)\ (h\mathcal{I}t_1 \ldots h\mathcal{I}t_n) = \mathcal{I}(P)\ (\mathcal{I}t_1 \ldots \mathcal{I}t_n) = \mathcal{I}\left(P(t_1 \ldots t_n)\right)$ (3)
$\mathcal{I}'(P)\ (\mathcal{I}'t_1 \ldots \mathcal{I}'t_n) = \mathcal{I}'\left(P(t_1 \ldots t_n)\right)$
and similarly for the cases involving constants, product etc.

If h is an isomorphism:

we can rewrite the first equalities of (1), (2) and (3) as:

- $(\mathcal{I}'_c(f))\ (d'_1 \ldots d'_n) = h\left(\mathcal{I}_c(f)\ (h^{-1}d_1 \ldots h^{-1}d_n)\right)$ (1')
- $(\mathcal{I}'F)\ (\mathcal{I}'t_1 \ldots \mathcal{I}'t_n) = h\left(\mathcal{I}(F)\ (h^{-1}\mathcal{I}'t_1 \ldots h^{-1}\mathcal{I}'t_n)\right)$ (2')
- $(\mathcal{I}'P)\ (\mathcal{I}'t_1 \ldots \mathcal{I}'t_n) = \mathcal{I}(P)\ (h^{-1}\mathcal{I}'t_1 \ldots h^{-1}\mathcal{I}'t_n)$ (3').

In these three equalities h is really mapping an element of D into an element of D'.

Using the notation $g' = h \circ g$ (2) for instance becomes:

$$(h \circ g)(F)(g't_1 \ldots g't_n) = h\left(g(F)(h^{-1}g't_1 \ldots h^{-1}g't_n)\right)$$

which is the formula that the hint at the end of handout 15 was precisely alluding to.

All this is not very deep but you should try to work on manipulating all these symbols in a consistant way.

In the same way if h is an isomorphism for any two terms t_1 and t_2

$gt_1 = gt_2$ iff $g't_1 = g't_2$

" $h(gt_1)$ " $h(gt_2)$.

(A counter-example of this is given by problem 1 of handout 15).

In equations (1), (1'), (2) we see that any valuation on $\mathcal{M}_1$ gives a valuation on $\mathcal{M}_2$. Using equations (1'), (2'), (3') we see that actually this correspondance is a bijective one. Namely:

To any valuation on $\mathcal{M}_2$ corresponds a valuation on $\mathcal{M}_1$ and the correspondances

$$g \longmapsto g' \longmapsto g$$

on $\mathcal{M}_1$ on $\mathcal{M}_2$ on $\mathcal{M}_1$ gives back the original one.

to our problem.

$(\mathcal{M}_1) = h_2(\mathcal{M}_2)$ iff

$(\mathcal{M}_1, g) \models A$ iff $(\forall g' \ (\mathcal{M}_2, g') \models A)$

our previous consideration the set of g' is

previous assertions rewrites

iff $\left[(\forall J \ (\mathcal{M}_1, J) \models A) \text{ iff } (\forall J \ (\mathcal{M}_2, h \circ J) \models A) \right]$

And this will hold when we will have proven that:

$$\forall J \left[(\mathcal{M}_1, J) \models A \text{ iff } (\mathcal{M}_2, h \circ J) \models A \right] \qquad (*)$$

(Note the change of position of the quantifier !)

Claim $h \circ J(t) = J'(t)$ for all terms t over S

proof look at the corresponding claim page 3 of handout 15.

(*) can be rewritten as $\forall J \quad J(A) = (h \circ J)(A)$.

In Problem 1 c) of handout 15 this was established for any first order wff A without equality. To prove (*) for a general wff in the case where h is an isomorphism we just need to prove (*) holds if

A is of the form $P^n(t_1 \ldots t_n)$

or $t_1 = t_2$.

(because then the cases $\neg A_1$, $A_1 \wedge A_2$, $A_1 \vee A_2$, $A_1 \supset A_2$, $\forall x A_1$, $\exists x A_1$ follow as in handout 15)

case A is of the form $P^n(t_1 \ldots t_n)$

But in equation (3) of the beginning of these solutions we established that $J(A) = J(P^n(t_1 \ldots t_n)) = J'(P^n(t_1 \ldots t_n)) = J'(A)$

case A is of the form: $[t_1 = t_2]$

Using (3) of def 10 of handout 11, and the fact that for any isomorphism h, $Jt_1 = Jt_2$ iff

we get that $J(A) = J'(A)$.

6.044 / 18-423 J Handout 22

Solutions to Problem Set 4

Problem 1: from: Theodore Ts'o

Problem 2: a, b from Aaron Goodisman.

+ Problem 2 c, d

Theodore Ts'o
6.044
Problem Set 4
November 23, 1988

Problem 1. Evaluating a WFF in Scheme

In order to evaluate a WFF in Scheme, you must first be able to represent a WFF in scheme. A WFF A is represented as follows:

Let A, B, and C be WFF's and A_{rep}, B_{rep}, and C_{rep} their representations in Scheme, respectively. Let P^0 be a perpositional variable and p^0 be a prepositional constant, and P^0_{rep} and p^0_{rep} their representation in Scheme. Let P^n be a predicate variable and let p^n be a predicate constant, and P^n_{rep} and p^n_{rep} be their representation. Finally, let t_i be a term and $t_{i_{rep}}$ its representation. The representation of prepositional variables, prepositional constants, and terms will be described later. Then a WFF A can be represented as follows: it must be one of the following forms:

- If A is of the form P^0, then A_{rep} = ('pvar P^0)
- If A is of the form p^0, then A_{rep} = ('pconst P^0)
- If A is of the form $t_1 = t_2$, then A_{rep} = ('eqv $t_{1_{rep}}$ $t_{2_{rep}}$)
- If A is of the form $P^n(t_1, \ldots, t_n)$, then A_{rep} = ('predv P^n_{rep} $t_{1_{rep}} \ldots t_{n_{rep}}$)
- If A is of the form $p^n(t_1, \ldots, t_n)$, then A_{rep} = ('predc p^n_{rep} $t_{1_{rep}} \ldots t_{n_{rep}}$)
- If A is of the form $\neg B$, then A_{rep} = ('neg B_{rep})
- If A is of the form $B \supset C$, then A_{rep} = ('imply B_{rep} C_{rep})
- If A is of the form $B \wedge C$, then A_{rep} = ('and B_{rep} C_{rep})
- If A is of the form $B \vee C$, then A_{rep} = ('or B_{rep} C_{rep})
- If A is of the form $\forall_x B$, the A_{rep} = ('forall 'x B_{rep})
- If A is of the form $\exists_x B$, the A_{rep} = ('exists 'x B_{rep})

Let t be a term and t_{rep} its representation. Let f^n be a function variable and f^n_{rep} its representation. Finally, let F^n be a function constant and let F^n_{rep} be its representation. Then a term t can be represented as follows: it must be one of the following terms:

- If t is a variable x, then t_{rep} = ('var 'x)

- If t is a variable c, then t_{rep} = ('const 'c)
- If t is of the form $f^n(t_1, \ldots, t_n)$, then t_{rep} = ('fvar $t_{1_{rep}} \ldots t_{n_{rep}}$)
- If t is of the form $F^n(t_1, \ldots, t_n)$, then t_{rep} = ('fconst $t_{1_{rep}} \ldots t_{n_{rep}}$)

In order to keep evaluate variable names and constants names in a context, the following binding data structure, called **environ** is used. It consists of a list consisting of a pointer to the "parent environ," if it exists (if it does not, nil is placed there), followed by (data . value) pairs. The value of a predicate is a lambda expression which when evaluated witht the proper number of arguments, returns t or nil. The value of a function is a lambda which when evaluated, returns a representation of a term. The value of a constant or a variable is a representation of a term.

An interpretation $\mathcal{I}$ is represented by a list consisting of the following elements:

- A list of all elements in the Domain.
- An **environ** binding the variables to their values
- An **environ** binding the predicate variables to their values
- An **environ** binding the function variables to their values
- An **environ** binding the constants to their values
- An **environ** binding the predicate constants to their values
- An **environ** binding the function constants to their values

The following procedure is used to evaluate a term t_{rep}, given an interpretation $\mathcal{I}$: Look at the car if t_{rep}...

- If it is 'var, look up the cadr of the t_{rep} in the variable **environ.**
- If it is 'const, look up the cadr of the t_{rep} in the constant **environ**
- If it is 'fvar, look up the cadr of the t_{rep} in the function variable **environ.** Recursive apply this procedure to the following arguments and apply those arguments to the cadr of the t_{rep},
- If it is 'fconst, look up the cadr of the t_{rep} in the function constant **environ.** Recursive apply this procedure to the following arguments and apply those arguments to the cadr of the t_{rep},

Finally, the following procedure is used to evaluate a WFF A_{rep}, given an intepretation $\mathcal{I}$. Look at the car of A_{rep}...

- If it is 'pvar, look up the cadr of A_{rep} in the variable **environ**.
- If it is 'pconst, look up the cadr of A_{rep} in the constant **environ**.
- If it is 'eqv, evaluate the caadr and the caaadr of A_{rep} and apply the scheme operator "equal?" to them; the result of "equal?" is the evaluation of the wff.
- If it is 'predv, look up the cadr in the predicate variable **environ**, and apply it with arguments obtained by recursively apply this procedure to the arguments (found as a list as the cddr of A_{rep}).
- If it is 'predc, look up the cadr in the predicate constant **environ**, and apply it with arguments obtained by recursively apply this procedure to the arguments (found as a list as the cddr of A_{rep}).
- If it is 'neg, evaluate the cadr of A_{rep} by recursively using this procedure, and return t of the evaluation is nil, and nil if the evaluation is t.
- If it is 'imply, evaluate the cadr and caddr of A_{rep}; the evaluation is true if the cadr evaluates to false (nil) or if the caddr evaluates to true (t); otherwise, the evaluation is false.
- If it is 'and, evaluate the cadr and caddr of A_{rep}; the evaluation is true if the cadr and cadr evaluates to true (t); otherwise, the evaluation is false.
- If it is 'or, evaluate the cadr and caddr of A_{rep}; the evaluation is true if the cadr or cadr evaluates to true (t); otherwise, the evaluation is false.
- If it is 'forall, create an interpretation which is identical to the current interpretation except that the variable **environ** is replaced with an empty **environ** whose parent environ is the original environ. Recursively evaluate the caddr of A_{rep} with every element listed in the domain (the car of $\mathcal{I}$) bound to the cadr of A_{rep} in the new **environ**.. If the evaluations are all true, the evaluation is true; otherwise, it is false.
- If it is 'exists, create an interpretation which is identical to the current interpretation except that the variable **environ** is replaced with an empty **environ** whose parent environ is the original environ. Recursively evaluate the caddr of A_{rep} with every element listed in the domain (the car of $\mathcal{I}$) bound to the cadr of A_{rep} in the new **environ**.. If one of the evaluations is true, the evaluation is true; otherwise, it is false.

This completes the demonstration that a WFF A can be evaluated with an interpretation $\mathcal{I}$, as long as the domain of the model is finite.

Aaron Goodisman 6.044J/18.423J Problem Set 3 Problem 2 24 October 1988

This problem was discussed with Mary Vogt.

Problem 2:

2(a).

F is

$$(\neg(z = z \cdot z)) \wedge (\forall x, y[(z = x \cdot y) \supset ((z = x) \vee (z = y))])$$

That is, z is not the empty string, and *for all* parses of z into two strings x and y, z equals either x or y.

2(b).

Let

$$M_1 = \langle \{a, b\}^*, a, b, \cdot \rangle \text{ and}$$

$$M_2 = \langle \{a, b\}^*, \cdot \rangle$$

In order to conclude that

$$\mathrm{Th}(M_1) \leq_m \mathrm{Th}(M_2)$$

we must show that

$$\{A \mid M_1 \models A\} \leq_m \{A \mid M_2 \models A\}$$

That is, we must show that there is some function f such that

$$\forall W \in \{A \mid A \text{ is a wff}\}$$

$$[W \in \{A \mid M_1 \models A\} \text{ iff } f(W) \in \{A \mid M_2 \models A\}]$$

constant $\underset{\sim}{a}$, $\underset{\sim}{b}$

Now if W is in $\{A \mid M_1 \models A\}$, then there must be exactly two symbols in the signature of W, ~~call them x_a and x_b,~~ such that $\mathcal{I}_c(\underset{\sim}{a}) = a$ and $\mathcal{I}_c(\underset{\sim}{b}) = b$. Let f be the function

$$f : \{A \mid A \text{ is a wff}\} \rightarrow \{A \mid A \text{ is a wff}\}$$

$$f(W) \equiv \exists x_a[\exists x_b[W \wedge F(x_a) \wedge F(x_b) \wedge (\neg(x_a = x_b))]]$$

$W\ x_a, x_b$

$\underset{\sim}{a}, \underset{\sim}{b}$ (distinct)

where x_a, x_b are fresh individual variables

where F is the wff defined in problem 2(a). In other words, f maps a wff W (in $\{A \mid M_1 \models A\}$) into a wff $f(W)$ (in $\{A \mid M_2 \models A\}$) in which the symbols x_a and x_b, which were mapped into a and b for W, are restricted to being of length one and unequal to each other; that is, x_a and x_b are restricted to being either a and b or b and a respectively. It can be assumed without loss of generality that the truth value of a wff W is unaffected if every constant symbol mapped by $\mathcal{I}_c$ to a is mapped to b and every constant symbol mapped to b is mapped to a. That is, the many-to-one reduction still holds if x_a is mapped to b and x_b is mapped to a.

$\Longrightarrow$: If the model M_1 satisfies a wff W for all valuation functions $\mathcal{I}_v$, then the model M_2 certainly satisfies $f(W)$ under all valuation functions, since $x_a = a$ and $x_b = b$ satisfies the existential portion of $f(W)$ and guarantees that all the parts of the conjunctions in $f(W)$ are satisfied.

$\Longleftarrow$: If the model M_2 satisfies a wff $f(W)$ for all valuation functions $\mathcal{I}_v$, then the model M_1 certainly satisfies W since two of the requirements that $f(W)$ be true are that W be true when $x_a = a$ and $x_b = b$ and that $x_a = a$ and $x_b = b$.

Therefore, $M_1 \models W$ iff $M_2 \models f(W)$ and so $\mathrm{Th}(M_1) \leq_m \mathrm{Th}(M_2)$.

2c) Let us first analyze the problem :

- the general idea is to encode all the strings in Σ^* into strings over $\{a,b\}^*$. This encoding has to preserve the concatenation ((ie) be a homomorphism)
- to be 1-1, so that we can recognize without ambiguity the encoded string from its encoding.

More formally, we are looking for :

$f : \Sigma \xrightarrow{1-1} \{a,b\}^*$ such that its natural homomorphism extension $f^* : \Sigma^* \longrightarrow \{a,b\}^*$ defined by :

$\forall x \; \forall i_1 \ldots i_x \in \{1..k\}, \; f^*(t_{i_1} \cdot t_{i_2} \cdot \ldots \cdot t_{i_x}) = f(t_{i_1}) \cdot f(t_{i_2}) \cdots \cdot f(t_{i_x})$.

is one to one too.

- If we have found such an f^* we can express all the strings in Σ^* as strings in $\{a,b\}^*$.

We then notice that $\text{Range}(f^*)$ is a priori included strictly in $\{a,b\}^*$

But $(x \in \Sigma^*)$ iff $(f^*(x) \in \text{Range}(f^*))$ or more significantly :

$(y \in \text{Range}(f^*))$ iff $(y$ is the encoding of a string in $\Sigma^*)$.

We have to be able to translate the expressions $\exists x \in \Sigma^*$ or $\forall x \in \Sigma^*$ into valid subwff's in $\text{Th}(\langle\langle \{a,b\}^*, a, b, \cdot \rangle)$.

These are the two general subproblems we want to solve to answer the question. Let's now go to business.

there are of course many possible encodings - (I have a chosen what is called a prefix-encoding).

Define for $i = 1 \dots k$ $\quad \boxed{f(t_i) = b^i . a}$.

(where b^i means $\underbrace{b . b \dots b}_{i \text{ times}}$).

f^* is then the associated homomorphism:

$$f^*(t_{i_1} . t_{i_2} \dots t_{i_x}) = b^{i_1} . a . b^{i_2} . a \dots b^{i_x} . a$$

The a serves as a delimiter, and we see that f^* is one to one:

$\forall y \in Im f^*$ there is only one $x \in \Sigma^*$ such that $f^*(x) = y$.

The expression $\forall x$ appearing in wff's of $th(\langle \Sigma^*, t_1 \dots t_k, \circ \rangle)$ (which actually means $\forall x \in \Sigma^*$) corresponds through f^* to:

$\underline{\forall y \in Range(f^*)}$ and we want to write this in predicate calculus syntax.

But: $y \in Range(f^*)$ can be rewritten as:

$$\{ \forall s_1, t_1, u_1 \; [(y = s_1 . a . t_1 . a . u_1) \wedge (t_1 \text{ has no } a\text{'s}) \supset ((t_1 = b) \vee (t_1 = b . b) \vee \dots \vee (t_1 = b^k))] \}$$

$$\wedge \{ \forall s_2, t_2 \; [(y = s_2 . a . t_2) \wedge (s_1 \text{ has no } a\text{'s}) \supset ((s_1 = b) \vee \dots \vee (s_1 = b^k))] \}.$$

$$\wedge \{ \text{the last letter of } y \text{ is } a \}$$

- the expression (t has no a's) can be expressed as:

$$\forall x, \forall \alpha, \forall y \left[(t = x \cdot \alpha \cdot y) \wedge ((\alpha = a) \vee (\alpha = b)) \supset (\alpha \neq a) \right].$$

- the expression (the last letter of y is a) can be expressed as:

$$\forall \alpha, \forall x \left[(y = x \cdot \alpha) \wedge ((\alpha = a) \vee (\alpha = b)) \supset (\alpha = a) \right].$$

To finish let us say precisely how a wff A in $Th(\langle \Sigma^*, \dots \rangle)$ is transformed (in a computable way) into a wff A' in $Th(\langle \{a,b\}^*, a, b, \cdot \rangle)$ so that

$$Th(\langle \Sigma^*, \dots \rangle) \leq_m Th(\langle \{a,b\}^*, a, b, \cdot \rangle) :$$

- whenever a free variable or a variable bound by a quantifier is used we add the fact that this variable is "in Range f^*" (we first saw how to express this as a wff)
- whenever a variable function F (free or bound by a qualifier) is used we add the fact that, calling n the arity of F:

$$F : \underbrace{Range(f^*) \times \dots \times Range(f^*)}_{n \text{ times}} \longrightarrow Range(f^*)$$

(ie) $$x_1, \dots x_n \in Range(f^*) \Longrightarrow F(x_1 \dots x_n) \in Range(f^*).$$

Note: we do not have to specify how functions or predicates act outside $Range(f^*)$: we are just interested in the behaviour over $Range(f^*)$.

Ex the wff

" $\exists x_1 \; \forall F^1 \quad F^1(y.x_1) \wedge P^2(x,y)$ " would become

" $\exists x_1 \Big[\Big(x_1 \in \text{Range}(f^*)\Big) \wedge \forall F^1 \Big(\forall x_2 \; \forall x_3 \; \Big(F^1(x_2) = x_3\Big) \wedge \Big(x_2 \in \text{Range}(f^*)\Big) \supset \Big(x_3 \in \text{Range}(f^*)\Big)$

$\supset \Big[\Big(y \in \text{Range}(f^*)\Big) \wedge F^1(y.x_1) \wedge P^2(x_1, y) \Big]$ "

2-d] From Quiz 2, Problem 3 we know that

$$Th\Big(\langle \{a,b\}^*, a, b, \cdot, \text{eq} \rangle\Big) \leq_m Th\Big(\langle \{a,b\}^*, a, b, \cdot \rangle\Big)$$

From b] of this problem $Th\Big(\langle \{a,b\}^*, a, b, \cdot \rangle\Big) \leq_m Th\Big(\langle \{a,b\}^*, \cdot \rangle\Big)$

From Gödel's incompleteness theorem the theory over binary strings is neither re nor co-re. By up-inheritance we deduce that

$Th\Big(\langle \{a,b\}^*, \cdot \rangle\Big)$ is not re nor co-re either.

Solutions to Quiz 3

Problem 1. Consider the two following wffs:

$$\begin{aligned} A_1 &: \quad \forall x\, [p(x) \supset q(x)] \\ A_2 &: \quad (\forall x\, p(x)) \supset (\forall x\, q(x)) \end{aligned}$$

(a) Show that A_1 implies A_2.

We shall give two solutions.

- Suppose that A_1 does not imply A_2 *i.e.* that there is some interpretation $\mathcal{I}$ s.t. $\mathcal{I}(A_1)$ is true but $\mathcal{I}(A_2)$ is false. $\mathcal{I}(A_2)$ is false iff $\mathcal{I}(\forall x\ p(x))$ is true whereas $\mathcal{I}(\forall x\ q(x))$ is false. Then $\exists d \in D$ s.t. $\mathcal{I}_c(q)(d)$ is false and $\mathcal{I}_c(p)(d)$ is true. But $\mathcal{I}(A_1)$ says that $\mathcal{I}(\forall x\ (p(x) \supset q(x))$ is true. Hence $\mathcal{I}_c(p)(d) \supset \mathcal{I}_c(q)(d)$, so that $\mathcal{I}_c(q)(d)$ should be true, which is a contradiction. Hence A_1 implies A_2.
- In this solution we procede using systematic rules as described in Manna.

(1)	$\forall x\, [p(x)$	$\supset$	$q(x)]$	hypothesis
(2)	$p(x)$	$\supset$	$q(x)$	$\forall$-elimination
(3)	$(\forall x\, p(x))$	$\supset$	$(\forall x q(x))$	$\forall\forall$ -introduction (Manna p. 120).

If you do not feel comfortable with 3) which is just an exercise in Manna, proceed as follows from 2):

(3)	$(\forall x\, p(x))$	$\Rightarrow$	$(p(x) \supset q(x))$	assumption introduction from (2)
(4)	$(\forall x\, p(x))$	$\Rightarrow$	$(\forall x\, p(x))$	assumption axiom
(5)	$(\forall x\, p(x))$	$\Rightarrow$	$p(x)$	$\forall$ elimination from (4)
(6)	$(\forall x\, p(x))$	$\Rightarrow$	$q(x)$	$\supset$ elimination from (3,5)
(7)	$(\forall x\, p(x))$	$\Rightarrow$	$(\forall x\, q(x))$	$\forall$ introduction fom (6)
(8)	$(\forall x\, p(x))$	$\supset$	$(\forall x\, q(x))$	$\supset$ introduction: rule II-(c) of Manna, page 110.

(b) Give a countermodel showing that A_2 does not imply A_1.

Consider a model $\mathcal{M}$ such that D has at least two elements $d_1 \neq d_2$, and such that $\mathcal{I}_c(p)(d)$ is "$d = d_1$" and $\mathcal{I}_c(q)(d)$ is "$d = d_2$". Then $\mathcal{I}(A_1)$ is false, and the premises of $\mathcal{I}(A_2)$ being false, $\mathcal{I}(A_2)$ is vacuously true.

Problem 2. Let A and B be two wffs.

(a) Prove that if $\models A \supset B$, then $A \models B$.

- $\models A \supset B$ means:

$$\begin{array}{rl} & (\mathcal{M}, \mathcal{I}_v) \models (A \supset B) \text{for all } \mathcal{M}, \mathcal{I}_v, \\ i.e. & (\mathcal{M}, \mathcal{I}_v) \models A \text{ implies } (\mathcal{M}, \mathcal{I}_v) \models B \text{ for all } \mathcal{M}, \mathcal{I}_v \quad \text{(Assertion 1).} \end{array}$$

- $A \models B$ means:

$$\begin{array}{rl} & \mathcal{M} \models A \text{ implies } \mathcal{M} \models B \text{ for all } \mathcal{M}, \\ i.e. & \text{if for all } \mathcal{I}_v \ (\mathcal{M}, \mathcal{I}_v) \models A \text{ then for all } \mathcal{I}_v (\mathcal{M}, \mathcal{I}_v) \models B \quad \text{(Assertion 2).} \end{array}$$

Suppose for contradiction that Assertion 1 does not implie Assertion 2, *i.e.* Assertion 1 is true whereas Assertion 2 is false. Assertion 2 is false iff there exists some $\mathcal{M}_0$ such that $\forall \mathcal{I}_v (\mathcal{M}_0, \mathcal{I}_v) A$ is true whereas $\forall \mathcal{I}_v (\mathcal{M}_0, \mathcal{I}_v) B$ is false. The last fact happens iff there is a valuation $\mathcal{I}_{v,0}$ such that $(\mathcal{M}_0, \mathcal{I}_{v,0}) B$ is false. But $(\mathcal{M}_0, \mathcal{I}_{v,0}) A$ is true, so from Assertion 1 $(\mathcal{M}_0, \mathcal{I}_{v,0}) B$ is then also true. This is a contradiction!

(b) Explain why if A is closed, then $(\mathcal{M}, \mathcal{I}_v) \models A \quad \text{iff} \quad \mathcal{M} \models A$.

Let $\mathcal{I}_{v,0}$ be any specific valuation. We want to prove that $(\mathcal{M}, \mathcal{I}_{v,0}) \models A$ iff $\mathcal{M} \models A$.

- $\mathcal{M} \models A$ means $\forall \mathcal{I}_v \ (\mathcal{M}, \mathcal{I}_v) \models A$, so that the implication $\Leftarrow$) is obvious.
- The implication $\Rightarrow$ follows from lemma 1 of handout 11, page 5.

(c) Conclude that if A is closed, then $\models A \supset B \quad \text{iff} \quad A \models B$.

We saw in part a) that $\models A \supset B$ implies that $A \models B$. Suppose now that $A \models B$. To show $\models A \supset B$ let $(\mathcal{M}, \mathcal{I}_v)$ be any iterpretation such that $(\mathcal{M}, \mathcal{I}_v) \models A$, and we must show that $(\mathcal{M}, \mathcal{I}_v) \models B$. But since A is closed, we have by part (b) that $\mathcal{M} \models A$. Now, since $A \models B$, we conclude that $M \models B$, and hence in particular that $(\mathcal{M}, \mathcal{I}_v) \models B$.

(d) Give an example of A, B such that $A \models B$, but it is not true that $\models A \supset B$.

Consider $A : p(x)$ and $B : \forall x \ p(x)$. Then $A \not\vdash B$ by $\forall$ introduction. So $A \models B$ by soundness of the rules. To see that A does not implie B, consider an interpretation for which the domain has at least two elements. Let d_0 be in D and $\mathcal{I}(p(x))$ be $\mathcal{I}_v(x) = d_0$. Then $\mathcal{I} \models A \supset B$ means $(\mathcal{I}_v(x) = d_0)$ implies $(\forall d \ (d = d_0))$, which is not true!

Problem 3. Find a prenex form wff equivalent to the following (from Manna Prob. 2-13(c)):

$$(\forall x \, p(x)) \supset \exists x [\forall z \, q(x, z) \vee \forall z \, r(x, y, z)]$$

We will write successive equivalent forms of the previous wff.

$(\forall x \, p(x)) \supset \exists x [\forall z \, q(x, z) \vee \forall z \, r(x, y, z)]$

$\neg(\forall x \, p(x)) \vee (\exists x_1 [\forall z \, q(x_1, z) \vee \forall z_1 \, r(x_1, y, z_1)])$

$\exists x \exists x_1 \forall z \forall z_1 \ \ \neg p(x) \vee (q(x_1, z) \vee r(x_1, y, z_1))$

Problem 4. Let $A(z_1, z_2, z_3)$ be a wff with free variables z_1, z_2, z_3 and no occurrences of the variables x_1, x_2, y or of the binary function constant g.

(a) Suppose $\mathcal{I} \models \forall x_1 \forall x_2 \exists y\, A(x_1, x_2, y)$. Describe an $\mathcal{I}'$ such that $\mathcal{I}' \models \forall x_1 \forall x_2\, A(x_1, x_2, g(x_1, x_2))$.

For convenience, call B_1 the wff $\forall x_1 \forall x_2 \exists y\, A(x_1, x_2, y)$ and B_2 the wff $\forall x_1 \forall x_2\, A(x_1, x_2, g(x_1, x_2))$. By asumption, $\mathcal{I} \models B_1$. So by definition, for all $d_1, d_2 \in D$, there is a d_3 in D so that:

$$\mathcal{I}(x_1 \rightarrow d_1, x_2 \rightarrow d_2, y \rightarrow d_3) \models A(x_1, x_2, y).$$

So for each $d_1, d_2 \in D$, let $\eta(d_1, d_2)$ be a possible d_3. Then $\eta : D \times D \longrightarrow D$ is total since there is always such a d_3. Finally let

$$\begin{aligned} D' &= D \\ \mathcal{I}'_v &= \mathcal{I}_v \\ \mathcal{I}'_c &= \mathcal{I}_c[g \mapsto \eta]. \end{aligned}$$

(b) Conclude that $\forall x_1 \forall x_2 \exists y\, A(x_1, x_2, y)$ and $\forall x_1 \forall x_2\, A(x_1, x_2, g(x_1, x_2))$ are equisatisfiable.

- In a) we established that if B_1 is satisfiable then so is B_2.
- As
$$\models A(x_1, x_2, g(x_1, x_2)) \supset \exists y A(x_1, x_2, y),$$
we have that $\forall \mathcal{I}\ (\mathcal{I}(B_2) \supset \mathcal{I}(B_1))$, which trivially implies that if B_2 is satisfiable, then so is B_1.

Solutions to Quiz 3

Problem 1. Consider the two following wffs:

$$A_1: \quad \forall x\,[p(x) \supset q(x)]$$
$$A_2: \quad (\forall x\; p(x)) \supset (\forall x\; q(x))$$

(a) Show that A_1 implies A_2.

We shall give two solutions.

- Suppose that A_1 does not imply A_2, *i.e.*, that there is some interpretation $\mathcal{I}$ s.t. $\mathcal{I}(A_1)$ is true but $\mathcal{I}(A_2)$ is false. $\mathcal{I}(A_2)$ is false iff $\mathcal{I}(\forall x\; p(x))$ is true whereas $\mathcal{I}(\forall x\; q(x))$ is false. Then $\exists d \in D$ s.t. $\mathcal{I}_c(q)(d)$ is false and $\mathcal{I}_c(p)(d)$ is true. But $\mathcal{I}(A_1)$ says that $\mathcal{I}(\forall x(p(x) \supset q(x)))$ is true. Hence $\mathcal{I}_c(p)(d) \supset \mathcal{I}_c(q)(d)$, so that $\mathcal{I}_c(q)(d)$ should be true, which is a contradiction. Hence A_1 implies A_2.
- In this solution we proceed using systematic rules as described in Manna.

(1)	$\forall x\,[p(x)$	$\supset$	$q(x)]$	hypothesis
(2)	$p(x)$	$\supset$	$q(x)$	$\forall$-elimination
(3)	$(\forall x\; p(x))$	$\supset$	$(\forall x q(x))$	$\forall\forall$-introduction (Manna p. 120).

If you do not feel comfortable with (3) which is just an exercise in Manna, proceed as follows from (2):

(3)	$(\forall x\; p(x))$	$\Rightarrow$	$(p(x) \supset q(x))$	assumption introduction from (2)
(4)	$(\forall x\; p(x))$	$\Rightarrow$	$(\forall x\; p(x))$	assumption axiom
(5)	$(\forall x\; p(x))$	$\Rightarrow$	$p(x)$	$\forall$-elimination from (4)
(6)	$(\forall x\; p(x))$	$\Rightarrow$	$q(x)$	$\supset$ -elimination from (3,5)
(7)	$(\forall x\; p(x))$	$\Rightarrow$	$(\forall x\; q(x))$	$\forall$-introduction fom (6)
(8)			$(\forall x\; p(x)) \supset (\forall x\; q(x))$	$\supset$ -introduction from (7).

(b) Give a countermodel showing that A_2 does not imply A_1.

Consider a model $\mathcal{M}$ such that D has at least two elements $d_1 \neq d_2$, and such that $\mathcal{I}_c(p)(d)$ is "$d = d_1$" and $\mathcal{I}_c(q)(d)$ is "$d = d_2$". Then $\mathcal{I}(A_1)$ is false, and the premises of $\mathcal{I}(A_2)$ being false, $\mathcal{I}(A_2)$ is vacuously true.

Problem 2. Let A and B be two wffs.

(a) Prove that if $\models A \supset B$, then $A \models B$.

- $\models A \supset B$ means (Assertion 1): if $(\mathcal{M}, \mathcal{I}_v) \models A$, then $(\mathcal{M}, \mathcal{I}_v) \models B$, for all $\mathcal{M}, \mathcal{I}_v$.

- $A \models B$ means (Assertion 2): if $\mathcal{M} \models A$, then $\mathcal{M} \models B$ for all $\mathcal{M}$, *i.e.*, if $(\mathcal{M}, \mathcal{I}_v) \models A$ for all $\mathcal{M}, \mathcal{I}_v$, then $(\mathcal{M}, \mathcal{I}_v) \models B$ for all $\mathcal{M}, \mathcal{I}_v$.

Suppose for contradiction that Assertion 1 does not imply Assertion 2, *i.e.*, Assertion 1 is true whereas Assertion 2 is false. Assertion 2 is false iff there exists some $\mathcal{M}_0$ such that $(\mathcal{M}_0, \mathcal{I}_v) \models A$ for all $\mathcal{I}_v$, but it is not true that $(\mathcal{M}_0, \mathcal{I}_v) \models B$ for all $\mathcal{I}_v$. The last fact happens iff there is a valuation $\mathcal{I}_{v,0}$ such that $(\mathcal{M}_0, \mathcal{I}_{v,0}) \not\models B$. But $(\mathcal{M}_0, \mathcal{I}_{v,0}) \models A$, so from Assertion 1, $(\mathcal{M}_0, \mathcal{I}_{v,0}) \models B$ also, a contradiction!

(b) Explain why if A is closed, then $(\mathcal{M}, \mathcal{I}_v) \models A$ iff $\mathcal{M} \models A$.

Let $\mathcal{I}_{v,0}$ be any specific valuation. We want to prove that $(\mathcal{M}, \mathcal{I}_{v,0}) \models A$ iff $\mathcal{M} \models A$.

- $\mathcal{M} \models A$ means $\forall \mathcal{I}_v \ (\mathcal{M}, \mathcal{I}_v) \models A$, so that the implication $\Leftarrow$) is obvious.
- The implication $\Rightarrow$ follows from lemma 1 of handout 11, page 5.

(c) Conclude that if A is closed, then $\models A \supset B$ iff $A \models B$.

We saw in part (a) that $\models A \supset B$ implies that $A \models B$. Suppose now that $A \models B$. To show $\models A \supset B$ let $(\mathcal{M}, \mathcal{I}_v)$ be any interpretation such that $(\mathcal{M}, \mathcal{I}_v) \models A$, and we must show that $(\mathcal{M}, \mathcal{I}_v) \models B$. But since A is closed, we have by part (b) that $\mathcal{M} \models A$. Now, since $A \models B$, we conclude that $M \models B$, and hence in particular that $(\mathcal{M}, \mathcal{I}_v) \models B$.

(d) Give an example of A, B such that $A \models B$, but it is not true that $\models A \supset B$.

Consider $A : p(x)$ and $B : \forall x \ p(x)$. Then $A \vdash B$ by $\forall$ introduction. So $A \models B$ by soundness of the rules. To see that A does not imply B, consider an interpretation for which the domain, D, has at least two elements. Fix $d_0 \in D$ and let $\mathcal{I}_c(p)(d)$ be **true**iff $d = d_0$. Let $\mathcal{I}_v(x) = d_0$. Then $\mathcal{I} \models A \supset B$ means that $\mathcal{I}(p(x))$, which simplifies to the assertion $d_0 = d_0$ and hence is true, implies $\mathcal{I}(\forall x \, p(x))$, which simplifies to the assertion that $d = d_0$ for all $d \in D$, which is false since D has some element other than d_0. So true implies false, a contradiction.

Problem 3. Find a prenex form wff equivalent to the following (from Manna Prob. 2-13(c)):

$$(\forall x \, p(x)) \supset \exists x [\forall z \, q(x, z) \vee \forall z \, r(x, y, z)]$$

We will write successive equivalent forms of the previous wff.

$$(\forall x \, p(x)) \supset \exists x_1 [\forall z \, q(x_1, z) \vee \forall z_1 \, r(x_1, y, z_1)]$$

$$\neg(\forall x \, p(x)) \vee (\exists x_1 [\forall z \, q(x_1, z) \vee \forall z_1 \, r(x_1, y, z_1)])$$

$$\exists x \exists x_1 \forall z \forall z_1 \ \neg p(x) \vee q(x_1, z) \vee r(x_1, y, z_1)$$

Problem 4. Let $A(z_1, z_2, z_3)$ be a wff with free variables z_1, z_2, z_3 and no occurrences of the variables x_1, x_2, y or of the binary function constant g.

(a) Suppose $\mathcal{I} \models \forall x_1 \forall x_2 \exists y\, A(x_1, x_2, y)$. Describe an $\mathcal{I}'$ such that $\mathcal{I}' \models \forall x_1 \forall x_2\, A(x_1, x_2, g(x_1, x_2))$.

For convenience, call B_1 the wff $\forall x_1 \forall x_2 \exists y\, A(x_1, x_2, y)$ and B_2 the wff $\forall x_1 \forall x_2\, A(x_1, x_2, g(x_1, x_2))$. By asumption, $\mathcal{I} \models B_1$. So by definition, for all $d_1, d_2 \in D$, there is a d_3 in D so that:

$$\mathcal{I}(x_1 \to d_1, x_2 \to d_2, y \to d_3) \models A(x_1, x_2, y).$$

So for each $d_1, d_2 \in D$, let $\eta(d_1, d_2)$ be a possible d_3. Then $\eta : D \times D \longrightarrow D$ is total since there is always such a d_3. Finally let

$$\begin{aligned} D' &= D \\ \mathcal{I}'_v &= \mathcal{I}_v \\ \mathcal{I}'_c &= \mathcal{I}_c[g \mapsto \eta]. \end{aligned}$$

(b) Conclude that $\forall x_1 \forall x_2 \exists y\, A(x_1, x_2, y)$ and $\forall x_1 \forall x_2\, A(x_1, x_2, g(x_1, x_2))$ are equisatisfiable.

- In (a) we established that if B_1 is satisfiable then so is B_2.
- $A(x_1, x_2, g(x_1, x_2))$ obviously implies $\exists y A(x_1, x_2, y)$. Hence, B_2 implies B_1 (by $\forall$-intro and Problem 1, if this does not seems equally obvious), which trivially implies that if B_2 is satisfiable, then so is B_1.

Problem Set 5

Problem 1. (Compactness) Show that any finitely satisfiable set of first-order wff's is satisfiable. *Hint*: An easy corollary of the lemmas in class about Herbrand models.

Problem 2. Prove that there is a model $\mathcal{M}$ such that $\mathcal{M} \models A$ for every *first-order* wff A valid in $\langle \{\{\mathtt{a}, \mathtt{b}\}^*\}, \cdot \rangle$, but such that $\mathcal{M}$ is not isomorphic to $\langle \{\{\mathtt{a}, \mathtt{b}\}^*\}, \cdot \rangle$. *Hint*: Compactness.

Some examples of Herbrand's procedure.

1 The theory

Let us first recall the results seen in Handout 11. Let A and B be wff's (not necessarily over the same signature).

Definition 1. A and B are *equisatisfiable* iff

A is satisfiable iff B is satisfiable.

Definition 2. A and B are *equivalent* iff for every interpretation $\mathcal{I}$ which assigns meaning to the symbols of A and B, $\mathcal{I} \models A \quad \text{iff} \quad \mathcal{I} \models B$.

We will be interested in various effective transformations between wff's which have the property of yielding a wff either equivalent to the original one or at least equisatisfiable with it. The first lemma in this vein is about the reduction to prenex form.

Lemma 1. There is an effective procedure to transform any given wff into an equivalent wff in prenex form and with the same signature.

Further simplifications of the structure of a wff are possible if one asks only to preserve satisfiability.

Lemma 2. There is an effective procedure to transform any given first-order wff into an equisatisfiable closed first-order wff in universal form and with the same signature.

The technique used to reduce a wff A in prenex form to an equisatisfiable universal wff B is called *Skolemization*. For this reason, B is sometimes called the *Skolemized* form of A.

Vocabulary: A closed term is also referred to as a *ground term*.

Let A be a first-order universal wff, $\forall x_1 \ldots \forall x_n B$ where B is quantifier-free, and let $\mathcal{S}$ be the signature which consists of all the nonlogical symbols occurring in A. If A has no individual constants, add a new individual constant c to $\mathcal{S}$.

Definition 3. A *ground instance* of A is a wff obtained by replacing each free occurrence of a variable in B by a ground term over $\mathcal{S}$. That is, a ground instance of A is a wff of the form $[B]^{t_1,\ldots,t_n}_{x_1,\ldots,x_n}$ where $t_1, \ldots, t_n$ are ground terms of A.

We consider next the notion of propositional satisfiability of a conjunction of ground instances of A. Let $G_1 \wedge \ldots \wedge G_n$ be a finite conjunction of ground instances of A. Treat each distinct atomic subwff of the wff's $G_1, \ldots, G_n$ as a propositional variable and assign it the value `true` or `false`. The wff $G_1 \wedge \ldots \wedge G_n$ now has a `true` or `false` value determined by the assignments to the atomic subwff's and the rules of propositional logic.

Definition 4. $G_1 \wedge \ldots \wedge G_n$ is *propositionally satisfiable* iff there is an assignment of truth values to the atomic subwff's under which $G_1 \wedge \ldots \wedge G_n$ has the value `true`. $G_1 \wedge \ldots \wedge G_n$ is *propositionally unsatisfiable* iff it is not propositionally satisfiable.

Theorem 1. (Herbrand's theorem) A first-order universal wff is (*logically*) unsatisfiable iff there is a finite conjunction of ground instances of the wff which is *propositionally* unsatisfiable.

Theorem 1 provides a semi-decision procedure for the validity problem of the first order predicate calculus.

To summarize, for a *first-order wff, A_0, without equality*, the different steps involved in Herbrand's procedure are:

- Take the universal closure of A_0 getting A_1 (which is equivalid with A_0).
- Rename all the different bound variables of A_1 so that they are distinct from each other; call this A_2 (which is equivalent to A_1).
- Using the fact that any wff of the type "$A \supset B$" is equivalent to the wff "$\neg A \vee B$", rewrite A_2 into a wff A_3 having no occurrences of "$\supset$" and "$\equiv$", and which is equivalent to A_2. (This step need not be done for those occurrences of $\supset$ or $\equiv$ all of whose subwff's are quantifier-free.)
- Let A_4 be the negation of A_3. So A_4 is unsatisfiable iff A_3 is valid.
- Push all the negation-symbols "$\neg$" inside their respective sub-wffs, getting A_5 equivalent to A_4, in which the only negations which occur are of atomic formulas.
- Move all quantifiers in A_5 to the left, obtaining A_6 in prenex form equivalent to A_5.
- Skolemize A_6, getting A_7, equisatisfiable with A_6. (Manna offers an alternative, more efficient Skolemization method which does not require putting in prenex form. You are welcome to use, but are not required to learn Manna's method.)
- Generate all ground instances of the matrix of A_7 looking a for a finite set of them which is propositionally unsatisfiable. By Theorem 1, such a finite set will be found iff A_7 is logically unsatisfiable, *viz.*, false in all interpretations.

2 Example of a valid wff

For readability we write $(pfff(x))$ instead of $p(f(f(f(x))))$, etc., and will use the symbols [,{ as an alternative to (.

$$
\begin{array}{ll}
A_0: & [\forall x\ p(x) \equiv \neg p(f(x))] \supset [p(x) \equiv \neg p(fff(x))] \\
A_1: & \forall x\ \{(\forall x\ [p(x) \equiv \neg p(f(x))]) \supset [p(x) \equiv \neg p(fff(x))]\ \} \\
A_2: & \forall z\ \{\ (\forall x\ [p(x) \equiv \neg p(f(x))]) \supset [p(z) \equiv \neg p(fff(z))]\ \} \\
A_3: & \forall z\ \{\ \neg\ (\forall x\ [p(x) \equiv \neg p(f(x))]) \vee [p(z) \equiv \neg p(fff(z))]\ \} \\
A_4: & \neg A_3 \\
A_5: & \exists z\ \{\ \forall x\ [p(x) \equiv \neg p(f(x))] \wedge [p(z) \equiv p(fff(z))]\ \} \\
A_6: & \exists z\, \forall x\ \{\ [p(x) \equiv \neg p(f(x))] \wedge [p(z) \equiv p(fff(z))]\} \\
A_7: & \forall x\ \{\ [p(x) \equiv \neg p(f(x))] \wedge [p(c) \equiv p(fff(c))]\ \}
\end{array}
$$

The matrix of the A_7 is:

$$[p(x) \equiv \neg p(f(x))] \wedge [p(c) \equiv p(fff(c))]\,.$$

Let us build the 3 ground instances G_1, G_2, G_3 deduced from it by substituting successively $c, f(c), f(f(c))$ to x:

$$
\begin{array}{ll}
G_1: & [p(c) \equiv \neg p(f(c))] \wedge [p(c) \equiv p(fff(c))], \\
G_2: & [p(f(c)) \equiv \neg p(ff(c))] \wedge [p(c) \equiv p(fff(c))], \\
G_3: & [p(ff(c)) \equiv \neg p(fff(c))] \wedge [p(c) \equiv p(fff(c))].
\end{array}
$$

Then the conjunction $G_1 \wedge G_2 \wedge G_3$ of these three ground instances is:

$$
\left\{
\begin{array}{l}
[p(c) \equiv \neg\ \ p(f(c))] \wedge \\
\qquad [p(f(c)) \equiv \neg\ \ p(ff(c))] \wedge \\
\qquad\qquad [p(ff(c)) \equiv \neg\ \ p(fff(c))] \wedge [p(c) \equiv p(fff(c))].
\end{array}
\right.
$$

But, using the transitivity of the binary relation $\equiv$, and the general fact that the wffs $A \equiv B$ and $\neg A \equiv \neg B$ are equivalent, we see that,

$$\models [G_1 \wedge G_2 \wedge G_3] \supset [p(c) \equiv \neg p(fff(c))] \wedge [p(c) \equiv p(fff(c))],$$

which shows that $(G_1 \wedge G_2 \wedge G_3)$ is not propositionally satisfiable. Hence A_0 is valid.

3 Example of a non valid wff

$$
\begin{array}{ll}
A_0: & (\exists x\ p(x,x)) \vee [\forall x\, \exists y\ p(x,y) \supset \exists z\ p(z,0)] \\
A_2: & (\exists x_1\ p(x_1,x_1)) \vee [\forall x\, \exists y\ p(x,y) \supset \exists z\ p(z,0)] \\
A_3: & [\exists x_1\ p(x_1,x_1)] \vee \neg[\forall x\, \exists y\ p(x,y)] \vee [\exists z p(z,0)] \\
A_5: & [\forall x_1\ \neg p(x_1,x_1)] \wedge [\forall x\, \exists y\ p(x,y)] \wedge [\forall z \neg p(z,0)] \\
A_6: & \forall x\, \exists y\, \forall x_1\, \forall z\, [\neg p(x_1,x_1) \wedge p(x,y) \wedge \neg p(z,0)] \\
A_7: & \forall x\, \forall x_1\, \forall z\ [\neg p(x_1,x_1) \wedge p(x,f(x)) \wedge \neg p(z,0)]
\end{array}
$$

The matrix of the previous expression is

$$\neg p(x_1, x_1)) \wedge p(x, f(x)) \wedge \neg p(z, 0)).$$

As there is only one function f, we see that all the ground terms are $0, f(0), f(f(0)), \ldots$. Let us consider propositional truth assignment $\mathcal{I}$ which assigns values as follows; for any arbitrary ground term t,

$$\begin{array}{ll} I(p(t,t)) & \text{is } \texttt{false}, \\ \mathcal{I}(p(t,0)) & \text{is } \texttt{false}, \\ \mathcal{I}(p(t,f(t))) & \text{is } \texttt{true}, \end{array}$$

and all the $\mathcal{I}(p(t,t_1))$'s not considered through the three previous allocations are given the arbitrary `true` value. This assignemnt is well-defined because the cases of atomic formulas with arguments matching (t,t) and $(t,0)$ can overlap only in the case of t being the ground term 0, and in this case the truth values assigned $p(t,0)$ and $p(t,t)$ are compatible, namely true. We see that under this interpretation $\mathcal{I}$, all the ground instances are *simultaneously* true. Herbrand's theorem allows us to conclude that A_0 is not valid, since we will never find a propositionally unsatisfiable set of ground instances of A_7.

Master

6.044J/18.423J: Computability, Programming, and Logic
Massachusetts Institute of Technology

Handout 26
14 December 1988

Quiz 4

Instructions. There are two problems of equal weight. This exam is *open book.* In doing part of a problem, you may assume the results of previous parts. You have 50 minutes. Good luck.

Problem 1. Define a *u-wff* to be a closed universal first-order wff without equality, and a *uf-wff* to be a u-wff in which no function symbols appear.

(a) Explain why the set of satisfiable u-wffs is not r.e.

(b) Explain why the set of valid u-wffs is decidable.

(c) Explain why the set of satisfiable uf-wffs is decidable.

Problem 2. We defined a (first-order) partial correctness assertion (pca) to be a triple $\{A\}W\{B\}$ where A and B are (first-order) wff's and W is a while-program scheme. We also defined satisfaction for pca's by the condition that $\mathcal{I} \models \{A\}W\{B\}$ iff either $\mathcal{I} \not\models A$ or else W halts under interpretation $\mathcal{I}$ and $[\![W]\!](\mathcal{I}) \models B$, where $[\![W]\!](\mathcal{I})$ is the interpretation left when W halts.

For example, $\{A\}W\{\texttt{false}\}$ is valid iff W diverges (runs forever) under every interpretation satisfying A.

Let W be the while-program

$$x := c;$$
$$\textit{while } p(x) \textit{ do } x := g(x);\ x := g(x) \textit{ od},$$

(a) For each $n \geq 0$, describe a first-order wff L_n which is satisfied by precisely those interpretations for which W executes the body of its while-loop *at least* n times.

Hint: $L_0 \equiv \texttt{true}$ and $L_1 \equiv p(c)$. Let $g^0(t)$ denote t, and $g^{n+1}(t)$ denote $g(g^n(t))$ for any term t.

(b) Let T be the infinite set of formulas $\neg(\{\texttt{true}\}W\{\texttt{false}\}), L_0, L_1, \ldots, L_n, \ldots$. Explain why T is finitely satisfiable.

(c) Conclude from part 2(b) that the pca $\{\texttt{true}\}W\{\texttt{false}\}$ is not equivalent to any first-order wff. *Hint*: Compactness would be contradicted.

END

Sept 16, 85

future topics.

1°) • (discussion about) understanding quantifiers.
2°) • computability
3°) • How to prove a program is correct

Computability Theory

the theory is not caring about efficiency - (Ex: exhaustive search is a good way!)
Actually it is more involved into proving what is NOT computable
(Ex: Hilbert's 10th problem).

1°) the book begins with:
Simulation thesis says: set of computable functions are the same for Turing machines...

2°) the halting function is not computable.

Scheme programs. (S.C)

3 many ways to encode. We'll use $\{a,b\}^*$ as the code for an ASCII character.
A S.C is ! encoded as a sequence of a and b's.

Scheme procedure:

notations:
F with one formal character.
d(F) $\stackrel{\text{def}}{=}$ string $\in \{a,b\}^*$ which codes F.
$\{a,b\}^* \stackrel{\text{def}}{=}$ all sequences of finite length...
λ = finite word.

Property: $\exists!$ 1-1, onto function : $\{$Schemes$\}$ $\longrightarrow$ $\{$encoded sequences, $\{a,b\}^*\}$

for $x \in \{a,b\}^*$ let $F_x \stackrel{\text{def}}{=} \varphi^{-1}(x)$.
precisely: • let $\mathcal{F}_x$ be the partial function : $\{a,b\}^* \longrightarrow \{a,b\}^*$ computed by the scheme procedure coded by x.
• If x is a well formed code, let $\mathcal{F}_x(y)$ = undefined (say the constant function a).

for all y, if x is not a code

Notations:

$F \in \{$ string of ASCII with parses as scheme $\}^{def} = \{$ ~~scheme~~ $\}$. SCHEME.

$\mathcal{F}: \{0,1\}^* \dashrightarrow \{0,1\}^*$. partial function

$d: \{$ scheme $\} \xrightarrow{1-1} \{0,1\}^*$. (The encoding of a scheme).

Interpret_0: SCHEME $\xrightarrow{\mathcal{J}_0}$ $\{$ functions: $\{0,1\}^* \xrightarrow[\text{partial}]{} \{0,1\}^*$ that are computed by the schemes $\}$.

Take. $F \in$ SCHEME. Then:

$\mathcal{F}_{d(F)}$ is the object that Interpret_0 is obtaining

$\mathcal{F}_{d(F)} = \mathcal{J}_0(F)$

$H: \{0,1\}^* \longrightarrow \{0,1\}^*$

where $H(x) = a$ iff $\mathcal{F}_x(x)$ is defined.

$= b$ ——— undefined.

Thm: $\nexists x \in \{0,1\}^*$ s.t. $H = \mathcal{F}_x$

Halting function:

$x \in \{a,b\}^*$

$$H(x) = \begin{cases} a & \text{if } \mathcal{F}_x(x) \text{ is defined.} \quad (\text{alg}) \; x \in \text{Dom}(\mathcal{F}_x) \\ b & \text{otherwise.} \end{cases}$$

Thm: H is not computable

((e) in any language it is not programmable)

proof: if H was computable by a Scheme procedure denoted by Halt

(so that (Halt "aba") gives H(aba).)

let us consider

```
( Define (Self x)
  ( If ( (Halt x) = "a" ) ( Self x ) ("abb") ) )
```

(e) runs forever, diverges.

else

Let s be the binary code of procedure Self and consider $\mathcal{F}_s$; then:

$\forall x \in \{a,b\}^*$ ~~$\mathcal{F}_s(x)$ is defined iff~~ $\mathcal{F}_s(x) = abb$ if $H(x) = b$

if $\mathcal{F}_x(x)$ undefined. (by definition).

Now take $x = s$. Then:

$\mathcal{F}_s(s)$ is defined iff $\mathcal{F}_s(s) = abb$ if $\mathcal{F}_s(s)$ undefined.

Contradiction.

※.

Philosophie of the result:

It is the answer b to H(x) which is delicate.

Diagonal argument of G. Cantor:

Define an order on $\{a,b\}^*$:

$x \leq y$ iff $|x| < |y|$

or ($|x| = |y|$ and $x \leq y$ lexicographically).

strings	Λ	a	b	aa	ab	ba	bb	aaa ...	y.
Λ									
a									
b									
aa									
ab									
ba									
aaa									
.									
.									
x									

$\bar{\sigma}_x(y)$

Put $\bar{\sigma}_x(y)$ if it is defined
c if it is undefined.

In the proof of Thm,
We did:

if $\sigma_{x_i}(x_j)$ is defined put c in box i
if ——— undefined put abb ——— | denote this by α_i.

so that the row $\bar{\sigma}_{x_i}(\alpha_i)_i$ is not a row of the matrix.

6.044

9/23 LAST TIME: HALTING PROBLEM ~~[illegible]~~ UNDECIDABLE
THIS TIME: CODING. PROPERTIES + SUCH.
WE WON'T LOOK AT CODING AFTER TODAY.

THM $K_1 = \{M : M\downarrow \text{ on } d(M)\}$
K_1 IS NOT DECIDABLE

(A RESTATEMENT OF WHAT WE'VE BEEN DOING, RELATED TO →

3 LECTURES AGO:
$H(x) = \begin{cases} a & \text{if } F_x(x) \text{ DEFINED} \\ b & \text{OTHERWISE} \end{cases}$
$x \in \{a,b\}^*$
UNDECIDABLE

PROPERTY OF MACHINE HERE. BEFORE, PROPERTIES OF LANGUAGES.
THM IS ACTUALLY

$\{d(M) : M\downarrow \text{ on } d(M)\}$
($d(M)$: ENCODING OF A TURING MACHINE)
A MACHINE THAT HALTS ON ITS OWN ENCODING

WE ARE ALWAYS SPEAKING OF PROPERTIES OF STRINGS, NOT MACHINES.
ORIGINAL STATEMENT OF THM WAS SLOPPY.

NOTE $K_1 = K_{1,d}$ ~~d'~~
IF d' IS ANOTHER CODING SCHEME OF TURING MACHINES THEN $K_{1,d} \neq K_{1,d'}$.
BUT $K_{1,d}$ AND $K_{1,d'}$ ARE NOT DECIDABLE.

DEFN A IS RECURSIVE IFF A IS R.E.
$\bar{A}$ IS R.E. ($\equiv A$ IS CO-RE)

TO BE PRECISE, K_1 IS NOT CO-RE
PROOF: IF IT WAS, THERE WOULD BE A T.M. M_1 THAT WOULD ACCEPT $\bar{K}_1$.
ie, $\bar{K}_1 = \text{DOMAIN}(M_1)$

$M \in \bar{K}_1 \Leftrightarrow M_1\downarrow \text{ on } M$
$\Leftrightarrow M\uparrow \text{ on } M$

SET $M_1 = M$ ↯ SO K_1 NOT CO-RE

IF WE CHANGED d TO d', WOULD GET ANOTHER LANGUAGE
$\Rightarrow \exists \infty$ SET OF UNDECIDABLE LANGUAGES

PROP FOR REASONABLE d K_1 IS R.E.
MORE PRECISELY $\exists$ v: UNIVERSAL MACHINE
$\exists z_\psi = d(M)$ GÖDEL NUMBER, A WAY TO ENCODE ANY T.M.

$v(z_\psi, x) = M(x) \in \{a,b\}^* \; \forall x \in \{a,b\}^*$

SO K_1 IS R.E. BECAUSE $M(d(M)) = v(d(M), d(M)) \in \{a,b\}^*$

K_1 NEVER CO-RE. K_1 MAY NOT BE RE IF d IS 'UNREASONABLE'

v HALTS ON THIS INPUT $\Rightarrow M \in K_1$

L RE $\Rightarrow \exists M$ WILL STOP SOMETIME ON ANY INPUT IN L.

NO WAY TO KNOW THE MACHINE WILL STOP ON A STRING NOT IN L

GIVING A CANONICAL ORDERING TO STRINGS IN A LANGUAGE

ORDERING OF $\{a,b\}^*$

$a, b, aa, ab, ba, bb, aaa, \ldots$

DEFN $x \leq y$ IFF $|x| < |y|$

OR $x \leq y$ LEXICOGRAPHICALLY.

CHARACTERIZATION OF A DECIDABLE SET

PROP A DECIDABLE $\Leftrightarrow \exists$ STRICTLY INCREASING (TOTAL, COMPUTABLE) FN $f: \{a,b\}^* \to \{a,b\}^*$ s.t. $A = \text{RANGE}(f)$

⇒ PRESENTED W/ ANY INPUT, GET SOME RESULT

STRICTLY INCREASING: $x < y \Rightarrow f(x) < f(y)$

HOW TO USE f TO DETERMINE MEMBERSHIP IN A SET?

CONSIDER M THAT COMPUTES f.

RUN M ON ALL POSSIBLE INPUTS SUCCESSIVELY

IF M OUTPUTS x AT SOME POINT, $x \in A$

IF M OUTPUTS $y > x$, $x \notin A$ (AND x HAS NOT APPEARED AS AN OUTPUT)

IF A LANGUAGE IS FINITE, IT IS DECIDABLE

EX: POLYNOMIAL W/ COEFFICIENTS IN $\mathbb{Z}$ WITH INTEGRAL ROOTS.

$E = \{P \in \mathbb{Z}[x] : P \text{ HAS A ROOT IN } \mathbb{Z}\}$ (SINGLE-VARIABLE POLYNOMIALS)

$x - 2 \in E$

$x - \frac{3}{2} \notin E$

E IS R.E. BUT NOT CO-RE

↳ VERY HARD RESULT, OPEN FOR YEARS

WE KNOW IT'S RE:

COMPUTE P(EVERY INTEGER). IF GET ZERO, THAT'S THE ROOT

6.044

9/26 CLARIFICATION OF LAST LECTURE

IN PROVING THAT K_1 IS NOT CO-RE (BY CONTRADICTION)

"WHY ARE WE JUSTIFIED IN SAYING BOTH $M \in K_1$ AND SETTING $M_1 = M$?"

REDO THE PROOF:

ASSUME K IS CO-RE — $K_1 = \{M : M\downarrow \text{ on } d(M)\}$

i.e. $\overline{K_1}$ IS RE — $\overline{K_1} = \{a,b\}^* \setminus K_1$

$\exists M_1 : M_1 \downarrow \text{on } \overline{K_1}$ (AND $M_1 \uparrow$ ON K_1)

$M \in \overline{K_1} \Leftrightarrow M \notin K_1$

$\Leftrightarrow M\uparrow$ ON $d(M)$

$M\uparrow$ ON $d(M) \Leftrightarrow M_1\downarrow$ ON $d(M)$ FOR ANY TURING MACHINE M (IN K_1 OR NOT)

SO, SET $M = M_1$.

$*$

IF IN $\mathbb{Q}$, CAN CODE W/FINITE AMT OF INFO. A TECHNICALITY.

EX: $A_1, A_2, A_3 \in M_2(\mathbb{R})$ 2×2 MATRICES — GENERAL MATRICES DON'T COMMUTE

$E = \{\prod_1^n A_i : (\prod_1^n A_i)_{1,1} = 0\}$

E IS RE, NOT CO-RE

CONSIDER THE SET OF ALL TURING MACHINES (OVER FINITE ALPHABETS, ELSE NOT COUNTABLE)

ENUMERATE THEM $(M_i)_{i=1,2\ldots}$

CONSIDER THE FUNCTION F (A REWRITING OF THE HALTING PROBLEM)

↳ $\Rightarrow F$ NOT COMPUTABLE

$$F(M_i) = \begin{cases} 0 & \text{IF } M_i\uparrow \text{ ON } d(M_i) \\ 1 & \text{IF } M_i\downarrow \text{ ON } d(M_i) \end{cases}$$

CONSIDER $\sum_{i=1}^{\infty} 2^{-i} \cdot F(M_i) = \alpha$ IF $\forall i$ $F(M_i)=0$, SUM IS 0

WE'LL USE THE FACT THAT $\forall x \in [0,1]$, x CAN BE WRITTEN $x = \sum_{i=1}^{\infty} 2^{-i} a_i(x)$

WHERE $a_i(j) = 0$ OR 1

GIVEN α, KNOW SOMETHING ABOUT THE BEHAVIOR OF EVERY T.M.

BUT NO MACHINE CAN COMPUTE α

WRITE α AS $0.\sigma_1\sigma_2\sigma_3\ldots$

CONSIDER THE SET $E = \{\sigma_1, \sigma_1\sigma_2, \sigma_1\sigma_2\sigma_3 \ldots\}$

} DECIMAL EXPANSION DIGITS FROM 0 TO 9

CLAIM E IS NOT DECIDABLE

ELSE α WOULD BE.

NOW CONSIDER THE SET $\{\alpha_i : \alpha_i$ IS SOME WFF USED IN $\alpha\}$
THIS SET IS FINITE AND SO MUST BE DECIDABLE.

THEOREM 5: WAYS FOR A SET TO BE R.E.

THESE ARE EQUIVALENT:

1) A IS R.E.
2) A IS IN THE DOMAIN OF A PARTIAL RECURSIVE FN (OF 1 ARGUMENT).
3) A IS IN THE RANGE OF A PARTIAL RECURSIVE FN.
4) A IS IN THE RANGE OF A TOTAL RECURSIVE FN.

OUR DEFN: A R.E. $\Leftrightarrow \exists M_A$: M_A HALTS EXACTLY ON THE STRINGS IN A
$\Leftrightarrow$ A IS THE DOMAIN OF M_A

PROVE $1 \Rightarrow 2$

CONSTRUCT $f : \{a,b\}^* \to \{a,b\}$ PARTIAL FUNCTION
S.t. $Dom(f) = A$

I HAVE MACHINE M_A.

CONSIDER THE MACHINE N_A: WHENEVER M_A HALTS, N_A GIVES OUTPUT a.

PROVE $2 \Rightarrow 1$

IF, FOR INSTANCE, $f(x) = c$, THEN $x \in Dom(f)$

MACHINE M' COMPUTES f

CONSTRUCT N': $N'(x) = \begin{cases} M'(x) \text{ IF } M' \downarrow \text{ ON } x \text{ AND } M'(x) \in \{a,b\} \\ \text{DIVERGES OTHERWISE } (N' \uparrow \text{ ON } x) \end{cases}$

PROVE 4 (DOVETAILING)

What I covered on the 3rd and last lecture during the leave of Prof. MEYER

- A is an re language $\Rightarrow$ { $A = \emptyset$ or / A is the range of a total recursive function }

Recall that we can order all the inputs in the following way: $i_1 = a$, $i_2 = b$, $i_3 = aa$, $i_4 = ab$, $i_5 = ba$, $i_6 = bb$, $i_7 = aaa$, $i_8 = aab$ ...

proof | Assume $A \neq \emptyset$ / $(\exists)\ y_0 \in A$ | Let M_A halting on A

Construct the Turing machine N that will compute the total function f in the following way:

For any input x: Have M_A work on parallel on all inputs in the following way:

- Spend 1 unit of time on i_1 store the result
- then resume on i_1 / i_2
- then resume computation on i_1 / i_2 / i_3 spend one unit of time ...
- then spend 1 unit of time on i_1 / i_2 / i_3 / i_4
- i_1 / i_2 / i_3 / i_4 / i_5 .

... and stop after exactly $|x|$ unit of time.

If at this step M_A has just halted ~~on the computation~~ at the end of its work on some input i_n, then display this input. ($(\exists)$: $f(x) = i_n$). //

Else display y_0 ($(\forall)$ $f(x) = y_0$).

We check that :

- f is totally computable
- Range(f) = exactly A.

- Justification of $\leq_m$ in terms of K_1 and K_0.
- **Def of $\leq_m$** : It is a transitive relation
- **Prop** $A \leq_m B$

  B re (recursive) $\Rightarrow$ A re (recursive)
- **Prop** $A \leq_m B$

  A not re $\Rightarrow$ B not re
- **Ex** : formally : $K_1 \leq_m K_0$
- **Corollary** : K_0 is not co-RE

  proof uses:

  Lemma $A \leq_m B \iff \overline{A} \leq_m \overline{B}$
- **Note** : Although transitivity will hold for all orderings of languages ($\leq_T$) this last property is specific to $\leq_m$.

Meyer
dept 29
φ

Exos related to $\leq_m$:

$P = \{n \geq 2 ;\ n \text{ is prime}\}$. undecidable
$\leq_m \{4\}$.

- All recursive sets are many one reducible to each other.

A first approximation of $\leq_m$ means: " $A \leq_m B$ iff "A is simpler than B" "

Def: $A \equiv_m B$ iff $\begin{cases} A \leq_m B \\ B \leq_m A \end{cases}$.

- proof. Define $f(n)$ (the translator) by: $\begin{cases} f(n) = 4 & \text{if } n \text{ prime} \\ = 3 & \text{otherwise} \end{cases}$

We have to verify that.
1°) f is computable, total. obvious.
2°) $n \in P$ iff $f(n) \in \{4\}$. ——.

---

if $\begin{cases} A \\ B \end{cases}$ • $\neq \begin{cases} \emptyset \\ \text{everything} \end{cases}$
• both recursive
then $A \equiv_m B$. :

This works even we do not have any idea of whether question A or B are true or not

---

Lemma : if A is Re then $A \leq_m K_0$:

proof: let $A = \text{Dom}(M_A)$ for some Turing Machine M_A.

$x \in A$ iff $M_A \downarrow$ on x.
iff $(M_A, x) \in K_0$

Let $f(x) \overset{\text{def}}{=} (M_A, x)$ which is Hly computable.

※

$K_2 = \{M ;\ M \text{ halts on } \Lambda\}$. (Blank tape halting problem).

Lemma if R is re Then $R \leq_m K_2$:

Let $R = \text{dom}(M_R)$. Define a new T.M for each $x \in \{0,1\}^*$:

"Given input w, erase w, simulate M_R on input x"

Call this machine $M_{f(x)}$

$x \in R$ iff M_R halts on x iff $M_{f(x)}$ halts on some input iff ——— Λ (since $M_{f(x)}$ is constant w.r.t all inputs) iff $f(x) \in K_2$.

※

- Note: K_2 is Re
- Have to check that f is total computable.

Corollary: K_2 is not coRE

proof: $K_1 \leq_m K_2$

$K_{tot} = \{ M ;\ M \text{ halts on all inputs} \}$

Lemma: If K is Re then $K \leq_m K_{tot}$

Corollary: K_{tot} is not coRe

proof: exactly as above.

Lemma: $\overline{K} \leq_m K_{tot}$

Corollary: K_{tot} is not Re.

proof: Given input w, simulate M_x on input x for $|w|$ steps. If it has not halted yet then halt. Else diverge.

$M_{g(x)}$ →

(Thm Lemma): $\overline{K_1} \leq_m K_{tot}$

proof: $\forall x \in \{a,b\}^*$ consider a program that behaves as follows:

"Given input w, let $n = |w|$
Simulate M_x on input x for n steps
If it is not discovered to halt yet, then halt, else diverge".

Let $M_{g(x)}$ be a machine which meets the quoted specification.

$x \notin \overline{K_1}$ iff M_x does not halt on input x.
(to be precise: (02) x is not a code of a Machine)
iff $\forall n \geq 0$ M_x does not halt on input x in $\leq n$ steps.
iff $M_{g(x)}$ halts on all inputs.
iff $g(x) \in K_{tot}$.

Moreover g is total computable because the quoted specification is obviously "uniform" in x.

So $\overline{K_1} \leq_m K_{tot}$.

QED.

Rice's thm:

Def: A property of languages is not trivial on RE-sets iff { some re language has the pty ; some does not.

Ex: $P_1(\mathcal{L}) \stackrel{def}{=} [\Lambda \in \mathcal{L}]$

Def: $K_P = \{ M \mid P(\text{domain}(M)) \}$.

Ex: . $K_{P_1} = K_1$
. for $\mathcal{L} \subseteq \{a,b\}^*$ then: $K_{P_2} = K_{tot}$

- $P_3(\mathcal{L}) =_{def}$ [$\mathcal{L}$ is infinite].
- $P_4(\mathcal{L}) =_{def}$ [$\mathcal{L}$ is re] is <u>trivial</u> among re. sets!
- $P_5(\mathcal{L}) =$ ~~true if Fermat~~ [truth of Fermat's last conjecture]

But P_5 is trivial;

<u>Rice's thm</u>:

Let P be a non trivial property of Re-sets such that $P(\emptyset)$ is false.
Then for every re set R

$$R \leq_m K_P \quad ;$$

<u>Corollary</u>:

If P is not trivial then K_P is not recursive.

- $P_3(\mathcal{L}) =_{def}$ [$\mathcal{L}$ is infinite]
- $P_4(\mathcal{L}) =_{def}$ [$\mathcal{L}$ is re] is trivial among re. sets!
- $P_5(\mathcal{L}) =$ ~~true if Fermat~~ [truth of Fermat's last conjecture]

But P_5 is trivial;

Rice's thm:

Let P be a non trivial property of Re-sets such that $P(\emptyset)$ is false.
Then for every re set R

$$R \leq_m K_P \quad ;$$

Corollary.

If P is not trivial then K_P is not recursive.

Rice's thm

Oct 5th, 89

More precisely: K_P is $\leq_m$-hard for the re sets

proof: By hypothesis, $P(\text{domain}(M_1))$ for some machine M_1.

Note: domain $M_1 \neq \emptyset$ because $\neg P(\emptyset)$.

Let R be any re set.

To show $R \leq_m K_P$ consider the following procedure.

"On input w, save w temporarily, and simulate M_R on input x
If this simulation halts then act, then act like M_1 on w."

Let $f(x)$ be the code of this procedure.

- Note: ~~Suppose~~ $x \in R$ iff M_R halts on x.
  so that $M_{f(x)}$ acts like M_1
  so that $P(\text{domain } M_{f(x)})$
  and $f(x) \in K_P$.
- $x \notin R$ iff M_R diverges on x
  so that domain $(M_{f(x)}) = \emptyset$ so that $\neg P(\text{domain } M_{f(x)})$
  (re) $f(x) \notin K_P$

Hence $\boxed{x \in R \iff f(x) \in K_0}$

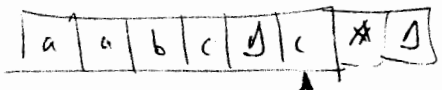
On the other hand f is "obviously" computable.

QED.

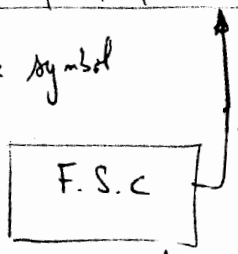
(1)
Oct 5

Turing Machines

- Storage unit: a two ways tape.



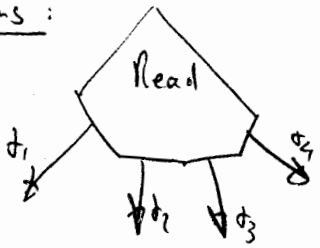
Δ. blank symbol



2-way read-write head that can sit on a cell of the tape.

At each time there is but a finite number of cells not blank.

- instructions:



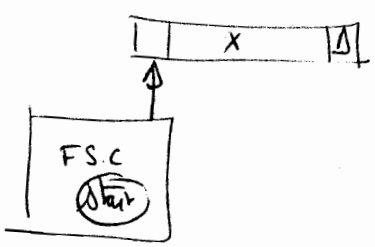
ϕ letter ϕ we have a command

left	right	Print ϕ

- the F.S.C can be thought as a finite flowchart built out of this boxes.

If for instance we wanted to compute $f: \Sigma_1^* \longrightarrow \Sigma_2^*$ we would require that $\Sigma_1 \cup \Sigma_2 \subset$ working alphabet of T.M.

To compute $f(x)$, we begin as follows:



... and let the machine run till it halts.

- If it halts we insist that the head halts with the head on the left of the non blank symbols.

- If it does not halt $f(x)$ is undefined.
- If it halts with a non acceptable final configuration then we say too that $f(x)$ is undefined.

Note: ψ is Turing computable $\Rightarrow$ ψ is partial recursive.
(any language is OK (simulation theorem)).

(re) ψ is computable by a SCHEME procedure

$\Leftarrow$ If we want to give more than one tape to a TM. then need other instructions. | Print & on1 | | Left on1 | ...
| Print & on2 |

Claim: { ψ Turing computable by 2-tape TM } = { ——— 1 ——— }

proof: W.o.l.g. assume the 2 tapes have same length.
"odd cells are tape 1"
even ——— 2
further pb. how to simulate the 2 heads?

| a_1 | b_1 | a_2 | b_2 | c_3 | b_3 | - - | b_5 |

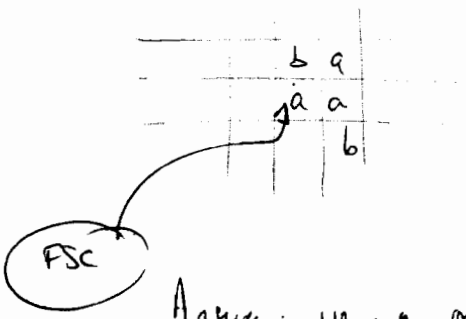
V: will denote where ↑ stands on each tape.
: is encoded with additionnal letters.

Lemma: If φ is 2-tape TM computable then φ is 1-tape TM computable
(where $\varphi: \Sigma_1^* \xrightarrow{\text{partial}} \Sigma_2^*$).

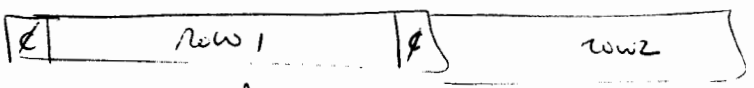
Note: In terms of time efficiency there is a quadratic loss from 2-tape TM to 1-tape TM

$\mathcal{C}_\Sigma$: polynomials are easily checked with 2-tapes TM's.
With one tape it is more difficult.

Question: if we had a 2-D tape would it add power to our T.M?

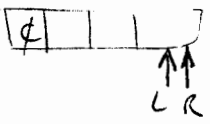


Answer: yes • one solution is to index all the cells of the plane in a "spiral way"
• or put the lines one after another



. going left or right is easy
. going up means going to the same relative position in row 2.

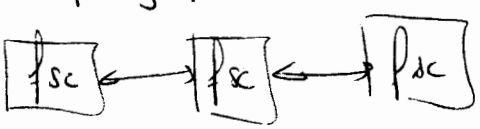
This can be done with 2 heads:



. L goes left as R goes right till L reaches ¢
. then translate everything to right... till L is on next ¢.

. A more general question is to simulate going through a tree with polynomial complexity of branches.

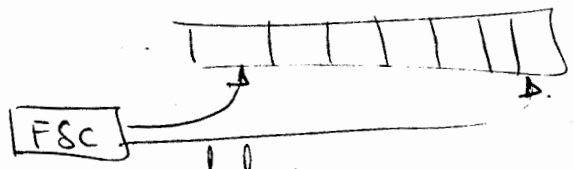
. Systolic array:



Post correspondence Problem

thm: the general jigsaw problem is undecidable.

Def: Post machines

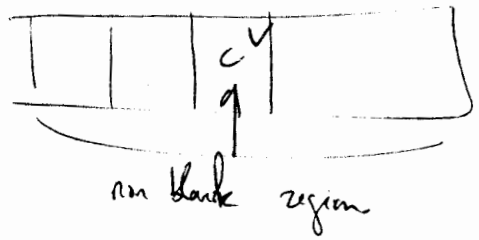


the heads are 1-way.
the left head is read only.

It is a special of a TM.

Prop: Any TM can be simulated by a Post machine.

pf: ⊆ a TM.



to construct the corresponding Post machine:

- duplicate the tape.
- have the right head copies what the left reads.
  for right shifts.

Reading assignment: Chapter 1 of Davis.

We studied post machines

{post-computable functions} = {Turing-computable functions}

Post correspondence problem

Def: A post system — of Davis

Ex: [3] $\{(b, bbb), (babbb, ba), (bb, a)\}$

Def A post system S is solvable iff: by repeatedly concatenating the pairs (coordinatewise), one can construct a pair with same first and second elements.

Ex: • $(b, b^3) \cdot (bab^3, ba) = (b^2 ab^3, b^4 a)$

• the above ~~that~~ example has solution 2.1.1.3.

Theorem: $\{S \mid S$ is a {post system, solvable}$\}$ is an re-set.

Ex 2: $S' = \{(ab, abb), (b, ba), (bb, b)\}$.

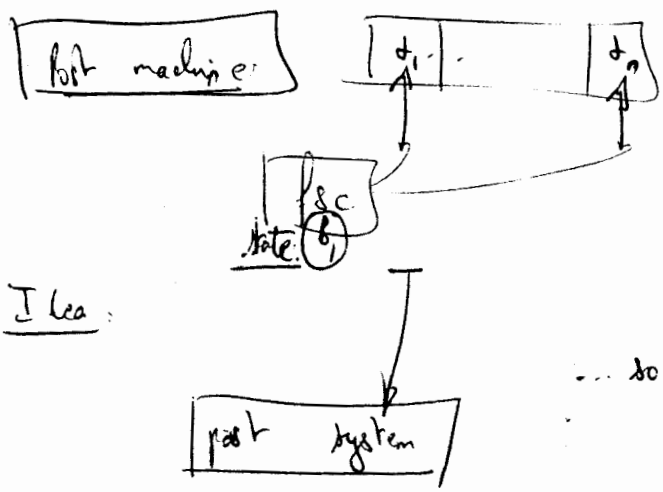
S' is not solvable because $\forall x \in S \quad |\Pi_2(x)| > |\Pi_1(x)|$

where Π_i is the i^{th} projection.

Read Davis p62

A More general problem would be:

given (A, x) or (y, A) can we add to it to get a solution.

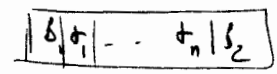


both head 1-way to the right.
so that the only thing which is relevant is what is on the right of the P.M.S head.

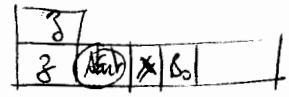
Idea:

... so that post machine halts on blank tape iff post system has a solution.

Post machine fsc : $B_1 \longrightarrow B_2$.

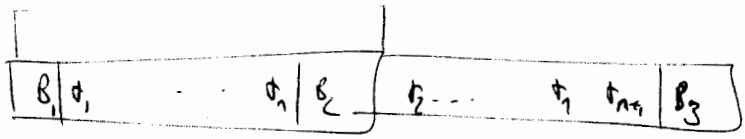


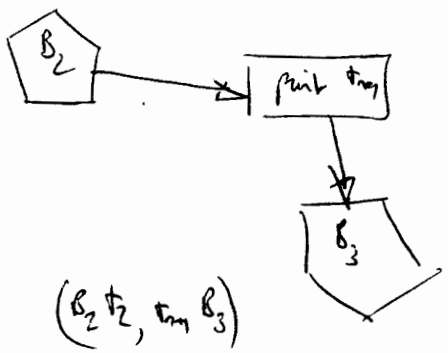
Post system :



Assignment check (Post system Problem) $\Longleftrightarrow$ (general Post system Problem)

- Use : (t_1, t_1) - (t_n, t_n) - - -



- 

$(B_2 t_2, t_{n+1} B_3)$

Matrix mortality problem

Manna p63.

Problem instance: $\{M_1 \dots M_n\}$ s.t. $\forall i$ M_i is a 3×3 integer matrix

An instance is mortal iff: $\exists k \geq 1$ $1 \leq \underset{i_k}{i_1} \leq n$ s.t. $\left(M_{i_1} M_{i_2} \dots M_{i_k}\right)_{3,2} = 0$

Mortality problem $= \{ I \mid I \text{ is a mortal instance} \}$. $= MMP$.

Note: Mortality problem is re.

Thm: (Floyd) The Mortality problem is undecidable

In fact ℓ con. Problem. $\leq_m$ MMP

(Consequence MMP is $\leq_m$-complete)

Proof: Observate: Fix any alphabet Σ

Then for any words $u, v, \in \Sigma^*$ there is a matrix (3×3, integer matrix) $M(u,v)$

s.t.: a) $M(u,v)_{3,2} = 0$ iff $u = v$

b) $M(u_1,v_1) \cdot M(u_2,v_2) = M(u_1 u_2, v_1 v_2)$ $\forall u_1, u_2, v_1, v_2$

Let S be a post system $= \{(\alpha_1,\beta_1),(\alpha_2,\beta_2) \dots (\alpha_n,\beta_n)\}$.

$i_1, i_2 \dots i_n$ is a solution to S iff $(\alpha_{i_1},\beta_{i_1}) \cdot (\alpha_{i_2},\beta_{i_2}) \dots (\alpha_{i_k},\beta_{i_k}) = (u,u)$ for some u.

Let a mortality instance be.

$\{ M_i \mid M_i = M(\alpha_i,\beta_i) \quad 1 \leq i \leq n \}$

So $M_{i_1} \dots M_{i_k} = M\left(\prod_{k=1}^{n} \alpha_{i_k}, \prod_{k=1}^{n} \beta_{i_k}\right)$

Therefore: if $i_1 \dots i_k$ solves S then $\prod_{k=1}^{k} M_{i_k} = M(u,u)$ hence mortal.

. The converse is true .

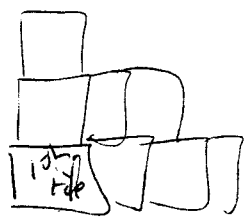
One can show that:

$PCP \leq_m \{ G \mid \begin{cases} G \text{ is a BNF (i.e. context free grammar)} \\ G \text{ is "ambiguous"} \end{cases} \}$.

Tiling Instance:  (a sequence)

Solution.

Rule: adjacent tiles must have same colours



Handout 8 : PS2 + A Hachman given to the Students. October 17

Logic

wff, truth, satisfaction, model, validity, proof, soundness, completeness, | decidability / incompleteness

$$(\exists F)\{F(a)=b \wedge \forall x\,[\,p(x) \supset (F(x) = g(x, F(f(x))))\,]\}.$$

($\wedge$: and ; $\supset$: implies)

Interpretation 1 : Domain $\mathbb{D} = \mathbb{N}$ $(x \in \mathbb{N})$

$\mathcal{I}_c(a) = 0$ (value of the constant a)

$\mathcal{I}_c(b) = 1$

$\mathcal{I}_c(f)$ = predecessor $= \lambda x$ { if $x=0$ then 0 ; else $x-1$ } $\quad (f(x) = x \dot{-} 1)$

$\mathcal{I}_c(g) = \lambda x, y \;\; x \cdot y$

$[\mathcal{I}_c(p)](n) = [n>0]$ where $n \in \mathbb{N}$

there is a function f ($\mathbb{N} \to \mathbb{N}$) such that $f(0)=1$ and for all numbers natural x, if $x>0$, then

$$F(x) = x \cdot F(x \dot{-} 1)$$

Interpretation 2 : $\mathbb{D} = \{a, b\}^*$

$\mathcal{I}(a) = \mathcal{I}(b) = \Lambda$

$\mathcal{I}_c(f)$ = tail

$\mathcal{I}_c(g)(x, y)$ = apend $(y, \text{head}(x))$

$\mathcal{I}_c(c) = \lambda x \quad x \neq \Lambda$

$\quad = \lambda x \quad |x| > 0$

Reverse function satisfies the condition and hence the whole formula is true

Note : | g b are specific values.

distinguish in between integers and numerals;

Interpretation 3 : $\mathcal{D} = \mathbb{N}$

$a : 0$

$b : 1$

$f(x) : id(x)$

$g(x,y) : y+1$ for all x

P : positive ?

We then get: for all $x > 0$ $\quad F(x) = 1 + F(x)$ which is wrong:
there is not total recursive function F satisfying this.

- Given an interpretation, a wff is true or false

Notation: $\mathcal{I} \models w$
(I satisfy)

Then: Interpret 1 $\not\models w_0$
Interpr 2 $\not\models w_0$
Interpret 3 $\not\models w_0$.

- a wff w is valid iff $\mathcal{I} \models w$ for all $\mathcal{I}$ (where implicitely $\mathcal{I}$ is related to the symbols appearing in w).

D, $\mathcal{I}_c$, $\mathcal{I}_v$ — (we distinguish in between constants and arguments).

Arity $(+) = 2$
(Arity (eqe) = 1
Arity (if then else) = 3 | nber of arguments it takes.

~~A model $\mathcal{I}$ will be~~ Note $\mathcal{I} = ((D, \mathcal{I}_c), \mathcal{I}_v)$ called valuation.
domain.

then a model m is $(D, \mathcal{I}_c)$

Let $m = \mathbb{N}$ with $+, \div, *$ (called arithmetic)
(we keep it fuzzy whether numerals are there).

$\mathcal{I}_c(\underline{+}) = +$
like a numeral.

Definition: $m \models w$ means "w is valid in m"
iff $(m, \mathcal{I}_v) \models$ for all $\mathcal{I}_v$
(... where implicitely $\mathcal{I}_v$ a correct valuation)

eg: • arithmetic $\models \exists y \; \forall x \; x - x = y$

a more interesting:
• arithmetic $\models (\exists y)[(y + y = y) \wedge (x \div x = y)]$

Thus ~~value~~ formula is valid in arithmetic eventhough x is a free variable

• $\models w$ (e) "w is valid"
iff $m \models w$ for all m.

Propositionally valid: $w \vee (\neg w)$

Claim:

$\models \left(\forall x \; P(x) \Rightarrow \exists x \; P(x) \right)$

the only free variable is predicate P.

$\models ? \left(\exists x \; P(x) \supset \forall x \; P(x) \right)$ not valid.

false in any interpretation in which some element of the domain has property P and some other element does not

(*) $\boxed{m \models \exists P \left[\neg \left(\exists x \; P(x) \supset \forall x \; P(x) \right) \right].}$ iff $|D_m| \geq 2$.

Remark: $m \models w$ iff $m \models \underbrace{\forall \vec{x}, \vec{P}}_{\text{free variables of } w}. w$

(so that not putting any quantifier for some variable, means using implicitly the universal quantifier).

Defs: 1st order wff : no function or predicate variable

2nd order wff : ∃ ———————

Thm 1 (Valid-2): $\{ w \mid w \text{ is 2nd order wff and } \models w \}$ is neither re, nor co-re.

(tantamount to Gödel's incompleteness thm).

Thm 2 (Valid-1): (Gödel's completeness theorem (in abstract form)) $\{ w \mid w \text{ is 1st order wff and } \models w \}$ is re

Thm 3 (Church's thesis): Valid-1 is not decidable

$\exists x \exists y \; x \neq y$ is equivalent to (*) above

Expressing stuff formally in predicate calculus

$\mathcal{I}_c(+) = [f : D^2 \to D]$. ($+$: arity $= 2$).

a function

- "the domain is finite": (does not depend on the valuation).

interpretation 1

(a) want closed wff, ω_{fin} s.t. $m \models \omega_{fin}$ iff D_m finite.

(b) for every 1-1 function f, everything is in the range of f.

(c) $\forall f$ [if f is 1-1, then $\forall x$ [$x \in$ range f].]

(implicitely $f : D \to D$).

(d) $\forall f$ [f 1-1 $\supset$ $\forall x$ [$x \in$ range (f)]]

ω_{fin1} (e) $\forall f$ [$\forall_{x,y}$ ($f(x) = f(y) \Rightarrow x = y$)] $\supset$ [$\forall x$ $\exists y$ $x = f(y)$].

interpretation 2

for all transitive binary relation R, there is an R-maximal element.

(a) $(\forall R)$ [R transitive $\supset$ $(\exists x)$ (x is R-maximal)]

ω_{fin2} (e) $\forall R$ [$(\forall_{x,y,z})$ $R(x,y) \wedge R(y,z) \supset R(x,z)$] $\supset$ [illegible]

$\exists x (\forall y \; R(x,y) \supset R(y,x))$

$\models \omega_{fin_1} \equiv \omega_{fin_2}$

Quiz : Wed.

Binary strings

$\Sigma = \{a, b\}$

Define inductively : 1° Λ is a binary string.
2° If x is a binary string so is $a.x$ and $b.x$.
3° That's all.

Axioms :

1° Λ, a, b are binary strings.
2° $\cdot$: binary strings $\longrightarrow$ b.s.
3° $(x.y).z = x.(y.z)$
4° $\Lambda . x = x . \Lambda = x$.
5° $ax \neq by$ $\forall x, y$.
6° $\begin{bmatrix} ax = ay \\ bx = by \end{bmatrix}$ iff $x = y$.
7° eqlength (Λ, Λ) holds. eqlength is a predicate
eqlength (ax, ay) iff .... eqlength (bx, by) iff eqlength (x, y).

Actually eqlength (Λ, z) iff eqlength (z, Λ) iff $z = \Lambda$

Notation: $\omega_1 = \omega_2$ iff $(\omega_1 \supset \omega_2) \wedge (\omega_2 \supset \omega_1)$.

We get $W_{b,s}$;

Assume we have $m_i \models W_{b,s}$ $i = 1, 2$
Want : m_1 and m_2 are isomorphic.
($f : D_1 \longrightarrow D_2$ a bijection st $f(d_1 \cdot_1 d_1') = f(d_1) \cdot_2 f(d_1')$
$\forall d_1, d_1' \in D_1$

Then eqlength (d_i, d_i') iff eqlength $(f(d_i), f(d_i'))$

But all that holds is:

if $m \models \omega_{bs}$ then $\langle \{a,b\}^*, a, b, \Lambda, \cdot, \text{eqlength} \rangle$ is ~~isomorphic~~ isomorphic to a substructure of m.

A nonstandard model of ω_{bs}:

$$D = \{a,b\}^* \cup \{a,b\}^\omega$$

Def:

$$x \cdot y \overset{\text{def}}{=} \begin{cases} xy & \text{if } x \text{ is finite} \\ x & \text{if } x \text{ is infinite} \end{cases}$$

eqlength (x) = as before if x is finite
= A if x is infinite.

Remark $\forall x,y,z$

$\not\models \omega_{bs} \supset \forall (y \cdot x = z \cdot x \supset y = z)$

(We saw a counter example last time : a^ω. strings :).

$a \cdot a^\omega = aa \cdot a^\omega$ but $a \neq aa$.

$W_{bs\text{-}konect} \overset{def}{=} \omega_{bs} \wedge only_{b.s.}$

Note: There is a 1st-order formula such that characterizes uniquely binary strings.

which means that $only_{bs}$ has to be 2nd order

So $m' \models W_{bsh}$ iff m' isomorphic to $\langle \{a,b\}^*, \cdot, eqlength \rangle$.

So that we can now define binary strings inductively.

$\forall P \left\{ \left(P(1) \wedge (\forall x) \left[P(x) \supset \left(P(ax) \wedge P(bx) \right) \right] \right) \supset \forall z \; P(z) \right\}$

... is the principle of induction on strings.

$only_{b.s.}$

$\langle \{a,b\}^*, \cdot, eqlength \rangle$.

Although this property does not hold for the previous example

this property is what we call only-bs.

Thm | $\exists$ 2nd order ~~predicate~~ closed formula w_{bsh}. s.t.

$m' \models W_{b.s.h}$ iff m' isomorphic to $\langle \{a,b\}^*, \cdot, eqlength \rangle$

$\{ w \mid$ binary strings $\models w$ and w 1st order $\}$

$\Vert$ def

Corollary: $Th_1(\underbrace{\text{binary-string}}_{\langle \{\varepsilon, 0\}, \cdot, \text{eqlength} \rangle})$ is the set of 1st order formulas valid in the framework of binary string.

$\leq_m Valid_2(\emptyset)$

$\Vert$ def

$\{ w \mid \models w$ and w has no non logical symbols $\}$
w is a wff of 2nd order predicate calculus.

~~(w) formula of~~

proof: let w be a 1st order formula with signature $\cdot$, eqlength.

$w \in Th(\text{binary-strings})$ iff binary-strings $\models w$.

$\Vert$ by def.

iff $\forall m$ [if m is binary string then $m \models w$]

iff (for all m $m \models w_{bsf} \supset w$)

iff $\models w_{bsf} \supset w$

iff $(\forall \cdot, \text{eqlength})$ $[w_{bsf} \supset w]$

where we quantify over constants in the same way as in the following example:

$\forall F \models [\forall \cdot \; f(f(\cdot)) = \cdot]$ is a closed form formula.

implies $\models \forall F \; \forall \cdot \; F(\cdot) = \cdot \supset F(F(\cdot)) = \cdot$

iff $(\forall \cdot)(\forall \text{eqlength})$ $[w_{bsf} \supset w] \in Valid_2(\emptyset)$.

thm: (Gödel's incompleteness essentially)

Th(binary-strings) is neither r.e. nor co-r.e. (Will be proven).

Cor: Valid$_2(\emptyset)$ is _____.

Thm $R \leq_m$ th(binary strings) for any r.e. set R.

Pf: Note: th(strings over alphabet Σ) $\leq_m$ th(binary strings)

~~We'll prove $R \leq_m$~~

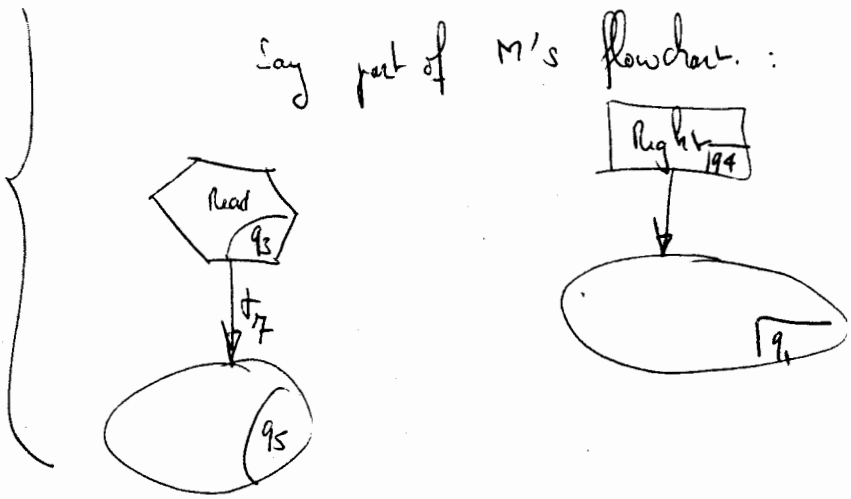
Let $R = \text{dom}(M)$ and Σ_M = work type symbols of M.

Q_M = flowchart boxes of M.

Then We'll show that:

$R \leq_m$ th(strings over $\Sigma_m \cup Q_m \cup \{\#\}$)

Say part of M's flowchart:



where $q_4, q_5 \in Q_m$.

I want a wff called $\text{Next}_m(u,v)$ meaning

u describes a configuration of M and

v —— next ——.

u is of the form: $u_1 s c u_2$ where $s \in Q_m$, $c \in \Sigma_m$, $u_1, u_2 \in \Sigma_m^*$

and if $s = q_3$, $c = t_7$ then $v = u_1\, q_5\, c\, u_2$

and (if $s = q_4$ and $u_2 \neq \Lambda$, then $v = u_1 c q_1 u_2$)

and (if $s = q_4$ and $u_2 = \Lambda$, then $v = u_1 c q_1 \cdot \Delta$ and...

↑ blank symbol)

Everything can be expressed as binary strings and diff's:

$\underline{\mathcal{L}_0}$: • $u_2 \neq \Lambda \iff \neg\, \text{eqlength}(u_2, u_2 \cdot u_2)$

• $c \in \Sigma_m \iff (c = \sharp_1) \vee (c = \sharp_2) \cdots (c = \sharp_m)$

• $u_1 \in \Sigma_m^*$ can be written as a formula in the following way:

$\forall u_1', c, u_2' \quad \left(u_1 = u_1' \cdot c \cdot u_2' \wedge \text{eqlength}(c, \sharp_1)\right) \supset c \in \Sigma_m$

• ~~$u_1 \$ c \$ u_2$~~ u is of the form $u_1 \$ c\, u_2 \iff$

$\exists\, u_1, \$, c, u_2$ s.t. $u = u_1 \$ c\, u_2$

$\mathcal{L} \in R$ if M halts on input α

$\exists$ history word ζ of M of the form:

$\$\, \underline{q_{start}}\, \alpha \$ \cdot u_1 \$ u_2 \$ \cdots \$ u_n \$$ s.t.

u_0

$\text{Next}(u_i, u_{i+1})$ for $0 \le i \le n$ and u_n contains q_{halt}.

$(\exists \zeta)\Big\{ \underbrace{(\exists \zeta')^{\exists} \left[\zeta = \$ \cdot q_{start} \cdot \alpha \$ \cdot \zeta'\right]}_{(*)\ \zeta \text{ starts OK.}}$

$\wedge \quad (\exists u, u', \zeta')\ [\zeta = \zeta' \$ u \$ u' \$ \quad \wedge \text{ next}(u, u')$

$\Big(\text{i.e. there is no } \$ \text{ in } u \text{ and } u'\Big)$

$\wedge \quad (\exists u'', u''' \quad u' = u'' q_{halt} u''')\Big]$

$(*)\ \zeta$ ends OK

$\wedge \quad \Big[\forall \zeta_1, \zeta_2, \zeta_3, \zeta_4 \quad \zeta = \zeta_1 \$ \zeta_2 \$ \zeta_3 \$ \zeta_4 \wedge (\zeta_2, \zeta_3 \in (\Sigma_m \cup Q_m)^*)$

$\supset \text{Next}(\zeta_2, \zeta_3)\Big]\Big\}.$

Subjects covered since last quiz :

Machines

1°) simulation of a 2-tape T.M by a 1-tape TM.
(simulation thesis)
description of the simulation of 2-D tape

2°) Post correspondance problem
- TM can be simulated by Post machines.
- Post systems.
- Manna (p63) : ~~HP~~ reducible to PCP

3°) a) Rabin's mortality problem (Manna p63)
b) Tiling problem.

Logic

4°) Models and interpretations :
example of a formula with 3 different interpretations
$\mathcal{M} \models A$.

5°) Gödel's thms.

6°) Translation stuff into predicate calculus syntax.

7°) Binary strings : • axiomatization of W_{bS}
• $\mathcal{M} \models W_{bS} \Rightarrow \langle \{0,1\}^+, 0, 1, \cdot, \text{eps} \rangle \cong \mathcal{M}$.
description of "only 1-s" (1st order)
• Th_1 (binary strings) $\leq_m$ Valid$_2$ ($\not\emptyset$)
• R re $\Rightarrow R \leq Th$ (binary strings)

8°) Morphisms.

About the Notes

$\mathfrak{I} = (D, \mathfrak{I}_C, \mathfrak{I}_V)$ $\quad \mathfrak{I}' = (D', \mathfrak{I}'_C, \mathfrak{I}'_V)$

$\mathfrak{I}(t) \in D$ for all term t.

$\mathfrak{I}(A) \in \{\text{true}, \text{false}\}$

$\eta \quad \mathfrak{I} \longrightarrow \mathfrak{I}'$

have $\eta : D \to D'$

$\eta(\mathfrak{I}(c)) = \mathfrak{I}'(c)$ and more generally:

$\eta\left(d_1 \left(\mathfrak{I}_C(\odot)\right) d_2\right) = \eta(d_1)\ \mathfrak{I}'_C(\odot)\ \eta(d_2)$

some binary symbol.

Nov 4

$\eta\left[\mathfrak{I}_C(f)(d_1, d_2)\right] = \mathfrak{I}'_C(f)\left[\eta(d_1), \eta(d_2)\right]$

in the same way: $\eta\left(\mathfrak{I}_V(x)\right) = \mathfrak{I}'_V(x)$.

For predicate symbols we do the same thing:

$\mathfrak{I}_C(\varphi)(d_1, d_2, d_3) = \mathfrak{I}'_C(\varphi)\left(\eta(d_1), \eta(d_2), \eta(d_3)\right)$

Lemma: if $\eta : \mathfrak{I} \longrightarrow \mathfrak{I}'$ is onto hom.
then: for all first order wff's A without equality, $\mathfrak{I}(A) = \mathfrak{I}'(A)$.

Def: A wff is of the unary calculus if signature has only unary predicates

Corollary: $\{A \mid A$ is unary, 1st order w/o equality, $\models A\}$ is decidable

Lemma Let $\mathcal{M} = (D, \mathfrak{I}_C)$ have a unary predicate signature

Say $d_1 \equiv_{\mathcal{M}} d_2$ iff $\mathfrak{I}_C(P_i)(d_1) = \mathfrak{I}_C(P_i)(d_2)$ for all P_i in signature of $\mathcal{M}$.

Ex: $D = \mathbb{Z}$

P_1 = positive

P_2 = perfect square

P_3 = prime.

then, $3 \equiv_{\mathcal{M}} 4$

- all negative numbers are equivalent.

Rail for class:
owner - 6044 - forum@athena - mit.edu

$\mathcal{I} = \left((D, \mathcal{I}_C), \mathcal{I}_V\right)$, $\underline{\mathcal{I}(t)}$, $\mathcal{I}(A)$

extension of the interpretation.

lemma: let $\eta : \mathcal{I} \to \mathcal{I}'$ be an onto homomorphism.
For any term t $\quad \eta(\mathcal{I}(t)) = \mathcal{I}'(t)$

(η: canonical surjection: $(\mathbb{Z}, +) \longrightarrow (\mathbb{Z}_2, +)$

proof: by induction on definition of t.

case 1 (a): $\eta(\mathcal{I}(t)) = \eta(\mathcal{I}(x)) = \eta(\mathcal{I}_V(x)) = \mathcal{I}'_V(x) = \mathcal{I}'(x)$

by def of an homomorphism.

case 2 (c): same thing. mutatis mutandis: $\mathcal{I}_V \leftrightarrow \mathcal{I}_C$.

case $f(t_1 .. t_n)$

$$\eta\left(\mathcal{I}\left(f(t_1 .. t_n)\right)\right) = \eta\left(\mathcal{I}_C(f)\left(\mathcal{I}_V(t_1) .. \mathcal{I}_V(t_n)\right)\right)$$

$$= \mathcal{I}'_C(f)\left(\eta(\mathcal{I}(t_1)) \cdots \eta(\mathcal{I}(t_n))\right)$$

$$= \mathcal{I}'_C(f)\left(\mathcal{I}'(t_1) .. \mathcal{I}'(t_n)\right)$$

induction.

case $F(t_1 .. t_n)$: same thing.

Thm: If $\mathcal{I}_1 \sim \mathcal{I}_2$ then $\mathcal{I}_1(A) \simeq \mathcal{I}_2(A)$.

proof: by induction on A.

case $p(t_1 \ldots t_n)$ is notes | (Ample notation ↦ Pb set 3.)

case $P(t_1 \ldots t_n)$

case $t_1 = t_2$:

$\mathcal{I}_1(t_1 = t_2)$ iff $\left[\begin{array}{l} \mathcal{I}_1(t_1) \text{ equals } \mathcal{I}_1(t_2) \\ \mathcal{I}_2(t_1) \text{ equals } \mathcal{I}_2(t_2) \\ \mathcal{I}_2(t_1 = t_2) \end{array}\right]$ by previous lemma.

Hence we are done for atfs.

case $A_1 \wedge A_2$, $A_1 \vee A_2$, $A_1 \supset A_2$, $\neg A_1$, $\forall x\, A_1$, $\exists x\, A_1$

↑ done page 4 of the notes.

case $\forall F.\ A_1$

$\exists F.\ A_1$

$\forall P.\ A_1$

$\exists P.\ A_1$

Thm $\mathcal{I}_1 \sim \mathcal{I}_2$ implies $\mathcal{I}_1(A) = \mathcal{I}_2(A)$

following and end of proof:

case: $A = \exists F.\ B$ where F is a functor variable

For notational easy, say F is binary.

$\mathcal{I}_1(\exists F.\ B)$ iff there is a binary function $\alpha : D_1 \times D_1 \longrightarrow D_1$ such that

$\left(\mathcal{I}_1\left(F \mapsto \alpha\right)\right)(B)$

where $\left\{\begin{array}{l} \text{if } f : S \to T \text{ is a functor.} \\ f' \overset{\text{def}}{=} f[s \to t] : S \to T \text{ is defined by} \\ \quad f'(s') = f(s') \text{ if } s' \neq s \\ \quad\quad = t \text{ if } s' = s \end{array}\right.$

Actually $\left(\mathcal{I}_1[F \to \alpha]\right)$ stands for $\left(M_1, \mathcal{I}_{1,v}[F \to \alpha]\right)$.

$$\left(\mathcal{I}_1[F \to \alpha]\right)(B) = \bigvee_{\alpha : D_1^2 \to D_1} \left(\mathcal{I}_1[F \to \alpha]\right)(B)$$

Lemme: If $\mathcal{I}_1 \sim \mathcal{I}_2$, then $\mathcal{I}_1[F \to \alpha] \sim \mathcal{I}_2[F \to \alpha']$ where:

η

$$\alpha' : D_2 \times D_2 \longrightarrow D_2$$

$$\alpha'(d_2', d_2'') = \eta\left(\alpha\left(\eta^{-1}(d_2'), \eta^{-1}(d_2'')\right)\right)$$

$$= \bigvee_{\alpha'} \left(\mathcal{I}_2[F \to \eta \circ \alpha \circ \eta^{-1}]\right)(B)$$

(reduction)

$$\sim \bigvee_{\beta : D_2 \times D_2 \to D_2} \mathcal{I}_2[F \to \beta](B) = \mathcal{I}_2(\exists F.\ B)$$

(def of $\mathcal{I}_2(\exists)$)

<u>Remark</u>: because every $\beta : D_2 \times D_2 \longrightarrow D_2$ is equal to $\eta \circ \alpha \circ \eta^{-1}$ for some $\alpha : D_1 \times D_1 \longrightarrow D_1$, namely $\alpha = \eta^{-1} \circ \beta \circ \eta$.

$\langle \mathbb{Z}_7, 0, 1, 2, \ldots 6, + (\mathrm{mod}\ 7), * (\mathrm{mod}\ 7) \rangle$

Given a wff A with free variables x_1, F and a valuation of x_1, F (i.e. a pair ~~$\langle n_1, F \rangle \in \mathbb{Z}_7^2$~~ $n_1 \in \mathbb{Z}_7$ and 7×7 table for F [entries in $\mathbb{Z}_7$]).

Output: ($\mathbb{Z}_7$, sym. value) (A)

(Define (TruthVal A V)
COND ((Atomic pred? A) (V (pred-symbol) A)
[TermVal (1st term of A) (— 2nd 3 A)]
(Equation? A)
((Conjunction? A) (and (TruthVal (left conjunct A) V) (— Right conjunct A) V)))

Quiz 3 Wed Nov 30 evening.
Quiz 4 Wed Dec 14 (Dec 8 Mon, Dec 9) evening

- Chris Craig (with Anthony Schneider)
- Rory Vogt (with goodness)
- Mark Wicks (with Bryan Dennis)

Church's Thm the valid wff's of 1st order pred. calculus (without equality) are not recursive.

proof PCP $\leq_m$ 1-tasks

(tautologies of 1st order predicate calculus.

$$S = \{(\alpha_1, \beta_1) \cdots (\alpha_n, \beta_n)\} \longmapsto \omega_S.$$

Claim: S has a solution iff $\vdash \omega_S$ iff $\omega_S \in$ 1-tasks.

$$\omega_S \overset{\text{def}}{=} \left[\bigwedge_{i=1}^{n} P\left(f_{\alpha_i}(c), f_{\beta_i}(c)\right) \wedge \forall x, y \quad P(x,y) \supset \bigwedge_{i=1}^{n} P\left(f_{\alpha_i}(x), f_{\beta_i}(y)\right) \right] \cup \left[\exists z \quad P(z,z) \right]$$

$\Leftarrow$ If since $\vdash \omega_S$, it is true in the interpretation where

$D = \Sigma_S^*$, $I_c(f_\sigma) = \lambda z.\, \sigma z$ for all $\sigma \in \Sigma_S^*$

and $I_c(P)(\alpha_1, \alpha_2)$ iff $(\alpha_1, \alpha_2) \in S^*$

("obvious solution")

$I_c(c) = \Lambda$

$f_{t_1 t_2 t_3}(t)$ abbreviates $f_{t_1}\left(f_{t_2}\left(f_{t_3}(t)\right)\right)$

$$\omega_S = \left[\bigwedge_{i=1}^{n} p\left(t_i(c), t_{f_i}(c)\right) \wedge \left\{ \forall x,y \quad p(x,y) \Rightarrow \bigwedge_{i=1}^{n} p\left(t_i(x), t_{f_i}(y)\right) \right\} \right] \Rightarrow \exists \xi \; p(\xi,\xi).$$

$\Leftarrow$ Suppose S has a solution $\alpha_{i,0} \cdots \alpha_{i,k_i} = \beta_{i,0} \cdots \beta_{i,k_i}$

Rq ω_S is a closed formula.

to prove $\models \omega_S$ it suffices to take a model

case 1 $m \not\models [\,\cdots\,]$ ok.

case 2 otherwise, prove by induction on $\mathcal{U}$ that for any integers $i_1 \cdots i_m \in [1 \cdots n]$

$$\mathcal{U} \models p\left(t_{\alpha_{i_1} \cdots \alpha_{i_m}}(c), t_{\beta_{i_1} \cdots \beta_{i_m}}(c)\right).$$

therefore $pm \models p\left(t_{\alpha_{i,0} \cdots \alpha_{i,k_i}}(c), t_{\beta_{i,0} \cdots \beta_{i,k_i}}(c)\right)$

the same

Hence $m \models \exists \xi \; p(\xi,\xi)$

Completeness theorem : the set of first order valid formulas is re

$$\vdash \text{taut} = \bigcup_{\text{signatures } S} \text{taut}(S)$$

Gödel really gives decidable axioms and rules which are

1) sound
2) complete.

$\left(A_1 \wedge \ldots \wedge A_n\right) \supset B \qquad (n \geq 0)$
$\{A_1, \ldots, A_n\} \supset B$ — absurdity formulas —

- axioms: $\dfrac{\Gamma, A}{\Gamma \cup \{A\}} \Rightarrow A$ — Γ is a finite set of wff's —

(Fact $\vdash \Gamma, A \Rightarrow A$
namely $\vdash \left(\left(A_1 \wedge \ldots \wedge A_n\right) \wedge A\right) \supset A$)

e.g. $\Gamma \Rightarrow$ True
$\Gamma \Rightarrow \neg$ false

- inference rules

$$\frac{\Gamma \Rightarrow B}{\Gamma, A \Rightarrow B}$$

My Notes

Discussion avec Roger

$\forall x,y \quad L(x,y) \supset R(x,y)$

$\forall x \quad [(\forall y\ L(x,y)) \supset (\forall y\ R(x,y))]$

$(\forall x,y\ L(x,y)) \supset \forall x,y\ R(x,y)$ ↓

a la equivalence.

- Implies & $\not\Vdash (T \supset B) \not\Vdash \forall (\mathcal{M}, \mathfrak{I}_\sigma)\ [(\mathcal{M}, \mathfrak{I}_{\sigma^-})(T) \supset (\mathcal{M}, \mathfrak{I}_\sigma) B]$

(Recall Soundness Thm: 1°) $(\vdash A$ implies $\models A$ not constructive
constructive
2°) $Ax \vdash A$ implies $Ax \models A$)

- $\frac{T}{B}$ sound $\not\Vdash$

$\forall \mathcal{M} [\forall \mathfrak{I}_\sigma\ (\mathcal{M}, \mathfrak{I}_\sigma) T$ then $\forall \mathfrak{I}_\sigma\ (\mathcal{M}, \mathfrak{I}_\sigma)(B)]$

- $\frac{T}{B}$ weakly sound $\not\Vdash$

$(\forall (\mathcal{M}, \mathfrak{I}_\sigma)\ (\mathcal{M}, \mathfrak{I}_\sigma)(T)) \supset (\forall (\mathcal{M}, \mathfrak{I}_\sigma), (\mathcal{M}, \mathfrak{I}_\sigma)(B))$

Ex Sheet

Axioms of rules

inference rules

$$\frac{\Gamma \Rightarrow B}{\Gamma, A \Rightarrow B}$$

$\Gamma, A \Rightarrow A$ axiom

$$\frac{(\Gamma, A \Rightarrow B) \text{ and } (\Gamma, \neg A \Rightarrow B)}{\Gamma \Rightarrow B}$$ (elim assumption)

Notation $(A_1, A_2 \mid\Rightarrow B$ means any arrangement of $A_i \supset B$ for wffs.

rule

$$\frac{S_1 \dots S_n}{S}$$

Def: a rule is sound if

$\forall \mathcal{U}$ $\left(\text{if } \mathcal{U} \models S_i \text{ for } 1 \le i \le n \text{ then } \mathcal{U} \models S \right)$

Def a rule is weakly sound if $\models S_i$ for $1 \le i \le n$ then $\models S$.

a) Ex: $\dfrac{x=0}{x=1}$ is weakly sound, vacuously because $\not\models x=0$

- another equivalent way is $\not\models (\forall x, x=0) \supset 1=0 \supset \forall x=1$ so that the rule is sound.

b) $\dfrac{x=0 \vee x=1}{x=1}$ weakly sound, but not sound.

Observation:

a) Axioms are valid
b) rules are sound.

Remark $\models \left((\Gamma, A) \Rightarrow B \wedge (\Gamma, \neg A) \Rightarrow B \right) \supset (\Gamma \Rightarrow B)$

Connectives $\dfrac{(\Gamma \Rightarrow A)}{\Gamma \Rightarrow (A \vee B)}$ ($\vee$-introduction)

Def. A proof of $(\rho \vee \neg\rho)$

1) $\rho \Rightarrow \rho$ (assumption axiom)

2) $\rho \Rightarrow \rho \vee (\neg\rho)$ ($\vee$ intro)

3) $(\neg\rho) \Rightarrow (\neg\rho)$ (assumption axiom)

4) $(\neg\rho) \Rightarrow \rho \vee (\neg\rho)$ ($\vee$-intro, symmetric)

5) $\rho \vee (\neg\rho)$ (for 2 and 4 elim assumption)

Corollary Soundness theorem.

1) If A is provable ($\vdash A$) implies $\models A$

2) $Ax \vdash$ implies $Ax \models A$ (actually only weakly soundness required)

Def $Ax \models A$ iff $\forall m$ s.t. $m \models \bigwedge_{B \in Ax} B$ implies $m \models A$

wffs A, B

- $\frac{A}{B}$ weakly sound $\quad \nvdash (\models A \text{ implies } \models B)$
  $\nvdash [(\forall \mathcal{U}\ \mathcal{U} \models A) \text{ implies } (\forall \mathcal{U}\ \mathcal{U} \models B)]$
  $\nvdash [(\forall \mathcal{I}\ \mathcal{I} \models A) \text{ implies } (\forall \mathcal{I}\ \mathcal{I} \models B)]$
- $\frac{A}{B}$ is sound $\quad \forall \mathcal{U}\ [\mathcal{U} \models A \text{ implies } \mathcal{U} \models B]$

- A implies B $\quad \nvdash \models A \supset B$
  $\nvdash \forall \mathcal{I}\ [\mathcal{I} \models A \text{ implies } \mathcal{I} \models B]$

Rules for quantifiers

- $$\frac{\Gamma \Rightarrow A(x) \qquad x \notin \text{Free variable } (\Gamma)}{\Gamma \Rightarrow \forall x\, A(x)}$$ (Universal generalization)

Remark $\quad \not\models (A(x) \supset \forall x\, A(x))$

Ex: $\not\models (x=3) \supset \forall x\, (x=3)$

- $$\Gamma \Rightarrow \forall x\, A(x)$$
  $$\Gamma \Rightarrow A(t)$$

- $$\frac{\Gamma \Rightarrow t_1 = t_2}{\Gamma \Rightarrow (A(t_1) = A(t_2))}$$

Thm (soundness) Let A_x any set of wffs (possibly $\emptyset$)

Then $A_x \vdash A$ implies $A_x \models A$

(*) in any $\mathcal{I}$ where A_x is true
A is true also

2nd order

conversely : false !

proof Note that if Ax is r.e. then $\{A \,/\, Ax \vdash A\}$ is r.e. also.
(generate A_x in //, and "at the same time" work out the deductions from A_x ...")

But, even in the case $Ax = \emptyset$ i.e. no case, $\{A \mid Ax \models A\}$ is neither

In fact, since $\vdash$ is sound
$$\{A \,/\, \vdash A\} \subsetneq \{A \,/\, \models A\}.$$
Hence there is a closed second order wff A_0 st:
$\models A_0$ is valid and $\nvdash A_0$ (of course $\nvdash \neg A_0$)

$\text{Taut}_2(\emptyset)$ is not re.

$\{A \mid \vdash A\}$ is $\left|\begin{matrix}\text{re}\\ \text{not rec}\end{matrix}\right.$ (even for 1st order wffs)

no hivation of the theory of recursive fcts.

Thm: Taut_1 is re.

As a matter of fact, Gödel showed that $\forall$ first order wffs A

$$\models A \quad \text{iff} \quad \vdash_{\text{Nana}} A$$

validity. provability

| Completness

Herbrand's procedure and theorems

To see if 1st order wff A is valid, let $B = \neg A$ and prove that B is unsatisfiable ($\mathcal{I}(B) \equiv \text{false}$)

1°) Simplify B to a "prenex form" B' st $\models (B \equiv B')$

↑ see Nanan (put all the quantifiers at the left).

2°) "Skolemize" B' to obtain a wff B'' in universal form s.t.

B' is satisfiable iff B'' is satisfiable. (very weak relation)

(cf $\leq_m$-reduction)

(B'' "gets rid of all $\exists$")

3°) Use Herbrand's thm: (no equality)

Auniversal fla is unsatisfiable iff there is some finite conjunction of its ground instances which is propositionally unsatisfiable.

Ex: $\forall x \forall y \quad [P(x, P(x,y)) \vee Q(f(x)) \wedge \neg P(x, f(y))]$

Ground terms

c.

$f(c,c)$

$f(c, f(c,c))$, $f(f(c,c), c)$, $f(f(c,c), f(c,c))$

Monday after thanksgiving

Def : A, B wff's are equisatisfiable iff $(\exists\, \mathcal{I}\ \ \mathcal{I} \models A)$ iff $(\exists\, \mathcal{I}'\ \ \mathcal{I}' \models B)$

Notation $A \underset{sat}{\sim} B$

Rq : $A \underset{sat}{\sim} B$ iff [(A unsatisfiable) iff (B is unsatisfiable)]

Def $A \underset{valid}{\sim} B$ iff [$\models A$ iff $\models B$]

Rq $A \underset{valid}{\sim} B$ iff [$\neg A \underset{sat}{\sim} \neg B$]

(Note A valid iff $\neg A$ is unsatisfiable)

About Herbrand's procedure :

Everything is 1st order

To see if A_0 is valid

1°] Construct A_1, closed, $\approx$ = symbol and $A_1 \underset{valid}{\sim} A_0$

Example : A_0 : $(x_0 = 0) \vee (x = 1) \vee \forall y\ P(x, f(y))$

A_0' : $\forall x\ \ x=0 \vee x=1 \vee \forall y\ P(x, f(y))$

A_1 : $\Big[\forall x\ \ x \approx 0 \vee x \approx 1 \vee \forall y\ P(x, f(y))\Big] \wedge$

$(\forall x\ \ x \approx x) \wedge (\forall x, y\ \ x \approx y \supset y \approx x) \wedge$ transitivité $\wedge \Big[\forall x, y\ \ x \approx y \supset f(x) \approx f(y)\Big]$

$\wedge \Big(\forall x_1, x_2, y_1, y_2\ \ \big((x_1 \approx x_2) \wedge (y_1 \approx y_2) \supset P(x_1, y_1) \equiv P(x_2, y_2)\big)\Big)$

where $\approx$ is by def a binary relation constant.

Now,

- Obviously $A_0 \underset{valid}{\sim} A_0'$
- Assume obtained that $A_0' \underset{valid}{\sim} A_1$

e.g. Call $A_2 = \neg A_1$

to find if A_2 is unsatisfiable

A_3 : in prenex disjunctive normal form and $\vdash (A_3 \equiv A_2)$

Ex of mono plus

Dec 2 1959

$\forall x, y, y_1, z \quad p(x, f(a)) \wedge q(y, g(y)) \wedge (\neg p(a, y_1) \vee \sim q(y_1, z))$

Manna p 135

$x, y, y_1, z \longrightarrow a \qquad p(a, f(a)) \wedge q(a, g(a)) \wedge (\neg p(a,a) \vee \neg q(a,a))$

$\underbrace{p(a, f(a))}_{p_1} \wedge \underbrace{q(f(a), g(f(a)))}_{q_1} \wedge [\underbrace{\neg p(a, f(a))}_{\neg p_1} \vee \underbrace{\neg q(f(a), g(f(a)))}_{\neg q_1}]$

$x \longrightarrow a$
$y, y_1 \longrightarrow f(a)$
$z \longrightarrow g(f(a))$

Remark: A first order closed wff implies all its ground instances.
(obvious: $\forall$-elim)

Claim (1) and (2) is unsatisfiable, therefore (0) is unsatisfiable

Summary of the last step in the validity procedure.
From A_4, search for a finite set of ground instances whose conjunction is propositionally unsatisfiable.
If found halt and report that A_4 is unsatisfiable, hence A_0 is valid.
(otherwise diverge).

König's lemma: A finitely rooted branching tree is infinite iff it has an infinite path

Thm: This procedure halts on precisely the valid formulas.

Corollary: it is decidable whether a function-free, equality-free prenex $\forall \forall \ldots \forall \exists \ldots \exists$ 1st order wff is valid.

Pf: the Herbrand universe is finite in this case and so there are only finitely ground instances to check

$A_1: \forall x \; \exists y \; M(x,y)$
$A_3: \exists x \; \forall y \; \neg M(x,y)$
$A_4: \forall y \; \neg M(c,y) \qquad H_{A_4} = \{c\}.$

Lemma 1 Suppose that $A_1, A_2 \ldots$ is a set of quantifier-free closed 1st order wffs which is propositionally satisfiable.

(i.e. it lacks the $\underbrace{p(f(a),b)}_{A_1}$ or $q(g(f(a),b), \underbrace{f(a)}_{A_2}) \vee \neg p(a,b)$)

then $\{A_1, A_2 \ldots\}$ is logically satisfiable.

(i.e. there is a model $\mathcal{M}$ s.t. $\mathcal{M} \models A_i \quad \forall i \geq 1$)

Lemma 2 (Hyperstition compactness) Let $B_1, B_2 \ldots$ be a set of propositional formulas which is finitely satisfiable

((i) for every finite subset of B_i's there is a truth assignment which makes everything in the subset true) then $\{B_1, B_2 \ldots\}$ is satisfiable

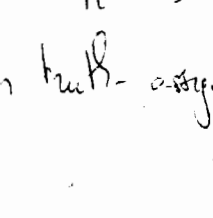
((ii) there is a single truth assignment which makes all B_i's true)

Lemma 0 If all ground instances of a universal formula A, is satisfiable then so is A, the set of

proof of Lemma 2 Let $p_1, p_2 \ldots$ be the propositional constant symbols in $B_1, B_2 \ldots$. Define a truth assignment to be a finite sequence of truths and falses.

A truth assignment $b_1, b_2 \ldots b_n$ is ON iff every formula B_i which only propositional constants are among $p_1 \ldots p_n$ is true under the assignment $p_i \longrightarrow b_i$

Define a "son of" relation on truth-assignments to be



iff

$$a_2 = \begin{cases} a_1 \cdot \text{true} \\ a_1 \cdot \text{false} \end{cases}$$

Note that if $b_1 \ldots b_n$ is ON then so is any ancestor of it, namely $b_1 \ldots b_i$ for $i \leq n$.

So that the ON assignments for a (binary) tree

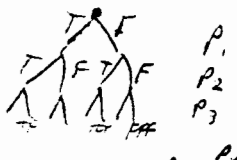
Moreover $\forall n$, $\exists$ assignment of $\{p_1, \dots, p_n\}$
(otherwise there would be formulas B_j $1 \le j \le 2^n$ s.t. B_j would be false under the j possible assignment, hence

$\bigwedge_{1 \le j \le 2^n} B_j$ is not satisfiable, contradicting fact that it is satisfiable

Apply Horn.

$\bigwedge$ RESTRICTED TO

(OTHERWISE THERE WOULD BE FORMULAS B_{ij} $1 \le j \le 2^n$ S.T. B_{ij} WAS FALSE UNDER THE j^{TH} POSSIBLE ASSIGNMENT, HENCE $\bigwedge_{1 \le j \le 2^n} B_j$ IS NOT SATISFIABLE, CONTRADICTING FINITE SAT.)



CLAIM: I MUST BE ABLE TO FIND AN OK NODE AT LEVEL n.

A NODE IS NOT OK BECAUSE A PARTICULAR FORMULA CAME OUT FALSE. IF ALL AT LEVEL n ARE NOT OK, THEN $\bigwedge_{1 \le j \le 2^n} B_{ij}$ IS UNSATISFIABLE: FOR ANY SET OF TRUTH ASSIGNMENTS, AT LEAST ONE OF THE FORMULAS IS FALSE.

∴ BY KÖNIG'S LEMMA, WE HAVE AN INFINITE PATH, I.E. AN INFINITE TRUTH ASSIGNMENT WHICH MAKES ALL B_i TRUE.

12/7 LEMMA 0 IF THE SET OF ALL GROUND INSTANCES OF A (CLOSED) UNIVERSAL FORMULA, A, IS SATISFIABLE, THEN SO IS A. (WITHOUT EQUALITY)

A SET (OF WFFS) IS SATISFIABLE = ONE INTERPRETATION MAKES ALL THE WFFS TRUE
(THIS IS STRONGER THAN SAYING THAT EVERY WFF IN THE SET IS SATISFIABLE)
FINITE SATISFIABILITY ⇒ SATISFIABILITY FOR PROPOSITIONAL WFFS + 1ST-ORDER WFFS
BUT NOT FOR 2ND-ORDER WFFS

PROOF SUPPOSE $\mathcal{I} \models A_i$ FOR ALL $i \ge 0$, WHERE $A_0, A_1, \ldots$ ARE THE GROUND INSTANCES OF A.

DEFINE $\mathcal{I}' = (\langle D', \mathcal{I}_c', \mathcal{I}_v' \rangle)$ WHERE D' = THE RELEVANT ELEMENTS OF THE MODEL $\mathcal{I}$.

$D' = \{ \mathcal{I}(t) \mid t$ IS A CLOSED TERM OVER THE SIGNATURE OF $A \}$

↳ THE DEFINABLE ELEMENTS - THOSE NAMEABLE AS THE VALUES OF TERMS

$\mathcal{I}_c'(f) = \mathcal{I}_c(f)$ RESTRICTED TO D' (NOTE THAT $\mathcal{I}_c(f)(d_1, d_2) \in D'$ IF $d_1, d_2 \in D'$ SO THE DEFINITION MAKES $\mathcal{I}_c'(f)$ STILL TOTAL.

$\mathcal{I}' = \mathcal{I}$ ↾ DEFINABLE ELEMENTS

f NEVER MAPS FROM D' TO $D \setminus D'$, ALWAYS FROM D' TO D'.

$\mathcal{I}_c'(P)$ IS DEFINED LIKEWISE.

$\mathcal{I}_v'$ I DON'T CARE ABOUT BECAUSE THE FORMULAS ARE CLOSED.

CLAIM $\mathcal{I}' \models A_i$ FOR ALL $i \ge 0$

PROOF OBVIOUS FROM DEFS SINCE A_i's ARE CLOSED.

BUT SINCE THE ONLY ELEMENTS IN $\mathcal{I}'$ ARE THE DEFINABLE ONES, IT FOLLOWS THAT A IS VALID IN $\mathcal{I}'$.

WHY HAVE UNDEFINABLE/UNNAMEABLE ELEMENTS AT ALL?

SUPPOSE $A = \forall x \forall y\ p(x,y,y) \vee q(f(x))$

GROUND INSTANCES
$p(c,c,c) \vee q(f(c))$
$p(f(c),c,c) \vee q(f(f(c)))$
⋮

IT IS POSSIBLE FOR A MODEL TO HAVE ELEMENTS THAT ARENT DEFINABLE
THEN THE CONJUNCTION OF FORMULAS HOLDS BUT THE UNIVERSAL FORMULA DOESN'T.
(THEY SNEAK IN VIA THE $\forall$ BUT AREN'T IN ANY GROUND INSTANCES).

HERBRAND UNIVERSE: ELTS THAT ARE THE VALUES OF CONSTANTS IN THE SIGNATURE. (THAT ARE IN A GIVEN TERM?)

LEMMA 0 GETS RID OF $\forall$, BUT CHANGES OUR FORMULA INTO AN INFINITE SET OF FORMULAS

LEMMA 1: A SET OF CLOSED QUANTIFIER-FREE FIRST-ORDER WFFS WITHOUT EQUALITY (NOT NEC. GROUND TERMS OF A FORMULA, BUT THATS A SPECIAL CASE HERE) WHICH IS PROPOSITIONALLY SATISFIABLE IS ALSO LOGICALLY SATISFIABLE

PROOF: SUPPOSE WE HAVE A TRUTH ASSIGNMENT $\{at_1 \to b_1,\ at_2 \to b_2, \ldots\}$ $\quad b_i \in \{\text{true}, \text{false}\}$

THE DOMAIN OF DISCOURSE IS WHAT? — ALL DISTINCT GROUND TERMS OCCURRING IN THE SET OF FORMULAS (EVERY GROUND FORMULA IS HERE ONCE)

SUCH THAT OUR SET OF FORMULAS IS PROPOSITIONALLY SATISFIED UNDER THIS ASSIGNMENT.

HOW TO GET THE TRUTH ASSIGNMENT:

$D = \{t \mid t \text{ IS A TERM OVER THE SIGNATURE OF SYMBOLS OCCURRING IN THE } \{at_i\}\}$

TRICKY HERE: DOMAIN IS SET OF TERMS

DISTINCTION TERM-MEANING IS BEING BLURRED HERE.

$I_c(f)(d_1, d_2) = "f(d_1, d_2)"$ MEANING OF $f(d_1, d_2)$ IS A STRING: $f(d_1, d_2)$.
(SYNTAX)

(CHECK BY INDUCTION ON DEFINITION THAT $I(t) = t$.)

$I_c(p)(d_1, d_2) = b_i$ WHERE $at_i = "p(d_1, d_2)"$ QED.

Thm compactness

Any finitely satisfiable set of 1st order wff's is satisfiable

Corollary: there is a model $\mathcal{N} \models \underbrace{Th_1(\langle \{a,b\}^*, a, b, \cdot \rangle)}_{\text{undecidable}}$ s.t. $\mathcal{N} \not\cong \langle \{a,b\}^*, a, b, \cdot \rangle$

proof: let P be a binary predicate etc, d an individual etc.

Let $\mathcal{F} = Th_1(\langle \{a,b\}^*, \cdot, a, b \rangle) \cup \{ \forall x\, P(x,x) \} \cup \{ \neg P(d,d) \mid s \text{ is a closed term} \} \cup \{ \forall x \forall y\, (P(x,y) \wedge P(y,x) \supset P(x,y)) \}$

eg: s is of the form: $(b \cdot (a \cdot a) \cdot b)$

Remark: $\mathcal{F}$ is finitely satisfiable. Choose $(\langle \{a,b\}^*, a, b, \cdot, =, P \rangle, \text{string } \not\equiv_d)$

where string $\neq$ value of any term in the given finite set of formulas to be satisfied.

$\therefore \mathcal{F}$ is satisfiable: let $\mathcal{N}$ satisfying $\mathcal{F}$.

$\mathcal{I}_c(d) \in Dom$, $\mathcal{I}_c(d) \neq \mathcal{I}_c(\delta)$ for any δ.

While Schemes -

$W :=$ skip | $(x := t)$ | $W_1 ; W_2$ (sequencing) | if A then W else W fi (at will) | while A do W od | var $x := t$ in W raw | (not recursive definition).

---

meaning: $[\![\ ]\!] : \text{Intps} \to \text{Intps}$ (partial) (actually the only thing that changes is I_v).

- $[\![x := t]\!] I = I[x \to I(t)]$
- $[\![W_1 ; W_2]\!] I =$ undefined if $[\![W_1]\!] I$ is undefined. Otherwise $[\![W_2]\!] ([\![W_1]\!] I)$
- $[\![\text{var } x := t \text{ in } W \text{ raw}]\!] = \left([\![W]\!] \left(I[x \to I(t)] \right) \right)[x \to I(x)]$

"WARNINGS ARE OBSOLETE"

6.044

10/19 DO DETERMINE IF A WFF IS TRUE, NEED

1) I_c, I_v — MEANINGS OF VARIABLES — 1st ORDER (LOTS), FUNCTIONS (HAVE ARITY), PREDICATES (OF VARIOUS ARITY)
   - MEANINGS OF CONSTANTS (CONSTANT-INTERPRETING FN) — CAN CONSIDER CST AS FCS OF NO ARGS. THEN J ONLY PREDICATES AND FUNCTIONS
   - DOMAIN

WHY DISTINCTION VARIABLES-CONSTANTS?

I_v — VALUATION

(D, I_c) — MODEL, $\mathcal{M}$

$= ((D, I_c), I_v)$ — AN INTERPRETATION

LET $\mathcal{M}$ = NONNEGATIVE INTEGERS WITH $+, \dot{-}, \times$ (SUBTRACTION RESTRICTED TO BE NONNEGATIVE.) THIS IS SOMETIMES CALLED ARITHMETIC

WE DON'T CARE WHETHER $\exists$ NUMERALS IN THE LANGUAGE.

$I_c(\dot{+}) = +$ — SYMBOL (BOLDFACE) — ADDITION.

DEF: $\mathcal{M} \models w$ "w IS VALID IN $\mathcal{M}$"

$\mathcal{M} \models w \Leftrightarrow_{\text{def}} (\mathcal{M}, I_v) \models w \quad \forall I_v$ (FOR ALL 'APPROPRIATE' I_v's. ASSIGN FCSS ON NONNEG. INTS, LOTS IN THE PROPER DOMAIN)

EX: $\text{ARITH} \models \exists y. \forall x. \; x \dot{-} x = y$ (NO FREE VARS)

$\text{ARITH} \models \exists y. \; y \dot{+} y \doteq y \wedge x \dot{-} x = y$ TRUTH VALUE NOT DETERMINED UNTIL I KNOW VALUE OF y

BUT NO MATTER WHAT INTEGER x IS, THIS EQUIVALENT TO TRUE

VALID IN ARITHMETIC DESPITE THE EXISTENCE OF A FREE VARIABLE.

$\models w \Leftrightarrow_{\text{def}} \mathcal{M} \models w \quad \forall \mathcal{M}$

"w IS VALID"

SIMPLEST VALID FORMULAS ARE THE PROPOSITIONALLY VALID ONES: $w \vee (\neg w)$

LOGICAL FORMULAS: $\models (\forall x. P(x)) \supset \exists x. P(x)$ — P IS A PROPOSITIONAL VARIABLE.

PROVE $A \supset B$ BY SHOWING IT'S NEVER THE CASE THAT $A \wedge \neg B$

THE DOMAIN MUST BE NONEMPTY: CRITICAL TECHNICAL ASSUMPTION

$(\forall x \in \{\}.$ ANYTHING$)$ IS TRUE

$\not\models \exists x. P(x) \supset \forall x. P(x)$

FALSE IN ANY INTERPRETATION IN WHICH SOME ELEMENT OF THE DOMAIN HAS P AND SOME OTHER ELT DOES NOT.

$\exists P. \neg((\exists x. P(x)) \supset \forall x. P(x))$ — FOR WHAT MODELS DOES SUCH A PREDICATE EXIST?

P IS NONTRIVIAL (ON THE DOMAIN) — WHAT SORTS OF SETS HAVE A NONTRIVIAL P?

$(\mathcal{M} \models \exists P.\ P \text{ NONTRIVIAL}) \Leftrightarrow |\text{Dom}| \geq 2$

MEANING OF THIS FORMULA IS THAT THE DOMAIN HAS AT LEAST TWO ELEMENTS.

REMARK: $\mathcal{M} \models w$ IFF $\mathcal{M} \models \vec{\forall}_{\vec{x}, \vec{P}}. w$

FREE VARIABLES OF w.

→ SO $(\exists x.\exists y.\neg(x=y)) \equiv (\exists P.\ P(x) \supset \forall x.\ P(x))$ IS A TRUE FORMULA

[ANOTHER WAY OF SAYING THE UNIVERSE HAS AT LEAST 2 ELTS: $\exists x.\exists y.\ x \neq y$ -OR- $\exists x.\exists y.\neg(x=y)$]

TECHNICAL DETAIL: DIFF. 1ST-ORDER WFFS AND 2ND-ORDER WFFS
↳ NO FUNCTION OR PREDICATE VARIABLES.
(SPECIAL CASE OF 2^{nd}-ORDER)

VALID-2:

THEOREM: $\{w \mid w$ IS A 2^{nd} ORDER WFF AND $\vdash w\}$ IS NEITHER R.E. NOR CO-R.E.

VALID-2 (TANTAMOUNT TO GÖDEL'S INCOMPLETENESS THM).

SUPPOSE I WRITE AUTOMATIC THEOREM-PROVING RULES TO GENERATE 2^{nd}-ORDER WFFS.
THEN IT CAN CHECK WHETHER A WFF IS VALID, BUT THAT MEANS THE SET OF TRUE FORMULAS IT CAN VERIFY IS R.E.
SO FORMAL RULES FOR PROVING (ALL) VALID TRUTHS CAN'T BE FOUND.

THEOREM (GÖDEL'S COMPLETENESS THEOREM IN ABSTRACT FORM)

VALID-1 = $\{w \mid w$ IS A 1^{st}-ORDER WFF AND $\vdash w\}$ IS R.E.
(GÖDEL ACTUALLY PROVIDED A SET OF RULES)

THEOREM (CHURCH'S THEOREM) VALID-1 IS NOT DECIDABLE.
CAN KNOW IF A 1^{st}-ORDER FORMULA ISN'T VALID.

AXIOMATIZING BINARY STRINGS $\{a,b\}^*$ (TO AVOID OVERLOADING 0-1)

10/24 DEFINE INDUCTIVELY:

1) Λ IS A BINARY STRING Λ, a, b ARE 3 OBJECTS, ALL DIFF F/EACH OTHER [NO OTHER PROPERTIES]

2) IF x IS A BINARY STRING, SO IS $a \cdot x$ AND SO IS $b \cdot x$ (CONCATENATION)

3) (EXTREMAL CLAUSE) THAT'S ALL. ALL BINARY STRINGS CAN BE BUILT F/PREVIOUS

BUT THIS IS NOT AN INDUCTIVE DEFN BECAUSE WE DON'T UNDERSTAND YET WHAT $\cdot$ (CONCATENATION) IS YET, AS STRINGS ARE JUST BEING DEFINED.

TAKE ANOTHER APPROACH.

AXIOMS

1) Λ, a, b ARE BINARY STRINGS

2) $\cdot$ MAPS BINARY STRINGS × BINARY STRINGS → BINARY STRINGS $\cdot : BS \times BS \to BS$

3) $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ ASSOCIATIVITY

4) $\Lambda \cdot x = x \cdot \Lambda = x$ Λ IS 2-SIDED IDENTITY

5) $ax \neq bx$; FURTHER, $ax \neq by$

6) $ax = ay \Leftrightarrow x = y$ ← TRIVIAL, UNINTERESTING.
$bx = by \Leftrightarrow x = y$

7) eqlength $(\Lambda, x) \Leftrightarrow$ eqlength $(x, \Lambda) \Leftrightarrow x = \Lambda$
eqlength $(ax, ay) \Leftrightarrow$ eqlength $(bx, ay) \Leftrightarrow$ eqlength $(ax, by) \Leftrightarrow$ eqlength $(bx, by) \Leftrightarrow$ eqlength (x, y)
CONNECT THE TWO PARTS OF 7 WITH $\wedge$ TO MAKE IT A SINGLE FORMULA

IFF IS $\equiv$ $w_1 \equiv w_2 \Leftrightarrow (w_1 \supset w_2) \wedge (w_2 \supset w_1)$ $\supset$: IMPLICATION

(FUNDAMENTAL WEAKNESS OF 1ST-ORDER DEFINITIONS IS PRESENT HERE)

ALL 7 AXIOMS CAN BE MADE 1 BIG FIRST-ORDER PREDICATE CALCULUS FORMULA (WHICH HAS THE OMITTED UNIVERSAL QUANTIFIERS).
(#2 WOULDN'T BE PART OF THE FORMULA)

W_{BS} IS THAT FORMULA (CLOSED, HAS EQUALITY). (3 CONSTANTS, 1 FUNCTION, 1 PREDICATE): eqlength. NO FREE VARS

SUPPOSE $\mathcal{M}_i \models W_{BS}$ FOR $i = 1, 2$. 2 MODELS SATISFY THE FORMULA.
IF THIS DID A PERFECT JOB THEN $\mathcal{M}_1$ AND $\mathcal{M}_2$ ARE ISOMORPHIC
INTUITIVELY: SAME SIGNATURE

BIJECTION BETWEEN THE TWO DOMAINS: IS-NAME-OF-SAME-STRING?
└ 1-1 ONTO MAPPING

$(f: D_1 \to D_2$, A BIJECTION s.t. $f(d_1 \cdot d_1') = f(d_1) \cdot f(d_1')$
↳ $\cdot$: CONCAT IN MODEL 1 → CONCAT IN MODEL 2, $\cdot$:
eqlength$_1(d_1, d_1') \Leftrightarrow$ eqlength$(f(d_1), f(d_1'))$
a's CORRESPOND, b's CORRESPOND.
THEN IT FOLLOWS THAT THE Λ's CORRESPOND (FROM 4) — ONLY ONE THING HAS THE Λ IDENTITY

WITH OUR DEFN, THOUGH, THIS DOESN'T HOLD! ☹

ALL THAT HOLDS IS THE FOLLOWING:

$\mathcal{M} \models W_{bs} \Rightarrow \langle \{a,b\}^*, a, b, \Lambda, \cdot, \text{eqlength} \rangle$ is ~~[illegible]~~ ISOMORPHIC TO A SUBMODEL OF $\mathcal{M}$.

SUB MODEL: GIVEN SUBSET of DOMAIN CLOSED UNDER OPERATIONS (eg CONCATENATION)

THE AXIOMS DON'T PREVENT THE EXISTANCE OF OTHER GARBAGE.

(YOU CAN'T, WITH A FIRST ORDER FORMULA. NO WAY of SAYING "THAT'S ALL")

NONSTANDARD MODEL of W_{bs}:

$D = \{a,b\}^* \cup \underline{\{a,b\}^\omega}$

└ THE ONE-WAY INFINITE STRINGS (BEGINNING BUT NO END)

$x \cdot y = \begin{cases} xy & \text{IF } x \text{ IS FINITE} \\ x & \text{IF } x \text{ IS INFINITE} \end{cases}$

eqlength ON FINITE STRINGS IS BEFORE

ALL INFINITE STRINGS HAVE EQUAL LENGTH.

NO FINITE STRING EQUENGTH TO AN INFINITE STRING.

2^{nd} ORDER DEFN WILL FIX THIS UP.

10/28 $\exists$ 1^{ST} ORDER FORMULA W_{bs} s.t. $\left(\mathcal{M} \models W_{bs} \Rightarrow \langle \{a,b\}^*, \cdot, \text{eqlength} \rangle \xrightarrow[\text{ISOMORPHISM}]{1-1} \mathcal{M}\right)$

WE WON'T PROVE THIS.

PROB F/LAST TIME: THE MODEL COULD INCLUDE OTHER STUFF.

IS THIS A WEAKNESS of W_{bs}? (WE DIDN'T WRITE ENOUGH AXIOMS)

$\mathcal{M}$ DOESN'T HAVE ALL THE PROPERTIES OF BINARY STRINGS, HOWEVER

WANT TO DEFINE $W_{bs\text{-}honestly} = W_{bs} \wedge ONLY_{bs}$

CAN'T DO THIS BY ADDING AXIOMS LIKE

$y \cdot x = z \cdot x \supset y = z$

THEOREM: CAN'T IMPROVE W_{bs} USING ONLY FIRST-ORDER PREDICATE CALCULUS

REMARK

$\not\vdash W_{bs} \supset \forall x,y,z \left(y \cdot x = z \cdot x \supset y = z\right)$

BUT THIS CONCLUSION ISN'T WARRANTED BY W_{bs}; RIGHT-CANCELLATION DOESN'T HOLD.

(THINK of $7 \cdot 0 \neq 9 \cdot 0$)

$a \cdot a^\omega = aa \cdot a^\omega$ BUT $a \neq aa$

2^{nd}-ORDER LOGIC CAN DO IT RATHERLY, AND IN FACT ONLY$_{bs}$ MUST BE 2^{nd} ORDER

$(*)$ $\mathcal{M}' \models W_{bsh} \Leftrightarrow \left(\mathcal{M}' \text{ ISOMORPHIC TO } \langle \{a,b\}^*, \cdot, \text{eqlength} \rangle\right)$

[AS LONG AS SIGNATURE of $\mathcal{M}'$ IS SAME; ie $\mathcal{M}'$ DOESN'T EXCLUDE OPERATIONS LIKE REVERSE AS WELL]

INDUCTION ON STRINGS

$\forall P.\ P(\Lambda) \wedge \forall x.\ P(x) \supset (P(ax) \wedge P(bx))) \Rightarrow \forall z.\ P(z)$

THE "$\forall P$" IS WHAT MAKES THIS FORMULA 2^{nd}-ORDER

$\langle \{a,b\}^*, \cdot, \text{eqlength} \rangle \models$ ● only/$_{bs}$

IF P = PROPERTY OF BEING A FINITE STRING, WE RULE OUT THE $\mathfrak{M}$ WHICH INCLUDES THE FINITE & INFINITE STRINGS.

WE COULD PRODUCE ALL THE STRINGS BY STARTING w/ Λ AND ADDING a's AND b's

THEOREM $\exists$ 2^{nd}-ORDER FORMULA w/ SIGNATURE $\cdot$, eqlength w_{bsh} s.t. (*)

RESTATING: $\exists\ w_{bsh}.\ (*)$

$Th(\text{binary-string})$ WHERE binary-string $= \langle \{a,b\}^*, \cdot, \text{eqlength} \rangle$

↳ 1^{st} ORDER THEORY OF THE STRUCTURE

ALL 1^{st}-ORDER FORMULAS VALID IN THE STRUCTURE

$Th_2(b\text{-}s)$ – ALL $1^{st}+2^{nd}$-ORDER FORMULAS VALID

I WANT A THEOREM PROVER: IS ϕ A PROPERTY IN $Th(b\text{-}s)$?

COROLLARY $Th(b\text{-}s) \leq_m Valid_2(\mathbb{C}) \triangleq \{w \mid \vdash w \wedge w$ HAS NO NONLOGICAL SYMBOLS$\}$ $\Big[$(w IS A WFF OF 2^{nd}-ORDER PRED. CALCULUS)

$\Leftarrow \{w \mid (b\text{-}s \vdash w) \wedge (w\ 1^{st}\text{-ORDER})\}$

→ FORMULAS OF 2^{nd}-ORDER PREDICATE CALCULUS W/O CONSTANTS (IE, IT IS CLOSED) WHICH ARE TRUE IN ALL MODELS.

INTUITIVELY, 2^{nd}-ORDER FACTS THAT HOLD IN GENERAL VS. 1^{st}-ORDER FACTS THAT HOLD IN PARTICULAR.

PROOF LET w BE A 1^{st}-ORDER FORMULA WITH SIGNATURE $\cdot$, eqlength.

$w \in Th(b\text{-}s)$; iff $b\text{-}s \vdash w$; iff $\forall \mathfrak{M}.$ [if $\mathfrak{M}$ is b-s. then $\mathfrak{M} \vdash w$]

iff ● $\mathfrak{M} \vdash w_{bsh} \supset w$ FOR ALL $\mathfrak{M}$

iff $\vdash w_{bsh} \supset w$ → HERE THERE ARE TWO CONSTANTS · AND eqlength

iff $\vdash \forall \cdot, \text{eqlength}.\ w_{bsh} \supset w$ → REPLACE CONSTANTS BY VARIABLES, THEN QUANTIFY

→ CLOSED, 2^{nd}-ORDER FORMULA W/O CONSTANTS

EX: $\vdash f(0)=0 \supset f(f(0))=0$

$\Leftrightarrow \vdash \forall F.\ \forall z.\ F(z)=z \supset F(F(z))=z.$

iff $(\forall \cdot.\ \forall \text{eqlength}.\ w_{bsh} \supset w) \in Valid_2(\phi)$

THEOREM (CLOSE TO GÖDEL'S INCOMPLETENESS THEOREM)

$Th(b\text{-}s)$ IS NEITHER R.E. NOR CO-RE.

COROLLARY ● $Valid_2(\phi)$ IS NEITHER R.E NOR CO-R.E

THEOREM

10/31 $Th(\text{bin-strings})$ IS NEITHER R.E. NOR CO-R.E.

CAN'T GET A PRECISE GRIP ON 2^{ND}-ORDER LOGIC FROM A COMPUTATIONAL POINT OF VIEW.

LEMMA: $Th(b\text{-}s) \leq_m \overline{Th(b\text{-}s)}$

PROOF: $w \in Th(b\text{-}s) \iff \neg w \notin Th(b\text{-}s)$ FOR ALL WFFs w WITH SIGNATURE $\{:, \text{eglength}\}$

IF INPUT NOT PARSEABLE, MAP IT TO $\forall x.\, x=x$. THEN [redacted] OUTPUT IS WELL-FORMED AND TRUE. THIS SATISFIES THE FORMULA ALL WELL-FORMED INPUTS ARE TRUE OR THEIR COMPLEMENTS ARE TRUE.

LEMMA $K_0 \leq_m Th(b\text{-}s)$

WANT TO DEFINE FORMULA $T(x,y,z)$ WITH EXACTLY THREE FREE VARIABLES

SUCH THAT $(b\text{-}s, \mathcal{A}_v) \models T \iff M_{\mathcal{A}_v(x)}$ ON INPUT $\mathcal{A}_v(y)$ HALTS AND $\mathcal{A}_v(z)$ IS THE CODE OF ITS COMPUTATION HISTORY.

(ASSUME TMs WORK ON INPUTS: a's AND b's)

MACHINE x ON INPUT y HALTS, AND z IS THE PROOF (HOW IT DOES IT).

z: WORD OVER (TM ALPHABET + STATES + DELIMITERS)

WANT [redacted] IF y = EMPTY WORD, PUT A BLANK SYMBOL THERE.

$z = \$ \text{(START)} by' \$ b \text{(BOX 2)} y' \$ \ldots \$ \ldots \text{(HALT)} \ldots \$$ COMPUTATION HISTORY WORD.

PICTURES OF TAPE AFTER EACH STEP.

COMPUTATION SNAPSHOTS DIFFER ONLY NEAR THE HEAD.

GIVEN SUCH A FORMULA T, TO REDUCE $K_0 \leq Th(b\text{-}s)$:

MAP $(M, y) \longmapsto \exists z.\, T(d(M), y, z)$

↳ TM [redacted] INPUT ALPHABET IS a's + b's

HERE WE CAN'T USE $d(M)$. WE MEAN "THE TERM THAT DENOTES $d(M)$."

$d(M)$ ISN'T A TERM BUT SHORTHAND FOR A TERM. eg $\cdot(\cdot(\cdot(\cdot(a,b),b),a),b)$

A BINARY WORD ISN'T A TERM

NOW BACK TO THE CONSTRUCTION OF THE FORMULA T (DO IT TOPDOWN):

STARTBOX$(x,u) \iff u$ IS THE CODE OF THE START BOX OF M_x.

NEXTBOX$(x,u,s,v) \iff v$ IS THE CODE OF THE FLOW CHART BOX WHICH M_x EXECUTES NEXT WHEN IT IS IN [redacted] READING THE SYMBOL CODED BY [redacted] ON ITS TAPE.

ST PAIR(x,y) = DOUBLE$(x) \cdot a \cdot b \cdot y$

DOUBLE$(\Lambda) = \Lambda$

DOUBLE$(a,z) = aaz$

DOUBLE$(b,z) = bbz$

CONVENTIONS RE: CODE OF A TM, x: BIT STRING VERSION OF A FINITE GRAPH.

$d(M) = \text{pair}(\text{STARTBOX}, \text{FLOWCHART-CODE})$

STARTBOX(x,u) IFF $\text{left}(x) = u$ IFF $\exists u'.\, [x = u'ab\cdot y] \wedge [u'$ DOES NOT HAVE ANY [redacted] ab's IN ODD-EVEN POSITIONS $] \wedge$ [illegible]