JOHNSON	KIRK	LAURITZ		6-3	'88
PRINT IN LARGE LETTERS (Family Name)	(First Name)	(Middle Name)		(Course)	(Year)

6.044J	COMPUTABILITY, PROGRAMMING & LOGIC	17 DEC 87
(Subject Number)	(Subject Name)	(Date)

A. MEYER
(Instructor's Name)

IF EXAMINATION FOR ADVANCED STANDING CHECK HERE ☐

IF A CONDITION EXAMINATION, FILL IN BELOW		IF A POSTPONED FINAL EXAMINATION, FILL IN BELOW	
(Year and term taken in Class)	(Instructor's Name)	(Year and term taken in Class)	(Instructor's Name)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Do not write below. Reserved for Examiner.

1. 10
2. 07
3. 10
4. (a) 3 (b) 3 (c) 0 (d) 10 total: 16
5. (a) 00 (b) 5 total: 05
6. (a) 10 (b) 10 total 20
7. 15
8. ~~20~~ 18 am
9.
10.
11.
12.
13.
14.
15.

~~100~~ ~~100~~

101 am / 200

6-72 30M S14

Problem 1:

Assume R is recursive. By virtue of R's recursiveness, f(x) as defined is total. Let M be a machine which computes f(x) on input x. For some M_0,

$M_0 \in R$ iff $f(M_0) = b$ [by defn of f(x)]

iff $M_0 \notin \bar{R}$

implies (⇒) ~~iff~~ $M_0 \notin K_b$ [$K_b \subseteq \bar{R}$]

iff M_0 does not output b on input $d(M_0)$ [by defn of K_b]

10/20

Letting $M_0 = M$ (which computes f(x)), we obtain:

$M \in R$ iff $f(M) = b$

iff M does not output b on input d(M)

iff $f(M) \neq b$ [because M computes f]

Contradiction. Thus the assumption that R is recursive is incorrect.

→ You have only shown that $M \notin R$. To complete the argument you need also to show that $M \notin \bar{R}$.

Problem 2:

(a) Manna's construction "simulates" Post machine behavior with a Post system. In effect, this simulation corresponds to generating a Post Machine computation word; the Post system solution provides a description of how to fit together finite length Post System rules to obtain this computation word. In the case of a divergent Post machine, the computation word becomes an (infinite) computation stream. The Post system rules can still be used to generate this stream, but since it is of infinite size and each rule is finite, ~~the~~ Post system solution is infinite ✓

(b) Under Manna's construction, a Post machine diverges (iff) if it corresponds to a Post system with an infinite solution. (part (a) ...) Call this construction $\mathcal{M}$.

P is a post machine. HP = { P | P halts on input Λ }.
HP is r.e. ($\overline{HP}$ is not r.e)

$P \in \overline{HP}$ iff P diverges on input Λ.
iff Post system $\mathcal{M}(P)$ has an infinite soln.
iff $\mathcal{M}(P) \in$ IPCP

−18

$\overline{HP} \leq_m$ IPCP by $\mathcal{M}$.

$\overline{HP}$ is not r.e., and non-r.e.-ness "inherits upward" over $\leq_m$.
Thus IPCP is not r.e.

Problem 3:

Define F to be a first-order wff which (when valid) says that the domain of the interpretation has only one element.

$$F ::= \forall x \forall y .\ x = y$$

$\mathcal{E} \in DG$ iff $\forall I, (I \models \mathcal{E}) \supset (\text{core}(I) \models F)$ [defn of DG]

iff' $\mathcal{E} \models F$ [rewrite]

iff $\mathcal{E} \vdash F$ [completeness]

only for semigroup interpretations

10/25

Since $\{\mathcal{E} \mid \mathcal{E} \vdash F\}$ is r.e., DG is also r.e.

mixing up semigroup logic and first-order predicate calculus logic.

Problem 4:

(a) $\forall Y \forall x .\ X \neq \langle x \rangle \cdot Y$ (3/3) [Y is a sequence variable]

(b) $\exists Y \exists Z .\ X \doteq Y \cdot \langle x \rangle Z$ (3/3) [Y, Z are sequence variables]

(c) (0/4)

(d) Statement of (In)compactness for s-wffs:

(10/10)

"Let $\{A_1, A_2, \ldots\}$ be a set of s-wffs which is finitely satisfiable. This does not imply $\{A_i\}$ is satisfiable."

- Let F be an s-wff which is valid precisely in those models with finite (but non-empty) domain.
- For each $n \geq 1$, let A_n be the first order wff
  $\exists x_1 \cdots \exists x_n [\bigwedge_{i \neq j} x_i \neq x_j]$
  which says that the domain has at least n elements.

Any finite subset of $\{F\} \cup \{A_n \mid n \geq 1\}$ is satisfiable by chosing a model with appropriately large finite domain.

$\{A_n \mid n \geq 1\}$ is only satisfiable by models with infinite domain. Thus $\{F\} \cup \{A_n \mid n \geq 1\}$ is unsatisfiable.

And thus we have the (In)compactness result stated initially.

Problem 5:

(a)

0/20

(b) $Th(\{a,b\}^*) \leq_m$ s-Valid,

4/5

$Th(\{a,b\}^*)$ is neither r.e. nor co-r.e.

Since non-re-ness "inherits-up" (over $\leq_m$),

s-Valid is neither r.e. nor co-r.e.

What exactly does Gödel's Incompleteness theorem say?

Problem 6:

(a) I is Λ-free. Show $(T)_I \neq \Lambda$ by induction.

$T ::= T \cdot T \mid mkseq(t) \mid X$

(1) $T = T_1 \cdot T_2$

$(T)_I = (T_1 \cdot T_2)_I = (T_1)_I \cdot (T_2)_I$

by inductive hypothesis, neither $(T_1)_I$ nor $(T_2)_I$ is Λ.

thus $(T)_I \neq \Lambda$.

(2) $T = mkseq(t)$

$(T)_I = (mkseq(t))_I = \langle (t)_I \rangle$

$\langle (t)_I \rangle$ is not the empty sequence.

thus $(T)_I \neq \Lambda$.

10/10

(3) $T = X$

$(T)_I = (X)_I = I_V(X)$

Since I is Λ-free, $I_V(X) \neq \Lambda$. thus $(T)_I \neq \Lambda$.

- so $(T)_I \neq \Lambda$ for all sequence terms T.

(b) If a swps halts with $X = \Lambda$, either

① $X = \Lambda$ before swps ran and was left unchanged

or ② the swps set $X = \Lambda$.

New sequence-terms (possible values of X in ②) can only be constructed via "$T ::= T \cdot T \mid mkseq(t) \mid X$", thus, under Λ-free interpretations there is no way to construct a Λ sequence term. Similarly, a Λ-free interpretation disallows ① as a method for halting with $X = \Lambda$.

In a Λ-free interpretation, X cannot start equal to Λ and X cannot be set to Λ by any swps (without ϵ). Thus there is no

Prob 6 continued:

swps (without ∈) which halts with $X = \Lambda$ for all interpretations.

(−5) implicit induction on # steps should be made explicit.

Problem 7:

Prove using Hoare logic that $\{true\}\ W\ \{\exists z.(X \doteq z \cdot z) \equiv (x = a)\}$

define $R(X, Y, x)$: $(Y \neq X) \vee (\exists z.((X) \doteq z \cdot z) \equiv (x = a))$ Y ?

① $\{true\} \supset R(X, \epsilon, a)$ [by expansion of R]

② $\{true\}\ Y := \epsilon;\ x := a\ \{R(X, Y, x)\}$ [assignment rule]

③ $R(X, Y, x) \supset R(X, \langle a \rangle \cdot \langle a \rangle \cdot Y, x)$ [by expansion of R]

④ $\{R(X, Y, x)\}\ Y := \langle a \rangle \cdot Y\ \{R(X, \langle a \rangle \cdot Y, x)\}$ [assignment rule]

⑤

?

don't understand how to formulate the loop invariant, R. this is unfortunate, because I think that I understand Hoare logic; it is just one silly trick/transformation that I can't see.

-15

KHALEEL ROHAN 6 4

PRINT IN LARGE LETTERS (Family Name) (First Name) (Middle Name) (Course) (Year)

6.044J Intro to Computability Theory Dec 17, 1987

(Subject Number) (Subject Name) (Date)

Prof. Meyer

(Instructor's Name)

IF EXAMINATION FOR ADVANCED STANDING CHECK HERE ☐

IF A CONDITION EXAMINATION, FILL IN BELOW		IF A POSTPONED FINAL EXAMINATION, FILL IN BELOW	
(Year and term taken in Class)	(Instructor's Name)	(Year and term taken in Class)	(Instructor's Name)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Do not write below. Reserved for Examiner.

1. 00
2. 00
3. 00
4. (a) 3 (b) 1 (c) 0 (d) 0 total: 04
5. (a) 00 (b) 0 total: 00
6. 00
7. 00
8. (a) 1 (b) 00 (c) 00 total: 01
9.
10.
11.
12.
13.
14.
15.

05/200

6-72 30M S14

(1) Consider $f(x) = \begin{cases} b & \text{if } x \in R \\ a & \text{if } x \notin R \end{cases}$

M halts with head @ b if X in R
M halts with head @ a if X not in R

Fact: K_a, K_b equivalent to K_0
K_0 is undecidable
K_0 is r.e.
K_0 is not co-r.e

(00/20)

- A set is recursive iff it is r.e. and co-r.e.

Hence: K_a, K_b Not recursive *true but not relevant.*

Using the example

$f(x) = \begin{cases} b & \text{if } x \in R \\ a & \text{if } x \notin R \end{cases}$ we conclude that R is not recursive since

what is the reduction?

$K_a, K_b \leq_m R$ and K_a, K_b Not recursive implies that R is not recursive.

2(a) In the Post Correspondence problem, the rewrite rules during computation are the following: $(*B_i*a, B_j*), (*B_i*b, B_k*) (*B_i*\#, B_l*)$

If infinite applications of these rules were allowed, a there would be an infinite soln.

? -5

(2b) -20

(3)

$\{ I(v) \mid v \in \{a,b\}^* \}$

$DG = \{ \varepsilon \mid \varepsilon$ is a degenate set of semigroup eqns $\}$
Prove M_{DG} halts for some input $x \in \{a,b\}^+$
$\{E\} \vdash F$ iff $\{E\} \models F$

By completeness $\{E\}$ proves F iff $\{E\}$ satisfies F

Fact: The Core has one element
DG is finite NO

We know that one DG has a valid interpretation? since the Core of DG has one element. Call the interpretation I.
(00/25)
$I \models DG$

$DG \models F$ since there is only one interpretation I such that $I \models DG$. This interpretation works for any particular F.

Given that there is only one interpretation I which satisfies the equations in DG and $DG \models F$ for F an equation in the set, It follows that we can construct a Turing Machine which halts for some input x.

(4)

a) $\forall B. B \doteq B.X$ ✓ (3/3)

b) $\forall B. \forall C. B. \langle x \rangle . C \doteq X$? (1/3)

c) $\forall A \forall \neg S. \neg (S. A \doteq A. S)$ ← for clarity (0/4)

what does this mean?

d) (00/10)

6. 00

7. 00

(8)

(a) spectrum(w) is re. Since it is reducible to K_0, the halting problem on input x.

(1/5)

Since K_0 is r.e. spectrum(w) is r.e.

PRINT IN LARGE LETTERS (Family Name) (First Name) (Middle Name) (Course) (Year)

PAGANO STEPHANIE FRANCES (WELLESLEY) 1988

(Subject Number) (Subject Name) (Date)

6.044J COMPUTABILITY/LOGIC & PROGRAMMING 12/17

MEYER

(Instructor's Name)

IF EXAMINATION FOR ADVANCED STANDING CHECK HERE ☐

IF A CONDITION EXAMINATION, FILL IN BELOW		IF A POSTPONED FINAL EXAMINATION, FILL IN BELOW	
(Year and term taken in Class)	(Instructor's Name)	(Year and term taken in Class)	(Instructor's Name)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Do not write below. Reserved for Examiner.

1. [redacted] 07
2. 07
3. 01
4. (a) 0 (b) [redacted]2 (c) 0 (d) 0 total [redacted] 02
5. (a) 0 (b) 5 total: 05
6. 17
7. 00
8. (a) 5 (b) 03 (c) 0 total: 08
9.
10.
11.
12.
13.
14.
15.

47/200

6-72 30M S14

(1) Proof by contradiction

Suppose R separates Ka and Kb then

$Ka \subseteq R$ and $Kb \subseteq \overline{R}$

M is the machine s.t.

$$f(x) = \begin{cases} b & \text{if } x \in R \\ a & \text{if } x \notin R \end{cases}$$

(07/20)

M will output a "b" on the input d(M′) for M′ ∈ Ka since Ka ⊆ R and will output an "a" on the input d(M″) for M″ ∈ Kb since Kb ⊆ $\overline{R}$ for all M′, M″. ✓ If we set M = M′ then M will output a "b" on the input d(M′) but M′ will output an "a" on d(M′) and we know that M = M′ so we have a contradiction. If we set M = M″ then M will output an "a" on the input d(M″) but M″ will output a "b" on d(M″) and we know that M = M″ so we have another contradiction

What does this mean? Do you mean assume M ∈ Ka?

② ⓐ This is because we know that for each step in the post machine flowchart adds a pair to the Post system. If the flowchart never halts then the number of pairs will never halt. The resultant pairs solve the PCP iff the post machine halts. We also know that if we start with the first pair of the resultant pairs and trace through the pairs that this simulates the flow chart so if the flowchart doesn't halt then there must be an infinite number of pairs in the PCP. ✓

ⓑ If we show that the comple of the halting problem for Post machine's $\leq_m$ to the complement of the halting prob for post machines then we know that IPCP is not r.e. We can do this because we know that the post machine halting problem $\leq_m$ PCP halting problem and if $A \geq_m B$ then $\overline{A} \geq_m \overline{B}$ and if $\overline{A}$ is not r.e. then $\overline{B}$ is not r.e. Therefore, the complement of the halting problem for Post machines $\leq_m$ IPCP and since the complement of the halting problem for Post machines is not r.e. then IPCP is not r.e.

(−18)

Look back a few pages for ③

(5)

a)

(00/10)

b) Since Th({a,b}*) ≤m S-Valid then we know that since Th({a,b}*) is not r.e. then S-Valid is not r.e. Therefore a Gödel's Incompleteness Th for sequence logic is that the set of valid S-wffs is not r.e.

(5/5)

(4)

a) $\forall Y,Z\,(Y \cdot Z \not\doteq X)$ × (0/3)

could have $Y = Z = \Lambda$.

b) $\exists Y \exists Z.\,(Y \cdot \langle X \rangle Z \doteq X)$ ✓ (2/3)

c) (0/4)

d) An infinite set of sequence logic formulas which are finitely satisfiable are satisfiable. This means that if any finite set of sequence logic formulas of the infinite set anded together has a model then the whole infinite set has a model.

From problem set seven we know that the domain of a model which satisfies an infinite set of 1st order wffs has an infinite domain and from part c of this problem

NO

we know that there is an s-wff which is valid in only those models with a finite domain, therefore, ↓

(0/10)

a model with infinite domain may not satisfy all of the sequence logic formulas in an infinite set of them.

⑥ ⓐ Prove by induction on the definition of sequence terms.

Prove for T·T, mkseq(t), and X

We know that $I_V(x) \neq \perp$ for all X. ~~therefore,~~ we know that ~~the interpretation of~~ any nonzero finite sequence of elements of the domain is not ⊥. Therefore, we know that any sequence made out of a first order term ≠ ⊥ so mkseq(t) ≠ ⊥. We know that $(T_1 \cdot T_2)_I = (T_1)_I (T_2)_I$. We have already shown that mkseq(t) ≠ ⊥ and X ≠ ⊥ so we know that $T_1 \neq \perp$ and $T_2 \neq \perp$ so we then know that the concatenation of these ≠ ⊥. ✓

ⓑ If an interpretation is not ⊥-free ~~then~~ the only way for X to be equal to ⊥ after the swps halts is for it to be set to ⊥ during the execution of the swps so the symbol ∈ is needed to do this.

unclear (−8)

(8)

(a) spectrum (W) is r.e. for every SWPs W because we simply need to run W started in the interpretation (Z_n, I_0) and if it halts then accept n.

(5/5)

(b) while $z \neq 0$ do

~~begin~~ —— assume initialized at 1

$x := \langle 0 \rangle \cdot x$

$z := z + 1$

~~end~~

od

(halts without ever executing the ~~loop since~~ body of the loop since z begins at 0)

(3/10)

because it is mod n addition we know that the loop will execute n times because z will become zero when it equals n under mod n addition.

(c)

(0/15)

problem 3 not problem 7

③ We know by completeness that $\{E_i\} \vdash F$ iff $\{E_i\} \models F$. To see if $\{E_i\}$ is a subset of D6 we must show that an equation F from D6? can be written from the Thue system whose rules are $\{E_i\}$. In essence if $\mathcal{I}$ satisfies all the equations in the set $\{E_i\}$ then $\mathcal{I} \models F$. Since we know that if F is a member of D6 then if F is $x=y$ then $\mathcal{I}(x) = \mathcal{I}(y)$ but we already know this since F is a member of the degenerate set.

-24

7 00

REALFF MATTHEW 10 G

PRINT IN LARGE LETTERS (Family Name) (First Name) (Middle Name) (Course) (Year)

6.044 COMP. PROG. LOGIC 12/17/87

(Subject Number) (Subject Name) (Date)

PROFESSOR MEYER

(Instructor's Name)

IF EXAMINATION FOR ADVANCED STANDING CHECK HERE ☐

IF A CONDITION EXAMINATION, FILL IN BELOW		IF A POSTPONED FINAL EXAMINATION, FILL IN BELOW	
(Year and term taken in Class)	(Instructor's Name)	(Year and term taken in Class)	(Instructor's Name)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Do not write below. Reserved for Examiner.

1. 08.
2. 07
3. 00
4. (a) 3 (b) 3 (c) 0 (d) 0 total : 06
5. (a) 00 (b) 5 total : 05
6. (a) 00 (b) 01 total : 01
7. 12
8. 02
9.
10.
11.
12.
13.
14.
15.

41/200

6-72 30M S14

1.

$$f(x) = \begin{cases} b & \text{if } x \in R \\ a & \text{if } x \notin R \end{cases}$$

This is a Recursive total function, thus it is computable

Much as in Problem set 2 Qu 4.

M: " On input x compute $f(x)$ iff $x \in R$ print a "b"
iff $x \notin R$ print an "a" "

Consider this Machine M computing on its own input $d(M)$.

$K_a \subseteq R$ and $K_b \subseteq \overline{R}$ iff R separates K_a and K_b

8/20

On input $d(M)$ compute $f(d(M))$ iff $d(M) \in K_a$ print a "b"

$d(M) \in K_a$ iff M on input $d(M)$ outputs an "a"
$\in R$ iff on input $d(M)$ M outputs a "b"
* contradiction

Thus we were wrong to assume that $f(x)$ was computable and total, since $f(x)$ is exactly the function necessary to separate K_a and K_b there can be no recursive set separating K_a and K_b.

Also if $d(M) \in K_b$ iff M on input $d(M)$ outputs a "b"
$\notin R$. iff M on input $d(M)$ output an "a"
* contradiction.

=> so $d(M) \notin K_a, K_b$; that is not yet a contradiction.

very confused.

2.

a) If we diverge then we never reach the Accept/Reject box. thus we never arrive at a (x←tail(x)) with Δ. We are looping and thus it must be possible to keep executing some subset of rules and keep going, we execute an infinite sequence of B_i ... which is the same as having an infinite sol^n to PCP, because failure of the rules will cause us to go to the Reject box. ✓

($* B_i * B_{m+1}$, B_{m+1}). is the corresponding PCP instruction

b)

1. Unmodified

$\bar{K}_{HP, \text{post machines}} \leq_m$ IPCP

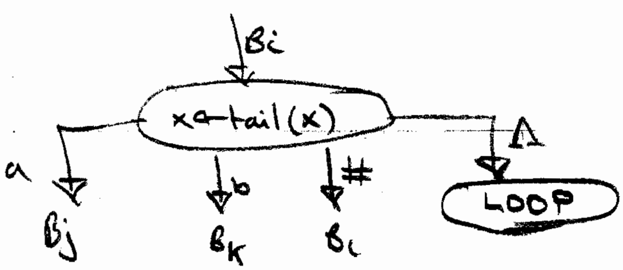
no, not w/o modification

$\bar{K}_{HP}$ is not-re

not-re inherits up.

∴ IPCP is not-re.

2.



?

−18

3.

Completeness states

Logically true iff provable.

$\models$ iff $\vdash$

0/25

Consider the interpretations of the set of equations E they will all yield only a single element each by definition, this will be an r.e set of elements and by completeness the sets of equations that

~~{set of equations} $\vdash x \equiv y$ iff $\models x \equiv y$ but this case single element~~

~~$\models x \equiv y$ means (2x) =~~

derive these elements will also be r.e.

4.

a) [illegible] ~~[illegible]~~

$$X . X \doteq X$$ (3/3) (only the empty Λ concatenates with itself to give itself.)

b) ~~$\exists X_2 ((\langle x \rangle . X_2) \doteq X) \wedge \exists X_3 ($~~

~~$\exists X_1 X_2 ((X_1 . \langle x \rangle) . X_2 =$~~

$$\exists X_1 X_2 \left(((X_1 . \langle x \rangle) . X_2 = X \right)$$ (3/3)

There exist sub sequences X_1, X_2 such that when concatenated with the subsequence formed from x, form X.

(c) "There exists a sequence which is the concatenation of all the distinct elements of the domain."

(0/4) true in all models of $\geq n$ elements, an ending infinite ones.

$$\exists x_1 \ldots \exists x_n \bigwedge_{i \neq j}^{n} x_i \neq x_j \wedge \exists X \left(X = (\langle x_1 \rangle . \langle x_2 \rangle) . x_3\rangle) \ldots \right))$$

(d) Compactness: if there exists an arbitrarily large but finite model that satisfies a formula then there exists a model with an infinite domain that satisfies the formula. But from part (c) we have a formula which is satisfied only by finite models, therefore sequence logic cannot satisfy the Compactness property.

NO (0/10)

5.

a) Posing the question "is a string a member of the theory of strings" is many-one-reducible to S-Valid.

~~$\{a,b\}^* \in Th\{a,b\}^*$ $f((\{a,b\}^*) \in S\text{-Valid}$~~

(0/20)

$\{a,b\}^* \in Th\{a,b\}^* \quad f\{a,b\}^* \in S\text{-Valid}.$

f is the function that constructs this formula, it is total and computable.

$\exists X\,(X = \langle \{, a,b\}^* \rangle)$ ← is this a valid formula? if it is then the $\{a,b\}^*$ is a member of the theory of strings.

(b) We know from class that $Th(\{a,b\}^*)$ is non-re. (Valid formulas valid in $< \{a,b\}, a, b, \circ >$) (Proof by reduction to T.M. computation word).

(5/5) Thus since non-re inherits up S-Valid is non-re.

Non-re implies non-axiomatizable implies Gödel Incompleteness

6.

(a) $(T)_I \neq \Lambda$ if $(X)_I \neq \Lambda$

$I_v(T) \neq \Lambda$ by defn

(0/10)

$I_v(X) \neq \Lambda$ by defn for all assignment statements $X := T$

Thus if there was $I_v(T) = \Lambda$ then since $X := T$ there would be an $I_v(X) = \Lambda$

Thus if $I_v(X)$ is Λ free so must $I_v(T)$

$\therefore$ $(T)_I \neq \Lambda$ for all sequence terms T.

(b)

different I's

$\phi = \{$ swps : which halt with without context E $I_v(X) = \Lambda$ for all I $\}$.

Consider those interpretations in which $(T)_I \neq \Lambda$ for all sequence terms T. It is impossible for $X := \Lambda$ since the only assignment statements are $X := T$. Thus I is an interpretation where swps cannot halt with $X = \Lambda$.

(01/15)

7.

Assignment Axiom

1) $R(X, Y, x) \quad (Y, x) \leftarrow (\varepsilon, a) \quad R(X, Y, x).$

2) $\text{True} \supset R(X, \varepsilon, a).$ — Trivial

3) Consequence

$\{\text{True}\}$

$(Y, x) \leftarrow (\varepsilon, a).$

$R(X, Y, x).$

4)

$R(X, \langle a \rangle . Y, x) \quad Y \leftarrow (\langle a \rangle . Y) \quad R(X, Y, x).$ — Assignment

4a) $R(X, \langle a \rangle . Y, x) \supset R(X, Y, x) \wedge X \doteq Y$

5)

$(R(X, Y, b) \wedge Y \neq X)$

$x \leftarrow b$

$(R(X, Y, x))$

6) $R(X, \langle a \rangle Y, x) \wedge Y \doteq X)$

$Y \leftarrow \langle a \rangle . Y$

$R(X, Y, x).$

7) $R(X, Y, b) \wedge Y \neq X) \supset R(X, Y, x) \wedge Y \doteq X.$

8) $R(X, \langle a \rangle . Y, x) \wedge Y \doteq X) \supset R(X, Y, x) \wedge Y \neq X.$

from
5,6,7,8. - conditional

9. $\{R(X,Y,x)\}$
$\quad$ if $Y \doteq x$ then $X := b$ else $Y := \langle a \rangle . Y$ fi.
$\{R(x,Y,x)\} \wedge Y \not\doteq x\}$
↑ from the conditional terminating (don't need it)).

from 4. and concatenation

10 $\{R(x,Y,x)\} \wedge Y \doteq x\}$.
$\quad Y \leftarrow \langle a \rangle . Y$;
$\quad$ if $Y \doteq x$ then $X := b$ else $Y : \langle a \rangle . Y$ fi.
$\{R(x,Y,x)\} \wedge Y \not\doteq x\}$.

from while rule. (note the condition on the bottom?
is ¬ of the one on the top.

11 $\{R(x,Y,x)\}$.
$\quad$ while $Y \doteq x$ do
$\quad\quad Y := \langle a \rangle . Y$
$\quad\quad$ if $Y \doteq x$ then $x :\doteq b$ else $Y := \langle a \rangle . Y$ fi
$\quad$ od.
$\{R(x,Y,x)\}$.

Finally

12 $$(R(x,Y,x)) \supset (\exists z. (x \doteq z.z) \equiv (x = a))$$

by concatenation (3) and (11) and Lemma (12)

∴ {true}.

$Y := \varepsilon \quad x := a$

while $Y \neq X$ do

$Y := \langle a \rangle . Y$;

if $Y \doteq X$ then $x := b$ else $Y := \langle a \rangle . Y$ fi.

od

$\{ \exists Z . (X \doteq Z.Z) \equiv x = a \}$.

Collect the Lemmas. – Need Loop invariant to prove Lemmas true.

$R(X, Y, b) \wedge Y \not\doteq X \supset R(X, Y, x) \wedge Y \doteq X$

$R(X, \langle a \rangle . Y, x) \wedge Y \doteq X \supset R(X, Y, x) \wedge Y \not\doteq X$.

$R(X, \langle a \rangle . Y, x) \supset R(X, Y, x) \wedge Y \doteq X$

$R(X, Y, x) \supset (\exists Z . (X \doteq Z.Z) \equiv x = a)$.

$\exists Z (Y \doteq Z.Z) \wedge x \equiv a$.

$\exists Z (Y \doteq Z.Z) \supset x = a$

aaa. Y

a.

aa.

aaa.

x = b.

Yes, but what is R?

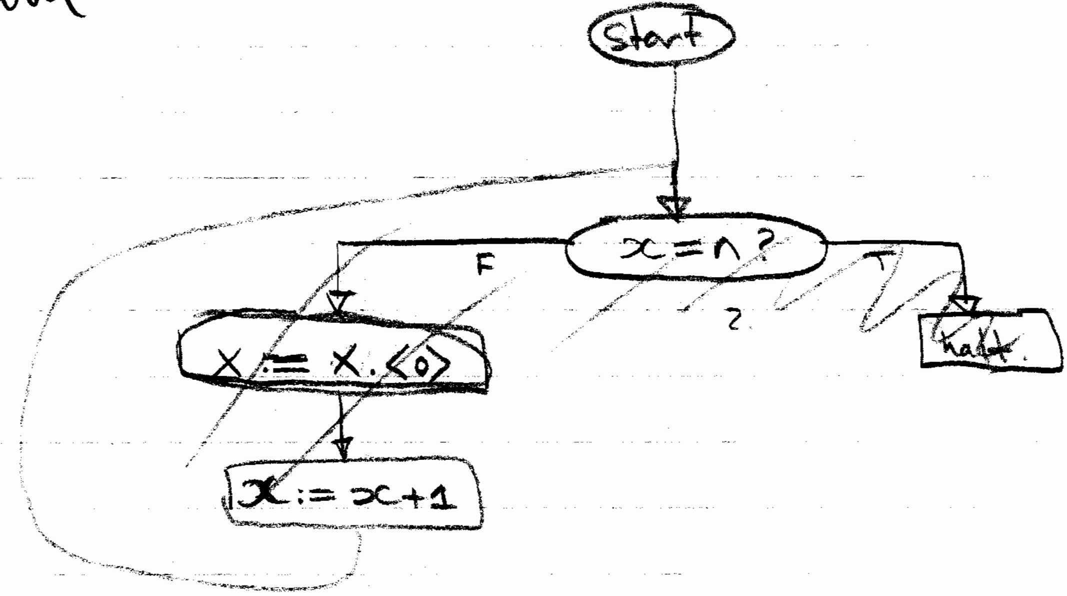
−18

8).

a) For a given while programme we can construct a machine that acts like W on an initial valuation I_0, we can then check the valuation at each stage

~~0/5~~ 02/5 am

b)



This can be expressed as while

while $X \neq n$ do.

$x := x+1$

$X := \langle 0 \rangle . X$

od.

While program contains only elements 0, 1, +, =.

Can't have $x \neq n$.

00/10

c)

I guess there must be some repeating structural element that we can customize such that for any function that enables you to enumerate a part of $\{n|n>1\}$ there is a corresponding piece of flowchart/while prog (they are equivalent) which can be made thus it is possible to have W halt on exactly those elements of $\{n|n>1\}$.

X

PRINT IN LARGE LETTERS Pugg (Family Name) Kenneth (First Name) William (Middle Name) 6-3 (Course) '88 (Year)

6044 (Subject Number) Computability Programming and Logic (Subject Name) Dec 17, 1987 (Date)

A. Meyer (Instructor's Name)

IF EXAMINATION FOR ADVANCED STANDING CHECK HERE ☐

IF A CONDITION EXAMINATION, FILL IN BELOW		IF A POSTPONED FINAL EXAMINATION, FILL IN BELOW	
(Year and term taken in Class)	(Instructor's Name)	(Year and term taken in Class)	(Instructor's Name)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Do not write below. Reserved for Examiner.

1. 00
2. 25
3. 00
4. (a) 3 (b) 2 (c) 0 (d) 0 total: 05
5. (a) 03 (b) 4 total: 07
6. 00
7. 00
8. (a) 4 (b) 10 (c) 00 total: 14
9.
10.
11.
12.
13.
14.
15.

51/200

6-72 30M S14

1. let $K_{f_1} = \{M \mid M$ on input x $\begin{cases} b \text{ if } x \in R \\ a \text{ if } x \notin R \end{cases}\}$

~~K_{f_2}~~ $= \{M \mid M$ on input x $\begin{cases} a \text{ if } x \in \bar{R} \\ b \text{ if } x \notin \bar{R} \end{cases}\}$

~~let $K_{f_3} = \{M \mid M$ runs K_{f_1} and K_{f_2} in parallel~~ ~~input x~~

Assume R seperates K_a, K_b

then K_{f_1} run on K_a outputs b

then K_{f_1} run of K_{f_1} outputs a

1. let $K_{f_1} = \{M \mid M$ on input x $\begin{cases} b \text{ if } x \in R \\ a \text{ if } x \notin R \end{cases}\}$

K_a is a set

K_{f_1} on input K_a? outputs a

K_{f_1} on input K_b outputs b

K_{f_1} on input K_f outputs a

00/20

let $K_{f_2} = \{M \mid M$ on input x $\begin{cases} b \text{ if } x \in \bar{R} \\ a \text{ if } x \notin \bar{R} \end{cases}$

K_{f_2} on input K_a out puts a

K_{f_2} on input K_b outputs b

K_{f_2} on input K_{f_2} outputs b

2. a.) because if the TM diverges there is a pair that can be selected at any point that will not cause the the top and bottom to become different, but there is no pair that will complete the pattern there for pairs may be added infinitely ✓

b.) [illegible]

Show $\overline{K_{0post}} \leq_m K_{IPCP}$

label the boxes $b_1, b_2, \ldots b_m$ as in standard construction and run as in the standard construction if the post machine does not halt this will correspond to an infinite solution to the post correspondance problem [illegible] the construction is altered in that the pair corresponding to the halt box is changed so that its second element is Λ therefore if the machine halts this will correspond to an insolvable IPCP ✓

3. Let $DG = \{ E \mid E$ is a finite degenerate set of semigroup eq'ns $\}$

4. a.) $X \doteq X \cdot X$ (3/3)

b.) $\exists X_1, X_2 .\ X \doteq X_1 \cdot \langle x \rangle \cdot X_2$ (2/3)

c.) $\exists X_1, X_2 .\ X_1 \not\doteq X_1 \cdot X_1 \wedge X_2 \not\doteq X_2 \cdot X_2 \wedge X_1 \cdot X_2 \doteq X_1$

(0/4)

d.) (0/10)

5. a.) map a to makseq(a), b to makseq(b) and • to •

(what about variables, etc ?)

(03/20) more explanation required.

b.) There is no complete deduction scheme for sequence logic since

(4/5) $Th(\{a,b\}^*) \leq_m$ to N

6. a.) assume $(T)_I = \Lambda$ for some sequence term T

then $X := T$ ✗

but then $I_V(X) = \Lambda$

but since I is empty-sequence-free

$I_V(X) \neq \Lambda$ ✱

b.) ✗

7. $\{True\}\ W\ \{\exists z.(x = z \cdot z) \equiv (x = 9)\}$

8. a) ~~if it halts then~~

(4/15)

Start running W starting in (Z_2, I_0) for 1 step. Then run W starting in (Z_3, I_0) for 1 step and then return to (Z_2, I_0) for 1 step. Continue adding (Z_n, I_0)'s in "parallel" and if one of them halts that is the spectrum of W

somewhat confused. It seems you are enumerating (not accepting) spectrum(W).

b.)

(10/10)

```
X := <0>; x := 1
while ¬(x = 0) do
    x := x + 1
    X := <0>·X
od
```

c.)

(0/15)

PRINT IN LARGE LETTERS Dovydaitis (Family Name) Vincent (First Name) (Middle Name) VI-3 (Course) 4 (Year)

6.044J (Subject Number) Computability, Prog & Logic (Subject Name) 12/17/87 (Date)

Meyer (Instructor's Name)

IF EXAMINATION FOR ADVANCED STANDING CHECK HERE ☐

IF A CONDITION EXAMINATION, FILL IN BELOW		IF A POSTPONED FINAL EXAMINATION, FILL IN BELOW	
(Year and term taken in Class)	(Instructor's Name)	(Year and term taken in Class)	(Instructor's Name)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Do not write below. Reserved for Examiner.

1. [redacted] 07
2. 02
3. 00
4. (a) 0 (b) 2 (c) 0 (d) 0 total: 02
5. (a) 03 (b) 4 total: 07
6. 12
7. 12
8. (a) 0 (b) 10 (c) 00 total: 10
9.
10.
11.
12.
13.
14.
15.

52/200

1) Assume R seperating K_a and K_b
$K_a \subseteq R$ and $K_b \subseteq R$

Assume R is a recursive set
Therefore there is a ~~A~~ totally computable function t s.t. t decides whether its input is in R

Now consider machine M computing

$$f(x) = \begin{cases} b & x \in R \\ a & x \notin R \end{cases}$$

07/20

Since R is decidable f(x) is totally computable. ✓

M executing f(M) returns "b" if $M \in K_a$ and "a" if $M \in K_b$ both of which are contradictions ✓

Therefore whether M is a member of K_a or K_b is undecidable
Therefore R is not recursive. why?

→ what if $M \notin K_a$, $M \notin K_b$?

2a) An infinite sequence of instruction executions ~~hoszan infinite is an infinite~~ ~~s~~ by Manna's proof is an infinite PCP

$$\alpha_{i_1} \ldots = \beta_{i_1} \ldots$$

-5

2b) IPCP is not r.e.

1) Halting problem for Post machines is r.e. (There is a totally computable function from Turing to Post and v.v. and K_0 is r.e.)

2) There is a totally computable func from Post Machine to Post system

$K_{pm} \leq_m K_{PCP} = \overline{IPCP}$ by 2a

$\overline{PCP}$ is not the same as IPCP.

$K_{pm} \leq_m \overline{IPCP}$

$\therefore$ $\overline{IPCP}$ is r.e.

$\therefore$ IPCP is not r.e.

(−18)

3 $\{E_i\} \vdash F$ is undecidable

4a) $\forall p \, [p(X) = \text{false}]$ (0/3)

b) $\exists T_1 \exists T_2 \, (T_1 \circ \langle X \rangle \circ T_2 \doteq X)$

↓ x, not X (2/3)

c) (0/4)

d) (0/10)

5a) $Th(\{a,b\}^*) \leq_m$ s-Valid

We can rewrite a formula in $Th(\{a,b\}^*)$ to s-Valid as follows:

? <u>string variable</u> $X \rightarrow$ seq variable X
? <u>element</u> $\sigma \rightarrow$ term t

$X \circ Y \rightarrow X \circ Y$

$X \circ \sigma \rightarrow X \circ mkseq(t)$

$X = Y \rightarrow X \doteq Y$

$\wedge, \vee, \forall, \exists$ etc. same

3/10

~~Note: If domain of St has more than two elements we~~

<u>Claim</u> that this rewrite is both validity preserving and totally computable

$\therefore Th(\{a,b\}^*) \leq_m$ s-Valid

In fact as this procedure is invertable (if domain of interpretation of formulas in s-Valid has more than two elements could code them)

$Th(\{a,b\}^*) \equiv_m$ s-Valid

5b) Notes on Pred Calc Thm 3:

First order theory of strings is not r.e.

(4|5) Since not r.e inherits ops "First order theory of sequence logic is not r.e."

6) a) Since $T \equiv_{def} T \circ T \mid mkseq(t) \mid X$

$mkseq(t) \neq \Lambda$

by def I Λ-free so $(X)_I \neq \Lambda$

if $T_1 \neq \Lambda$ and $T_2 \neq \Lambda$ then
$T_1 \circ T_2 \neq \Lambda$

$\therefore (T)_I \neq \Lambda$ ✓

b) Since from part a if we start Λ-free and do not allow the assignment $Y := \varepsilon$ which sets $(X)_I = \Lambda$

then $(T)_I \neq \Lambda$ for all terms. unclear

$\therefore (X)_I \neq \Lambda$ for all interpretations

-13

what is R?

-18

7) 1. true ⊃ $R(X, \varepsilon, a)$

2. {true}
$Y := \varepsilon$; $x := a$;
$\{R(X, Y, x)\}$ assign rule

3. $\{R(X, Y, x) \wedge X \neq Y\} \supset R(X, \langle a\rangle \circ Y, x)$
~~∨ ¬R(X, ⟨a⟩ ∘ ⟨a⟩ ∘ Y, b~~

4 $\{R(X, Y, x) \wedge X \neq Y\}$ ~~⊃ R(X, ⟨a⟩ Y, x~~
$Y = \langle a\rangle \circ Y$
$\{R(X, \langle a\rangle \circ Y, x)\}$ assign rule

5 $R(X, Y, x) \wedge X \doteq Y \} \supset R(X, \varepsilon Y, x)$
6. $R(X, Y, x) \wedge X \neq Y \supset R(X, \langle a\rangle \circ Y, x)$

7 & 8 assign rules for if ... then ... else

9: 5 & 6 $\{R(X, Y, x) \wedge (X \doteq Y)$ and $R(X, Y, x) \wedge (X \neq Y)\}$
if $X \doteq Y$ then $x = b$ else $Y = \langle a\rangle \circ Y$
$\{R(X, Y, x)\}$
if-then-else rule

10: $R(X, Y, x) \wedge X \neq Y \supset R(X, Y, x)$

11: from 4 & 9 & 10 :
$\{R(X, Y, x)\}$
while ... od $\{R(X, Y, x) \wedge X \doteq Y\}$

8a) to determine if $n \in$ spectrum(ω):

if ω halts started in $(Z_n, I_\emptyset)$

(0/15) $n \in$ spectrum(ω) — this is the definition of spectrum(ω). why is it r.e. ?

Therefore spectrum(ω) is r.e.

b) z = 1; X = <0>

while z ≠ 0 do

X = <0> ∘ X

z = z + 1

od

(10/10)

(note: in Z_1 $(1)_I = (0)_I$)

c) (00/15)

Course Information

Staff.

Lecturer:	Prof. Albert R. Meyer	NE43-315 x3-6024 `meyer@theory.lcs.mit.edu`
Teaching Assistant:	Mihir Bellare	NE43-334 x3-5866 `bellare@theory.lcs.mit.edu`
Secretary:	Arline Benford	NE43-316 x3-6025 `ah@theory.lcs.mit.edu`

Send netmail to the TA so we know your net address.

Lectures and office hours. Class meets Monday, Wednesday, and Friday from 3 to 4, in 34-302. There will be no regular sections, but tutorial/review sessions may be organized in response to requests. Office hours will be announced; you can also meet with the instructor or the TA 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. Course 6-3 students may use this course as a substitution in kind for the 6.045J/18.400J requirement.

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.

Textbooks. The required text is:

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

You may wish to consult the following supplemental texts:

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.

H. Enderton, *A Mathematical Introduction to Logic*, Academic Press, 1972.

Handouts and Notebook. We suggest that you 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 Arline's office (NE43-316). If you take the last copy of a handout, please inform Arline 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.

Grading. There will be problem sets, two evening quizzes, and 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 eight to ten problem sets. Homework will usually be assigned on a Friday and due the following Friday. 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.

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* use 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.

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.**

Problem Set 1

Reading assignment. For this assignment: Manna, sections 1-2 to 1-4.1.

Problem 1.

1(a). Design a Turing machine M over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ which does the following

- If the input is the string a then M moves to the right forever, without changing the contents of the tape.
- If the input is not the string a then M puts a b at the end of the input word and halts with its tape head on this b.

Present the program of this machine in the form of a graph as described in Manna.

1(b). Exhibit the behaviour of M, in the style of Manna's Example 1-8, on the two input words a and aa.

Problem 2. In class we described Turing machines as finite flowcharts built up of boxes (instructions) of the following kinds:

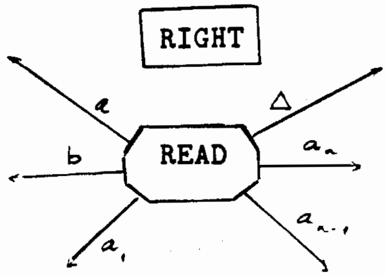
(START)

(HALT)

[PRINT σ] for all $\sigma \in \Sigma \cup V \cup \{\Delta\}$

[LEFT]

(over)



where $V = \{a_1, \ldots, a_n\}$. This is different from Manna's description of Turing machines as finite graphs with state transitions given by labelled arrows. The purpose of this exercise is to illustrate, by examples, the essential equivalence of these descriptions. Suppose F is a Turing machine flowchart. A *translation* of F to graph form is a graph description G of a Turing machine which has the following property: If the machines described by F and G are both started in their respective START states on a tape containing an arbitrary input word w, then either

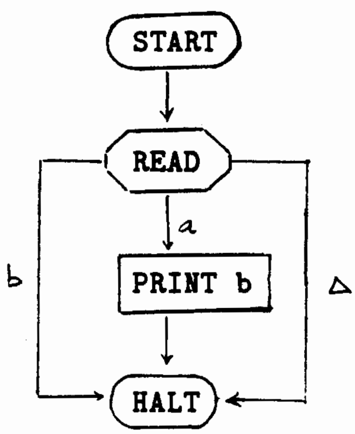
(i) both loop forever, or

(ii) both halt with identical tape contents and final head positions.

A translation of a graph description to a flowchart description is defined similarly.

2(a). Translate the graph of Figure 1-4 in Manna to an equivalent flowchart description.

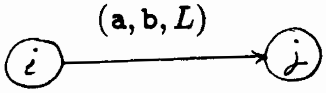
2(b). Translate to a graph description the following flowchart description



of a Turing machine over the alphabet $\{a, b\}$ with $V = \emptyset$.

Problem 3. We explained in class how to translate an arbitrary Turing machine state diagram to a Post machine state diagram, essentially by translating each arrow in the in the Turing machine diagram to a piece of a Post machine flowchart. The translation of a right shift

was described in class, and the corresponding piece of Post machine flowchart is reproduced in Figure 1. Produce a similar flowchart piece for a Post machine to translate a left move



of a Turing machine.

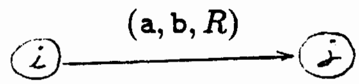
Problem 4. Briefly explain why r.e. sets are closed under union (i.e., if A and B are r.e. so is $A \cup B$).

Hint: Given Turing machines M_A and M_B accepting A and B respectively, describe a Turing machine M accepting $A \cup B$.

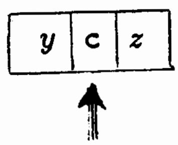
Try to explain the ideas of the construction at a high level without getting enmeshed in details of Turing machine code.

Figure 1: Translation of Turing machine right shift to Post machine

We reproduce the piece of Post machine flowchart that translates the Turing machine arrow



The Post machine uses two auxiliary symbols # and * not in the alphabet of the Turing machine. Recall that we are representing the Turing machine tape contents



where $y, z \in (\Sigma')^*$, $c \in \Sigma'$, and $\Sigma' = \Sigma \cup V \cup \{\Delta\}$, by the Post machine variable

$$x = cz\#y.$$

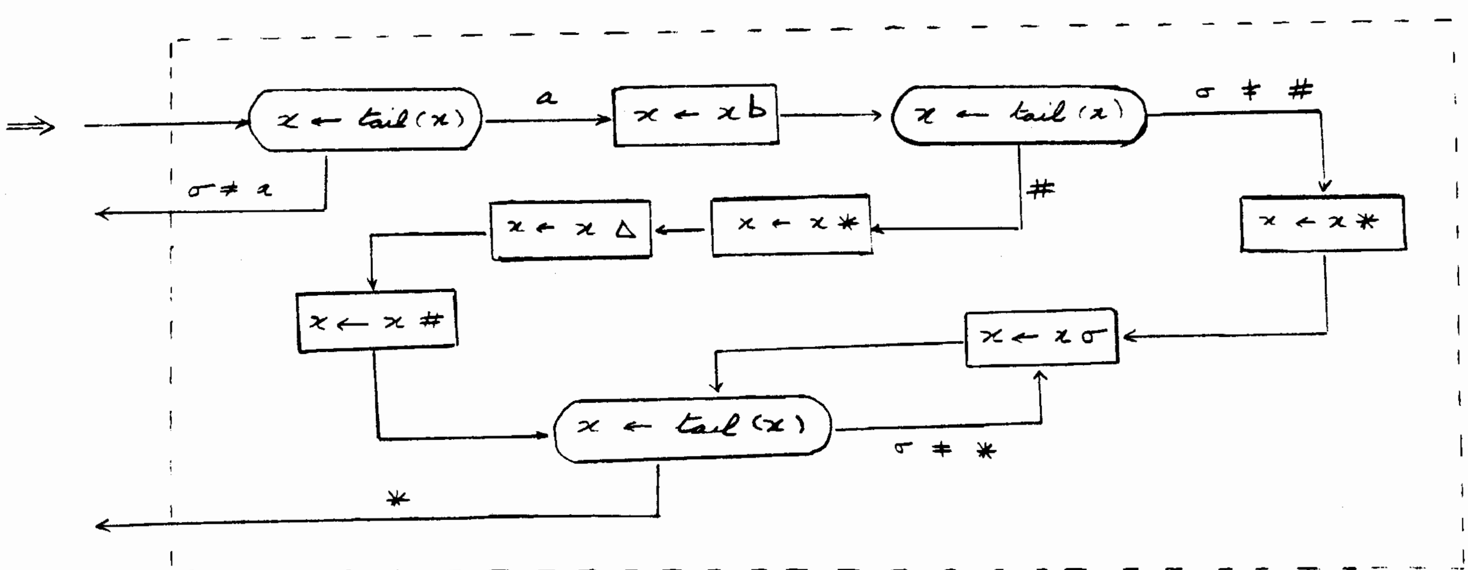
The Turing machine move we are considering should, in the case that $c = a$ and $z \neq \Lambda$, change x to

$$z\#yb.$$

In the special case that $c = a$ and $z = \Lambda$, x becomes

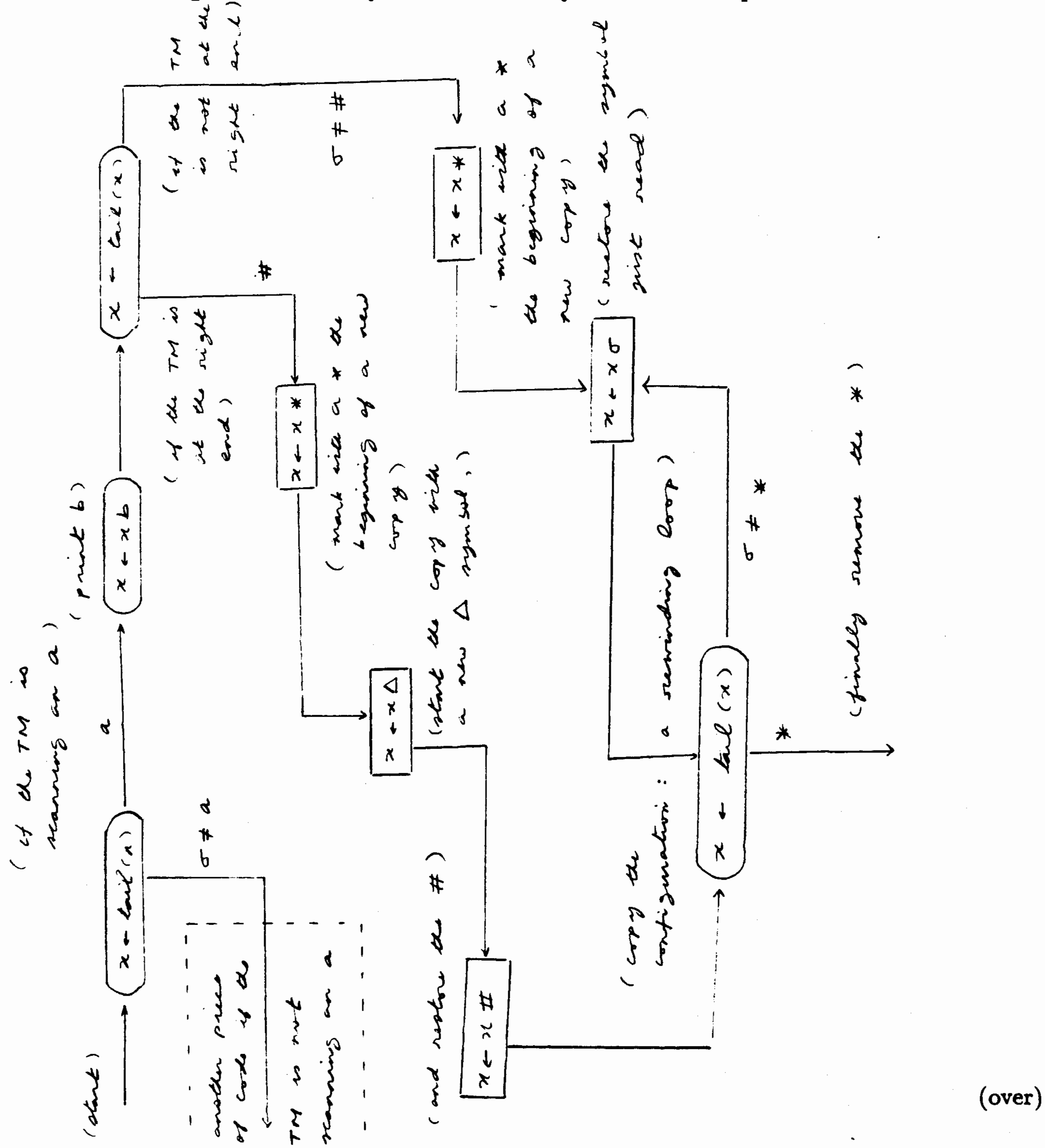
$$\Delta\#yb$$

Here is the Post machine flowchart piece:



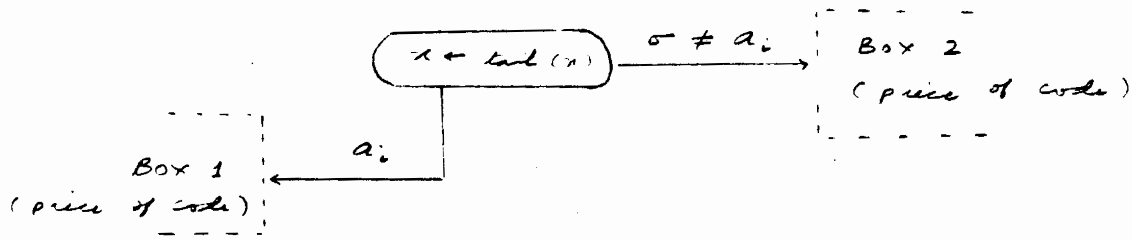
Problem Set 1 Errata

We provide here an annotated version of the Post machine code of problem 4, together with a more complete description of the notation and conventions used in this code. You are encouraged to annotate your own code in your solution to problem 4.

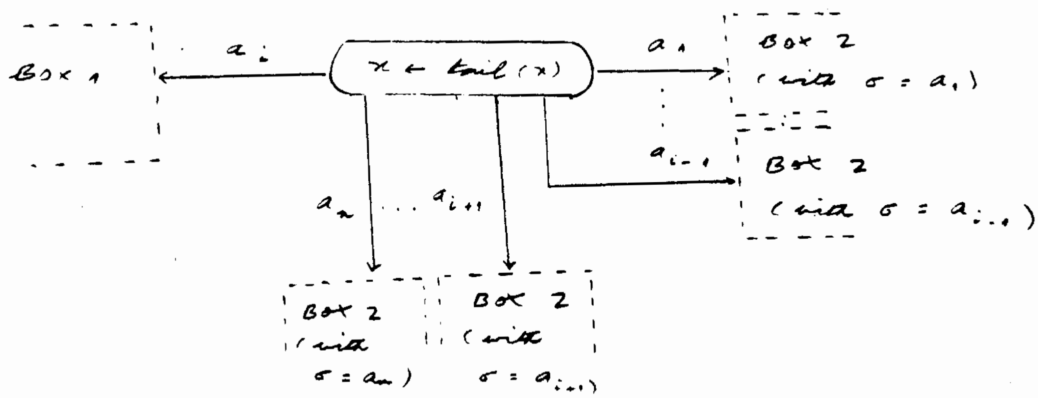


(over)

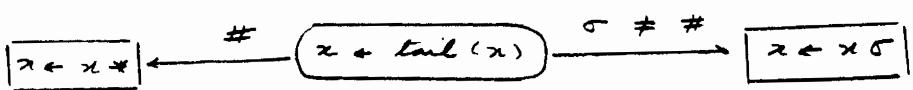
Say the Post machine alphabet is $\{a_1, \ldots, a_n\}$. A flowchart piece of the form



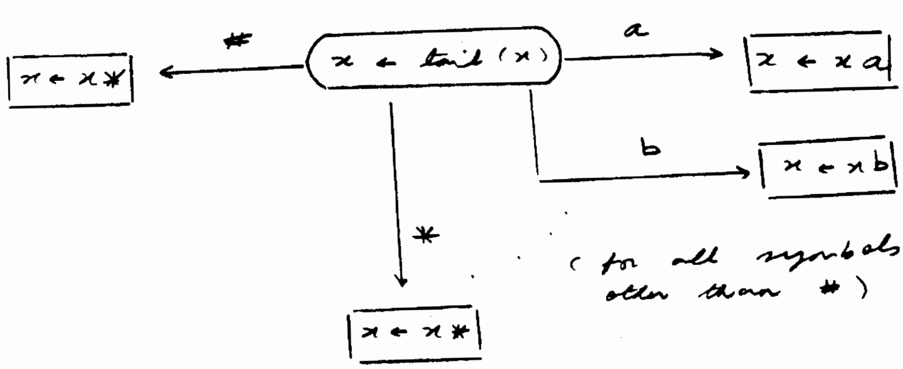
is a shorthand for the following more complete flowchart piece:



Here Box 2 is replicated for each arrow out of the $x \leftarrow$ tail(x) instruction which is labelled with a symbol different from a_i, and in the replication corresponding to the arrow labelled a_j, σ is replaced by a_j. So



just means



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 $f : \{\mathtt{a}, \mathtt{b}\}^* \to \Sigma^*$. (*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 $\{\mathtt{a}, \mathtt{b}\}$.)

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).

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 $(2.7)^{\ln \pi}$.)

Problem 4. Define

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

(a) Show that $K_2 \leq_m K_{tot}$, 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 $f : \{\mathtt{a},\mathtt{b}\}^* \to K_{tot}$ 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.

Notes on Computability Theory

The following notes outline the main definitions and results about computability theory developed in the course lectures which went 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, $\{a, b\}^*$, as are computable by Scheme programs, register machines, or Post machines. 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 $\{a, b\}$ 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 : Turing machines $\rightarrow \{a, b\}^*$ 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 $\{a, b\}^*$ 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, doesn't halt on any input.

We now define

$$K_1 = \{d(M) \mid M \text{ is a Turing machine that halts on input } d(M)\}\ .$$[1]

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 υ, the axiomatic definition of K_1 becomes

[1] This is the same set as the complement $\overline{L_1}$ of the set L_1 of section 1-3.1 of Manna, given our convention for interpreting strings not of the form $d(M)$ as the code of a never halting Turing machine.

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 $d(M)$ can be modified to obtain the code for simple variants of M. For example, from $d(M)$ and $y \in \{\mathtt{a}, \mathtt{b}\}^*$, one can easily construct a machine $M_{d(M),y}$ which replaces its input x by the word $pair(y, x)$ and then acts like M on the modified input. This is abstractly captured by thinking of the function $s(z, y) = d(M_{d(M),y})$ 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 $x, y, z \in \{\mathtt{a}, \mathtt{b}\}^*$

$$v(z, pair(y, x)) = v(s(z, y), x)$$

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 $x_1 * \ldots * x_n$ never halts.

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 $f_D : \Sigma^* \to \{\mathtt{a}, \mathtt{b}\}$ is total recursive, where

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

A language $R \subseteq \Sigma^*$ is *recursively enumerable* (r.e.) iff

$$R = \{x \in \Sigma^* \mid M_R \text{ halts on input } x\}$$

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. 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 Λ, **a**, **b**, **aa**, **ab**, **ba**, **bb**, **aaa**, **aab**, ...

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

Theorem 6. A language A is recursive iff it is finite or the range of a total increasing recursive function.

Theorem 7. Recursive sets are closed under union, intersection, and complementation.

Theorem 8. R.e. sets are closed under union and intersection.

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,

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

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

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. K_1 is a special case of K_0, so if K_0 were decidable, then K_1 would be too. ■

Claim 1. K_0 and K_1 are r.e.

Proof. Left to reader. ■

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. imply 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 flowchart 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^m_n-Theorem 2. This is the first result in these notes other than the recursive isomorphism Theorem 3 which depends on the S^m_n-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 $\leq_m$-hard for r.e. sets.

Proof. Suppose that $P(\emptyset)$ is not true. Since P is nontrivial, there exists a machine M_1 with $\text{domain}(M_1) = L_1 \neq \emptyset$ such that $P(L_1)$.

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 $R \leq_m K_P$ 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,

$$\begin{aligned} x \in R \quad & \text{iff} \quad M_R \text{ halts on } x \\ & \text{implies} \quad M_{f(x)} \text{ acts like } M_R \text{ on every input} \\ & \text{implies} \quad \text{domain}(M_{f(x)}) = L_1 \\ & \text{implies} \quad f(x) \in K_P \ , \end{aligned}$$

and also

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

Moreover, as in the proof for K_2, the function f is total recursive. Hence, $R \leq_m K_P$. ∎

10 Total Functions

Definition 11.

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

Lemma 4. K_{tot} is neither r.e. nor co-r.e.

Proof. In homework. ∎

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*, $\leq_T$, 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, f_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	$f_B(\Lambda)$	$f_B(a)$	$f_B(b)$	$f_B(aa)$	$f_B(ab)$	$f_B(ba)$	$f_B(bb)$	$f_B(aaa)$	...

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 $A \leq_{\mathrm{T}} B$ 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 13. 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.
- $A \leq_{\mathrm{T}} B$ and $B \leq_{\mathrm{T}} C$ imply $A \leq_{\mathrm{T}} C$.
- $A \leq_{\mathrm{m}} B$ implies that $A \leq_{\mathrm{T}} B$.
- It is not the case that $A \leq_{\mathrm{T}} B$ implies $A \leq_{\mathrm{m}} B$. (For example, $K_0 \leq_{\mathrm{T}} \overline{K}_0$, but it is false that $K_0 \leq_{\mathrm{m}} \overline{K}_0$.)
- $A \leq_{\mathrm{T}} \overline{A}$ for any set A.
- $A \leq_{\mathrm{T}} B$ iff $A \leq_{\mathrm{T}} \overline{B}$ iff $\overline{A} \leq_{\mathrm{T}} B$.

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. p

Theorem 14. (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

$$B <_T B' .$$

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

Likewise Theorem 4 relativizes to:

Theorem 16. $A \leq_T B$ iff A is both r.e. and co-r.e. in B.

Some further relativizations:

Theorem 17. 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.

Lemma 5. $B' \equiv_m \{M \mid M \text{ halts on blank string input with language input } B\}$. (That is, the relativized halting problem is $\equiv_m$ to the relativized blank-tape halting problem.)

Theorem 18. 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 16, A' is r.e. A' and $\overline{A'}$ is r.e. in A'. Therefore, by Theorem 15, $A' \leq_m A''$ so $\overline{A'} \leq_m \overline{A''}$, and also $\overline{A'} \leq_m A''$. ∎

Corollary 6. K_0' is neither r.e. nor co-r.e.

Define

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

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

$$\emptyset <_{\mathrm{T}} \emptyset' <_{\mathrm{T}} \emptyset^{(2)} <_{\mathrm{T}} \cdots$$

has strictly more difficult successive membership problems. This sequence, or more precisely the sequence of families $\{L \mid L \leq_{\mathrm{m}} \emptyset^{(n)}\}$ 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 sche¡mes turn out to be $\equiv_{\mathrm{m}} \emptyset^{(2)}$. 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....

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

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

(a) Show that $K_2 \leq_{\mathrm{m}} A$.

(b) Show that $\overline{K}_2 \leq_{\mathrm{m}} A$.

(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.)

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 $f : \{a, b\}^* \to \Sigma^*$. (*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 $\{a, b\}$.)

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

Suppose $L \subseteq \Sigma^*$ is empty or the range of a total recursive function $f : \{a, b\}^* \to \Sigma^*$. In either case it is the range of a partial recursive function $\psi : \{a, b\}^* \to \Sigma^*$, 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 $\psi : \{a, b\}^* \to \Sigma^*$. 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 $f : \{a, b\}^* \to \Sigma^*$ by

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

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: range(f) = range$(\psi) = L$.

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).

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 $x \in \{\mathtt{a}, \mathtt{b}\}^*$,

- 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 $x \in \{\mathtt{a}, \mathtt{b}\}^*$,

- 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 $\psi : \{\mathtt{a}, \mathtt{b}\}^* \to \{\mathtt{a}, \mathtt{b}\}^*$ 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 $\{\mathtt{a}, \mathtt{b}\}^*$. Then $\psi^{-1}(E) = E$ is not recursive. This provides the counterexample for (b).

For (c) let $D = \{\mathtt{a}, \mathtt{b}\}^*$ 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) $\{M \mid 10^9 < |d(M)|\}$.

(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 $A = \{M \mid M$ does not halt on x for all $x \in \{a, b\}^*\}$ 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 $x \in \{a, b\}^*$ of length $> 10^9$ which are well-formed Turing machine codes. In fact, given our convention of regarding any string in $\{a, b\}^*$ 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

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

(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 $\{a, b\}^*$ to $\{a, b\}^*$. (***Hint***: Let $f : \{a, b\}^* \to K_{\text{tot}}$ be onto and recursive. Define the "diagonal function" $\rho : \{a, b\}^* \to \{a, b\}$ by

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

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.

(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

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

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

(b) Assume $f : \{\mathtt{a}, \mathtt{b}\}^* \to$ [illegible]$_{t}$ is onto and recursive.

Claim 1. The function ρ is computable.

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

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

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

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.

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

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

(a) [10 points] Explain why Rice's theorem doesn't apply to *OddSquare*. More precisely, show that $OddSquare \neq K_P$ 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 $T : \{\mathtt{a},\mathtt{b}\}^* \to \mathbf{N}$ 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

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

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

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 $x \not\to_S^* y$.

(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 $T(w) = 2^{|w|}$.

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.

Glossary and notation

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

$\mathrm{N} = \{0, 1, 2, ...\}$ 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.

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

Problem Set 3

Reading assignment. For this assignment: Manna, sections 1-5.3,1-5.4.

Problem 1. Show that if A is a recursive set, then $A \leq_m B$ for essentially *any* set B. Identify the exceptional B's.

Problem 2. Show that the Post Correspondence Problem over the alphabet $\{a, b\}$ is $\equiv_m$ to the Post Correspondence Problem over arbitrary alphabets.

Problem 3. An *input-tape limited machine (ILM)* is a Turing machine variant in which a shiftright off the portion of the tape initially occupied by the input is interpreted in the same way as shiftleft off the tape, namely as "halt-and-reject". Thus, the entire computation on any input x occurs in the first $|x|$ tape squares.

3(a). Show that the halting problem for ILM's is decidable.

3(b). Show that the Emptiness Problem for ILM's is an $\leq_m$-complete co-r.e. set. (The Emptiness Problem for ILM's is $\{M \mid M \text{ is an ILM and } \text{domain}(M) = \emptyset\}$

Problem 4. The *Busy Beaver* function, $b : \mathbf{N} \to \mathbf{N}$, is defined as

$$b(n) = max\{m \geq 0 \mid \text{some Turing machine } M \text{ with } d(M) \leq n \text{ halts in exactly } m \text{ steps on input } \Lambda\} .$$

(By convention, $max\emptyset = 0$). A total function $f : \mathbf{N} \to \mathbf{N}$ *majorizes* b if $f(n)$ is greater than $b(n)$ for sufficiently large n. More precisely, f majorizes b if

$$\exists n_0 \forall n \geq n_0 \, (f(n) > b(n)) .$$

Show that the Busy Beaver function is not majorized by any total computable function $f : \mathbf{N} \to \mathbf{N}$.

Quiz 1 Solutions

Problem 1 [30 points]. Let

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

(a) [10 points] Explain why Rice's theorem doesn't apply to *OddSquare*. More precisely, show that $OddSquare \neq K_P$ 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 $M \in OddSquare$ and $M' \notin OddSquare$. 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 $\leq_m$-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 $K_2 \leq_m OddSquare$. Since K_2 is a $\leq_m$-complete r.e. set this will imply that *OddSquare* is $\leq_m$-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

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

So $K_2 \leq_m OddSquare$ via f.

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

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

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..

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

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

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

Proof. For any Turing machine M let M' be the machine whose description is: "on input x, if $x = \texttt{a}$ 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 a, and halts on input b iff M halts on Λ. So

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

and $\overline{K_2} \leq_m A$ 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

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

$\mathbf{N} = \{0, 1, 2, \ldots\}$ 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 $\leq_m$-complete for r.e. sets if and only if

(i) A is an r.e. set

(ii) $B \leq_m A$ for any r.e. set B.

Δ is the blank symbol for Turing machines.

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

Problem Set 4

Reading assignment. Manna Chapter 2: Introduction and 2-1.1 to 2-1.4.

Problem 1. Manna exercise 2-9 (a).

Problem 2. Manna exercise 2-10 (a),(b),(c),(d),(n).

Problem 3. Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. Recall that an *equation* over Σ is an expression of the form

$$x{=}y$$

where x and y are semigroup terms over Σ. Let

$$AX = \{F_1, \ldots, F_k, \ldots\}$$

be a set of equations.

Definition 1. Let E be an equation. We define the notion that E is *provable from axioms* AX, written $\vdash_{AX} E$, inductively as follows:

(1) $\vdash_{AX} E$ if $E \in AX$.

(2) $\vdash_{AX} x{=}x$ for all $x \in \Sigma^+$.

(3) If $\vdash_{AX} x_1{=}x_2$ and $\vdash_{AX} x_1{=}x_3$ then $\vdash_{AX} x_2{=}x_3$.
A rule such as the above is usually displayed in the format

$$\frac{\vdash_{AX} x_1{=}x_2 \ , \ \vdash_{AX} x_1{=}x_3}{\vdash_{AX} x_2{=}x_3} .$$

The assertions above the horizontal line are usually called the *antecedents* of the rule, and the assertion below the line is called the *consequent*.

(4)

$$\frac{\vdash_{AX} x_1{=}x_2 \ , \ \vdash_{AX} y_1{=}y_2}{\vdash_{AX} x_1y_1{=}x_2y_2} .$$

Prove the completeness of this system. That is, show that for any equation E,

$$\vdash_{AX} E \quad \text{iff} \quad AX \models E .$$

Hint: By the completeness theorem proved in class $AX \models E$ iff $AX \vdash E$ (i.e. $AX \models E$ iff the left and right hand sides of E rewrite to each other under the Thue system whose rules are AX). Show that the Thue system rewriting process can be simulated in the above system so that $AX \vdash E$ implies $\vdash_{AX} E$. Conversely prove by induction on the definition of $\vdash_{AX}$ that $\vdash_{AX} E$ implies $AX \vdash E$.

Problem Set 5

Reading assignment. Manna sections 2-1, 2-2 and the notes on logic (to appear).

Problem 1. Let Σ be a finite alphabet. For each $\sigma \in \Sigma$ let f_σ be a unary function symbol. For any word $x = \sigma_1 \ldots \sigma_n \in \Sigma^*$ where $\sigma_1, \ldots \sigma_n \in \Sigma$, and for any individual variable v, let $f_x(v)$ abbreviate the term

$$f_{\sigma_1}(f_{\sigma_2}(\cdots f_{\sigma_n}(v)\ldots)) .$$

(By convention $f_\Lambda(v)$ abbreviates v.) For any monoid equation $x = y$ let $\ulcorner x = y \urcorner$ be the first order formula

$$\forall v(f_x(v) = f_y(v)) .$$

In this problem we will use the symbol $\models_m$ to denote satisfaction over monoids, and the symbol $\models_p$ to denote satisfaction in the predicate calculus.

(a) Let $\mathcal{I} = ((M, *), I)$ be a monoid interpretation of Σ. We define a first order logical interpretation $\mathcal{I}' = (M, \mathcal{I}'_c, \mathcal{I}'_v)$ over the signature $\{f_\sigma \mid \sigma \in \Sigma\}$ as follows: $\mathcal{I}'_c(f_\sigma) : M \to M$ is the function defined by

$$\mathcal{I}'_c(f_\sigma)(m) = I(\sigma) * m$$

for all $m \in M$. Prove that for any monoid equation $x = y$ over Σ,

$$\mathcal{I} \models_m x = y \quad \text{iff} \quad \mathcal{I}' \models_p \ulcorner x = y \urcorner .$$

(b) Prove that for any first order logical interpretation $\mathcal{I} = (D, \mathcal{I}_c, \mathcal{I}_v)$ over the signature $\{f_\sigma \mid \sigma \in \Sigma\}$ we can define an associated monoid interpretation $\mathcal{I}^0$ over Σ with the property that for any monoid equation $x = y$,

$$\mathcal{I}^0 \models_m x = y \quad \text{iff} \quad \mathcal{I} \models_p \ulcorner x = y \urcorner .$$

(c) Use parts **(a)** and **(b)** to show that for any set of monoid equations AX, and for any monoid equation E,

$$AX \models_m E \quad \text{iff} \quad \left(\bigwedge_{F \in AX} \ulcorner F \urcorner\right) \supset \ulcorner E \urcorner \text{ is valid.}$$

(d) Conclude that the validity problem for the predicate calculus is undecidable.

Problem 2. Let $\mathcal{S}$ be a first order predicate calculus signature consisting of n unary predicate constants,

$$\mathcal{S} = \{P_1, \ldots, P_n\}\ .$$

Let $\mathcal{I} = (D, \mathcal{I}_c, \mathcal{I}_v)$ be an interpretation of $\mathcal{S}$. We say that two elements d and d' of D *have the same truth pattern* iff

$$\mathcal{I}_c(P_i)(d) \ \text{ iff } \ \mathcal{I}_c(P_i)(d') \ \text{ for all } i = 1, \ldots n\ .$$

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 $\overline{\mathcal{I}} = (\overline{D}, \overline{\mathcal{I}}_c, \overline{\mathcal{I}}_v)$ 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

$$\overline{\mathcal{I}}_c(P_i)([d]) \ \text{ iff } \ \mathcal{I}_c(P_i)(d)\ .$$

  ($\overline{\mathcal{I}}_c(P_i)$ is well defined because if $[d] = [d']$ then $\mathcal{I}_c(P_i)(d)$ iff $\mathcal{I}_c(P_i)(d')$).

- For each individual variable x we let

$$\overline{\mathcal{I}}_v(x) = [\mathcal{I}_v(x)]\ .$$

(a) Prove by induction on the definition of a first order wff A without equality over the signature $\mathcal{S}$ that

$$\mathcal{I} \models A \ \text{ iff } \ \overline{\mathcal{I}} \models A\ .$$

(b) Use part (a) to show that the validity problem for $\mathcal{S}$ is decidable. 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.

Problem Set 6

Quiz: Quiz 2 is scheduled for Monday 09 November, 7:00-9:00 PM, in 34-302.

Problem 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 $\mathcal{S}$.

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 $\mathcal{I}$.

(a) Let $\mathcal{S} = \{\mathbf{c}_1, \mathbf{c}_2\}$ where $\mathbf{c}_1$ and $\mathbf{c}_2$ are constant symbols. Describe a pair of interpretations $\mathcal{I}$ and $\mathcal{I}'$ such that $\mathcal{I}'$ is a homomorphic image of $\mathcal{I}$ and

$$\mathcal{I} \models \neg(\mathbf{c}_1 = \mathbf{c}_2) \quad \text{but} \quad \mathcal{I}' \not\models \neg(\mathbf{c}_1 = \mathbf{c}_2) .$$

(b) Suppose $\mathcal{I}'$ is a homomorphic image of $\mathcal{I}$. Prove that for any wff A over the signature $\mathcal{S}$ such that A does not contain the equality symbol,

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

(*Hint*: Use induction on the definition of A. Begin by using induction on the definition of terms to show that $h((t)_{\mathcal{I}}) = (t)_{\mathcal{I}'}$ for any term t over $\mathcal{S}$).

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 $(\mathcal{M}, \mathcal{I}_v) \cong (\mathcal{M}', \mathcal{I}'_v)$.

Problem 2. Let $\mathcal{S} = \{\cdot, \boldsymbol{EqLen}\}$ be a signature consisting of one binary function constant $\cdot$ and one binary predicate constant $\boldsymbol{EqLen}$. Let Σ be an alpahabet, and let $\mathcal{M}_{\Sigma^*}$ be the standard interpretation of $\mathcal{S}$. That is, $\mathcal{M}_{\Sigma^*} = (\Sigma^*, \mathcal{I}_c)$ where $\mathcal{I}_c$ is defined by

- $\mathcal{I}_c(\cdot)$ is $\cdot$, the usual concatenation operator on strings.
- $\mathcal{I}_c(\boldsymbol{EqLen})$ is the equal length predicate $EqLen$:

$$EqLen(x, y) = \mathtt{true} \quad \text{iff} \quad |x| = |y|$$

for $x, y \in \Sigma^*$.

Write down a second-order predicate calculus formula F_{Σ^*} over the signature $\mathcal{S}$ with the property that for any model $\mathcal{M}$ over the signature $\mathcal{S}$,

$$\mathcal{M} \models F_{\Sigma^*} \quad \text{iff} \quad \mathcal{M} \cong \mathcal{M}_{\Sigma^*} \,.$$

Briefly explain why your formula works. (*Hint*: This is similar to the construction of the formula F_{Arith} in class).

Problem 3. Let F be the conjunction of the following first-order formulas over the signature $\mathcal{S} = \{\mathbf{o}, \boldsymbol{suc}, +\}$:

- $\forall x \forall y[(\boldsymbol{suc}(x) = \boldsymbol{suc}(y)) \supset (x = y)]$ $\quad$ ($\boldsymbol{suc}$ is 1-1)
- $\forall x(\boldsymbol{suc}(x) \neq \mathbf{o})$ $\quad$ ($\mathbf{o}$ is not a successor)
- $\forall x \forall y[(x + \mathbf{o} = x) \wedge (x + \boldsymbol{suc}(y) = \boldsymbol{suc}(x + y))]$ $\quad$ (inductive definition of $+$)

Give an example of a model of F such that the interpretation of $+$ is *not* a commutative operation.

(*Optional Problem*: Same thing when the formula

$$\forall x[(x \neq \mathbf{o}) \supset \exists y(\boldsymbol{suc}(y) = x)]$$

(every non-zero element has a successor) is conjuncted into F).

Problem Set 4 Solutions

10/10

Boris Goldowsky
6.044J/18.423J Problem Set #4 (23 Oct 87)
Problem 1. (Manna 2-9a)

Show $\exists x \forall y [(p(x,y) \land \neg p(y,x)) \supset (p(x,x) \equiv p(y,y))]$ to be invalid

A counterexample:

Domain = D = Integers

$$p(x,y) = \begin{cases} T \text{ if } x > y \\ F \text{ if } x < y \\ T \text{ if } x = y \text{ and } x \text{ is even} \\ \text{~~F if x=y and x is odd~~} \end{cases}$$

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

$p(x,y) = p(x, x-1) = T$

$p(y,x) = p(x-1, x) = F$

$(p(x,y) \land \neg p(y,x)) = (T \land \neg F) = T$ 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 (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.

Boris Goldowsky
6.044J / 18.423J Problem Set 4, 23 Oct 87
Problem 3

Show that given any equation E, $\vdash_{AX} E$ iff $AX \models E$
we know that $AX \models E$ iff $AX \vdash E$
$AX \models E$ iff left + right sides of E rewrite to each other under Thue system whose rules are $AX = \{F_1 \ldots F_k\}$

1. If $AX \vdash E$ then $\vdash_{AX} E$.

Assume $AX \vdash E$. Then there are some rules (α_i, β_i) of the Thue system such that $E_L \xrightarrow{(\alpha_1,\beta_1)} E_1 \xrightarrow{(\alpha_2,\beta_2)} E_2 \cdots E_{k-1} \xrightarrow{(\alpha_k,\beta_k)} E_R$. Take each step in turn; each (E_L = left side of E, E_R = right side of E; $E_i \in \Sigma^*$)
can be simulated by the rules in the definition of $\vdash_{AX}$.
So if $E_i \to E_{i+1}$ in one step by some rule (α, β), then
$E_i = A\alpha B$ and $E_{i+1} = A\beta B$, $A, B \in \Sigma^*$.
But since the rules of the Thue system are AX, if (α, β) is a rule then $\alpha = \beta$ is an equation in AX. Thus by part 1 of the definition, $\vdash_{AX} \alpha = \beta$. Also by part 2, $\vdash_{AX} A = A$ and $\vdash_{AX} B = B$, unless $A = \Lambda$ or $B = \Lambda$.

→ If $A = \Lambda$, $E_i = \alpha B$ and $E_{i+1} = \beta B$. By part 4, $\dfrac{\vdash_{AX} \alpha = \beta,\ \vdash_{AX} B = B}{\vdash_{AX} \alpha B = \beta B}$

→ If $B = \Lambda$, $E_i = A\alpha$ and $E_{i+1} = A\beta$. By part 4, $\dfrac{\vdash_{AX} A = A,\ \vdash_{AX} \alpha = \beta}{\vdash_{AX} A\alpha = A\beta}$

→ If A and B both are Λ, $E_i = \alpha$ and $E_{i+1} = \beta$, so $\vdash_{AX} \alpha = \beta$ by part 1.
→ If neither one is Λ, then $\dfrac{\vdash_{AX} A = A,\ \vdash_{AX} \alpha = \beta}{\vdash_{AX} A\alpha = A\beta}$, then $\dfrac{\vdash_{AX} A\alpha = A\beta,\ \vdash_{AX} B = B}{\vdash_{AX} A\alpha B = A\beta B}$.

Thus $\vdash_{AX} A\alpha B = A\beta B$ for any A and B and any rule (α, β).
By applying this ~~simulation~~ technique you can simulate any number of Thue replacement steps, and thus show $\vdash_{AX} E$ if $AX \vdash E$.

2. If $\vdash_{AX} E$ then $AX \vdash E$.

If $\vdash_{AX} E$ then it is so by at least one of the four parts of the definition of $\vdash_{AX}$. I will show that anything produced by these 4 rules satisfies $AX \vdash E$, that is that the Thue system with rules AX can rewrite the two sides of E to each other.

(1) $\vdash_{AX} E$ if $E \in AX$. Clearly the two sides of a rule rewrite to each other

(2) $\vdash_{AX} X = X$. X rewrites to itself trivially.

(3) If $\vdash_{AX} x_1 = x_2$ and $\vdash_{AX} x_1 = x_3$ then $\vdash_{AX} x_2 = x_3$.

If x_1 rewrites to x_2 and also to x_3, then to get x_2 from x_3 (or vice versa) do the substitutions to make it into x_1 and then the subs to make x_1 into the desired result.

(4) If $\vdash_{AX} x_1 = x_2$ and $\vdash_{AX} y_1 = y_2$ then $\vdash_{AX} x_1 y_1 = x_2 y_2$.

Starting with $x_1 y_1$, the substitutions that make x_1 into x_2 will change it into $x_2 y_1$, then the subs that make y_1 into y_2 make this into $x_2 y_2$.

QED

Thue Systems and Semigroups

We provide here a summary of the notation, definitions, and theorems in the logic of Thue systems and semigroups.

Definition 1. A *semigroup* is a pair $(S, *)$ where $S \neq \emptyset$ is a set whose members are called the *elements* of the semigroup, and $*$ is an associative binary operation on S (that is, $* : S \rightarrow S$ is such that $s_1 * (s_2 * s_3) = (s_1 * s_2) * s_3$ for all $s_1, s_2, s_3 \in S$).

Definition 2. An element e of a semigroup $(S, *)$ is an *identity element* iff $e * s = s * e = s$ for all $s \in S$.

Definition 3. A *monoid* is a semigroup which has an identity element.

Lemma 1. There is exactly one identity element in a monoid.

Notation. Let Σ be an alphabet. Then $\Sigma^+ = \Sigma^* - \{\Lambda\}$ denotes the set of non-empty strings over Σ.

Example 1. $(\{\mathtt{a},\mathtt{b}\}^+, \cdot)$ is a semigroup ($\cdot$ denotes the operation of concatenation of strings), and $(\{\mathtt{a},\mathtt{b}\}^*, \cdot)$ is a monoid with Λ as its identity element.

Example 2. $(\{\mathbf{true},\mathbf{false}\}, \wedge)$ and $(\{\mathbf{true},\mathbf{false}\}, \equiv)$ are monoids with **true** as identity element, and $(\{\mathbf{true},\mathbf{false}\}, \oplus)$ is a monoid with **false** as identity element (the operation $\oplus$ is called exclusive-or and is defined by $x \oplus x = \mathbf{false}$ and $x \oplus \neg x = \mathbf{true}$ for all $x \in \{\mathbf{true},\mathbf{false}\}$).

Definition 4. A *semigroup term* over an alphabet Σ is a word in Σ^+. A *monoid term* over Σ is a word in Σ^*.

Definition 5. Let Σ be an alphabet. A *semigroup interpretation over signature* Σ consists of a pair $\mathcal{I} = ((S, *), I)$ where $(S, *)$ is a semigroup and $I : \Sigma \rightarrow S$. We extend I to a map from Σ^+ to S (calling the extension I by an abuse of notation) by induction on the length of semigroup terms as follows:

$$I(\sigma x) = I(\sigma) * I(x) \qquad (x \in \Sigma^+, \sigma \in \Sigma)\,.$$

$I(x)$ is called the *meaning* of the word x. By a further abuse of notation the mapping I is sometimes itself called the interpretation, and we write $\mathcal{I}(x)$ for $I(x)$.

Monoid interpretations are defined similarly. In the case of a monoid interpretaion $((M, *), I)$ the extension of $I : \Sigma \rightarrow M$ is a map from Σ^* to M defined as before on Σ^+ and in addition mapping the emptry string Λ to the identity element of $(M, *)$.

Lemma 2. $\mathcal{I}(xy) = \mathcal{I}(x) * \mathcal{I}(y)$ for a semigroup or monoid interpretation $\mathcal{I}$ over Σ and strings x, y over Σ.

Proof Sketch: Use induction on the length of x for each fixed y. ■

Example 3. $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, $S = (\{\mathtt{true}, \mathtt{false}\}, \oplus)$. Let

$$\mathcal{I}(\mathtt{a}) = \mathtt{true}\ ,\ \mathcal{I}(\mathtt{b}) = \mathtt{true}\ .$$

Then $\mathcal{I}(\mathtt{abaab}) = \mathtt{true}$. In general, the meaning of a word is $\mathtt{true}$ iff it is of odd length.

If x, y are semigroup terms over Σ we call "$x = y$" a ***semigroup equation*** over Σ. A semigroup equation is a syntactic entity consisting of two elements of Σ^+ seperated by the symbol "$=$". Monoid equations are defined analogously.

Given a semigroup interpretation $\mathcal{I}$ we can talk about the meaning of a semigroup equation under $\mathcal{I}$.

Definition 6. $\mathcal{I}$ ***satisfies*** $x = y$, written $\mathcal{I} \models x = y$, iff $\mathcal{I}(x) = \mathcal{I}(y)$.

Note that the first two occurences of "=" in Definition 6 indicate the formal symbol for equality while the last occurence denotes mathematical equality.

Definition 7. A set of equations $\{E_i\}$ ***logically implies*** an equation F, written $\{E_i\} \models F$, iff for any $\mathcal{I}$ such that $\mathcal{I} \models E_i$ for all i, it is also the case that $\mathcal{I} \models F$.

Definition 8. A set of equations $\{E_i\}$ ***proves*** an equation $x = y$, written $\{E_i\} \vdash x = y$, iff x rewrites to y under the Thue system whose rules are $\{E_i\}$.

Example 4. Let E_1, E_2 and E_3 be the equations $\mathtt{aa} = \mathtt{a}$, $\mathtt{bb} = \mathtt{b}$, and $\mathtt{ab} = \mathtt{ba}$ respectively. Then

$$\{E_1, E_2, E_3\} \vdash \mathtt{aaaaba} = \mathtt{abb}\ ,$$

since $\mathtt{aaaaba} \overset{*}{\longleftrightarrow}_P \mathtt{abb}$ where P is the Thue system $\{(\mathtt{aa}, \mathtt{a}), (\mathtt{bb}, \mathtt{b}), (\mathtt{ab}, \mathtt{ba})\}$.

Theorem 1. (Completeness theorem) $\{E_i\} \vdash F$ iff $\{E_i\} \models F$.

Proof Sketch: To prove that $\{E_i\} \vdash F$ implies $\{E_i\} \models F$ use induction on the length of the rewriting. For the converse, suppose that $\{E_i\} \not\vdash F$ and derive a contradiction by constructing a "term model" $\mathcal{I}$ with the property that $\mathcal{I} \models \{E_i\}$ for all i but $\mathcal{I} \not\models F$. ■

Corollary 1. The problem of determining whether $\{E_i\} \models F$ is undecidable.

Problem Set 3 Solutions

Problem 1. Show that if A is a recursive set, then $A \leq_m B$ for essentially *any* set B. Identify the exceptional B's.

Given a recursive set A and any set B.
When is $A \leq_m B$?
Define a reduction function as follows:

$$f(x) = \begin{cases} \alpha & \text{iff } x \in A \\ \beta & \text{iff } x \notin A \end{cases}$$

where $\alpha \in B$ and $\beta \notin B$.

- f is computable, since A is recursive
- α and β exist unless $B = \Sigma^*$ or $B = \emptyset$.
- f exists if α and β do, even if α and β are difficult to actually find.
- If $x \in A$ then $f(x) = \alpha \in B$.
- If $x \notin A$ then $f(x) = \beta \notin B$.
- Thus $x \in A$ iff $f(x) \in B$ and f is the required function that converts a problem of A'ness to one of B'ness. A can be reduced to any set B EXCEPT Σ^* and $\emptyset$.

Problem 2. Show that the Post Correspondence Problem over the alphabet $\{a, b\}$ is $\equiv_m$ to the Post Correspondence Problem over arbitrary alphabets.

Part 1. Show that a Post Correspondence Problem over $\Sigma_0 = \{a, b\}$, (P_0), is $\leq_m$ a PCP over Σ_1, an arbitrary alphabet (P_1).

This is possible iff Σ_1 has at least 2 symbols. Then pick one symbol to stand for "a" and one to stand for "b," and construct P_1 by replacing all the a's and b's in P_0 by these new symbols. Any other symbols in Σ_1 are superfluous; ignore them.

Part 2. Show that $P_1 \leq_m P_0$.

Replace each symbol in Σ_1 with a sequence of a's and b's. Pick some order for the symbols in Σ_1 and rewrite the (α, β) pairs with these substitutions:

1st symbol:	a
2nd "	ba
3rd "	bba
4th "	bbba
... etc.	

Since there is no way to duplicate one of these strings with any combination of the others, the new system P_0 will have the same solutions as P_1.

Since $P_0 \leq_m P_1$ and $P_1 \leq P_0$, $P_0 \equiv_m P_1$, as required.

Problem 3. An *input-tape limited machine (ILM)* is a Turing machine variant in which a shiftright off the portion of the tape initially occupied by the input is interpreted in the same way as shiftleft off the tape, namely as "halt-and-reject". Thus, the entire computation on any input x occurs in the first $|x|$ tape squares.

A notion central to this problem is that of a *configuration* of a Turing machine computation. A configuration C of a computation of M on x consists of the state of the finite control of M (or box of M's flowchart), the non-blank portion of M's tape, and the position of the head. We view a computation of M on x as a sequence of configurations.

The configuration at any point completely determines the future behaviour of M. In particular, if in a computation of M on input x a certain configuration C is repeated, then M will not halt on halt on input x.

3(a). Show that the halting problem for ILMs is decidable.

The halting problem for ILMs is the set

$$H = \{(M, x) \mid M \text{ is an ILM and } M \text{ halts on input } x\}\,.$$

The computation of an ILM M on input x is confined to the first $|x|$ tape squares. So the computation has only a finite number of possible configurations. Thus in a non-halting computation of M on input x some configuration C must be repeated. This is the basis for the following description of a decision procedure for H:

"On input (M, x), run M on input x, and keep track of the succesive configurationss of the computation. If at any point M moves beyond the $|x|$-th tape sqaure then halt and reject (M was not an ILM). If a configuration is repeated, halt and reject (M will not halt). Else if M halts, halt and accept."

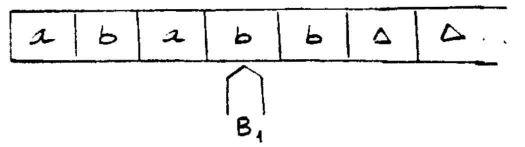
3(b). Show that the Emptiness Problem for ILMs is an $\leq_m$-complete co-r.e. set. (The Emptiness Problem for ILMs is $E = \{M \mid M \text{ is an ILM and } \text{domain}(M) = \emptyset\}$).

We first observe that E is co-r.e. The complement of E is

$$\overline{E} = \{M \mid M \text{ is not an ILM or } \text{domain}(M) \neq \emptyset\}\,.$$

and the machine with the following description accepts $\overline{E}$: "On input M, run M on all input strings $\Lambda, a, b, aa, ab, \ldots$ in parallel. If M uses more than $|x|$ tape squares in a computation on input x then accept (M is not an ILM). If M halts on input x then accept ($\text{domain}(M) \neq \emptyset$)."

We show that E is co-r.e. hard by reducing the co-r.e. complete set $\overline{K_2}$ to E. We first need some conventions about representing configurations. We represent configurations of M on x as strings over the alphabet $\Sigma_M \cup \{\Delta\}$. The string vBw ($v \in (\Sigma_M \cup \{\Delta\})^*, w \in (\Sigma_M \cup \{\Delta\})^+$, B a state of M) represents the configuration in which the tape of M holds the string vw and M is in state B scanning the first symbol of w. As an example, the configuration corresponding to



is written **aba**B_1**bb**. Blanks to the right of the last non-blank symbol are ignored in the representation of a configuration unless the head is scanning them.

Given a Turing machine M let M' be the machine with the following description: "Regard the input as a sequence $C_1, \ldots, C_n$ of configurations of a computation of M (if it is not of this form, diverge). Check that C_{i+1} follows from C_i by one move of M, for all $i < n$. Then if C_n is a configuration of a halting state of M and C_1 is a starting configuration of M on input Λ, halt. Else diverge."

Note that the computation of M' on input x can be carried out in the first $|x|$ tape squares. So M' is an ILM. Then observe that $\overline{K_2} \leq_m E$ via the reduction $f(M) = M'$.

Problem 4. The *Busy Beaver* function, $b : \mathbf{N} \to \mathbf{N}$, is defined as

$$b(n) = max\{m \geq 0 \mid \text{some Turing machine } M \text{ with } d(M) \leq n \text{ halts in exactly } m \text{ steps on input } \Lambda\} .$$

(By convention, $max\emptyset = 0$). A total function $f : \mathbf{N} \to \mathbf{N}$ *majorizes* b if $f(n)$ is greater than $b(n)$ for sufficiently large n. More precisely, f majorizes b if

$$\exists n_0 \forall n \geq n_0 \,(f(n) > b(n)) .$$

Show that the Busy Beaver function is not majorized by any total computable function $f : \mathbf{N} \to \mathbf{N}$.

Suppose $f : \mathbf{N} \to \mathbf{N}$ is a total computable function that majorizes b,

$$\exists n_0 \forall n \geq n_0 \,(f(n) > b(n)) .$$

Define $g : \mathbf{N} \to \mathbf{N}$ by

$$g(m) = \begin{cases} f(m) & \text{if } n_0 \leq m \\ f(n_0) & \text{otherwise.} \end{cases}$$

Observe that g is total computable and

$$\forall n \,(g(n) > b(n)) .$$

We now derive a contradiction by giving a description of a machine M which uses g to decide K_2. This implies that g, and hence f, could not exist. M works as follows:

"On input M', compute $t = g(d(M'))$. Run M' on input Λ for t steps. Accept if M' halts within these t steps and reject otherwise."

The Predicate Calculus I

1 Syntax

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

- Constants: "(", ")" (parentheses), "," (comma), `true`, `false`.
- Connectives: $\neg$ (not), $\supset$ (implies), $\wedge$ (and), $\vee$ (or).
- The equality operator: $=$.
- Quantifiers: $\forall$ (universal quantifier), $\exists$ (existential quantifier).
- Variables:

  (1) n-ary function variables F_i^n $(i \geq 1, n \geq 0)$; F_i^0 is called an *individual variable* and is also denoted by x_i).

  (2) n-ary predicate variables P_i^n $(i \geq 1, n \geq 0)$; P_i^0 is called a *propositional variable*.

The non-logical symbols are

- n-ary function constants f_i^n $(i \geq 1, n \geq 0)$; f_i^0 is called an *individual constant* and is also denoted by c_i).
- n-ary predicate constants p_i^n $(i \geq 1, n \geq 0)$; p_i^0 is called a *propositional constant*.

Definition 1. A *signature* is a set of non-logical symbols.

Terms and formulas over a given signature are defined by induction. Let $\mathcal{S}$ be a signature.

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 $\mathcal{S}$ $(n \geq 1)$ 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 $\mathcal{S}$ $(n \geq 1)$ 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 expressions $f_i^n(t_1, \ldots, t_n)$ and $F_i^n(t_1, \ldots, t_n)$ of Definition 2 are the ~~non~~-logical symbols "(" , ")" and "," listed above.

Definition 3. An *atomic formula* over $\mathcal{S}$ is 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. A *formula* (wff) over $\mathcal{S}$ is defined as follows:

(1) An atomic formula over $\mathcal{S}$ is a formula over $\mathcal{S}$.
(2) If A and B are formulas 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 formula over $\mathcal{S}$ then $(\forall x_i A)$ and $(\exists x_i A)$ are formulas over $\mathcal{S}$.

We emphasize that terms and formulas over $\mathcal{S}$ are *syntactic* entities construced out of the symbols in $\mathcal{S}$ and the logical symbols.

Parentheses will be dropped in writing wffs 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. See Manna for details of this nature.

We omit here the technical definitions of free and bound variables in a formula. See Manna for this. We can now define closed terms and formulas:

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

We write $[A]_{x_1,\ldots,x_n}^{t_1,\ldots,t_n}$ for the formula 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 will not give here a complete formal definition of substitution; the above will suffice for our purpose.

2 Semantics

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

(called the domain of $\mathcal{M}$)

Definition 6. A *model* of $\mathcal{S}$ is a pair $\mathcal{M} = (D, I_c)$ where D is a non-empty set 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 each individual constant $c_i \in \mathcal{S}$.
(2) $I_c(p_i^0)$ is either the value `true` or the value `false` for each propositional constant $p_i^0 \in \mathcal{S}$.
(3) $I_c(f_i^n) : D^n \to D$ is a n-ary function over D for each function constant $f_i^n \in \mathcal{S}$ $(n \geq 1)$.
(4) $I_c(p_i^n) \subseteq D^n$ is a n-ary predicate over D for each predicate constant $p_i^n \in \mathcal{S}$ $(n \geq 1)$.

Definition 7. A *valuation function* 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) $I_c(x_i)$ is an element of D for each individual variable x_i.
(2) $I_c(P_i^0)$ is either the value `true` or the value `false` for each propositional variable P_i^0.
(3) $I_c(F_i^n) : D^n \rightarrow D$ is a n-ary function over D for each function variable F_i^n.
(4) $I_c(P_i^n) \subseteq D^n$ is a n-ary predicate over D for each predicate variable 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 function 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 $(t)_\mathcal{I}$ of D defined as follows:

(1) $(x_i)_\mathcal{I} = I_v(x_i)$ for each individual variable x_i.
(2) $(c_i)_\mathcal{I} = I_c(c_i)$ for each individual constant $c_i \in \mathcal{S}$.
(3) If $t_1, \ldots, t_n$ are terms over $\mathcal{S}$ ($n \geq 1$) and f_i^n is a function variable then

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

(4) If $t_1, \ldots, t_n$ are terms over $\mathcal{S}$ ($n \geq 1$) and $F_i^n \in \mathcal{S}$ is a function constant then

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

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 $\mathcal{S}$. $\mathcal{I}$ *satisfies* A, written $\mathcal{I} \models A$, is defined inductively as follows:

(1) For each propositional variable P_i^0,

$$\mathcal{I} \models P_i^0 \quad \text{iff} \quad I_v(P_i^0) = \texttt{true} \ .$$

(2) For each propositional constant $p_i^0 \in \mathcal{S}$,

$$\mathcal{I} \models p_i^0 iff I_c(p_i^0) = \texttt{true} \ .$$

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

$$\mathcal{I} \models t_1 = t_2 \quad \text{iff} \quad (t_1)_\mathcal{I} = (t_2)_\mathcal{I} \ .$$

(Note that the first $=$ here is the syntactic symbol for equality while the second $=$ is the equality amongst elements of D).

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

$$\mathcal{I} \models P_i^n(t_1, \dots, t_n) \quad \text{iff} \quad I_v(P_i^n)((t_i)_\mathcal{I}, \dots, (t_n)_\mathcal{I}) .$$

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

$$\mathcal{I} \models p_i^n(t_1, \dots, t_n) \quad \text{iff} \quad I_c(p_i^n)((t_i)_\mathcal{I}, \dots, (t_n)_\mathcal{I}) .$$

We need some preliminary notation before proceeding to define the satisfaction of formulas. 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 $S \to T$ 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 formulas) Let A be a formula over $\mathcal{S}$ and let $\mathcal{I} = ((D, I_c), I_v)$ be an interpretaion of $\mathcal{S}$. $\mathcal{I}$ *satisfies* A, written $\mathcal{I} \models A$, is defined inductively as follows:

(1) $\mathcal{I} \models A$ for an atomic formula A iff $\mathcal{I} \models A$ according to Definition 10.

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

- $\mathcal{I} \models \neg A$ iff it is not the case that $\mathcal{I} \models A$.
- $\mathcal{I} \models A \supset B$ iff whenever $\mathcal{I} \models A$ it is also the case that $\mathcal{I} \models B$.
- $\mathcal{I} \models A \wedge B$ iff $\mathcal{I} \models A$ and $\mathcal{I} \models B$.
- $\mathcal{I} \models A \vee B$ iff $\mathcal{I} \models A$ or $\mathcal{I} \models B$.

(3) If x_i is an individual variable and A is a formula over $\mathcal{S}$ then $\mathcal{I} \models \forall x_i A$ iff for all $d \in D$ it is the case that $\mathcal{I}[x_i \mapsto d] \models A$.

(4) If x_i is an individual variable and A is a formula over $\mathcal{S}$ then $\mathcal{I} \models \exists x_i A$ iff there is a $d \in D$ such that $\mathcal{I}[x_i \mapsto d] \models A$.

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

Definition 12. Let A be a formula 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 valuation functions I_v for $\mathcal{M}$ it is the case that $(\mathcal{M}, I_v) \models A$.

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

Proof Sketch: Work through the definitions. ■

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

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

3 Subsets of the Predicate Calculus

The predicate calculus as defined above is usually called the ***second-order predicate calculus***. The terms and formulas 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 ($i \geq 1$). The first-order terms and formulas over a signauture $\mathcal{S}$ are thus those second-order terms and formulas which contain no predicate variables 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 formula A over a signature $\mathcal{S}$ is ***valid*** iff for all interpretations $\mathcal{I}$ of $\mathcal{S}$ it is the case that $\mathcal{I} \models A$.

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

Definition 14. A formula A over a signature $\mathcal{S}$ is ***satisfiable*** iff there is an interpretation $\mathcal{I}$ of $\mathcal{S}$ such that $\mathcal{I} \models A$.

Definition 15. A formula A over a signature $\mathcal{S}$ is ***unsatisfiable*** iff it is not satisfiable.

The following remark will be of importance when we desribe 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. The valid formulas of the first-order predicate calculus are recursively enumerable.

The proof of this theorem will require a fairly substantial development which will be outlined in later sections.

Theorem 2. The valid formulas of the first-order predicate calculus are not recursive.

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

Theorem 3. The first order theory of strings is not recursively enumerable.

Details on the proof and definitions for this theorem are postponed.

6 Special Classes of Formulas

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

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

7 Relations Between Formulas

Let A and B be formulas (not necessarily over the same signature).

Definition 17. A and B are ***equisatisfiable*** iff

A is satisfiable iff B 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 formulas which have the property of yielding a formula 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 2. There is an effective procedure to transform a given formula A into an equivalent formula B which is in Prenex normal form.

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

Lemma 3. There is an effective procedure to transform a given first-rder formula A into an equisatisfiable first-order formula B which is a universal wff.

The technique used to reduce a formula 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

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

Definition 19. A *ground term* of A is a term over the signature $\mathcal{S}$.

Definition 20. A *ground instance* of A is a formula obtained by replacing each free occurence of x_i in B by a ground term of A. That is, a ground instance of A is a formula 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 subformula of the formulas $G_1, \ldots, G_n$ as a propositional variable and assign it the value `true` or `false`. The formula $G_1 \wedge \ldots \wedge G_n$ now has a `true` or `false` value determined by the assignments to the atomic subformulas 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 subformulas 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 formula is unsatisfiable iff there is a finite conjunction of ground instances of the formula which is propositionally unsatisfiable.

Theorem 4 provides a semi-decision procedure for the validity problem of the first order predicate calculus. Lemmas [redacted] 2 and 3 and theorem 4 provide the proof of theorem 1.

Problem Set 5 Solutions

Problem 1. Let Σ be a finite alphabet. For each $\sigma \in \Sigma$ let f_σ be a unary function symbol. For any word $x = \sigma_1 \ldots \sigma_n \in \Sigma^*$ where $\sigma_1, \ldots \sigma_n \in \Sigma$, and for any individual variable v, let $f_x(v)$ abbreviate the term

$$f_{\sigma_1}(f_{\sigma_2}(\ldots f_{\sigma_n}(v)\ldots)) .$$

(By convention $f_\Lambda(v)$ abbreviates v.) For any monoid equation $x = y$ let $\ulcorner x = y \urcorner$ be the first order formula

$$\forall v(f_x(v) = f_y(v)) .$$

In this problem we will use the symbol $\models_m$ to denote satisfaction over monoids, and the symbol $\models_p$ to denote satisfaction in the predicate calculus.

(a) Let $\mathcal{I} = ((M, *), I)$ be a monoid interpretation of Σ. We define a first order logical interpretation $\mathcal{I}' = (M, I'_c, I'_v)$ over the signature $\{f_\sigma \mid \sigma \in \Sigma\}$ as follows: $I'_c(f_\sigma) : M \to M$ is the function defined by

$$I'_c(f_\sigma)(m) = I(\sigma) * m$$

for all $m \in M$. Prove that for any monoid equation $x = y$ over Σ,

$$\mathcal{I} \models_m x = y \quad \text{iff} \quad \mathcal{I}' \models_p \ulcorner x = y \urcorner .$$

Claim 1. $(f_x(v))_{\mathcal{I}'[v \mapsto m]} = I(x) * m$ for any $x \in \Sigma^+$.

Proof: By induction on the length of x. ■

Suppose $\mathcal{I} \models_m x = y$. By definition of $\models_m$ this implies that $I(x) = I(y)$. Using Claim 1 we thus have

$$(f_x(v))_{\mathcal{I}'[v \mapsto m]} = I(x) * m = I(y) * m = (f_y(v))_{\mathcal{I}'[v \mapsto m]}$$

for all $m \in M$. Hence $\mathcal{I}' \models_p \ulcorner x = y \urcorner$.

Conversely suppose $\mathcal{I}' \models_p \ulcorner x = y \urcorner$. So $(f_x(v))_{\mathcal{I}'[v \mapsto m]} = (f_y(v))_{\mathcal{I}'[v \mapsto m]}$ for all $m \in M$. By claim 1 this implies that $I(x) * m = I(y) * m$ for all $m \in M$. In particular, setting $m = e$ (the identity element of M) we get $I(x) = I(y)$. Thus $\mathcal{I} \models_m x = y$.

(b) Prove that for any first order logical interpretation $\mathcal{I} = (D, I_c, I_v)$ over the signature $\{f_\sigma \mid \sigma \in \Sigma\}$ we can define an associated monoid interpretation $\mathcal{I}^0$ over Σ with the property that for any monoid equation $x = y$,

$$\mathcal{I}^0 \models_m x = y \quad \text{iff} \quad \mathcal{I} \models_p \ulcorner x = y \urcorner .$$

The monoid interpretation $\mathcal{I}^0 = ((M, *), I)$ associated with $\mathcal{I} = (D, I_c, I_v)$ is defined as follows. Let

$$M = \{g : D \to D\}$$

be the set of all functions mapping D to D. Let $*$ be the operation of function composition,

$$g_1 * g_2 = g_1 \circ g_2$$

for all $g_1, g_2 \in M$. It is easy to see that $(M, *)$ is a monoid with identity element the identity function on D. Define $I : \Sigma \to M$ by

$$I(\sigma) = I_c(f_\sigma)$$

for all $\sigma \in Sigma$. We omit the easy check that $\mathcal{I}^0$ has the necessary properties.

(c) Use parts **(a)** and **(b)** to show that for any set of monoid equations AX, and for any monoid equation E,

$$AX \models_m E \quad \text{iff} \quad \left(\bigwedge_{F \in AX} \ulcorner F \urcorner\right) \supset \ulcorner E \urcorner \ \text{ is valid.}$$

Suppose that $AX \models_m E$. We need to show that for any predicate calculus interpretation $\mathcal{I}$ it is the case that

$$\mathcal{I} \models_p \left(\bigwedge_{F \in AX} \ulcorner F \urcorner\right) \supset \ulcorner E \urcorner .$$

Supose $\mathcal{I} \models_p \ulcorner F \urcorner$ for all $F \in AX$. We wish to show that $\mathcal{I} \models_p \ulcorner E \urcorner$. By the result of **(a)** it is the case that $\mathcal{I}' \models_m F$ for all $F \in AX$. By assumption $AX \models_m E$ so $\mathcal{I}' \models_m E$. The result of part **(a)** then implies that $\mathcal{I} \models \ulcorner E \urcorner$.

The converse implication is the same argument using part **(b)** instead of **(a)**.

(d) Conclude that the validity problem for the predicate calculus is undecidable.

The result of part **(c)** implies that the validity problem of the predicate calculus is at least as hard as the question of whether $AX \models_m E$, and we know from our work on monoid interprations that the latter is undecidable.

Problem 2. 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 2. $\overline{\mathcal{I}[x \mapsto d]} = \overline{\mathcal{I}}[x \mapsto [d]]$ 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 2. 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 $\{\texttt{true}, \texttt{false}\}^n$ 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 $(v_1, \ldots, v_n) \in \{\texttt{true}, \texttt{false}\}^n$,

$$I_c^D(P_i)((v_1, \ldots, v_n)) \quad \text{iff} \quad v_i = \texttt{true}\,.$$

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, $|\{\mathcal{M}^D \mid D \subseteq \{\texttt{true}, \texttt{false}\}^n\}| = 2^{n2^n}$) 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 $D \subseteq \{\texttt{true}, \texttt{false}\}^n$.

Quiz 2

Instructions. Do all 5 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 [10 points]. Show that

$$\{\mathtt{a} = \mathtt{bc}, \mathtt{bd} = \mathtt{db}, \mathtt{cd} = \mathtt{dc}\} \models_s \mathtt{a}^3\mathtt{d}^5 = \mathtt{d}^5\mathtt{a}^3$$

where $\models_s$ denotes logical implication over semigroups.

Problem 2 [10 points]. Explain why the following Post Correspondence Problem has no solution: (This is problem 1-20(a) from Manna's text)

$$\{(\mathtt{ba}, \mathtt{bab}), (\mathtt{abb}, \mathtt{bb}), (\mathtt{bab}, \mathtt{abb})\} .$$

Problem 3 [20 points]. Describe a countermodel to show that the following formula is not valid:

$$\{\forall x \forall y \forall z\, [p(x,y) \land p(y,z) \supset p(x,z)] \land \forall x \forall y\, [p(x,y) \lor p(y,x)]\} \supset \exists x \forall y\, p(x,y) \quad .$$

Problem 4 [20 points]. Let $S = \{p_1, p_2\}$ be a signature consisting of two unary predicate constants. Write down a satisfiable first-order formula, A, *without equality* over the signature S such that the domain of any model of A has ~~exactly~~ at least 4 elements.

Problem 5 [40 points]. 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

$$\forall x_1 \ldots \forall x_n \exists y_1 \ldots \exists y_m\, B \, .$$

(a) [10 points] Show that the validity problem for $\forall\exists$ wffs is many-one reducible to the unsatisfiability problem for $\exists\forall$ wffs.

(b) [15 points] Let A be the wff

$$\exists x_1 \exists x_2 \forall y_1 \forall y_2\, [p(x_1, y_2, y_1) \land \neg p(y_2, x_2, y_1)]$$

Exhibit all the ground instances of the Skolemized form of A. (Note: In the special case of Skolemizing an $\exists x$ preceded by *no* universal quantifiers, one introduces zero-ary function constants; that is, individual constants c_x).

(c) [15 points] Use the ground instances of part **(b)** to conclude that $\neg A$ is valid.

Quiz 2 Solutions

Problem 1 [10 points]. Show that

$$\{\mathtt{a} = \mathtt{bc}, \mathtt{bd} = \mathtt{db}, \mathtt{cd} = \mathtt{dc}\} \models_s \mathtt{a}^3\mathtt{d}^5 = \mathtt{d}^5\mathtt{a}^3$$

where $\models_s$ denotes logical implication over semigroups.

By the completeness theorem it suffices to show that

$$\{\mathtt{a} = \mathtt{bc}, \mathtt{bd} = \mathtt{db}, \mathtt{cd} = \mathtt{dc}\} \vdash_s \mathtt{a}^3\mathtt{d}^5 = \mathtt{d}^5\mathtt{a}^3.$$

By the definition of $\vdash_s$ this means we must check that $\mathtt{a}^3\mathtt{d}^5$ rewrites to $\mathtt{d}^5\mathtt{a}^3$ under the Thue system whose rules are

$$\{(\mathtt{a}, \mathtt{bc}), (\mathtt{bd}, \mathtt{db}), (\mathtt{cd}, \mathtt{dc})\}\ .$$

This rewriting is easily verified and we omit the details.

Problem 2 [10 points]. 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 $(\mathtt{ba}, \mathtt{bab})$ because the other pairs are mismatched in the first charecter. At this point there is one more $\mathtt{b}$ at the bottom than at the top. Since all three pairs have the property that the number of $\mathtt{b}$s in the second half of the pair is $\geq$ the number of $\mathtt{b}$s in the first half, further use of any of the pairs will not reduce the diference in the number of $\mathtt{b}$s between bottom and top. So no solution is possible.

Problem 3 [20 points]. Describe a countermodel to show that the following formula is not valid:

$$\{\forall x \forall y \forall z\, [p(x,y) \land p(y,z) \supset p(x,z)] \land \forall x \forall y\, [p(x,y) \lor p(y,x)]\} \supset \exists x \forall y\, p(x,y)\ .$$

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 4 [20 points]. 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

$$\begin{aligned}\exists x_1 \exists x_2 \exists x_3 \exists x_4 [\ & (p_1(x_1) \wedge p_2(x_1)) \\ & \wedge (\neg p_1(x_2) \wedge p_2(x_2)) \\ & \wedge (p_1(x_3) \wedge \neg p_2(x_3)) \\ & \wedge (\neg p_1(x_4) \wedge \neg p_2(x_4))]\end{aligned}$$

does the job.

Problem 5 [40 points]. 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

$$\forall x_1 \ldots \forall x_n \exists y_1 \ldots \exists y_m\, B\,.$$

(a) [10 points] 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.

(b) [15 points] Let A be the wff

$$\exists x_1 \exists x_2 \forall y_1 \forall y_2\, [p(x_1, y_2, y_1) \wedge \neg p(y_2, x_2, y_1)]$$

Exhibit all the ground instances of the Skolemized form of A. (Note: In the special case of Skolemizing an $\exists x$ preceded by *no* universal quantifiers, one introduces zero-ary function constants; that is, individual constants c_x).

The Skolemized form of A is

$$\forall y_1 \forall y_2\, [p(c_{x_1}, y_2, y_1) \wedge \neg p(y_2, c_{x_2}, y_1)]\,.$$

The ground terms are c_{x_1} and c_{x_2}. Substituting these in all possible ways for y_1 and y_2 yields four ground instances:

$$\begin{array}{ll}(1) & p(c_{x_1}, c_{x_1}, c_{x_1}) \wedge \neg p(c_{x_1}, c_{x_2}, c_{x_1}) \\ (2) & p(c_{x_1}, c_{x_2}, c_{x_1}) \wedge \neg p(c_{x_2}, c_{x_2}, c_{x_1}) \\ (3) & p(c_{x_1}, c_{x_1}, c_{x_2}) \wedge \neg p(c_{x_1}, c_{x_2}, c_{x_2}) \\ (4) & p(c_{x_1}, c_{x_2}, c_{x_2}) \wedge \neg p(c_{x_2}, c_{x_2}, c_{x_2})\end{array}$$

(c) [15 points] Use the ground instances of part **(b)** to conclude that $\neg A$ is valid.

$\neg A$ is valid iff A is unsatisfiable. By Herbrand's theorem, A is unsatisfiable iff the conjunction of the four ground instances of part **(b)** is propositionally unsatisfiable. This is indeed the case because the conjunction of the ground instances (1) and (2) includes the propositionally unsatisfiable pair

$$\neg p(c_{x_1}, c_{x_2}, c_{x_1}) \wedge p(c_{x_1}, c_{x_2}, c_{x_1}) .$$

Problem Set 7

Problem 1. Prove the validity problem for $\forall\exists$-wffs *without function symbols* is decidable. (*Hint*: Generalize the last problem on Quiz 2.)

Problem 2.

(a) Let A range over second-order wffs with signature $\{+, *, =\}$. Show that it is decidable whether $\langle Z_3, +, * \rangle \models A$, where Z_3 denotes the integers modulo 3.

(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.

Problem 3. Let T be (an infinite) set of first-order wffs. Let $Mod(T)$ be the set of models of (every formula in) T,

$$Mod(T) = \{\mathcal{M} \mid \mathcal{M} \models A \text{ for all } A \in T\}.$$

Show that $Mod(T) \neq \{\mathcal{M} \mid \text{ the domain of } \mathcal{M} \text{ is finite}\}$.

6.044J/18.423J: Computability, Programming, and Logic — Handout 22
Massachusetts Institute of Technology — 20 November 1987 (due 30 November)

Problem Set 8

Problem 1. Let F be the flowchart schema of Figure 1.

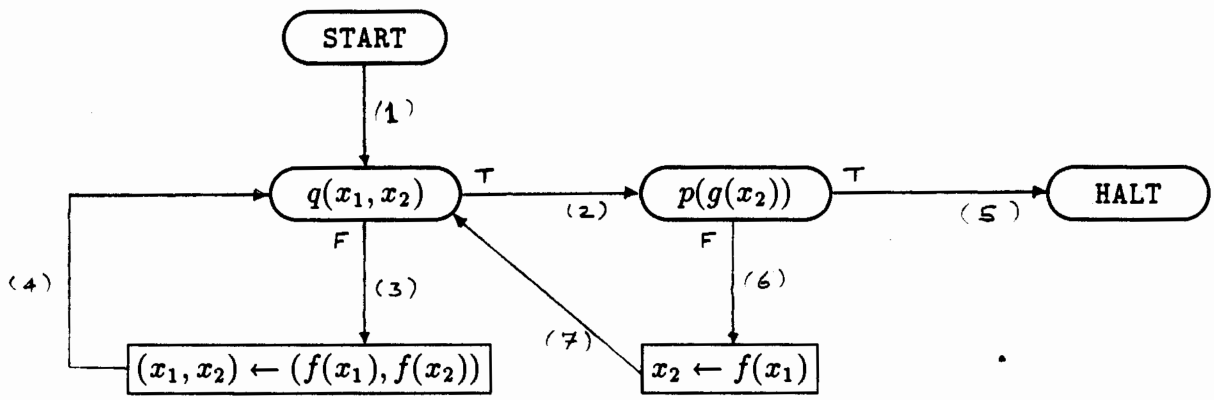


Figure 1: The flowchart schema F

The edges of the flowchart F are numbered so that we can refer to a path in F by the sequence of numbers of the edges that constitute it. The following table illustrates the construction of the formulas $S_{F,path}$. Fill in the blank entries of the table.

Path (p)	x_1	x_2	$S_{F,p}$
Λ	x_1	x_2	true
1	x_1	x_2	true
1, 3	x_1	x_2	$\neg q(x_1, x_2)$
1, 3, 4	$f(x_1)$	$f(x_2)$	$\neg q(x_1, x_2)$
1, 3, 4, 2	$f(x_1)$	$f(x_2)$	$\neg q(x_1, x_2) \wedge$ ~~$p(g(f(x_2)))$~~ $q(f(x_1), f(x_2))$
1, 3, 4, 2, 6			
1, 3, 4, 2, 6, 7			
1, 3, 4, 2, 6, 7, 3			
1, 3, 4, 2, 6, 7, 3, 4			
1, 3, 4, 2, 6, 7, 3, 4, 2			
1, 3, 4, 2, 6, 7, 3, 4, 2, 5			

Problem 2. Let F be the flowchart schema of Figure 2.

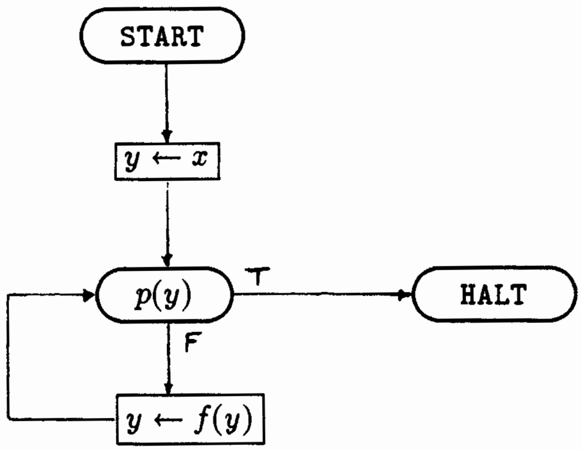


Figure 2: The flowchart schema F

We say that a wff A *expresses divergence* of F iff for all interpretations $\mathcal{I}$,

$$\mathcal{I} \models A \quad \text{iff} \quad F \text{ under } \mathcal{I} \text{ does not halt} .$$

(a) Write down a *second-order* wff which expresses divergence of F, and explain your answer.

(b) For each $n \geq 0$, let A_n be the wff

$$\neg p(f^0(x)) \wedge \neg p(f^1(x)) \wedge \ldots \wedge \neg p(f^n(x)) ,$$

where $f^n(x)$ abbreviates the term

$$\underbrace{f(\ldots(f(}_{n} x))\ldots)$$

($f^0(x)$ is x). Suppose A is a wff which expresses divergence of F. Show that the set of formulas

$$\{\neg A\} \cup \{A_n \mid n \geq 0\}$$

is finitely satisfiable but not satisfiable.

(c) Conclude from part **(b)** that there is no *first-order* wff which expresses divergence of F.

Problem 3. In this problem we assume for simplicity that wffs and flowchart schemes are without the equality symbol. We call a first-order wff a Σ_1 wff if it is of the form

$$\exists x_1 \ldots \exists x_n \, A$$

where A is quantifier-free.

(a) Use Herbrand's theorem to show that the satisfiability problem for *closed* Σ_1 wffs is decidable.

(b) Conclude from **(a)** that the validity problem for quantifier-free first-order wffs is decidable.

(c) We showed in class that given a flowchart schema F and an integer $n \geq 0$ one can effectively find a quantifier-free first-order wff $S_{F,n}$ with the property that for any interpretation $\mathcal{I}$,

$$\mathcal{I} \models S_{F,n} \quad \text{iff} \quad F \text{ under } \mathcal{I} \text{ halts in exactly } n \text{ steps} .$$

Use this and part **(b)** to show that

$$\{(F,n) \mid \text{the flowchart schema } F \text{ halts in } \leq n \text{ steps under } all \text{ interpretations } \mathcal{I}\}$$

is decidable.

6.044 / 18.423
Lecture Notes, 11/25/87

WHILE PROGRAM SCHEMES

1. SYNTAX

We define while program syntax through a BNF form grammar:

W ::= x := term | W; W |
<u>if</u> qff <u>then</u> W <u>else</u> W <u>fi</u> |
<u>while</u> qff <u>do</u> W <u>od</u> | <u>stop</u>

Underlined words are the keywords. The symbols ':=' and ';' used above are part of the syntax. The words "term" and "qff" stand for terms and quantifier free wffs over a given predicate calculus signature, respectively.

2. STRUCTURED OPERATIONAL SEMANTICS (SOS)

<u>Def</u>: Given a model $\mathcal{M}$, a pair (W, ϑ) is a while program <u>configuration</u> over $\mathcal{M}$ if W is a while program and ϑ is a store (or valuation function) over $\mathcal{M}$.

We define a relation between configurations which we call the <u>one-step</u> relation and denote by $\xrightarrow[\mathcal{M}]{}$. When the model $\mathcal{M}$ is fixed we usually just write $\longrightarrow$. The relation is defined as the smallest relation closed under the following list of axioms and deduction rules.

(1) Axioms for if:

If $(m, g) \models p$ then
$(\underline{if}\ p\ \underline{then}\ W_1\ \underline{else}\ W_2\ \underline{fi}\ , g) \longrightarrow (W_1, g)$

If $(m, g) \not\models p$ then
$(\underline{if}\ p\ \underline{then}\ W_1\ \underline{else}\ W_2\ \underline{fi}\ , g) \longrightarrow (W_2, g)$

(2) Axiom for stop

$$(\underline{stop}\ ; W\ , g) \longrightarrow (W, g)$$

(3) Axiom for assignment:

$$(x := t\ , g) \longrightarrow (\underline{stop}\ , g[x \mapsto (t)_{(m,g)}])$$

(4) Deduction rule for $W_1 ; W_2$:

$$\frac{(W_1, g) \longrightarrow (W_1', g')}{(W_1 ; W_2\ , g) \longrightarrow (W_1' ; W_2\ , g')}$$

(5) Axiom and deduction rule for while:

If $(m, g) \not\models p$ then
$(\underline{while}\ p\ \underline{do}\ W\ \underline{od}\ , g) \longrightarrow (\underline{stop}\ , g)$

$$\frac{(W, g) \longrightarrow (W', g')}{(\underline{while}\ p\ \underline{do}\ W\ \underline{od}\ , g) \longrightarrow (W' ; \underline{while}\ p\ \underline{do}\ W\ \underline{od}\ , g')}$$

Problem Set 8 Solutions

Problem 1. Let F be the flowchart schema of Figure 1.

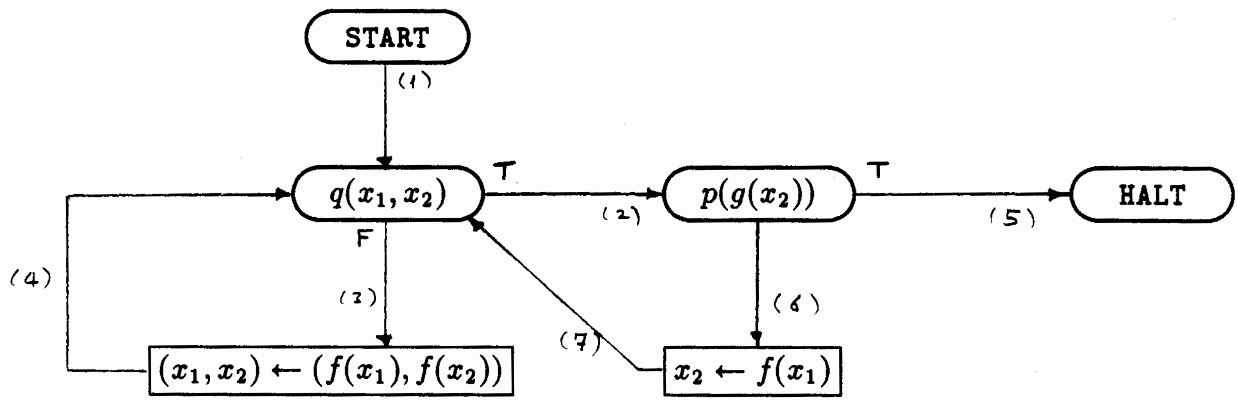


Figure 1: The flowchart schema F

The edges of the flowchart F are numbered so that we can refer to a path in F by the sequence of numbers of the edges that constitute it. The following table illustrates the construction of the formulas $S_{F;path}$. Fill in the blank entries of the table.

The completed table is as follows.

Path (p)	x_1	x_2	$S_{F;p}$
Λ	x_1	x_2	**true**
1	x_1	x_2	$S_{F;\Lambda} \wedge \mathbf{true}$
1, 3	x_1	x_2	$S_{F;1} \wedge \neg q(x_1, x_2)$
1, 3, 4	$f(x_1)$	$f(x_2)$	$S_{F;1,3}$
1, 3, 4, 2	$f(x_1)$	$f(x_2)$	$S_{F;1,3,4} \wedge q(f(x_1), f(x_2))$
1, 3, 4, 2, 6	$f(x_1)$	$f(x_2)$	$S_{F;1,3,4,2} \wedge \neg p(g(f(x_2)))$
1, 3, 4, 2, 6, 7	$f(x_1)$	$f^2(x_1)$	$S_{F;1,3,4,2,6}$
1, 3, 4, 2, 6, 7, 3	$f(x_1)$	$f^2(x_1)$	$S_{F;1,3,4,2,6,7} \wedge \neg q(f(x_1), f^2(x_1))$
1, 3, 4, 2, 6, 7, 3, 4	$f^2(x_1)$	$f^3(x_1)$	$S_{F;1,3,4,2,6,7,3}$
1, 3, 4, 2, 6, 7, 3, 4, 2	$f^2(x_1)$	$f^3(x_1)$	$S_{F;1,3,4,2,6,7,3,4} \wedge q(f^2(x_1), f^3(x_1))$
1, 3, 4, 2, 6, 7, 3, 4, 2, 5	$f^2(x_1)$	$f^3(x_1)$	$S_{F;1,3,4,2,6,7,3,4,2} \wedge p(g(f^3(x_1)))$

Problem 2. Let F be the flowchart schema of Figure 2.

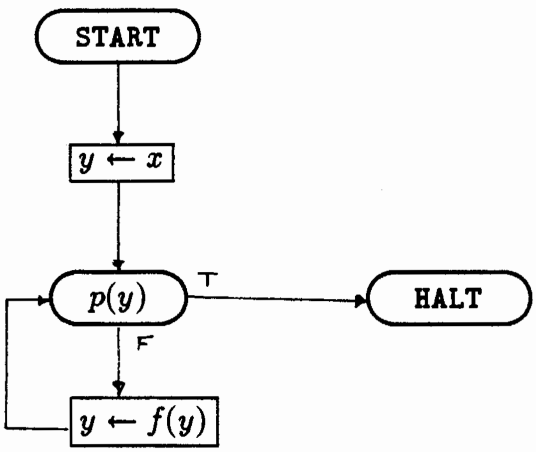


Figure 2: The flowchart schema F

We say that a wff A *expresses divergence* of F iff for all interpretations $\mathcal{I}$,

$$\mathcal{I} \models A \quad \text{iff} \quad F \text{ under } \mathcal{I} \text{ does not halt .}$$

(a) Write down a ***second-order*** wff which expresses divergence of F, and explain your answer.

Let A be the second-order wff $\exists Q\, B$ where B is

$$Q(x) \wedge \forall y[Q(y) \supset Q(f(y))] \wedge \forall y[Q(y) \supset \neg p(y)] .$$

We claim that A expresses divergence of F. For suppose F does not halt under $\mathcal{I}$. This means that for all $n \geq 0$ it is the case that

$$\mathcal{I} \models \neg p(f^n(x)) .$$

So

$$\mathcal{I}[Q \mapsto \{(f^n(x))_{\mathcal{I}} \mid n \geq 0\}] \models B ,$$

and hence $\mathcal{I} \models A$.

Conversely suppose $\mathcal{I} = ((D, I_c), I_v) \models A$. So there is a $D' \subseteq D$ such that

$$\mathcal{I}[Q \mapsto D'] \models B .$$

By definition of B it is the case that

$$\{(f^n(x))_{\mathcal{I}} \mid n \geq 0\} \subseteq D'$$

and

$$\mathcal{I}[y \mapsto d] \models \neg p(y)$$

for all $d \in D'$. So

$$\mathcal{I}[y \mapsto d] \models \neg p(y)$$

for all $d \in \{(f^n(x))_{\mathcal{I}} \mid n \geq 0\}$. That is,

$$\mathcal{I} \models \neg p(f^n(x))$$

for all $n \geq 0$. So F does not halt under $\mathcal{I}$.

(b) For each $n \geq 0$, let A_n be the wff

$$\neg p(f^0(x)) \wedge \neg p(f^1(x)) \wedge \ldots \wedge \neg p(f^n(x)) \ ,$$

where $f^n(x)$ abbreviates the term

$$\underbrace{f(\ldots(f(}_{n}x))\ldots)$$

($f^0(x)$ is x). Suppose A is a wff which expresses divergence of F. Show that the set of formulas

$$\{\neg A\} \cup \{A_n \mid n \geq 0\}$$

is finitely satisfiable but not satisfiable.

To show that $\{\neg A\} \cup \{A_n \mid n \geq 0\}$ is finitely satisfiable it suffices to show that for each $i \geq 0$ there is an interpretation $\mathcal{I}^i$ such that $\mathcal{I}^i \models \{\neg A\} \cup \{A_n \mid 0 \leq n \leq i\}$. We define $\mathcal{I}^i$ to be the interpretaion $((\mathbf{N}, I_c^i), I_v^i)$ where

- $I_c^i(f)$ is the succesor function.
- $I_c^i(p)(k) = \mathbf{true}$ iff $k = i + 1$.
- $I_v^i(x) = 0$.

Clearly $\mathcal{I}^i \models A_n$ for each $0 \leq n \leq i$. But $\mathcal{I}^i \models p(f^{i+1}(x))$ so F halts under $\mathcal{I}$. So $\mathcal{I}^i \models \neg A$.

The set $\{\neg A\} \cup \{A_n \mid n \geq 0\}$ is clearly unsatisfiable. For if $\mathcal{I} \models A_n$ for each $n \geq 0$ then F does not halt under $\mathcal{I}$. So by definition of A it must be the case that $\mathcal{I} \models A$.

(c) Conclude from part **(b)** that there is no ***first-order*** wff which expresses divergence of F.

If the A of part **(b)** were a first-order wff then $\{\neg A\} \cup \{A_n \mid n \geq 0\}$ would be satisfiable by the compactness theorem, which is a contradiction.

Problem 3. In this problem we assume for simplicity that wffs and flowchart schemes are without the equality symbol. We call a first-order wff a Σ_1 wff if it is of the form

$$\exists x_1 \ldots \exists x_n\, A$$

where A is quantifier-free.

(a) Use Herbrand's theorem to show that the satisfiability problem for *closed* Σ_1 wffs is decidable.

Skolemizing a closed Σ_1 wff $\exists x_1 \ldots \exists x_n\, A$ yields the quantifier-free wff

$$A^{c_1,\ldots,c_n}_{x_1,\ldots,x_n}$$

where the c_i are new 0-ary function constants. There is only one ground instance, so Herbrand's procedure terminates.

(b) Conclude from **(a)** that the validity problem for quantifier-free first-order wffs is decidable.

A quantifier-free wff is valid iff the negation of its universal closure is unsatisfiable. The latter wff is a Σ_1 closed wff whose satisfiability is decidable by part **(a)**.

(c) We showed in class that given a flowchart schema F and an integer $n \geq 0$ one can effectively find a quantifier-free first-order wff $S_{F;n}$ with the property that for any interpretation $\mathcal{I}$,

$$\mathcal{I} \models S_{F;n} \quad \text{iff} \quad F \text{ under } \mathcal{I} \text{ halts in exactly } n \text{ steps}\,.$$

Use this and part **(b)** to show that

$$\{(F,n) \mid \text{the flowchart schema } F \text{ halts in} \leq n \text{ steps under } \textit{all} \text{ interpretations } \mathcal{I}\,\}$$

is decidable.

For any interpreation $\mathcal{I}$,

$$F \text{ halts in} \leq n \text{ steps under } \mathcal{I} \quad \text{iff} \quad \mathcal{I} \models S_{F;0} \vee \ldots \vee S_{F;n}\,.$$

Thus F halts in $\leq n$ steps under *all* interpretations iff the wff $S_{F;0} \vee \ldots \vee S_{F;n}$ is valid. Since this wff is quantifier free, its validity is decidable by part **(b)**.

Problem Set 7 Solutions

Problem 1. Prove the validity problem for ∀∃-wffs *without function symbols* is decidable. (*Hint*: Generalize the last problem on Quiz 2.)

Let A be a ∀∃-wff without function symbols. The negation of its universal closure is a ∃∀ closed wff

$$\exists x_1 \dots \exists x_n \forall y_1 \dots \forall y_m\, B$$

without function symbols which is unsatisfiable iff A is valid. Skolemizing this wff introduces new constants $c_{x_1}, \dots c_{x_n}$, and since there are no function symbols these are the only ground terms. The Skolemized form thus has not more than n^m ground instances. Since the number of ground instances is finite, Herbrand's procedure terminates and says whether or not this wff is satisfiable.

Problem 2.

(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 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 3. Let T be (an infinite) set of first-order wffs. Let $Mod(T)$ be the set of models of (every formula in) T,

$$Mod(T) = \{\mathcal{M} \mid \mathcal{M} \models A \text{ for all } A \in T\} .$$

Show that $Mod(T) \neq \{\mathcal{M} \mid \text{ the domain of } \mathcal{M} \text{ is finite}\}$.

Let

$$\mathcal{F} = \{\mathcal{M} \mid \text{ the domain of } \mathcal{M} \text{ is finite}\} .$$

For each $n \geq 1$ let σ_n be the (closed) wff

$$\exists x_1 \ldots \exists x_n \left[\bigwedge\nolimits_{i \neq j} x_i \neq x_j\right]$$

which says that the domain has at least n elements. Let

$$S = T \cup \{\sigma_n \mid n \geq 1\} .$$

Claim 1. If $\mathcal{F} \subseteq Mod(T)$ then S is finitely satisfiable.

Proof: We wish to show that every finite subset of S has a model. For this it suffices to show that for each $i \geq 1$ the set $S_i = T \cup \{\sigma_n \mid 1 \leq n \leq i\}$ has a model. By assumption any model with finite domain is a model of T. Choose an $\mathcal{M} \in \mathcal{F}$ whose domain has $\geq i$ elements. Then $\mathcal{M} \models S_i$. ■

Claim 2. If $\mathcal{F} \subseteq Mod(T)$ then T has an infinite model.

Proof: By Claim 1 the set S is finitely satisfiable. By the compactness theorem it has a model $\mathcal{M}$. But $\mathcal{M} \models \sigma_n$ for all $n \geq 1$ so the domain of $\mathcal{M}$ must be infinite. And $\mathcal{M} \models T$ since $T \subseteq S$. ■

Claim 2 implies that $Mod(T) \neq \mathcal{F}$.

Alternatively, the previous axiom and deduction rule for while can be replaced by the single axiom

(5') $\langle$ <u>while</u> p <u>do</u> W <u>od</u> , σ)

$\longrightarrow$ (<u>if</u> p <u>then</u> W ; <u>while</u> p <u>do</u> W <u>od</u> <u>else</u> <u>stop</u> , σ)

Notice that some of the rules above, such as (a), do not depend on the model m.

We now define a partial function $[\![W]\!]_m$ from stores to stores (W a while program, m a model) by

$$[\![W]\!]_m(\sigma) = \sigma'$$

if

$$(W, \sigma) \xrightarrow[m]{*} (\underline{stop}, \sigma').$$

We can prove, by induction on the definition of $\xrightarrow[m]{}$, that every configuration arrows to at most one other configuration. So given σ there is at most one σ' such that $(W, \sigma) \xrightarrow[m]{*} (\underline{stop}, \sigma')$. So the above constitutes a good definition of a (partial) function.

SOS enables us to prove things about programs based on the structure of the program rather than the steps in its execution.

Master

Final Exam

Instructions. Do all 8 problems; a total of 200 points is allocated as shown on each problem. This exam is *open book*—you may appeal to any results from the text, handouts, or lectures. In doing a problem, you may also assume the results of *any preceding* problem (or problem part) on this exam. You have three hours. Good luck.

Problem 1 [20 points]. [Diagonalization] A language $R \subseteq \{\mathtt{a},\mathtt{b}\}^*$ is said to *separate* a pair of languages A and B iff $A \subseteq R$ and $B \subseteq \overline{R}$. Let $d(M) \in \{\mathtt{a},\mathtt{b}\}^*$ denote the code of a Turing machine M as in class notes. Let

$$K_{\mathtt{a}} = \{M \mid M \text{ on input } d(M) \text{ outputs } \mathtt{a}\}$$
$$K_{\mathtt{b}} = \{M \mid M \text{ on input } d(M) \text{ outputs } \mathtt{b}\}\ .$$

Prove that there is no recursive set R separating $K_{\mathtt{a}}$ and $K_{\mathtt{b}}$. (*Hint*: Consider the machine M which computes the function

$$f(x) = \begin{cases} \mathtt{b} & \text{if } x \in R \\ \mathtt{a} & \text{if } x \notin R.\ ) \end{cases}$$

Problem 2 [25 points]. [Post Correspondence Problem] A Post system $\{(\alpha_1,\beta_1),\ldots,(\alpha_k,\beta_k)\}$ (cf. Manna, §1-5.4) has an ***infinite solution*** iff there is an infinite sequence of integers $i_1, i_2, \ldots$ ($1 \le i_j \le k$ for all $j \ge 1$) such that

$$\alpha_{i_1}\alpha_{i_2}\cdots = \beta_{i_1}\beta_{i_2}\cdots$$

For example, the Post system with one pair $\{(\mathtt{a},\mathtt{aa})\}$ has no solution in the ordinary (finite) sense, but does have an infinite solution. Let

$$IPCP ::= \{S \mid S \text{ is a Post system with an infinite solution}\}.$$

(a) [5 points] Briefly explain why, in Manna's reduction of the halting problem for Post machines to PCP, if the machine diverges then the Post system Manna constructs has an infinite solution.

(b) [20 points] Show that IPCP is not r.e. (*Hint*: Slightly modify Manna's construction to reduce the *complement* of the halting problem for Post machines to IPCP.)

Problem 3 [25 points]. [Semigroup Word Problems] Consider semigroup terms and interpretations over the alphabet $\{\mathtt{a},\mathtt{b}\}$, i.e., words in $\{\mathtt{a},\mathtt{b}\}^+$ (cf. Handout 15). The *core* of a semigroup interpretation $\mathcal{I}$ is $\{\mathcal{I}(u) \in S \mid u \in \{\mathtt{a},\mathtt{b}\}^+\}$. A set of semigroup equations is *degenerate* iff the core of every interpetation which satisfies the equations has exactly one element. Let

$$DG = \{\mathcal{E} \mid \mathcal{E} \text{ is a finite, degenerate set of semigroup equations}\}$$

Prove that DG is r.e. (*Hint*: Use Completeness.)

Sequence logic is an extension of first-order logic in which there are two kinds of variables: individual (first-order) variables $x, y, z, \ldots$ which refer as usual to elements of the domain, and *sequence variables*, $X, Y, Z, \ldots$ which refer to finite sequences of elements of the domain; there will also be first-order terms t and sequence terms T. The definitions are precisely what you might expect, but to avoid doubts we now spell them out in more detail.

For any first-order signature, the terms and wffs of sequence logic (*s-wff's*) are defined by the following grammar using with three additional symbols $\doteq, \cdot,$ mkseq.

$$\begin{aligned}
t &::= \ \ldots\text{first-order terms}\ldots \\
T &::= \ T{\cdot}T \mid \mathrm{mkseq}(t) \mid X \\
\mathrm{atf} &::= \ \ldots\text{first-order atomic formulas}\ldots \mid T \doteq T \\
A &::= \ \mathrm{atf} \mid \neg A \mid A \wedge A \mid \forall x.\, A \mid \forall X.\, A
\end{aligned}$$

A *model* in sequence logic is simply a first-order model. Valuation functions, however, assign values to sequence variables as well as to individual variables. Namely, let

$$D^* = \{\langle d_1, \ldots, d_n\rangle \mid d_1, \ldots, d_n \in D, n \geq 0\}$$

be the set of finite sequences of elements from a set D. Then

Definition 1. A *sequence valuation function* over a model $\mathcal{M}$ with domain D is a function I_v which assigns an element of D to each individual variable and an element of D^* to each sequence variable, i.e., $I_v(x) \in D$ and $I_v(X) \in D^*$. A *sequence interpretation* $\mathcal{I}$ is a pair $(\mathcal{M}, I_v)$ where $\mathcal{M}$ is a model and I_v is a sequence valuation function over $\mathcal{M}$.

The meaning of sequence terms in a sequence interpretation $\mathcal{I} = ((D, I_c), I_v)$ is defined as in first-order logic with the additional feature that $\cdot$ is interpreted as concatenation, $\cdot$, of sequences and mkseq is interpreted as the function which promotes an element to the sequence of length one consisting of just this element. That is,

- $(t)_{\mathcal{I}}$ is defined exactly as in first-order logic.

- $(X)_\mathcal{I} = I_v(X)$.
- $(T_1 \cdot T_2)_\mathcal{I} = (T_1)_\mathcal{I} \cdot (T_2)_\mathcal{I}$.
- $(\text{mkseq}(t))_\mathcal{I} = \langle (t)_\mathcal{I} \rangle$

Satisfaction over a sequence interpretation $\mathcal{I}$ is then defined as in first-order logic with the addition that the symbol $\doteq$ is interpreted as equality of sequences:

- $\mathcal{I} \models A$ for a first-order atf A is defined exactly as in first-order logic.
- $\mathcal{I} \models T_1 \doteq T_2$ iff $(T_1)_\mathcal{I} = (T_2)_\mathcal{I}$.
- $\mathcal{I} \models \neg A$, $\mathcal{I} \models A \wedge B$, and $\mathcal{I} \models \forall x.A$ are defined exactly as in first-order logic.
- $\mathcal{I} \models \forall X.A$ iff for all $d^* \in D^*$ it is the case that $\mathcal{I}[X \mapsto d^*] \models A$.

Abusing notation, we write $\langle t \rangle$ instead of mkseq(t) in s-wff's. We also use the other logical connectives ($\supset$, etc.) and quantifiers $\exists x$ and $\exists X$ which can be expressed in terms of the connectives above in the usual way. For example, the s-wff

$$\forall x_1, x_2, X_1, X_2. (\langle x_1 \rangle \cdot X_1 \doteq \langle x_2 \rangle \cdot X_2) \supset (x_1 = x_2) \wedge (X_1 \doteq X_2)$$

asserts that every sequence which has a first element has a unique first and rest. This s-wff is valid in all models.

Problem 4 [20 points]. [Simple wff's and Compactness]

(a) [3 points] Write an s-wff whose only free variable is X and whose meaning is that X is the empty sequence $\Lambda \in D^*$. (Remember that no constant denoting Λ appears in s-wff's.)

(b) [3 points] Write an s-wff whose only free variables are x and X, and whose meaning is that x occurs in X.

(c) [4 points] Write an s-wff which is valid in precisely those models with finite (non-empty) domain.

(d) [10 points] State precisely and explain the conclusion that sequence logic does not satisfy the Compactness Property.

Problem 5 [25 points]. [Incompleteness]

Let Th($\{\mathtt{a},\mathtt{b}\}^*$) be the first-order wff's valid over the structure $(\{\mathtt{a},\mathtt{b}\}^*, \mathtt{a}, \mathtt{b}, \cdot)$ of strings of a's and b's under concatenation. Let *s-Valid* be the set of valid s-wff's.

(a) [20 points] Show that Th($\{\mathtt{a},\mathtt{b}\}^*$) $\leq_m$ s-Valid.

(b) [5 points] Conclude a "Gödel's Incompleteness" Theorem for sequence logic.

Sequence while program schemes (swps's) are while program schemes extended to include sequence variables. That is, swps's are while program schemes with sequence assignment statements of the form $X := T$ in addition to ordinary assignment statements of the form $x := t$. Tests in swps's are quantifier-free *s-wffs*. We also allow sequence assignment statements of the form $X := \epsilon$ whose effect is defined to be setting X to the empty sequence, $\Lambda \in D^*$. For example, if X is a sequence of a's, then the following swps has the effect of setting Y equal to X and x equal to a or b depending on the parity of the length of X.

$Y := \epsilon$; $x :=$ a;
while $Y \neq X$ **do**
  $Y := \langle \mathrm{a} \rangle \cdot Y$;
  if $Y \doteq X$ **then** $x :=$ b **else** $Y := \langle \mathrm{a} \rangle \cdot Y$ **fi**
od

Problem 6 [25 points]. [Induction on Terms and While Program Schemes] This problem explains why the "extra" sequence assignment statement $X := \epsilon$ is needed in swps's. A sequence interpretation $\mathcal{I}$ is Λ-*free* iff $I_v(X) \neq \Lambda$ for all X.

(a) [10 points] Prove that if $\mathcal{I}$ is Λ-free, then $(T)_{\mathcal{I}} \neq \Lambda$ for all sequence terms T.

(b) [15 points] Conclude that there is no swps *without the constant* ϵ which halts with X equal to Λ in all interpretations.

Problem 7 [30 points]. [Hoare Logic] Let W be the swps given above. Use the axioms and rules of Hoare logic, extended to allow s-wff's as pre- and post-conditions, to prove the partial correctness assertion

$$\{\text{true}\} W \{\exists Z. (X \doteq Z \cdot Z) \equiv (x = a)\}$$

Problem 8 [30 points]. [While Schemes and Computability] Let $\mathcal{Z}_n$ be the integers mod n under addition, and let $\mathcal{I}_0$ be the valuation under which all first-order variables are zero and all sequence variables are Λ. In this problem we consider swps's whose only first-order symbols are $0, 1, +, =$. Define spectrum(W) for a swps W to be

$$\{n > 1 \mid W \text{ halts started in } (\mathcal{Z}_n, \mathcal{I}_0)\}\ .$$

(a) [5 points] Explain why spectrum(W) is r.e. for every swps W.

(b) [10 points] Exhibit a swps which, for *all* $n > 1$, halts under interpretation $(\mathcal{Z}_n, \mathcal{I}_0)$ with variable X equal to a sequence of n zeroes.

(c) [15 points] Sketch an argument demonstrating that *every* r.e. subset of $\{n \mid n > 1\}$ is the spectrum of some swps.

DRAFT
12/22/87

6.044J/18.423J (Fall 1987)
LECTURE SUMMARY

1. Fri, 9/11/87 Overview: computability and logic.

2. Mon, 9/14/87 Overview: logic and logic of programs.

3. Wed, 9/16/87 Turing Machines: definitions and examples.

4. Fri, 9/18/87 Multi-Turing machines, systolic arrays, Post machines, Simulation Thesis.

5. Mon, 9/21/87 Simulation Thesis, RAMs; Defintion of Turing acceptable / recursive sets; Thm: Recursive iff r.e and co-r.e.

6. Wed, 9/23/87 Coding into strings; Halting Problem is r.e. not recursive.

7. Fri, 9/25/87 Diagonalization and Countability: Cantor's theorem.

8. Mon, 9/28/87 Computable functions; Thm: r.e. iff range of partial computable function; Definition of $\leq_m$; Halting Problem $\leq_m$ Blank Tape Halting problem.

9. Wed, 9/30/87 Basic properties of $\leq_m$; Rice's Theorem: statement and discussion.

10. Fri, 10/2/87 Rice's Theorem: proof; Thue / Semi-Thue system derivability problem: statement.

11. Mon, 10/5/87 There exist non r.e. non co-r.e. sets: counting argument; Undecidability of the semi-Thue system derivability problem.

12. Wed, 10/7/87 Undecidability of the Thue-system derivability problem.

    QUIZ I (EVENING)

13. Fri, 10/9/87 Post-Correspondance, Matrix mortality.

    ADD DATE

    Mon, 10/12/87 NO CLASS Columbus holiday

14. Wed, 10/14/87 Thue Systems and semigroups; Completeness theorem: statement.

15. Fri, 10/16/87 Completeness theorem: proof and corollaries.

16. Mon, 10/19/87 Predicate calculus: definitions.

17. Wed, 10/21/87 Predicate calculus: definitions; validity problem.

18. Fri, 10/23/87 Undecidability of the validities of the predicate calculus.

19. Mon, 10/26/87 Expressive power of the second-order predicate calculus; F_{N}; Overview of logic: completeness and incompleteness.

20. Wed, 10/28/87 F_{Arith}; First-order theory of strings is not r.e.

21. Fri, 10/30/87 Herbrand's procedures: equisatisfiability and equivalene, Prenex form.

22. Mon, 11/2/87 Every wff is equivalent to a wff in Prenex form: proof.

23. Wed, 11/4/87 Skolem form; Herbrand's theorem: statement and explanation.

24. Fri, 11/6/87 Towards the proof of Herbrand's theorem: Konig's lemma and propositional compactness.

25. Mon, 11/9/87 Compacthess theorem of first-order logic; Non-standard models of arithmatic.

    QUIZ II (EVENING)

    Wed, 11/11/87 NO CLASS Veterans Day holiday

26. Fri, 11/13/87 Complete axiom systems for first-order logic, soundness and completeness theorems.

27. Mon, 11/16/87 Overview of the proof of completeness: Henkinazation, complete theories.

28. Wed, 11/18/87 Flowchart Schemas.

29. Fri, 11/20/87 Park's lemma; $K_2 \leq_m \{F | \text{there is an } \mathcal{I} \text{ such that } F \text{ terminates in } \mathcal{I}\}$.
DROP DATE

30. Mon, 11/23/87 Flowchart scheme equivalence is not r.e.

31. Wed, 11/25/87 While program schemes and their structured operational semantics.

    Fri, 11/27/87 NO CLASS Thanksgiving day holiday

32. Mon, 11/30/87 Equivalence of flowcharts and while programs; dynamic logic.

33. Wed, 12/2/87 Floyd-Hoare logic, partial correctness; concatenation theory completeness of Hoare logic.

34. Fri, 12/4/87 Proof of the above, including expressiveness lemma.

35. Mon, 12/7/87 Resolution theorem proving.

36. Wed, 12/9/87 Recursive function schemes.

6.044 Lecture Notes #1

09/11/87 (Friday)

(1) General Introduction (Administrative)
Hard homeworks.

(2) Text : MANNA
First Reading assignment : sections 1-2, 1-3

outline :

Plan to follow book although it is ancient, the only part that is weak is the computability which will be supplemented. Convincing proofs for a mathematician. Want to teach how to do mathematics / proof.

Topics :

(1) Computability theory : what can computers not do ? Quite precise mathematical interpretation of this statement. What total functions $f: \mathbb{N} \to \mathbb{N}$ cannot be computed. say

$f(n)$ = the largest k s.t. $p^k \mid n$ for some prime p.

Then $f(9) = 2$ since $9 = 3^2$. f is computable.

Can code integers into bits, we define a function

$g(\langle M_1, \ldots, M_k \rangle) = 0$ iff $(M_1 \cdot \cdots \cdot M_k)_{3,2} = 0$

where $M_1, \ldots, M_k$ are 3 by 3 matrices with integer entries. This is computable. Now with same conventions

$h(\langle M_1, \ldots, M_k \rangle) = 0$ iff $\exists M_{i_1}, \ldots, M_{i_n}$, $n \geq 1$, $1 \leq i_j \leq k$ s.t. $(M_{i_1} \cdot \cdots \cdot M_{i_n})_{3,2} = 0$.

(i.e. take some of the matrices, can use any of them as often as you want, s.t. product has 0 in 3,2 position). Easy to write an exhaustive search program which will eventually print 0 if $h(\langle M_1, \ldots, M_k \rangle) = 0$. But what if $h(\langle M_1, \ldots, M_k \rangle) = 1$? No obvious way to program a "stop" into the search which results in computing h correctly.

Theorem: (proof later) h is not computable.

In order to understand and prove such a theorem we need a good mathematical definition of "computable". If h couldn't be computed in SCHEME, could it be done in FORTRAN? Will show that it could not through the primitive model (computer) of the Turing Machine, which we will show can simulate, say, a CRAY running SCHEME. Will eventually prove the undecidability of the halting problem. The proof is based on a "paradox" type argument such as

"This statement is false".

Then we code the halting problem into the question of whether h is 0 or 1.

Other undecidable problems -

* Hilbert's 10th problem: Take an integer polynomial in several variables,

$$3xy^2 - 17x^2z^3 + xz + 3$$

Does it have an integer (vector) root? Input is an arbitrary diophantine polynomial. Again, easy to write program which enumerates $\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$ and finds a solution if one exists. One method to enumerate is by increasing value of the sum $|x| + |y| + |z|$ of $\langle x, y, z \rangle$. (This, by the way, proves that $\mathbb{Z}^3 \overset{1-1}{\rightarrow} \mathbb{N}$). Again, no obvious way to stop search when you haven't found a root yet.

* Another example is the problem of deciding whether two given context free grammars are ambiguous.

(2) Development of Formal Logic: In order to know whether

$$\forall x \exists y (x \neq y)$$

is true or false, need to know what x and y are ranging over.

$$\neg \forall x \exists y (x \neq y) \supset \forall z \; f(z) = z$$

deduce —↑ world has only 1 element — for all z, if f is a function in this world, then $f(z) = z$.

$$f(\alpha) = \begin{cases} 1 & \text{if it is valid} \\ 0 & \text{otherwise} \end{cases}$$

is not computable.

6.044
09/14/87 (Monday)

Lecture # 2.

Topics:
(1) Computability (and non-computability)
(2) Formal logic.

Last time: overview, look at big topics of course.
This time: finish survey, start Turing machines.

We will do the following results in logic.
(1) Gödel's completeness theorem. In first order logic, provability holds iff validity holds. An astonishing theorem, given how different are the concepts of provability and validity.
(2) Gödel's Incompleteness theorem: Provability ⊊ validity for second order logic. Also Provability ⊊ True formulas over standard arithmetic.

Imagine having written a bunch of axioms about the successor function $S(n)$: $\forall x (S(x) \neq x)$, $\forall x \forall y (S(x) = S(y) \Rightarrow x = y)$, ...

Axioms to define addition:

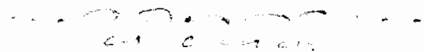
A1: $\forall y\, [add(0, y) = y]$

A2: $\forall x \forall y\, [successor(add(x, y)) = add(successor(x), y)]$

Ask whether the above two axioms of addition imply that

A3: $\forall u \forall v (add(u, v) = add(v, u))$

We know this formula is true for standard arithmetic, (N, succ, add). Claim that A3 is not valid. Why? There are strange models which have things satisfying above axioms but in which add is not commutative. An idea about such a model: It has elements in it which are not obtained from 0 by applying the successor any finite number of times.



Compactness is a corollary of completeness.

Topic 3) : models of programming languages (schemes)
Attempt to formulate models of flowcharts etc at compiler level of abstraction.

We will do things like: take assignment statement $x := f(x, y)$ from a flowchart. We don't say what f is (treat it like in logic), and analyse the flowchart.

Dijkstra: advocate of structured programming. "gotos considered harmful".
Theorem: An arbitrary program (flowchart) can be translated into one with one "while" and no "goto"'s.

Flowchart schemes, recursion schemes
Method of structured operational semantics (SOS)

(3.5) Mathematical semantics
Tries to find the right mathematical meaning of phrases for programming languages.

An example :

```
(define (COMPOSE f g)
(lambda (x)
    (f (g (x))))

(COMPOSE (lambda (x) (+ x 2))   1+  ) 3 )
> 6      ; SCHEME output)

(define (LEFT-ASSOCIATE  f g h)            this is (f∘g)∘h
  (COMPOSE (COMPOSE  f g) h ))

(define (RIGHT-ASSOCIATE f g h)            this is f∘(g∘h)
  (COMPOSE  f  (COMPOSE  g  h)))
```

would like to say:

Theorem: RIGHT-ASSOCIATE and LEFT-ASSOCIATE "mean the same thing".

"proof": obvious, because composition is associative:

$(f \circ g) \circ h = f \circ (g \circ h)$.

Mathematical semantics attaches to RIGHT-ASSOCIATE a function, as with LEFT-ASSOCIATE, such that we can prove the above theorem. Note in SCHEME functions can be applied to themselves, so our functions must have this property.

(4) Verification of programs

Hoare logic

Recursion - Induction

proof methods for programs in a language (extended formal language) which has programs as operators. Have completeness / incompleteness here too.

6.044
09/16/1987 Lecture # 3

Begin real subject matter today.

Turing Machines
Post machines
Pushdown machines
⋮

TURING MACHINES

Devices meant to process a finite set of symbols. Σ denotes a finite set of elements called symbols, also called an alphabet. Usually, and for definiteness, $\Sigma = \{a, b\}$. A word (or string) is something like aaba. A language is a set of words, usually infinite. Words are sequences of finite length. The empty word is a word of length 0 denoted Λ. Technically, there is a different empty word for every alphabet, but we usually don't bother about it. Operation of concatenation puts words next to each other. $x \cdot y = xy$. The properties defining Λ are

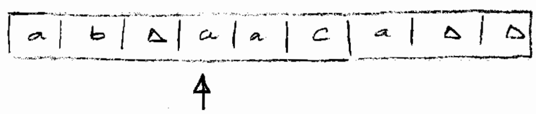
$$\Lambda \cdot x = x \cdot \Lambda = x \quad \text{for all } x.$$

Σ^* is the set of all words over Σ. So a language is a subset of Σ^*.

Basic data structure of a TM is a one way tape, each square of which can hold one symbol. A TM has two alphabets, one for internal bookkeeping. Σ is the input/output alphabet, and V is the alphabet of auxiliary symbols. The blank symbol is Δ.

The tape is really just a (finite) word over

$\Sigma \cup V \cup \{\Delta\}$. The right end of the tape cell can always be added to: new squares initialised with Δ appear as needed.



For this example, $\Sigma = \{a, b\}$, $V = \{c\}$,
Primitive instructions: shifting one square R or L, printing, and branching: read and branch.
TM program is a series of print instructions

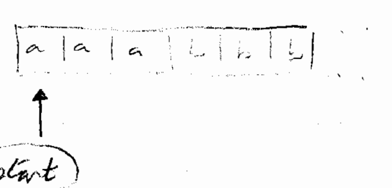
There is a START and two kinds of halting: ACCEPT REJECT. Book has

Moving right at end of tape: (add a blank)

Shifting off the left end means REJECT.

TM to accept language $\{a^n b^n \mid n \geq 0\}$:

Convention for starting



Note that Δ is a word, not a symbol, and can't occur in a tape square. The start configuration looks like

if the input is Δ.

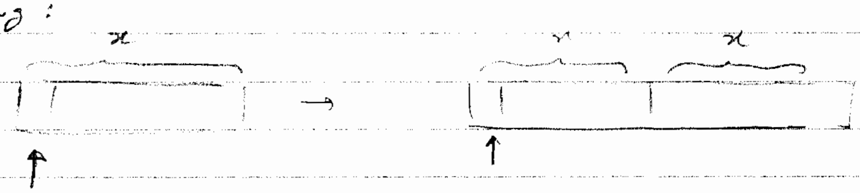
Recognizing $\{a^n b^n \mid n \geq 0\}$: to begin, if see b, reject, if see Δ accept, otherwise enter loop.

Loop:

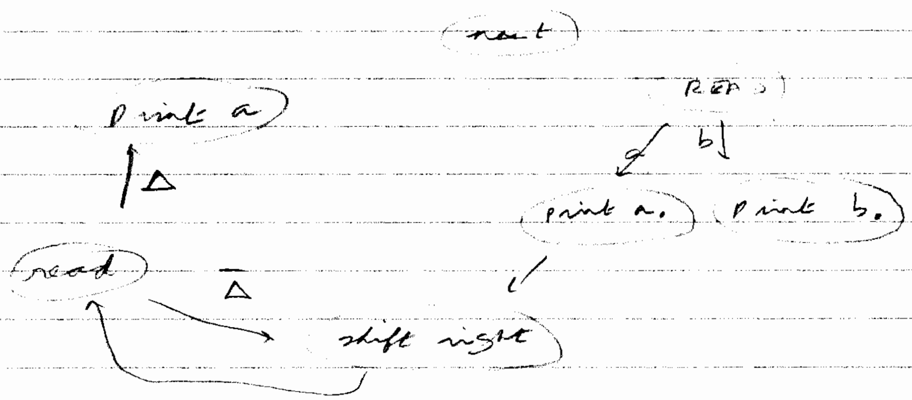
check the a: $a^{\checkmark}$
shift right past a's and checked b's.
Move left to right of first checked a met.

finally: go beyond all $b^{\checkmark}$, and see that there is a Δ there.

Doubling:



First mark left most symbol with an 'L' to indicate it is leftmost. Then 'carry' symbols across, marking the first one to indicate end of the word x.



6.044

09/18/1987 Lecture # 4.

Homework #1 handed out, due 09/25/87.

Want a model of computation that is as simple as possible yet captures general computability. TM is about the simplest. Not too hard to see that multiple headed, multi-tape, or 2 dimensional TMs can be simulated by the model we described.

How could we simulate (keeping meaning of "simulate" informal) a two-headed (one-tape) machine by a one headed one tape machine? Multiply alphabet size by 4: have marks $\uparrow_1$, $\uparrow_2$, $\uparrow_1\uparrow_2$ on each of the old symbols. Have to keep track ("remember") where the two heads are.

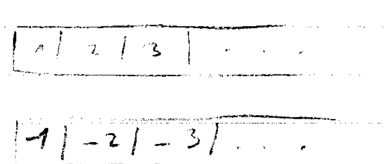
For a TM M,

$$L(M) = \{x \in \Sigma^* \,/\, M \text{ accepts } x\}$$

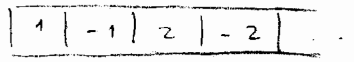
M accepts x iff starting properly on input x (i.e. according to our input conventions), M eventually enters an ACCEPT state.

so "M_2 simulates M_1" means $L(M_1) = L(M_2)$.

Now for multiple tapes: how do we simulate two tapes with one? One way: use even numbered squares for contents of tape 1, and odd numbered squares for contents of tape 2.

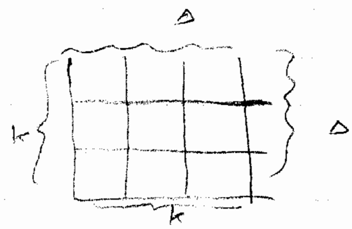


becomes

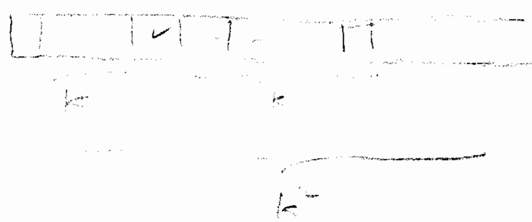


Two way infinite tapes: simulate with two one way infinite tapes.

Two dimensional:

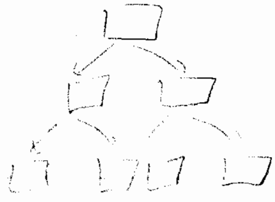


at any stage in 2D computation a finite square is all that is used. simulate by laying rows one after the other



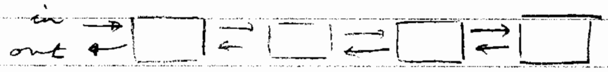
TMs with a growing number of tapes? Concept not clear ...

Consider TM with squares in tree structure so that in n moves can access any of 2^n cells:

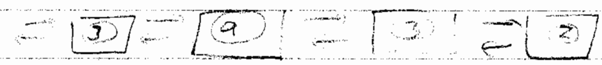


This can be simulated in the same way.

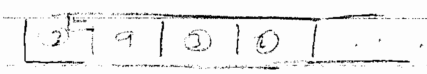
Systolic (Iterative) arrays:



each box is a finite state machine (same machine for each box). Each box in some state moving synchronously in time. State of box determined by its current state and state of its neighbours. How do we simulate this? suppose states are

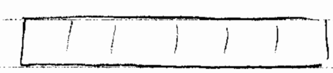


then tape has



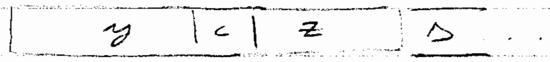
POST MACHINES

Tape divided into squares, input word put on tape same way as in TM.

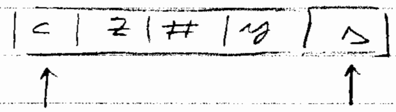


Think of tape as queue. One head shifts only to the right, and can't write. Another head sits on the first blank after end of input. It is a write only head. The heads together perform a destructive read.

Obvious that a TM can simulate this: we just described it as a TM with 2 heads. We explain strategy for reverse. Have a simple TM ,



PM tape (register) looks like



simulating a right shift looks easy, but is complicated by the fact that z may be Δ .

6.044
09/21/87 Lecture # 5

Next step in our list of machines is the RAM. These are given by (finite) flowcharts. Boxes are of the form

$$x := f(x, y)$$

where f is a fixed definite set of functions, say x and y are registers that can hold finite strings from $\{0,1\}^*$. There are branches.

$$x = y?$$

and one of the fs is probably concatenation. Another is $x :=$ first (y).
TM could simulate a RAM by keeping the (finite, fixed) number of identifiers on its tape.
RAM can also simulate a TM.

RAMs with indirect addressing,

$$x^* := x \cdot y^*$$

Think of all registers as indexed by binary words; memory address is anything in $\{0,1\}^*$, and contents of a memory are strings of $\{0,1\}^*$. so above means: store in memory indexed by x the concatenation x (contents of memory y).'
This too is equivalent to TMs.

Now, SCHEME. Want to write a SCHEME procedure which takes a TM and input, and simulates the TM on the input. Problem: SCHEME has only finite memory. "Idealized" SCHEME would be, for example, one where you could have an unlimited list size.

"simulation thesis": Essentially Church's thesis, but actually Church's thesis says people = Turing machine. Our simulation thesis just says any high level algorithmic description = TM

A language L is Turing acceptable / recursively enumerable (r.e.) iff $L = \{x \in \Sigma^* \mid M \text{ accepts } x\}$ for some machine M (any kind of machine).

A language is decidable / recursive iff there is a machine which on input x halts and accepts if $x \in L$, and halts and rejects if $x \notin L$.

Thm: L is recursive iff L and $\bar{L} = \Sigma^* - L$ are r.e.

Proof - suppose L and $\bar{L}$ are r.e. Let M_1 be an acceptor for L and M_2 an acceptor for $\bar{L}$. We construct a decider for L. Recipe

"Given input, supply x as input to M_1 and M_2. Run them in parallel. If M_1 halts first, then halt and accept. If M_2 halts first then halt and reject. If neither halts then run forever."

Note we need the code of M_1, M_2: we are actually constructing a dovetailing simulator.

6.044
09/23/87 Lecture # 6

Last time: A set is recursive iff it is r.e. and co-r.e.
We proved r.e. & co-r.e ⇒ recursive last time.
The other direction is trivial: given M deciding L want a "half decider" M' for L. Given input x, run M on x until it halts (If it doesn't halt then diverge). If M accepts then accept. Else diverge.

Remark:
L is r.e. iff L = {x / M accepts x} for some M
iff L = {x / M halts on x} for some M.

Encoding of numbers: $n \in \mathbb{N}$ can be encoded as either a unary or binary strings, similarly, other finite mathematical objects are coded into strings.

Recursive sets: even numbers, prime numbers

Diophantine polynomials with no integer root: co-r.e.
(deep fact: it is not r.e.)
Note on coding in a case like this: there are strings (ill formed expressions) which do not represent polynomials. These are assigned by default to the polynomial 0. We assume the coding is computable. There is a TM to decide whether or not a given string is a well formed expression.

{M / M is a TM on M halts on blank input}
is r.e. but we will show it is not recursive.

{(M, x) / M halts on x} is also r.e. not recursive.

note again the implicit coding: M and x into strings over {a, b}, (say), or any other fixed alphabet.

$\overline{K_1} = L_1 = \{ M \mid M$ does not halt on input $d(M) \}$ where $d(M) \in \{0,1\}^*$ is the code of M, is co-r.e.

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

Proof: Suppose $\overline{K_1}$ was r.e. Say M_0 accepts it. Thus M_0 halts on input $d(M)$ iff M does not halt on input $d(M)$ for all M, by definition of $\overline{K_1}$. Let M be M_0. Therefore

M_0 halts on $d(M_0)$
iff M_0 does not halt on $d(M_0)$ ☒

Corollary: The halting problem is r.e. but not recursive.

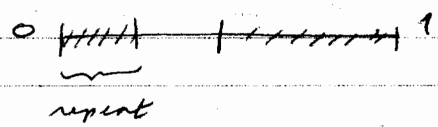
Proof - we know $\{(M, x) \mid M$ halts on $x\}$ is r.e. If it were recursive, we could decide K_1 ☒

6.044
09/25/84 Lecture # 7

Historical digression

Cantor : $\mathbb{Q}$ has measure 0 in $\mathbb{R}$.

Dividing the interval :



i.e. take $\frac{1}{2} + \frac{1}{2}\cdot\frac{1}{4} + \frac{1}{2}\cdot\frac{1}{16}$

$$= \frac{1}{2} \sum_{i=1} 2^{-i}$$

Borel : to find measure of set of points, try to cover them by finite number of intervals ~~and take~~ (disjoint) and take sum of lengths. This gives upper bound. For lower bound use intervals in the set. But this way can't measure $\mathbb{Q} \cap [0,1]$, or $\mathbb{Q}$.

Lebesgue : use countably many intervals. Now can prove that $m(\mathbb{Q}) = 0$. Enumerate $\mathbb{Q}$, $r_1, r_2, \ldots$

$0 \in$	$[0, \epsilon/2]$	length $\frac{\epsilon}{2}$
$1/1 \in$	$[1, 1+\frac{\epsilon}{2}]$	.
⋮		.
$r_n \in$	. . .	$\epsilon/2^n$

Cantor : countable sets are smaller than $[0,1]$.

Compare sets : $|A| \le |B|$ if $\exists f: A \to B$ injective (and total) iff $\exists g: B \to A$ total and onto

Def : A is countable iff $|A| \le |\mathbb{N}|$.
~~iff $\exists g: B \to A$ total and onto~~.

Theorem (Cantor): $|\mathbb{N}| = |\mathbb{Q}|$.

Theorem (Cantor): $|\mathbb{Q}| < |[0,1]|$

Note $|A| = |B|$ iff $|A| \le |B|$ and $|B| \le |A|$.

$|A| < |B|$ iff $|A| \le B$ but not $|B| \le |A|$.

Proof: $|\mathbb{Q}| \le |[0,1]|$ obvious: ~~suppose~~ we know $|\mathbb{Q}| = |\mathbb{N}| \le |[0,1]|$, ~~the~~ last because

$$f(n) = \begin{cases} 0 & n = 0 \\ \frac{1}{n} & n > 0 \end{cases}$$

is an injection. For the converse, suppose to the contrary that there is a function $g : \mathbb{N} \to [0,1]$ which is onto.

$\mathbb{N}$	
0	decimal expansion of $g(0)$
1	decimal expansion of $g(1)$
2	⋮
3	

Note: use $1 = .99\ldots$; decimal expansions are not unique.

By hypothesis every real number occurs on some row. Define

$$\text{my-}\pi = .\ldots$$

s.t. n-th digit $\ne$ digit in (n-th row, n-th column) above. So n-th digit $\ne$ n-th digit of $g(n)$. Small problem: decimal expansion of my-π doesn't appear, but the same real number might as another expansion. Fix by either (1) making all expansions appear in list (2) choosing my-pi a bit more carefully:

add 2 mod 10. Lemma: this gives
distinct number.

Prove: $|S| \leq |P(S)|$ (exercise)

Next time: definition of computable functions,
(total and partial).

6.044
09/28/87

Lecture # 8

Theorem: $A \subseteq \Sigma_1^*$ is r.e. iff
(i) A = domain(ψ) for some partial computable function $\psi: \Sigma_1^* \to \Sigma_2^*$, for some Σ_2.
(ii) A = range(ψ') for some partial computable function $\psi': \Sigma_0^* \to \Sigma_1^*$, for some Σ_0.
(iii) $A = \emptyset$ or A = range(f) for some total recursive $f: \Sigma_0^* \to \Sigma_1^*$, some Σ_0.

Note: If we have a TM M computing a function φ, M may halt on inputs not in domain(φ), i.e. on garbage input, to produce garbage output. (Technicality)

Proof of theorem (i): Say $A = \{x \in \Sigma_1^* / M$ halts on $x\}$. Fix M to be M' which acts like M but cleans up its tape when it halts. Then M' computes the constant function Λ with domain A. Converse is trivial; If A = dom(ψ) and M computes ψ, just fix M a little.

Proof (ii): Let A = domain(ψ) and say M computes ψ. Create M' which computes the identity function on domain(ψ), and fixes M for what it does on ill formed inputs. Conversely let M computing ψ' be given. Need a program which halts on inputs in range(ψ').

"Given input $y \in \Sigma_1^*$
search for $z \in \Sigma_0^*$ s.t. $\psi'(z) = y$
halt if you find one."

How do we search for z? Search in parallel over all $z \in \Sigma_0^*$. If there is a good z we will find it in finite time. This is called "dovetailing".
At stage n, spend n steps running each of processes $1, \ldots, n$ (where process i is computing $\psi'(z)$ with z being the i-th string).

Proof of (iii) is HW #2 (1). Hints: use dovetailing and remember there may be repetitions.

We talked about the blank tape halting problem $K_2 = \{ M / M$ halts on input $\Delta \}$. Special case of general halting problem K_0. But in fact just as hard.

<u>Lemma</u>: The decision problem for any r.e. set R reduces to the decision problem for K_2.

More precisely, if M could tell whether or not a TM halted on Λ, one could tell given x whether or not $x \in R$.

Proof - say $R = \{ x / M$ halts on $x \}$. To decide if $y \in R$, construct from M and y a new machine $M_{R,y}$ which operates as follows:

"given input z, erase z and print y on the input. Then run like M"

Thus $y \in R$ iff M halts on y (by def of M)
iff $M_{R,y}$ halts on all inputs (by def. of $M_{R,y}$)
iff $M_{R,y}$ halts on Δ.
iff $M_{R,y} \in K_2$

Note we can find $M_{R,y}$ given M and y; i.e. can construct it without knowing beforehand what it will do. We have a translator $T: y \mapsto M_{R,y}$. The construction of $M_{R,y}$ from y is computable. So if K_2 was recursive so would R be ☒

<u>Corollary</u>: If K_2 was recursive so would R be

Proof - remarks at end of previous proof. ☒

Two kinds of arguments that the field builds on:
(1) diagonal arguments: which get you off the ground. Sometimes unnatural problems.
(2) reductions: other more natural problems inherit the undecidability of the previous problems.

Def: $A \subseteq \Sigma_1^*$, $B \subseteq \Sigma_2^*$, say $A \leq_m B$ (many-one reducible to B) iff there is a total computable function $f : \Sigma_1^* \to \Sigma_2^*$ such that
$x \in A$ iff $f(x) \in B$,
for all $x \in \Sigma_1^*$.

The lemma used this. We proved
R is r.e. $\Rightarrow$ $R \leq_m K_2$.

Lemma: $\leq_m$ is transitive.

(Note: $A \leq_m B$ and $B \leq_m A$ does not imply $A = B$; they are called equivalent)

Lemma: If $A \leq_m B$ and B is recursive then A is recursive. (Contrapositive is interesting).

Lemma: If $A \leq_m B$ and B is r.e. so is A.

Note this is a weak reducibility. We ask B only one question: is $f(x) \in B$? We don't use the answer other than returning it, nor do we ask more questions.

6.044 Lecture # 9
09/30/87

Quiz: October 7 (Wednesday) 7-9 PM,
Room to be announced.

$A \le_m B$ iff ($x \in A$ iff $f(x) \in B$) for some computable f.

Example: $K_2 \le_m K_0$
$M \in K_2$ iff $(M, \Lambda) \in K_0$
i.e.
define
$f(x) = pair(x, code(x, \Lambda))$

$code(x, \Lambda)$ is code of Λ as a member of the alphabet specified by x, $code(x, \Lambda) \in \{a, b\}^*$.

If R is r.e., then $R \le_m K_0$. For say $R = dom(M_R)$, then define
$f(x) = (M_R, x)$

so K_0 is the hardest r.e. set: we say K_0 is r.e complete (with respect to $\le_m$), i.e.
(1) K_0 is r.e.
(2) $\forall R$, if R is r.e. then $R \le_m K_0$.

K_2 is also a complete r.e. set (proved last time). In particular this implies $K_0 \equiv_m K_2$ (many-one equivalent)
also $K_1 \equiv_m K_0$ (K_1 is the halting problem of M on M). same proof works.

Lemma: $A \leq_m B$ and B recursive $\Rightarrow$ A recursive

Proof - To decide whether $x \in A$,

(1) Compute $f(x)$ (where $x \in A$ iff $f(x) \in B$)

(2) Test whether $f(x) \in B$. Print yes if so, no otherwise. ☒

Lemma: If $A \leq_m B$ and B is r.e. then A is r.e.

Proof - Same idea. Use the B acceptor to halt iff $f(x) \in B$. ☒

Transitivity of $\leq_m$: $A \leq_m B$ and $B \leq_m C$ $\Rightarrow$ $A \leq_m C$.

Lemma: If R is r.e. and not recursive then R and $\bar{R}$ are $\leq_m$ incomparable.

Proof - Note: $x \in R$ iff $x \notin \bar{R}$. But this does not mean that the identity mapping can play the role of f in $\leq_m$. Now for the proof.

Case 1: Suppose $\bar{R} \leq_m R$. Then $\bar{R}$ is r.e. since R is r.e. and r.e. inherits downward.

So R is recursive since it is r.e. and co-r.e.

Case 2: Suppose $R \leq_m \bar{R}$. Use the lemma which follows to get $\bar{R} \leq_m R$, and then by Case 1. ☒

Lemma: $A \leq_m B$ iff $\bar{A} \leq_m \bar{B}$.

Proof - $x \in A$ iff $f(x) \in B$ so

$x \notin A$ iff $f(x) \notin B$. ☒

Rice's Theorem: (sloppy) Every I/O 'behavioral' property of programs is undecidable.

Is the set of inputs on which M halts r.e.? Yes, by definition (for any M). This is a trivial property.

A property P of r.e. sets is nontrivial iff $P(R_0)$ holds for some r.e. set R_0, and $\neg P(R_1)$ holds for some r.e. set R_1.

Let

$K_P = \{ M \mid P(\text{domain}(M)) \}$

Rice: Any r.e. set R is $\leq_m K_P$ if $\neg P(\emptyset)$. If $P(\emptyset)$ then any r.e. set R is $\leq_m \overline{K_P}$.

6.044
10/02/1987 Lecture # 10

Rice's theorem: Let P be a non-trivial property of r.e. sets such that $\neg P \emptyset$. Then for every r.e set R, $R \leq_m K_P$, where
$K_P = \{ M / P(\text{domain}(M)) \}$.

i.e in general, given any property, if you claim you have a program P to decide whether or not an r.e set has that property, then you are wrong: there will be inputs on which your program fails.

Note for any non-trivial P either P or $\neg P$ satisfies $\neg P(\emptyset)$. So by Rice either P or $\neg P$ is not recursive. So either P or $\neg P$ is not r.e.

Proof -
Let $L_1 \neq \emptyset$ be an r.e. set s.t. $P(L_1)$; it exists since P is not trivial: something $\neq \emptyset$ satisfies P. Let $R = \text{dom}(M_R)$. Consider the following machine $M_{f(x)}$ where $x \in \Sigma_R^*$.
"Given input $w \in \Sigma_{L_1}^*$, save w temporarily, and then simulate M_R on x. If M_R halts on x, then act like an L_1 acceptor on w"
Claim 1: This is programmable. In fact there is a translator program which maps x to $M_{f(x)}$; Here $f: \Sigma_{L_1}^* \to \{a,b\}^*$ is total recursive
$x \in R$ iff M_R halts on x
$\Rightarrow$ $M_{f(x)}$ acts like an L_1 acceptor
$\Rightarrow$ $\text{domain}(M_{f(x)}) = L_1$
$\Rightarrow$ $f(x) \in K_P$ since $P(L_1)$ holds.

$x \notin R$ iff M_R doesn't halt on x
$\Rightarrow$ $\text{domain}(M_{f(x)}) = \emptyset$
$\Rightarrow$ $f(x) \notin K_P$ since $\neg P(\emptyset)$.

Hence

$x \in R$ iff $f(x) \in K_P$.

i.e. $R \leq_m K_P$. ☒

Discussion of HW #2 problem 3(e): Moral: don't assume an object is complex because you are given a complex description of it.

We accept that we are not intuitionists: $A \vee \bar{A}$ = true. A proof of existence by cases as above is valid.

THUE SYSTEM DERIVABILITY PROBLEM
(Manna § 1-5)

<u>Def</u>: Semi Thue system: An alphabet Σ, a finite set P of <u>rewrite</u> <u>rules</u>, $P \subseteq \Sigma^* \times \Sigma^*$.

If $x, y \in \Sigma^*$, define $x \xrightarrow[P]{*} y$.

Ex: $(ab, ba), (ba, ab) \in P$
written $ab \to ba$, $ba \to ab$
then $aaba \xrightarrow[P]{*} baaa$

<u>Theorem</u>: There is a semi-Thue system (Σ, P) such that
$\{(x, y) \mid x \xrightarrow[P]{*} y\}$
is not recursive. (Note it is obviously r.e.)

We will go beyond Manna and remove the "semi". In a Thue system we have
$(x, y) \in P \Rightarrow (y, x) \in P$
i.e. it is a two-way rewrite system.
Then there is an undecidable Thue system.

<u>Theorem</u>: Can't understand simple algebraic expressions about an associative operation (Corollary of above), (semi-group).

6.044

10/05/87 Lecture # 11

We have lots of examples of r.e. non recursive sets.
So we have examples of r.e., non co-r.e. sets.
But

Theorem: There exist languages which are neither r.e. nor co-r.e.

Proof - There are more languages than r.e. or co-r.e. languages ∎

$|\mathcal{P}(\{a,b\}^*)| > |\{a,b\}^*|$, by Cantor's theorem.

But $|\{a,b\}^*| = |(\text{r.e. sets}) \cup (\text{co-r.e. sets})|$. Why?

Good onto mapping from $\{a,b\}^* \to$ r.e. sets is $d(M) \mapsto \text{domain}(M)$. Similarly for co-r.e. sets. Now get an onto mapping into the union by mapping strings which start with a as

$$a\,d(M) \mapsto \text{domain}(M)$$

and strings which start with b as

$$b\,d(M) \mapsto \overline{\text{domain}(M)}.$$

Actually the right way to see this is: union of two countable sets are countable.

Proof above does generate a non r.e., co-r.e set through Cantor's diagonalization. But for a more concrete example of such a set, we have $K_2 \times \overline{K_2}$.

Claim: $K_2 \times \overline{K_2}$ is neither r.e. nor co-r.e.

Proof: $\overline{K_2} \le_m K_2 \times \overline{K_2}$ through

$$f: M \mapsto (\text{start} \to \text{halt}, M),$$

so $K_2 \times \overline{K_2}$ is not r.e. (because non-r.e. inherits up).

Likewise $K_2 \le_m K_2 \times \overline{K_2}$ through

$$f: M \mapsto (M, \text{start} \to \text{right}).$$

so $\overline{K_2} \le_m \overline{K_2 \times \overline{K_2}}$ and therefore $\overline{K_2 \times \overline{K_2}}$ is not r.e. so $K_2 \times \overline{K_2}$ is not co-r.e. ∎

<u>Theorem</u>: The acceptance problem for Post-machines $\le_m$ derivability problem for semi-Thue systems.

Proof -

Given a Post machine M and input $w \in \{a, b\}^*$

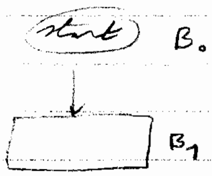
$$(M, w) \longmapsto (S, (B_0 \vdash w \dashv, \Lambda))$$

where $S = (\Sigma_S, P)$ is a semi-Thue system, B_0, $\vdash$, $\dashv$ are symbols in $\Sigma_S - \Sigma_M$. How do we build S? Remember the mapping must be a translation (i.e. computable).

We want

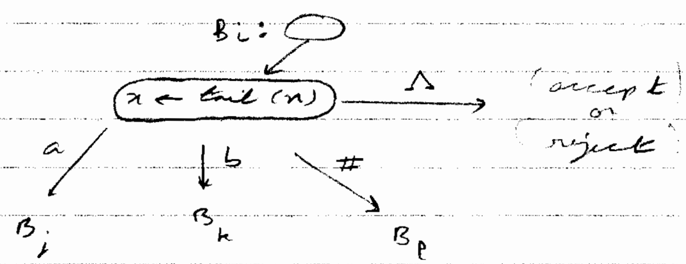
$$M \text{ halts on } w \text{ iff } B_0 \vdash w \dashv \xrightarrow[S]{*} \Lambda .$$

Assign names $B_0, B_1, \ldots$ to the boxes in the PM flowchart, with B_0 = (start), and



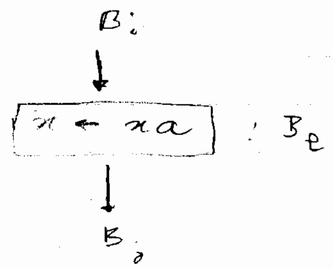
(call this B_1).

Put $B_0 \to B_1$ in P_S. Now corresponding to



we have rewrite rules

$$B_i \vdash a \longrightarrow B_j \vdash$$
$$B_i \vdash b \longrightarrow B_k \vdash$$
$$B_i \vdash \# \longrightarrow B_\ell \vdash$$
$$B_i \vdash \dashv \longrightarrow \Lambda$$



$$\dashv B_i \longrightarrow a \dashv B_j$$

Add rules

$$\sigma B_i \leftrightarrow B_i \sigma$$

for all B_i, $\sigma \in \{a, b, \#, \vdash, \dashv\}$

It is now "clear" (Manna) that this works.

The big idea here is the setting up of the reduction, not the programming details. ∎

Now would like to use the same proof for Thue systems. Problem: there are more derivations. For example if M accepts w_1 and w_2 then $w_1 \overset{*}{\leftrightarrow} w_2$

<u>Lemma</u>: Construction still works for Thue systems.

Why? Looking at words of the form

$\vdash w' \dashv$ (one B somewhere)

(call such words ok), we have

$ok_1 \overset{*}{\longleftrightarrow} ok_2$ iff $\exists ok_3$ s.t. $ok_1 \overset{*}{\rightarrow} ok_3 \overset{*}{\leftarrow} ok_2$

Suppose $B_0 \vdash w \dashv \longrightarrow B_i \vdash \dashv$ (both are ok)

6.044

10/07/86 Lecture # 12

Practice problem:

$A = \{ M \mid M$ diverges on input Λ and writes an infinite

(1) $K_2 \leq_m A$:

"given M, simulate M on Λ using a new symbol c instead of a. If M halts, diverge while printing a's."

(2) $\overline{K_2} \leq_m A$:

"given M, simulate M on Λ printing an "a" in between each step".

Theorem: The Thue-system word problem is undecidable.

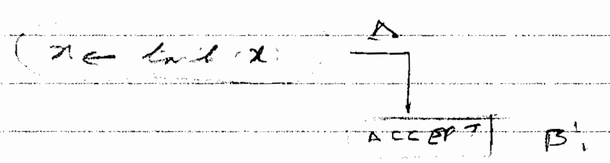
Proof - Reduce $PM_0 \leq_m$ Thue using Manna's reduction for semi-Thue, except:

(PM_0 = Post machine halting problem).

don't use the reduction rule

$$B_i \vdash\!\dashv \longrightarrow \Lambda$$

which came from



This was a troublesome rule in going backwards, because Λ could be replaced by $B_i \vdash\!\dashv$ anywhere and we would get all kinds of garbage.

Thus by Manna,

$$B_0 \vdash x \dashv \xrightarrow{*} B_i \vdash\!\dashv \quad \text{iff PM accepts } x.$$

(assume PM has only one ACCEPT Box B_i)

For Thue system,

Claim: $B_0 \vdash x \dashv \overset{*}{\leftrightarrow} B_i \vdash \dashv$ iff PM accepts x.

(Note Claim ⇒ QED theorem)

Proof sketch for Claim:

An OK word over the Thue alphabet is a word of the form

$\vdash W \dashv$, $W \in \{a, b, \#\}^*$

exactly one B somewhere.

Remark 1: If w is OK and $w_1 \to w$ or $w \to w_2$ (1 step) then w_1, w_2 are OK.

Proof. Look at rules. For example

rule: $B_i \vdash a \leftrightarrow B_j \vdash a$

clearly preserves OK-ness in either direction. ∎

Remark 1 says we don't have to worry about non-OK words. If w is OK and $w \overset{*}{\leftrightarrow} w'$ then by Remark 1 w' is OK.

Subclaim: If w is OK and $w \overset{*}{\leftrightarrow} w'$ then $\exists w_0$ s.t. $w \overset{*}{\to} w_0 \overset{*}{\leftarrow} w'$.

Proof - Look at the shortest derivation $w \overset{*}{\leftrightarrow} w'$. Then it looks like $w \overset{*}{\to} w_0 \overset{*}{\leftarrow} w'$, for if not we could get a shorter derivation $w \overset{*}{\leftrightarrow} w'$. ∎

Now if PM accepts x then certainly

$B \vdash x \dashv \overset{*}{\to} B_i \vdash \dashv$

so $B \vdash x \dashv \overset{*}{\leftrightarrow} B_i \vdash \dashv$. Conversely suppose $B \vdash x \dashv \overset{*}{\leftrightarrow} B_i \vdash \dashv$. Then by Subclaim

$B_0 \vdash x \dashv \overset{*}{\longrightarrow} z \overset{*}{\longleftarrow} B_i \vdash \dashv$.

But no rewrite rule except commutativity applies to $B_i \vdash \dashv$. So z looks like $B_i \vdash \dashv$ up to commutativity. But commutativity rules are two way. We conclude

$B_0 \vdash x \dashv \overset{*}{\longrightarrow} B_i \vdash \dashv$

so by Manna PM halts. ∎

Next time: §1-5.4: Post Correspondence & Matrix Mortality.

6.044

10/09/87 Lecture # 13

Quiz handed back.
Minor typo: $\not\to^*_S$ in problem 3 was supposed to be $\to^*_S$, which is trivial.

More sample problems for practice were requested.

POST CORRESPONDENCE

A Post Correspondence problem is a finite set of pairs of words over some finite alphabet Σ.

A solution to a PCP $\{(\alpha_1,\beta_1),\ldots,(\alpha_k,\beta_k)\}$ is a finite sequence of integers $i_1, i_2, \ldots, i_n$ $(n \geq 1)$ $(1 \leq i_j \leq k)$ such that

$$\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_n} = \beta_{i_1}\cdots\beta_{i_n}.$$

Sometimes this is written

$$\binom{\alpha_1}{\beta_1}, \ldots, \binom{\alpha_k}{\beta_k}.$$

want to concatenate so that top and bottom rows are identical.

Example (Manna): $\Sigma = \{a, b\}$,
$S = \{(b, b^3), (bab^3, ba), (ba, a)\}$
solution

babbb b b ba
ba bbb bbb a } equal.

Theorem: PCP = {S | S has a solution} is a $\leq_m$-complete r.e. set.

Example (Manna):

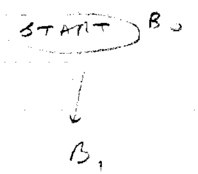
$\{(ab, abb), (b, ba), (b, bb)\}$

has no solution: lower words are always longer than upper words.

See Problem 1-20 (Manna).

Proof of the above theorem reduces HP (Post machine halting problem) to PCP.

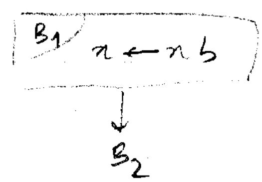
Input $z \in \{a, b, \#\}^*$ to PM.



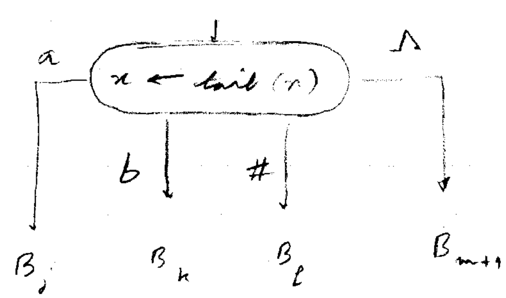
$(B_0, B_0 z B_1)$

Manna's asterisks (*) force you to begin with above pair: all other pairs have one side beginning with *, other side not. We will impose that we have to start with above rule.

Cycling rules: (σ, σ) for all $\sigma \in \{a, b, \#\}$.



$(B_1, b B_2)$



$(B_i a, B_j)$

$(B_i b, B_k)$

$(B_i B_{m+1}, B_{m+1})$

Matrix (i,j) - mortality : (Floyd)

PCP $\leq_m$ Matrix (3,2) mortality.

Instance : $M_1, \ldots, M_k$ finite set of 3×3 integer matrices.

Question : Is there some sequence $i_1, \ldots, i_n$ $(n \geq 1)$, $(1 \leq i_j \leq k)$ s.t. $M_{i_1} \cdots M_{i_n}$ has a 0 in its (3,2) position?

Given $P : \{(\alpha_1, \beta_1), \ldots, (\alpha_k, \beta_k)\}$ map it into $\{M(\alpha_1, \beta_1), \ldots, M(\alpha_k, \beta_k)\}$.

From p. 64 of Manna,

$u, v \in \Sigma_P^*$ = alphabet of PCP has n letters &

$$M(u,v) = \begin{bmatrix} n^{|u|} & n^{|v|} - n^{|u|} & 0 \\ 0 & n^{|v|} & 0 \\ u & v-u & 1 \end{bmatrix}.$$

if Σ_P has three letters, think of them as digits 1, 2, 3. Then interpret $u \in \Sigma_P^*$ as base 3 number. Note: 1, 2, 3, <u>not</u> 0, 1, 2.

$n^{|u|}$ is in base n using digits 0, .., n-1.

Properties:

(1) $u = v$ iff $M(u,v)_{(3,2)} = 0$ [Clearly]

(2) $M(u_1, v_1) \cdot M(u_2, v_2) = M(u_1 u_2, v_1 v_2)$.

6.044
10/14/87

Lecture # 14

THUE SYSTEMS AND SEMIGROUPS

serves as basis (case study) for some ideas that come up in Logic.

Def: A semigroup is an algebra with a single binary associative operation. That is, a semi-group is a pair $(S, *)$ where $S \neq \emptyset$ is a set whose members are called elements of the semigroup, and $* : S \times S \to S$ is such that

$$s_1 * (s_2 * s_3) = (s_1 * s_2) * s_3$$

for all $s_1, s_2, s_3 \in S$.

Remark: A semigroup with an identity element is called a monoid.

Def. $e \in S$ is an identity element iff $e * s = s * e = s$ for all $s \in S$.

Lemma: There is exactly one identity element in a monoid.

Proof - say e_1, e_2 are identity elements. Then

$e_1 = e_1 * e_2$ since e_2 is an identity element.

$e_2 = e_1 * e_2$ since e_1 is an identity element

so $e_1 = e_2$. ▨

Example 1: $(\{a,b\}^*, \cdot)$ ($\cdot$ is the operation of concatenation) is a monoid with Λ as identity element.

A <u>semigroup term</u> over alphabet Σ is simply a word in $\Sigma^+ = \Sigma^* - \{\Lambda\}$. A <u>monoid term</u> over Σ is a word in Σ^*.

<u>Example 2:</u> $(\{true, false\}, \wedge)$

$(\{true, false\}, \oplus)$ (mod 2 sum)

$(\{true, false\}, \equiv)$ (equivalence)

$(x \equiv y) \equiv z = x \equiv (y \equiv z)$ $(x + x = false$, $x + \neg x = true)$

because

$(x \equiv y) = (x \oplus y) \oplus true$

and replacing $\equiv$ by this will prove the associativity.

$(\{true, false\}, \vee)$.

Elementary questions we could ask ourselves? How many two element semigroups are there? How many of the above are distinct?

Formally, a pair $((S, *), \mathcal{I})$)

A <u>semigroup interpretation</u> over alphabet (signature) Σ consists of a semigroup $(S, *)$ and a mapping (sometimes itself called the interpretation) $\mathcal{I}: \Sigma \to S$. We now extend $\mathcal{I}$ to a mapping from $\Sigma^+ \to S$ (calling the extension $\mathcal{I}$ by an abuse of notation) by induction on the length of semigroup terms as follows:

$$\mathcal{I}(\sigma x) = \mathcal{I}(\sigma) * \mathcal{I}(x) \qquad (x \in \Sigma^*)$$

<u>Example 3</u>: $\Sigma = \{a, b\}$, $S = (\{true, false\}, \oplus)$.

Let

$\mathcal{I}(a) = true$, $\mathcal{I}(b) = true$

Then

$\mathcal{I}(abaab) = true$

The meaning of a word of odd length is true, and the meaning of a word of even length is false. If we wanted to extend to monoids we would set $\mathcal{I}(\Lambda) = false$. (Pure term model; free interpretation).

Example 4: $\Sigma = \{a, b\}$, $S = (\Sigma^*, \cdot)$

Let $\vartheta(\sigma) = \sigma$ (meaning of symbol σ is the word of length 1 which consists of the symbol σ).

so $\vartheta(z) = z$ for all $z \in \Sigma^+$.

The very syntax we use to talk about mathematical stuff can be used to assign meaning.

Given an interpretation can talk about meaning of a term. Write

$$\vartheta \models x \approx z \qquad (x, y \in \Sigma^+)$$

call $x \approx z$ an equation. It is a syntactic object: Two terms over our signature separated by a $\approx$ sign.

Say

$$\vartheta \models x \approx z \quad \text{iff} \quad \underbrace{\vartheta(x) = \vartheta(z)}$$

mathematical English word (we know what it means)

Def: A set of equations $\{E_i\}$ (possibly infinite) implies another equation F iff for any ϑ s.t. $\vartheta \models E_i$ for all i, it is also the case that $\vartheta \models F$.

Write $\{E_i\} \models F$. (semantic implications)

Examples:

$$\{E_i\} = \{\overbrace{aa \approx a}^{E_1}, \overbrace{bb \approx b}^{E_2}, \overbrace{ab \approx ba}^{E_3}\}$$

$$\models \underbrace{aaaaba = abb}_{F}$$

For example let $S = (\{\text{true}, \text{false}\}, \wedge)$ and $\vartheta(a) = \text{true}$, $\vartheta(b) = \text{false}$. $\vartheta \models \{E_1, E_2, E_3\}$.

and we see that $\vartheta \models F$.

To prove that the implication holds we need to look at all possible interpretations.

Under the term model interpretation the hypotheses are not true, so the implication is vacuously true.

Note: If we let P be the rewrite rules

$P = \{ aa \leftrightarrow a,\ bb \leftrightarrow b,\ ab \leftrightarrow ba \}$

then it is easy to see that

$$aaaaba \xleftrightarrow[P]{} abb$$

Check:

$$abb \xrightarrow[P]{} ab$$

$$aaaaba \xrightarrow[P]{} aaaba \xrightarrow[P]{} aaba \xrightarrow[P]{} aba \xrightarrow[P]{} aab \xrightarrow[P]{} ab$$

Notation:

$\{E_1, E_2, \ldots\} \vdash F$ iff the two sides of F rewrite to each other under the Thue system whose rules are $\{E_1, E_2, \ldots\}$ (a Thue system with an infinite number of rules)

This is new notation for a familiar idea. $E_1, E_2, \ldots$ are axioms. If F is $x \approx y$ then we say $x \approx y$ is provable from axioms $\{E_i\}$ if can get from x to y with $\{E_i\}$ as rewrite rules.

__Theorem:__ $\vdash$ iff $\models$. Namely,

$\{E_i\} \vdash F$ iff $\{E_i\} \models F$.

(Completeness theorem)

$\{E_i\} \vdash F$ is purely calculational (symbol pushing)

$\{E_i\} \models F$ is a mathematical fact.

$\Rightarrow$: reasoning is valid.

6.044
10/16/87

Lecture # 15

Theorem: $\{E_i\} \models F$ iff $\{E_i\} \vdash F$.

Easy direction: $\{E_i\} \vdash F \Rightarrow \{E_i\} \models F$.
Prove by induction on length of rewriting of left hand side of F to right hand side.
By hypothesis left side of F rewrites to right side using rules $\{E_i\}$. Induct on number of steps the rewriting took.

Base Case: 0 steps.
so F must be of the form $x = x$ for $x \in \Sigma^*$.
[Note: if $x_1 \leftrightarrow x_2 \leftrightarrow \cdots \leftrightarrow x_n$ the number of steps is $n-1$].
so $\{E_i\} \models x = x$ because $\mathcal{I} \models x = x$ for all $\mathcal{I}$.

Induction Case: say F is of form $x_1 = x_{n+1}$ and $x_1 \leftrightarrow x_2 \leftrightarrow \cdots \leftrightarrow x_n \leftrightarrow x_{n+1}$ where $n \geq 1$.
so certainly $\{E_i\} \vdash x_1 = x_n$ (the proof is the chain $x_1 \leftrightarrow \cdots \leftrightarrow x_n$). By the induction hypothesis $\{E_i\} \models x_1 = x_n$. Also $x_n \leftrightarrow x_{n+1}$ (i.e. in one step). So $x_n = u e_1 v$ and $x_{n+1} = u e_2 v$ for some $u, v \in \Sigma^*$ and equation $e_1 = e_2$ (or $e_2 = e_1$) $\in \{E_i\}$.

Claim: $\{E_i\} \models x_n = x_{n+1}$
Proof - $\mathcal{I}(x_n) = \mathcal{I}(u) * \mathcal{I}(e_1) * \mathcal{I}(v)$ [exercise: $\mathcal{I}(xy) = \mathcal{I}(x) * \mathcal{I}(y)$; use induction on length of x to prove it for all y]. And $\mathcal{I}(x_{n+1}) = \mathcal{I}(u) * \mathcal{I}(e_2) * \mathcal{I}(v)$. By hypothesis $\mathcal{I} \models \{E_i\}$ so in particular $\mathcal{I} \models e_1 = e_2$, i.e. $\mathcal{I}(e_1) = \mathcal{I}(e_2)$. So $\mathcal{I}(x_n) = \mathcal{I}(x_{n+1})$. ☒

At this stage we have $\{E_i\} \models x_1 = x_n$ and

$\{E_i\} \models x_n = x_{n+1}$. so $\{E_i\} \models x_1 = x_{n+1}$.
This ends this direction of proof.

Let's go the other way. Why are the rules powerful enough to capture all the possible situations, i.e. all the $\mathcal{I}$ s.t. $\mathcal{I} \models \{E_i\}$? Mathematician's trick: suppose $\{E_i\} \not\vdash F$ and find a countermodel. We will build $\mathcal{I}_0$ s.t. $\mathcal{I}_0 \models \{E_i\}$ but $\mathcal{I}_0$(left hand side of F) $\neq$ $\mathcal{I}_0$(right hand side of F). We will build $\mathcal{I}_0$ s.t.

$$\mathcal{I}_0 \models x = y \text{ iff } \{E_i\} \vdash x = y$$

Then $\mathcal{I}_0$ is our countermodel. Clearly $\mathcal{I}_0 \models \{E_i\}$ because $\{E_i\} \vdash E$ for all $E \in \{E_i\}$. But if $\{E_i\} \not\vdash x = y$ then $\mathcal{I}_0 \not\models x = y$ (where F is $x = y$).

Definition of $\mathcal{I}_0$:

__Def:__ $[x] = \{y \mid x \overset{*}{\leftrightarrow} y\}$, for $x \in \Sigma^+$.
where the rewriting is with respect to rules $\{E_i\}$.
The set $\{[x] \mid x \in \Sigma^*\}$ is a partition of Σ^*,
($\overset{*}{\leftrightarrow}$ is an equivalence relation)
Elements of the domain of our semigroup:

$$\{[x] \mid x \in \Sigma^*\}.$$

Define $[x] * [y] = [xy]$. We must check that the operation is well defined, i.e.
$[x] = [x']$ and $[y] = [y'] \Rightarrow [xy] = [x'y']$. obvious by rules of rewriting.
The operation $*$ is clearly associative:

$$[x] * ([y] * [z]) = [x] * [yz] = [xyz]$$
$$= [xy] * [z] = ([x] * [y]) * [z] .$$

Now define

$$\mathcal{I}_0(\sigma) = [\sigma]$$

__Exercise__: It follows that $\mathcal{I}_0(x) = [x]$, by induction on length of x.

Now

<u>Lemma:</u> $\mathcal{J}_0 \models x = y$ iff $\{E_i\} \vdash x = y$.

Proof -

$\mathcal{J}_0 \models x = y$ iff $\mathcal{J}_0(x) = \mathcal{J}_0(y)$ but

$\mathcal{J}_0(x) = [x] = \{z \mid x \overset{*}{\leftrightarrow} z\}$,

$\mathcal{J}_0(y) = [y] = \{z \mid y \overset{*}{\leftrightarrow} z\}$.

In particular $[x] = [y] \iff x \sim y \iff x \overset{*}{\leftrightarrow} y$. ◻

Top level summary: For $\{E_i\} \vdash F \Rightarrow \{E_i\} \models F$ use induction on length of proof to show that rewriting preserves truth. The converse direction is proved by looking at the contrapositive: build a "term model" as a countermodel.

<u>Corollary:</u> We know the problem $\{E_i\} \vdash F$? is undecidable (finite $\{E_i\}$); this is the undecidability of Thue systems. Our theorem shows that $\{E_i\} \models F$? is undecidable.

Word problem of groups is undecidable: Novikoff/Boone. Mathematicians thought this was the first "real" undecidable problem.

We could have more complicated formulas, like

$\mathcal{J} \models \exists z[(x \neq y) \wedge (x = z)]$

we start this next time. But we've already shown that a simple logic has an undecidable validity problem which will imply the same for more general logics.

6.044 Lecture # 16
10/19/87

Formal logic Syntax and semantics.

System to formalize reasoning.

signature : set of symbols (the nonlogical symbols).
Classified into 4 kinds :
- function symbols f, g
  - constant (arity = 0) c
  - arity > 0 f

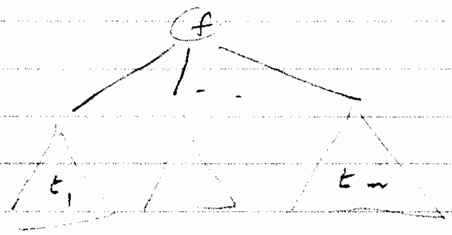
- predicate symbols
  - constants (arity = 0) P :
  - arity > 0

(each function / predicate symbol has an arity).

Variable symbols :
- first order : x, y, z, ...
- second order : function variables and predicate variables.

terms :
(1) x is a term (for x a first order variable symbol)
(2) c is a term (c is a constant symbol)
(3) If f is n-ary and $t_1, \dots, t_n$ are terms then $f(t_1, \dots, t_n)$ is a term.
↑↑ ↑ ↑ ↑ syntax.

We can actually talk about the parse tree:
if $t_1, \ldots, t_n$ are nodes of parse trees then



is a term.

(3.5) If F is n-ary, $F(t_1, \ldots, t_n)$ is a term

(4) if A then t_1 else t_2 fi is a term
if t_1, t_2 are terms and A is a wff.

On the first pass forget (3.5) and (4). It is a theorem that these are redundant (sugar).

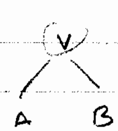
atf: (atomic formulas)

(a) If P is n-ary and $t_1, \ldots, t_n$ are terms then $P(t_1, \ldots, t_n)$ is a atomic formula.
(b) similarly for predicate variables P.
(c) T and F are atomic formulas.
(zero-ary predicate constant symbols)
Note: there is no inductive definition here.
(a'): $t_1 = t_2$ for terms t_1, t_2.

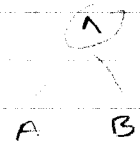
Are defining a parser, i.e. trees that are well formed.

wff: formula (well formed formula)

(1) atomic formulas are wff's.
(2) If A, B, C are wffs then so are



$(A \vee B)$

 : $(A \wedge B)$

$(\neg A)$ 

$A \supset B$, $A \equiv B$,
IF A THEN B ELSE C .

c) $\exists F.A$, $\exists P.A$, $\forall F A$, $\forall P.A$
↓ function variables ↓ predicate variables.

$\exists x . A$ is the special case of F being 0-ary.

First order formulas:
Special case when there are no predicate variables and only 0-ary function variables occur.

Interpretations: Say what every symbol means.

Logical interpretation (interpretation of the predicate calculus): $(D, \mathcal{I}_c, \mathcal{I}_v) = \mathcal{I}$.
$D \neq \emptyset$ is the domain whose members are elements.
$\mathcal{I}_v$: (valuation) says what the variables mean, $\mathcal{I}_v$: variable symbols → elements, functions and predicates on D, of appropriate kind and arity.
$\mathcal{I}_c$: says what the constants mean.

Define for terms t define $t_{\mathcal{I}} \in D$ inductively:
(1) $(x)_{\mathcal{I}} = \mathcal{I}_v(x)$

Example: $D = \mathbb{Z}$.
$\mathcal{I}_v(x_3) = -7$
$\mathcal{I}_v(F) =$ addition
(arity of $F = 2$)

$\mathcal{I}_v(P)$ = positive and odd.

P is a unary (arity 1) predicate variables.

$\mathcal{I}_c$ is similar on constants.

Call the above $\mathcal{I}_{today}$.

$$\mathcal{I}_{today}(F(x_3, F(x_3, x_3))) = -21.$$

$$(c)_g = \mathcal{I}_c(c)$$

$$(f(t_1, \ldots, t_n))_g = \mathcal{I}_c(f)((t_1)_g, \ldots, (t_n)_g)$$

We now define $\mathcal{I} \models A$ for a wff A, inductively as follows:

atomic formulas:

$\mathcal{I} \models P(t_1, t_2)$ iff $\underbrace{(\mathcal{I}_c(P))((t_1)_g, (t_2)_g)}_{\text{truth value}}$.

$\mathcal{I} \models \top$ is true, $\mathcal{I} \models F$ is false.

6.044
10/21/87 Lecture # 17

Define $\mathcal{I} \models A$ ($\mathcal{I} = (D, \mathcal{I}_c, \mathcal{I}_v)$)

A is of the form $\forall x.\, B$
$\mathcal{I} \models A$ iff $\mathcal{I}[x \mapsto d] \models B$ for all $d \in D_\mathcal{I}$

Def: $\mathcal{I}[x \mapsto d] = (D, \mathcal{I}_c, \mathcal{I}_v[x \mapsto d])$

Def: $f: S_1 \to S_2$, $s_1 \in S_1$, $s_2 \in S_2$. Then
$f[s_1 \mapsto s_2] : S_1 \to S_2$ is defined by

$$f[s_1 \mapsto s_2](s) = \begin{cases} f(s) & \text{if } s \neq s_1 \\ s_2 & \text{if } s = s_1 \end{cases}$$

think of this as the result of the assignment statement $f[s_1] := s_2$.

Note: we are not explaining what "for all" means; we are just putting the symbol $\forall$ into English.

Similarly $\mathcal{I} \models \exists x.B$ iff $\mathcal{I}[x \mapsto d] \models D$ for some $d \in D_\mathcal{I}$.

Similarly for other types of variables (function variables).

We re-package the definition (Manna's) of an interpretation as

$$\mathcal{I} = (\underbrace{(D, \mathcal{I}_c)}_{m}, \mathcal{I}_v)$$

$m = (D, \mathcal{I}_c)$ is called a model and $\mathcal{I}_v$ is called a valuation. While

$$m \models A \quad (A \text{ is valid in } m)$$

means $\mathcal{I} \models A$ for all $\mathcal{I} = (m, \mathcal{I}_v)$.

Remark If $I = (M, I_v) \models A$ and A is closed then $M \models A$. (A is closed means A has no free variables).

Pre-remarks

(1) If x is not free in A then:

$$I \models A \text{ iff } I[x \mapsto d] \models A$$

for any I and any $d \in D_I$.

This implies the above remark.

Lemma: $M \models A$ iff $M \models \underbrace{\forall \vec{x}\, A}_{\text{universal closure of } A}$

For example, over $M = (\mathbb{Z}, I_c)$ where:

$$I_c : \begin{cases} + \longmapsto \text{addition} \\ \cdot \longmapsto \text{multiplication} \\ 0 \longmapsto 0 \\ 1 \longmapsto 1 \\ = \longmapsto = \end{cases}$$

Then

$$\models \forall x \forall y\, (x+y = y+x)$$

iff $(\mathbb{Z}, I_c) \models x+y = y+x$

Def: $\models A$ (A is valid) means $M \models A$ for all M (of the appropriate signature).

Fact: $\models \overbrace{\forall x.\, p(x)}^{A} \supset \overbrace{\exists x.\, p(x)}^{B}$

Proof -

Must show that $I \models A \supset B$ for any I.

This holds if $I \not\models A$ by definition of satisfaction for $\supset$. So assume $I \models A$. Must show $I \models B$. It is sufficient to show that

$$I[x \mapsto d] \models p(x)$$

for some $d \in D_I$. But $I \models \forall x.\, p(x)$, so for all $d' \in D_I$ $I[x \mapsto d'] \models p(x)$.

But $D_g \neq \emptyset$. So choose $d \in D_g$ and conclude that $g[x \mapsto d] \models p(x)$ ▨

$$\models \forall x . A \supset A^t_x$$

syntactic substitution operator. Replace all <u>free</u> occurrences of x in A by t.

This could be written, more mnemonically,

$$\forall x\, A(x) \supset A(t)$$

Actually let's just claim that:

Let p be a unary predicate constant. Then

$$\models \forall x\, p(x) \supset p(t)$$

for any term t. Here $\forall x\, p(x) \supset p(t)$ defines a class of formulas, one for each t. Could also write

$$\models \underbrace{\forall x\, p(x) \supset p(y)}_{\text{a formula}}$$

To show that $\not\models \exists x\, p(x) \supset \forall x\, p(x)$ pick D s.t. $|D| = 2$ and P s.t. $P(0)$, $\neg P(1)$. Then $g = (D, I_c)$ where $I_c(p) = P$ doesn't satisfy the formula.

6.044
10/23/1987

Lecture # 18

Undecidability of the validities of predicate calculus: Manna's proof (Reduction from Post Correspondance)

6.044
10/26/1987

Lecture # 19

Expressive power of the second order predicate calculus

Theorem 1: There is a sentence (closed wff) $F_{\mathbb{N}}$ of the second order predicate calculus, over the signature $\{ 0, suc \}$ such that (suc is a function symbol of arity 1)

$M \models F_{\mathbb{N}}$ iff M is isomorphic to $\langle \mathbb{N}, 0, S \rangle$ where $S : \mathbb{N} \to \mathbb{N}$ is the successor.

Thus

$F_{\mathbb{N}} \supset G$ is valid iff $\langle \mathbb{N}, 0, S \rangle \models G$. (Note G may have free variables whereas $F_{\mathbb{N}}$ doesn't)

When the language is powerful enough to characterize a model, reasoning about all models is at least as hard as reasoning about a particular model.

Theorem 2: The above theorem doesn't hold if we replace second order by first order. One reason: the existence of non-standard models of arithmetic.

Theorem 3: Theorem 1 can be stated in more generality. For example we can find $F_{\{a,b\}^*}$ characterizing $\langle \{a,b\}^*, a, b, \cdot \rangle$.

Non-axiomatizability:

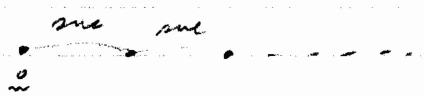
Theorem 4: The first order wffs valid in $\langle \mathbb{N}, 0, 1, S, +, * \rangle$ (or $\langle \{a,b\}^*, a, b, \cdot \rangle$) are not r.e. (Gödel's first incompleteness theorem).

Theorem: (Godel's Completeness theorem) the valid formulas of first order predicate calculus are r.e.

Corollary: The valid formulas of the second order predicate calculus are not r.e.

Godel's second incompleteness: the only theory that can prove its own consistency is one that is inconsistent.

Proving theorem 1: we want our models to look like



(1) $\forall x \forall y\, [suc(x) = suc(y) \supset (x = y)]$
(successor function is 1-1; geometrically only one arrow into each point)

(2) $\forall x\, [0 \neq suc(x)]$ (no arrow enters 0)

(3) $\forall P\, [P(0) \wedge \forall x\, (P(x) \supset P(Suc(x)))] \supset \forall z\, P(z)$

P: unary predicate variable
to see that (3) implies model is just the chain, interpret P as the chain.

6.044
10/28/87

Lecture # 20

Last time: we wrote down $F_{\mathbb{N}}$

This time we want to write down F_{arith} s.t.
$\mathcal{M} \models F_{arith}$ iff $\mathcal{M} \cong \langle \mathbb{N}, 0, 1, Succ, +, * \rangle$

F_{arith} : $F_{\mathbb{N}} \wedge \forall x \forall y [(x \pm 0 = x)$
$\wedge\ x + succ(y) = succ(x+y)]$
$\wedge \forall x \forall y [x * 0 = 0 \wedge x * (succ(y)) = (x * y) + x]$

Obviously $\langle \mathbb{N}, 0, succ, +, * \rangle \models F_{arith}$.

(Note: the precise definition of isomorphism will be a homework problem)

Conversely suppose $\mathcal{M} \models F_{arith}$. So $\mathcal{M} \models F_{\mathbb{N}}$, so domain$(\mathcal{M}) \cong \mathbb{N}$ and 0 and succ work right.
Must show that
$\mathcal{M}(\pm) =$ addition
$\mathcal{M}(*) =$ multiplication.

Sub proof: by induction on m a non negative integer prove
$n\ \mathcal{M}(\pm)\ m = n + m$ (real mathematical addition)

i.e. prove it for all n inducting on m.
If $m = 0$ then for any n, OK because of formula $x + 0 = x$.
Induction case easy too.

Interesting exercise: find a model of the first order part of F_{arith} which is not isomorphic to the standard model.

Gödel numbering: assign numbers to all possible Turing machines. TMs are strings over a countable alphabet.

Similarly, there is a formula F_{Σ^*} (Homework problem, HW #6) s.t.

$$\mathfrak{M} \models F_{\Sigma^*} \text{ iff } \mathfrak{M} \cong \langle \Sigma^*, \cdot, \mathrm{EqLen} \rangle$$

where $\cdot$ is concatenation

EqLen is a binary predicate s.t.

$$\mathrm{EqLen}(x, y) \text{ iff } |x| = |y|.$$

<u>Theorem</u>: 1-st Th(Σ^*) = { F / F is a first order wff and $\langle \Sigma^*, \cdot, \mathrm{EqLen} \rangle \models F$ } is not r.e.

(version of incompleteness theorem).

Proof sketch: Let M be an arbitrary Turing machine and $w \in \Sigma_M^*$ an input word for M. Write a first order formula $F_{M,w}$ with signature $\Sigma, \cdot, \mathrm{EqLen}$ s.t.

$$\langle \Sigma^*, \cdot, \mathrm{EqLen} \rangle \models F_{M,w} \text{ iff } M \text{ halts on input } w.$$

Then

$$K_0 \leq_m \bigcup_{\Sigma} (\text{1-st Th}(\Sigma^*))$$

But 1-st Th(Σ^*) $\leq_m$ $\overline{\text{1-st Th}(\Sigma^*)}$ (reduction: $F \mapsto \neg F$:

$F \in$ 1st Th(Σ^*) iff $\langle \Sigma^*, \cdot, \ldots \rangle \models F$

iff $\langle \Sigma^*, \ldots \rangle \not\models \neg F$ (because F is closed)

iff $\neg F \notin$ 1-st Th(Σ^*)

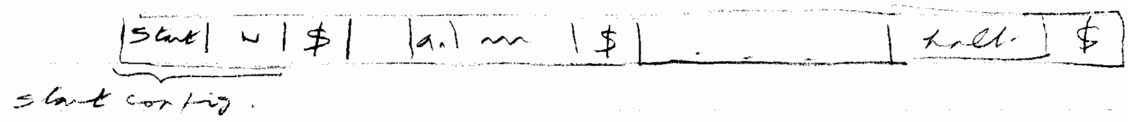
iff $\neg F \in$ $\overline{\text{1st Th}(\Sigma^*)}$.

So 1st Th(Σ^*) is not r.e.

We now write $F_{M,w}$:

$\exists z$ [z is the computation word of M's halting computation on input w].

computation word:



state symbol to the left of square head is reading.

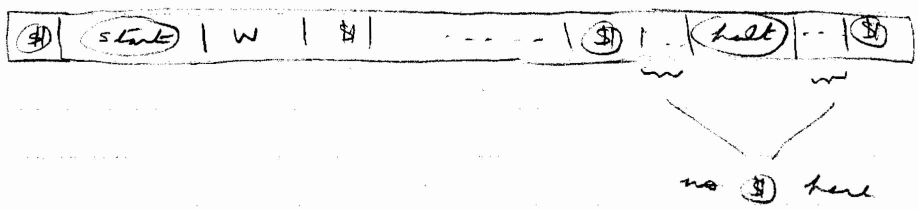
"z is a computation word": $\exists z' [z = \text{(start)} \cdot w \cdot \$ \cdot z']$ $\wedge \exists z'' \exists z_1 \exists z_2 [z = z'' \cdot \$ \cdot z_1 \cdot \text{(halt)} \cdot z_2 \cdot \$$ $\wedge$ "\$" does not occur in z_1, z_2]

6.044
10/30/87

Lecture # 21

$F_{M,w}$: so far we know how to say:

$\exists z.$ z looks like:

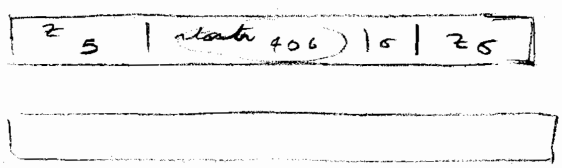


Now want

"consecutive words delimited by $ are descriptions of consecutive configurations of a computation of M".

$\forall z_1 z_2 z_3 z_4 . \{[(z = z_1 \$ z_2 \$ z_3 \$ z_4) \wedge (\text{" no \$ in } z_2, z_3 \text{"})] \supset \forall z_5 \forall z_6 \{[z_2 = z_5 \cdot \text{state}_{403} \cdot \sigma \cdot z_6$

$\sigma \in \Sigma$ — particular symbol in alphabet



$\supset ([z_3 = z_5 \cdot \sigma \cdot (\text{state}, 9) \cdot z_6 \wedge z_6 \neq \Delta]$

$\vee [z_3 = z_5 \cdot \sigma \cdot (\text{state}, 9) \cdot \Delta \wedge z_6 = \Delta])]$

$\wedge$ ... (for all arrows in flowchart } }

HERBRAND'S PROCEDURES

Remark: a wff B is valid iff for no m does $m \models \neg B$ iff $\neg B$ is unsatisfiable. (m an *interpretation*).

Def: $A \sim B$ (A is equi-satisfiable to B)
iff
A is satisfiable iff B is satisfiable.

Def: A is equivalent to B iff $\models A \equiv B$
iff for all interpretations $\mathfrak{I}$,
$\mathfrak{I} \models A$ iff $\mathfrak{I} \models B$.

Review: If F is a boolean combination of wffs $A_1, \dots, A_n$ (e.g. F is
$(((A_1 \vee \neg A_2) \supset A_3) \vee A_4) \supset A_5)$)
then there is a wff F' s.t.
(1) F is equivalent to F'
(2) F' is a "sum" of "products"
(disjunctions) (conjunctions)
of $A_1, \neg A_1, A_2, \neg A_2, \dots, A_n, \neg A_n$.
use:

* $A \equiv B$ is equivalent to $(A \supset B) \wedge (B \supset A)$
* $A \supset B$ is equivalent to $\neg A \vee B$
* $\neg\neg A$ is equivalent to A
* $\neg(A \vee B)$ is equivalent to $\neg A \wedge \neg B$.
* $\neg(A \wedge B)$ is equivalent to $\neg A \vee \neg B$
* distributive laws
  $A \wedge (B \vee C)$ is equivalent to $(A \wedge B) \vee (A \wedge C)$
  $(B \vee C) \wedge A$ " " " $(B \wedge A) \vee (C \wedge A)$
  etc.

Have to check that this procedure does terminate.

i.e. , for example , if A is very complicated, it looks like

$$A \wedge (B \vee C) \Rightarrow \underbrace{(A \wedge B) \vee (A \wedge C)}_{2 \text{ copies of } A}$$

makes things worse. Invent a measure of flatness which can be shown to always decrease.

<u>Remark</u>: $A \equiv B$ implies A is equisatisfiable with B $(A \sim B)$. The converse is not true.

<u>Lemma 1</u>: Every wff is equivalent to a wff in <u>Prenex</u> form.

<u>Lemma 2</u>: Every wff is equisatisfiable with a universal wff.
(Universal : all quantifiers in front and they are all $\forall$)

6.044
11/02/87

Lecture # 21.

Let p and q be propositional constants (i.e. they are either T or F). Then $p \sim q$ (they are equisatisfiable), obviously. But clearly $p \not\equiv q$.

Lemma: Every propositional wff (only 0-ary predicate constants) is equisatisfiable with a propositional wff in 3-CNF.

3-CNF : product of sums with only three summands in every clause. Eg:

$$(p_1 \vee \bar{p}_2 \vee p_3) \wedge (\bar{p}_1 \vee p_2 \vee p_4) \wedge \cdots$$

The size of the 3-CNF of lemma 1 is proportional to the size of the original wff.

Suppose had

$$(p_1 \wedge p_2) \vee (p_3 \wedge p_4) \vee \cdots \vee (p_{n-1} \wedge p_n)$$

When putting this into CNF, there will be an exponential number of terms. Theorem: Can't avoid this blowup if we want a product of sums equivalent to the original wff. But can prevent blowup if we only want to preserve satisfiability.

Eg: $((p_1 \supset \bar{p}_2) \vee p_3) \equiv \neg p_4$

Introduce additional variables for every binary connective

$$((p_1 \underbrace{\supset}_{p_5} \bar{p}_2) \underbrace{\vee}_{p_6} p_3) \underbrace{\equiv}_{p_7} \neg p_4$$

want

$(p_5 \equiv p_1 \supset \bar{p}_2) \wedge$

$(p_6 \equiv p_5 \vee p_3) \wedge$

$(p_7 \equiv (p_6 \equiv \neg p_4)) \wedge$

p_7.

example) Note this is equisatisfiable with the original wff. But they are not equivalent: because (for the second formula depends on P_y unlike the first.

Now put each 3 variable clause in 3-CNF

Lemma Every wff is equivalent to a wff in Prenex normal form.

Proof by example (p 103 of Manna)

$$\forall x\, [(\forall y\; P(x) \vee \forall z.\, q(z,y)) \supset \neg\forall y.\, r(x,y)]$$

(1) redundant quantifier: eliminate

$\rightarrow$ $\forall x\, [(P(x) \vee \forall z.\, q(z,y)) \supset \neg\forall y.\, r(x,y)]$

(2) a prudent thing to do is rename the bound variables to new things. (Just like $\int_0^1 x\,dx = \int_0^1 y\,dy$)

$\rightarrow$ $\forall x\, [(P(x) \vee \forall z.\, q(z,y)) \supset \neg\forall y_1.\, r(x,y_1)]$

(3) eliminate hook:

$\rightarrow$ $\forall x\, [\neg(P(x) \vee \forall z.\, q(z,y)) \vee \neg\forall y_1.\, r(x,y_1)]$

(4) Push in $\neg$

$\rightarrow$ $\forall x\, [(\neg P(x) \wedge \neg(\forall z.\, q(z,y))) \vee \neg\forall y_1.\, r(x,y_1)]$

(5) use $\neg\exists x.\, A \equiv \forall x.\, \neg A$ (equivalent)

$\rightarrow$ $\forall x\, [(\neg P(x) \wedge \exists z.\, \neg q(z,y)) \vee \exists y_1.\, \neg r(x,y_1)]$

(6) Have "$\wedge$"s and "$\vee$"s of quantified atomic formulas. Use:

$$(\exists z.\, A) \wedge B \equiv \exists z.\, (A \wedge B)$$

if z does not occur (free) in B.

$\rightarrow$ $\forall x\, \exists z\, \exists y_1.\, [(\neg P(x) \wedge \neg q(z,y)) \vee \neg r(x,y_1)]$

(by a few applications of the above rule and its complement for $\wedge$)

Lemma Every prenex formula is $\sim$ to a universal formula.

(Universal : $\forall x_1 \ldots \forall x_n$ (quantifier free wff)).

Proof by example : Take

$$\forall x \exists z \exists y_1 [(\neg p(x) \wedge \neg q(z, y)) \vee \neg r(x, y_1) \vee s(z, y_1)]$$

In any particular model, given x can find z and y_1 s.t. formula is true.

$$\forall x [(\neg p(x) \wedge \neg q(f_z(x), y))) \vee \neg r(z, f_{y_1}(x)) \vee s(f_z(x), f_{y_1}(x))]$$

$\forall x . \exists y_1 . \exists y_2 . \forall x_1 \exists z .$

6.044
11/06/87

Lecture # 23

To see if a formula of first-order predicate calculus (without = for now) is valid:

[Note the theorem is true for formulas with equality. The proof is a trick we haven't done yet. You should just know it can be done.]

step (1): negate it
(2): convert to prenex
(3): Skolemize (now have a universal formula)
(4): Enumerate all ground instances looking for propositional inconsistency. Halt when found.

The above procedure would be spectacularly inefficient. Resolution and Unification can be understood as making step (4) above more efficient.

An obvious validity:

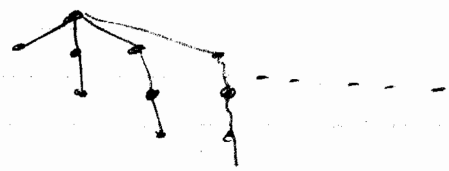
$$\forall \vec{x}\, A \supset A^{\vec{t}}_{\vec{x}}$$

To prove: If all finite sets of ground instances of a universal formula $\forall \vec{x}.\, F$ (F quantifier free without =) are propositionally satisfiable then so is the formula $\forall \vec{x}.\, F$.

Konig's lemma: If we have a tree with an infinite number of nodes and each node has only finitely many children then the tree has arbitrarily long paths.

Another statement: Finite branching tree with arbitrarily long finite paths has an infinite path.

Counterexample if tree is not finite branching:



Call a tree good if given n there is a path of length n.

$T =$ 

suppose all 3 sons are not good. Say there are no paths of length n_1, n_2, n_3 in the sub trees 1, 2, 3 respectively. So T has no paths of length

$$1 + \max\{n_1, n_2, n_3\}.$$

So if T is good it has a good sub tree. Keep picking left most good sub tree. This gives infinite path.

Lemma 1: If there is a single truth assignment to all atfs occurring in ground instances of $\forall \bar{x}\, F$ then $\forall x\, F$ is satisfiable.

Proof - Construct $\mathcal{I}$ satisfying $\forall x. F$ as follows: (Herbrand interpretation)

$$D_{\mathcal{I}} = \text{set of ground terms}.$$

$$\mathcal{I}(f)(t_1, t_2) = \underbrace{f(t_1, t_2)}_{\text{term}}.$$

$$\vartheta(P)(t_1,t_2) = \begin{cases} \text{assignment to } P(t_1,t_2) & \text{if } P(t_1,t_2) \text{ occurs} \\ \text{anything (say false)} & \text{otherwise} \end{cases}$$

By choice of truth assignment to atfs, $\vartheta \models$ ground instances of $\forall \vec{x}.F$.

<u>Sub lemma:</u> $\vartheta \models$ atf iff assignment makes atf true.

<u>Sub lemma:</u> $(t)_\vartheta = t$

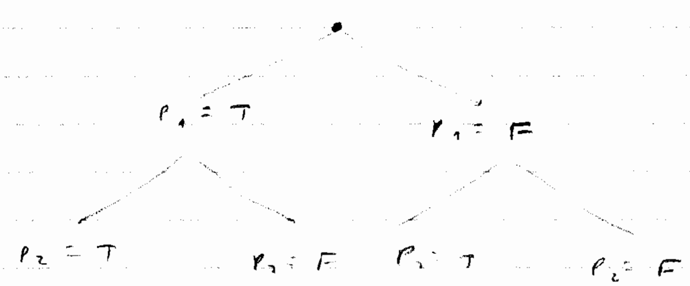
Therefore $\vartheta \models \forall \vec{x}.F$ since the only values of $\vec{x}$ possible are ground terms.

<u>Sub lemma:</u> $\vartheta[x \mapsto (t)_\vartheta] \models A$ iff $\vartheta \models A^{t}_{x}$ (Fundamental lemma of substitution) (non trivial but boring)

∎

<u>Lemma 2:</u> If every finite set of ground instances has a satisfying truth assignment to its atfs then there is one assignment to the atfs which satisfies all ground instances.

Think of this purely propositionally. We have propositional variables $P_1, P_2, \ldots$



•) terminal node: causes some finite set of ground instances which is made false.

6.044
11/6/87

Lecture # 24

[missed]

Topics: • Non standard models of arithmetic,
• Compactness theorem (both propositional and first order)

6.044
11/13/87

Lecture # 25

Complete axiom systems of first order wff+S
(Read Manna 2.2).

Write down some axioms which are valid.
(actually axiom schemes).

Write down "rules" consisting of antecedents and
a consequent.

Proof is a sequence of formulas which are either
axioms or consequents of rules whose antecedents
have already appeared in the list.

Example of an axiom (scheme):

$$(A_1 \wedge \ldots \wedge A_n) \supset \text{true}$$

for all $A_1, \ldots, A_n$ and all $n > 0$.

The set of axioms is infinite but recursive.

Manna's notation for the above would be

$$\{A_1, \ldots, A_n\} \Rightarrow \text{true}$$

This is more or less shorthand for the
above notation except that in ours
a formula A_i could occur twice, and
$(A_1 \wedge A_1) \supset \text{true}$ and $A_1 \supset \text{true}$ are
different.

Rule schemes:

$$\frac{\Gamma \Rightarrow B}{\Gamma, A \Rightarrow B}$$

where $M, A = M \cup \{A\}$. This says if $M = \{A_1, \ldots, A_n\}$ and $A_1 \wedge \ldots \wedge A_n \supset B$ then for any A, $A_1 \wedge \ldots \wedge A_n \wedge A \supset B$. This is a scheme: instance of it consists of actual wffs plugged in.

If $x \notin FV(M)$ then

$$\frac{M \Rightarrow A(x)}{M \Rightarrow \forall x . A(x)}$$

idea "x doesn't occur (free) in M, so saying we can get $A(x)$ means we can get $A(x)$ for any x".

If we are looking at rules like

$$\frac{C_1, C_2}{C_3}$$

then can consider 2 properties:

(1) inferentially sound

$$\models (C_1 \wedge C_2) \supset C_3$$

(2) validity preserving

If $\models C_1$ and $\models C_2$ then $\models C_3$.

(1) implies (2) but not vice versa.

We have, taking $M = \emptyset$,

$$\frac{A(x)}{\forall x . A(x)}$$

But $\not\models A(x) \supset \forall x . A(x)$. Counter model: take $A = P$ unary predicate. Then

$$\not\models P(x) \supset \forall x . P(x)$$

Say $D = \{0, 1\}$, $I_c(P)(a) =$ true iff $a = 0$.
and $I_v(x) = 0$.

But $\models A(x)$ implies $\models \forall x\, A(x)$

so the rule

$$\frac{M \Rightarrow A(x)}{M \Rightarrow \forall x . A(x)}$$

is validity preserving but not inferentially sound.

Look for: axioms are obviously valid and are a decidable set.

Note condition that $x \notin FV(M)$ is crucial. Say $M = \{x = 0\}$. Certainly

$$M \Rightarrow x = 0$$

but can't conclude

$$M \Rightarrow \forall x (x = 0).$$

Conclude:

$$\vdash A \quad \text{implies} \quad \models A.$$

$\vdash A$ means there is a finite sequence of wffs s.t every wff in sequence is
(1) an axiom
or
(2) consequent of antecedents which have appeared earlier in sequence

and last wff is A.

Basic completeness theorem (Gödel) $\models A$ implies $\vdash A$.

(We are using here the Gentzen / Manna $\vdash$.)

Completeness implies valid wffs are r.e. Just list all finite sequences of proofs.

Corollary 1: Valid first order wffs are an r.e. set

Stronger version of Completeness:

Def: $M \vdash A$ means have proof of A using wffs in M as additional axioms.

Def: $M \models A$ (M semantically implies A) means for all models $\mathcal{M}$, if
$\mathcal{M} \models \gamma$ for all $\gamma \in M$
then $\mathcal{M} \models A$.

It is easy to see
$M \vdash A$ implies $M \models A$
(called soundness).

Consider another property of proofs:
(1.5) $\dfrac{M \models C_1 \quad , \quad M \models C_2}{M \models C_3}$ (any M)
Call it validity preserving relative to arbitrary hypotheses).

Need this for above stronger soundness.

Stronger Completeness: $M \models A$ implies $M \vdash A$.

Note that M could be infinite.

Corollary to stronger form: Compactness.

6.044
11/16/87

Lecture # 26

Completeness: $\Gamma \vdash A$ iff $\Gamma \models A$.

The proof that $\Gamma \vdash A$ implies $\Gamma \models A$.
(1) Check axioms are valid
(2) Check inference rules preserve validity (validity of antecedents implies validity of consequents; property (1.5) of last lecture).

We chat about the proof that $\Gamma \models A$ implies $\Gamma \vdash A$:
Construct a "term model" from $\Gamma, \vdash$ such that

Term Model $\models A$ iff $\Gamma \vdash A$.

However there are two problems that need to be fixed to make this workable. Remember in semigroups the set of all E s.t.

$\{E_i\} \models E$

was the set of all E true in *some* model not true in f.o. logic — example:

Γ = axioms of group theory
$A = \exists x_1 \exists x_2 \exists x_3 [x_1 \neq x_2 \wedge x_2 \neq x_3 \wedge x_1 \neq x_3]$

Notice $\Gamma \not\models A$ and $\Gamma \not\models \neg A$.

Complete set of formulas: contains A or $\neg A$ for all formulas A.

$Th(M) = \{A \mid M \models A\}$ is complete for any model M. If Γ alone is not complete there is no term model s.t.

Term Model $\models A$ iff $\Gamma \vdash A$.

second problem is that there may not be enough function symbols. Intuitively if you are building a term model and need
$$\Gamma \vdash \exists x . B(x)$$
then want a constant c s.t $\Gamma \vdash B(c)$.

So we fix Γ by adding to it s.t it has the necessary properties. Idea is: we are supposing $\Gamma \nvdash A$ and want to find Term Model $\models \Gamma$ s.t Term Model $\nvDash A$. Extend Γ to Γ' which has properties
(1) being complete
(2) having enough constants.
and s.t $\Gamma' \nvdash A$. For (1), suppose $\Gamma \nvdash B$ and $\Gamma \nvdash \neg B$ where B is closed. then either $\Gamma \cup \{B\} \nvdash A$ or $\Gamma \cup \{\neg B\} \nvdash A$. this is how we proceed to extend Γ to a complete Γ' s.t $\Gamma' \nvdash A$. [proof: if $\Gamma \cup \{B\} \vdash A$ and $\Gamma \cup \{\neg B\} \vdash A$ then $\Gamma \vdash B \supset A$ and $\Gamma \vdash \neg B \supset A$. Hence $\Gamma \vdash A$, contradiction. we are using various proof system metatheorems here, such as deduction.]

Note: getting (1) and (2) can undo each other, need an infinite number of steps.

If Γ was complete ($\Gamma \vdash B$ or $\Gamma \vdash \neg B$ for any closed B) and Henkinized then could get a term model from Γ, the elements are
$$[c]_\Gamma = \{ d \mid \Gamma \vdash (c = d) \}$$
then by induction on formulas φ,
$$\text{Term Model} \models \varphi \text{ iff } \Gamma \vdash \varphi .$$

Compactness is a corollary of Completeness:
suppose $A_1, A_2, \ldots, A_n, \ldots$ is
finitely satisfiable. Then it is satisfiable.

Proof - suppose $\{A\}_{i \geq 1}$ is not satisfiable.
Then

$$\{A_i\} \models \text{false}.$$

why? There are no models of $\{A_i\}$. So
every model of $\{A_i\}$ satisfies false. By
Completeness $\{A_i\} \vdash$ false. But then there
is an n such that

$$\{A_1, \ldots, A_n\} \vdash \text{false}$$

because proofs are finite and use only
finitely many axioms. But provability is
sound. so

$$\{A_1, \ldots, A_n\} \models \text{false}.$$

But this can only happen if $\{A_1, \ldots, A_n\}$
has no model, a contradiction ∎

another statement of Compactness:

If $\Gamma \models A$ then there is a finite
$\Gamma_0 \subseteq \Gamma$ s.t. $\Gamma_0 \models A$.

6.044

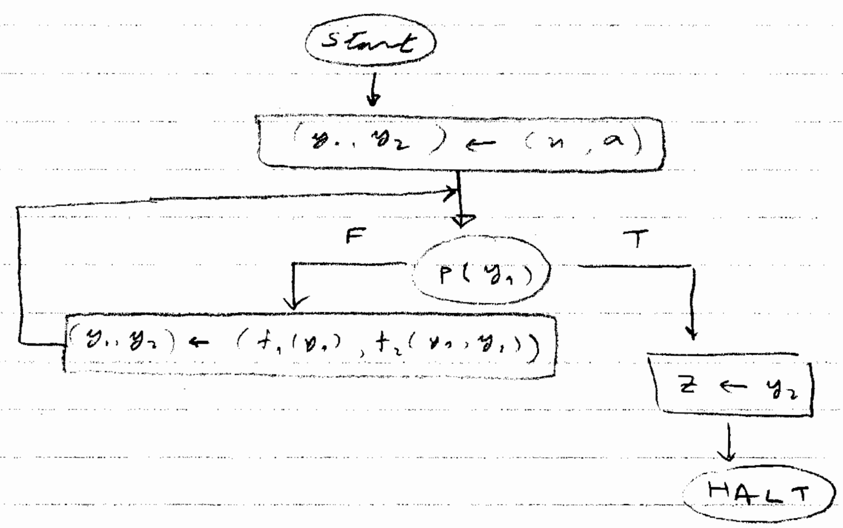
11/18/87 Lecture # 27

FLOWCHART SCHEMAS

Reading: Beginning of chapter 4.

Flowchart schema is to a flowchart what a predicate calculus is to a formula of mathematics.

Example (Manna p 244)



y_1, y_2 are locations; set their values to the values of x, a.

General form of assignment:

$(y_1, y_2) \leftarrow$ (term 1, term 2)

Test boxes, wlog, contain atomic formulas over the given signature.

Flowcharts are deterministic. Every box has one arrow out of it except test boxes which have 2 arrows, and halt box which has no arrow out of it. There is exactly one start box.

We want to interpret these things. Will arrange so that can use predicate calculus interpretations. Need to know about locations, meanings of symbols, and initial values.

Note: Computer Scientists call $y_1, y_2, \dots$ etc variables; Mathematicians would call them locations. Here we will be able to identify these things with what Mathematicians call variables. In a more realistic programming model there are environments (mapping identifiers to locations) and stores (mapping locations to their contents). In our simple model there is no aliasing and sharing and hence we don't need to distinguish between locations and environments.

Choose an interpretation $\mathcal{I} = (\overbrace{(D, \mathcal{I}_c)}^{m}, \mathcal{I}_v)$ of the above scheme.

$D = \mathbb{N}$

$\mathcal{I}_c(a) = 1$

$\mathcal{I}_c(f_1) = \text{predecessor}$

$= \lambda x : \mathbb{N}.\ x \dot{-} 1$

$= x \mapsto \begin{cases} 0 & \text{if } x = 0 \\ x - 1 & \text{if } x > 0 \end{cases}$

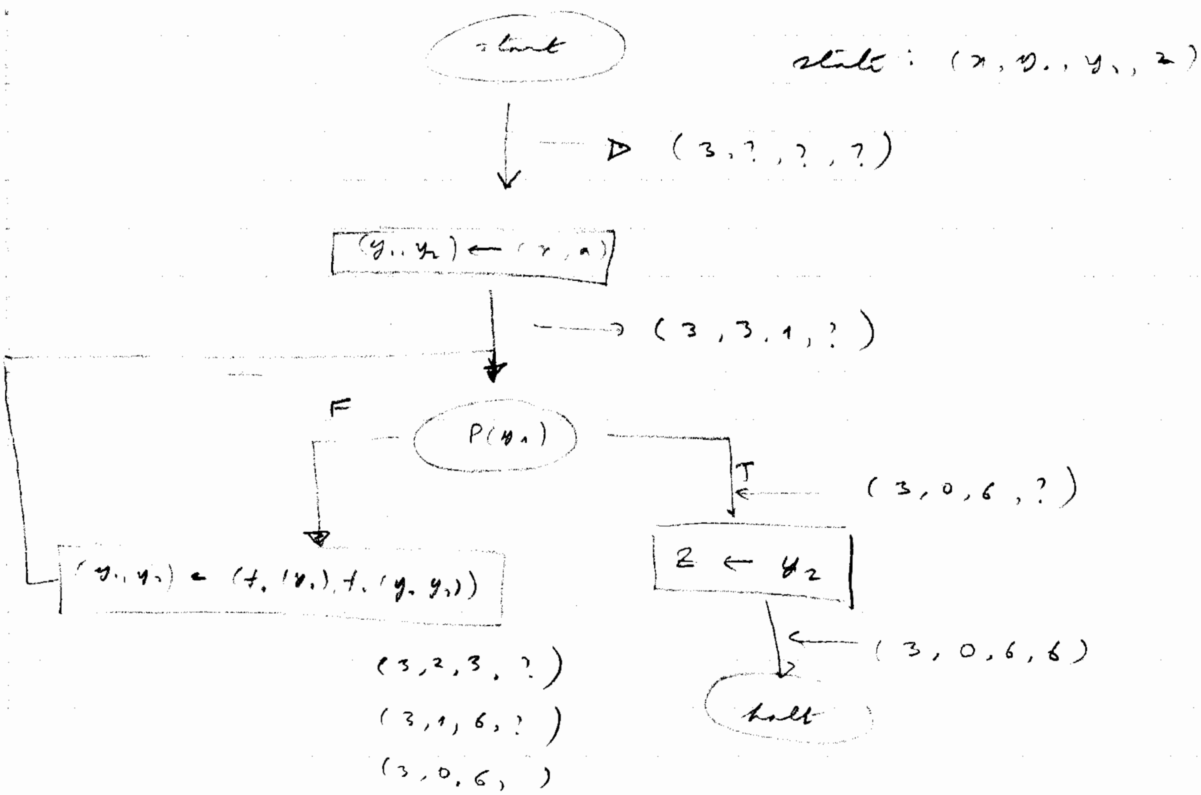
$\mathcal{I}_c(f_2) = \text{product} = *$

$\mathcal{I}_c(P) = \text{is zero}$

In this programming context we call $\mathcal{I}_v$ a <u>state</u> (of memory) or a <u>store</u>. Given m, the flowchart determines a partial map from stores to stores.

Eg: store $\equiv$ $\{x \mapsto 3, y_1 \mapsto ?; y_2 \mapsto ?; z \mapsto ?\}$ (initially). This is all we need to know (i.e. value of x) because initial values of y disappear and z is not used.

This maps to
$$\{x \mapsto 3; y_1 \mapsto 0; y_2 \mapsto 6; z \mapsto 6\}$$



<u>Claim</u>: $\{x \mapsto n; y_1 \mapsto ?, y_2 \mapsto ?, z \mapsto ?\}$ maps to $\{x \mapsto n; y_1 \mapsto 0; y_2 \mapsto n!; z \mapsto n!\}$

Can formalize the syntax of flowcharts.

Example: Let's write a first order wff which means that

F halts in exactly 6 steps

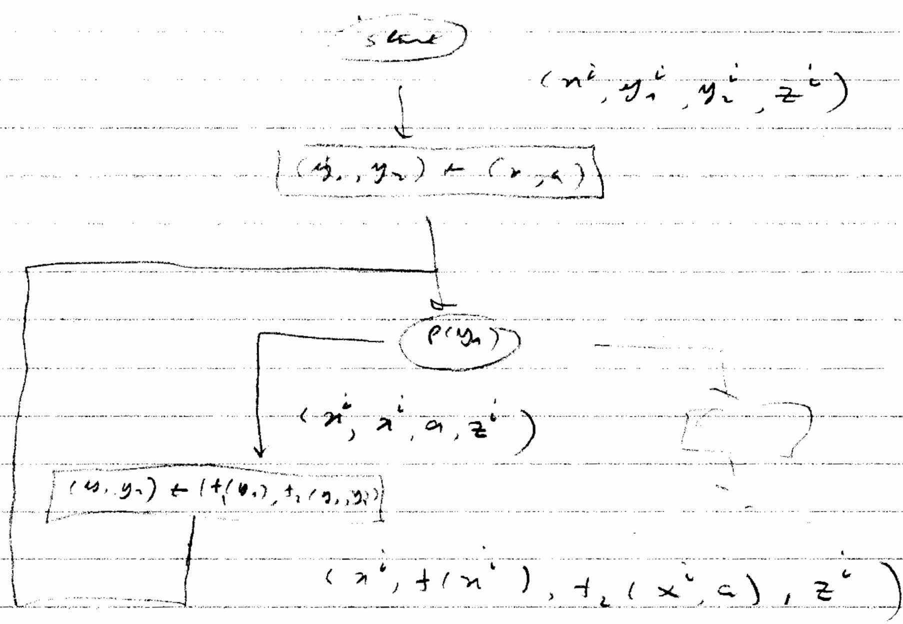
(where F is the previous flowchart).

$\neg p(x) \wedge p(f_1(x))$

$\neg p(x)$ guarantees does not halt in < 6 steps

$p(f_1(x))$: next time, $p(y_1)$ is true.

Look at this as symbolic computation on state



Lemma: For any flowchart F and any $n \geq 1$ one can effectively find a (quantifier free) first order wff $S_{F,n}$ s.t.

$I \models S_{F,n}$ iff F halts under I in exactly n steps.

Proof Idea: First note that for any path

from (start) to a box of F, there is a (q-free) 1st order wff $S_{F,\text{"path"}}$ s.t.
$\mathcal{I} \models S_{F,\text{"path"}}$ iff under $\mathcal{I}$ the computation of F begins by following "path".

We can construct this formula by induction on the length of the path. Have to use a stronger induction which says we can write down terms $t_1, t_2, t_3, t_4, \ldots$ whose values represent contents of registers.

Now $S_{F,n} = \bigvee_{p \in \text{Path-set}} S_{F,p}$

where Path-set = all paths through F of length n from (start) to (halt).

6.044

11/20/87 — Lecture # 28

Last time we saw that there is an effective transformation

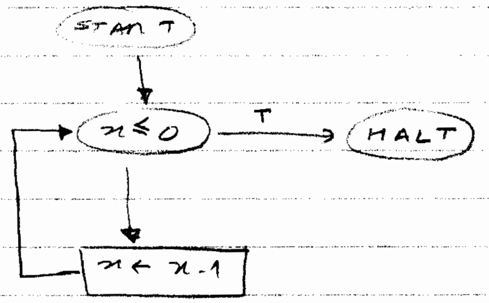
$$(F, n) \longmapsto S_{F,n}$$

where $S_{F,n}$ is a quantifier free first-order wff such that

$$\mathcal{I} \models S_{F,n} \text{ iff } F \text{ halts on } \mathcal{I} \text{ in exactly } n \text{ steps.}$$

Park's lemma: If F terminates in all $\mathcal{I}$ (i.e. F halts under $\mathcal{I}$ for all $\mathcal{I}$) then there exists an n s.t. for all $\mathcal{I}$, F terminates in $\mathcal{I}$ in $\leq n$ steps.

F_0:



Under the standard model < N, ≤, 0, 1, subtraction > the above always terminates, but the number of steps depends on the interpretation of x. Park's lemma does not apply to this particular interpretation; it applies to flowcharts halting under all interpretations.

But can conclude from Park's lemma that there is an interpretation in which the above flowchart does not terminate.

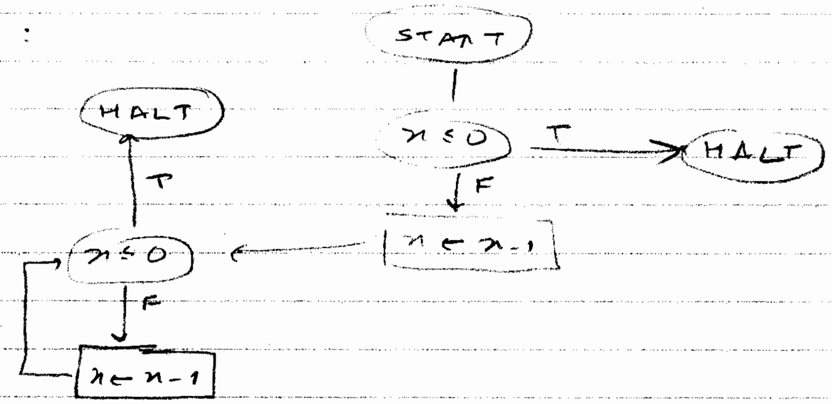
Corollary of Park's lemma: Every completely terminating flowchart is equivalent to a loop free flowchart.

"Completely terminating": terminates in all interpretations.

F and F' are equivalent if they compute exactly the same state to state partial function under all interpretations. i.e. state to state transformation is the same in all models

Given completely terminating flowchart F, there is an n s.t. F halts in ≤ n steps in all interpretations. Now can "unwind" F. For example, one step of unwinding the loop in the above F_0 yields:

F_1:



as this is equivalent to the original flowchart F_0. In general, unwinding is an equivalence preserving transformation. so unwind our F so that it looks like a tree upto depth n (i.e. no loops, or "coming back", before n steps). Now can throw away everything below level n because flowchart is guaranteed to halt in ≤ n steps.

Philosophical conclusion here is that flowcharts that terminate in all interpretations are not interesting.

<u>Proof of Park's lemma:</u>

If F terminates in all $\mathcal{I}$ then the set of wffs

$$\{ \neg S_{F,n} \mid n \geq 0 \}$$

is unsatisfiable. Why? Suppose

$$\mathcal{I} \models \neg S_{F,n} \quad \text{for all } n \geq 0,$$

then under $\mathcal{I}$, F doesn't halt in 0 steps, doesn't halt in 1 step, ... i.e. doesn't halt.

Now by compactness there is an n_0 such that

$$\bigwedge_{n \leq n_0} \neg S_{F,n}$$

is unsatisfiable. That is,

$$\bigvee_{n \leq n_0} S_{F,n}$$

is valid. This last formula says that F halt in $\leq n_0$ steps. Since it is valid, F always halts in $\leq n_0$ steps ∎

<u>Corollary</u>: Let $Th(\mathbb{Z})$ be the true sentences of $\langle \mathbb{Z}, +, *, \text{subtraction}, 0, 1 \rangle$. Clearly

$$\langle \mathbb{Z}, +, *, -, 0, 1 \rangle \models F_0 \text{ terminates}.$$

But

$$Th(\mathbb{Z}) \not\models F_0 \text{ terminates}.$$

Proof - Clearly

$$Th(\mathbb{Z}) \cup \{ \neg S_{F,n} \mid n \geq 0 \}$$

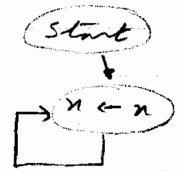
is finitely satisfiable. So it is satisfiable. So $Th(\mathbb{Z}) \not\models F_0$. ∎

Theorem (Paterson, Park, Luckham) $K_2 \le_m \mathcal{F}$ where

$\mathcal{F} = \{ F \mid$ there is an I s.t. F terminates in $I \}$

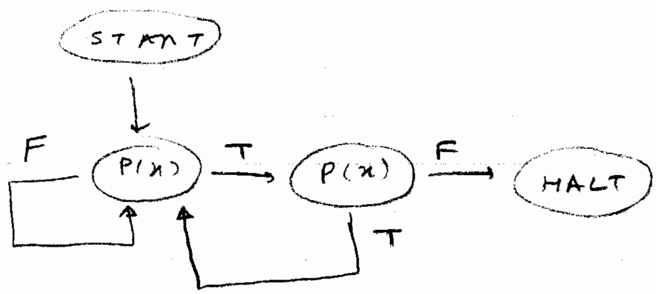
$= \{ F \mid F$ is not equivalent to the flowchart

Ω :



$\}$

(Note: if insist that flowcharts have a halt box, as Manna does, a always diverging flowchart is



)

Corollary: Flowchart scheme equivalence (even to Ω) is not r.e.

Proof of Theorem: Given a TM M, consider its blank tape computation word:

$\$1\ q_{start}\ \Delta\ \$ \ldots \# \ldots \ldots \$\ (halt)\ \#$.

Code the computation word into truth values, symbol by symbol.

With any infinite sequence of boolean values

$b_0, b_1, \ldots$

associate model M of symbols $P, f, 0$ by the rule:

$$\mathcal{M} \models p(f^n(0)) \quad \text{iff} \quad b_n$$

where

$$f^n(0) = \underbrace{f(\ldots(f(0))\ldots)}_{n} \qquad (f^0(0) = 0) .$$

and $\quad D_{\mathcal{M}} = \{ f^i(0) \mid i \geq 0 \}$.

Conversely given a model $\mathcal{M}$ with signature $p, f, 0$ associate the sequence

$$b_0, b_1, b_2, \ldots$$

where

$$b_n = \text{true} \quad \text{iff} \quad \mathcal{M} \models p(f^n(0))$$

6.044

11/23/87

Lecture #29

(lecture by Mishin)

Topic: Showed $\overline{K_2} \leq_m \{ F \mid F \text{ is equivalent to } \Omega \}$.

6.044 Lecture # 30
11/25/87

NON-Ω = { F | F is not equivalent to Ω }

Showed last time that

$K_2 \le_m$ NON-Ω.

We regard the two flowcharts

as equivalent because we define equivalence in terms of effect upon store.

Review of proof -

Associate with model M an infinite string

$b_M = b_0, b_1, \ldots$

of truth values (bits).

Given M construct flowchart F_M which works like a two-headed finite automaton on the sequence

$P(f^0(0)), P(f^1(0)), \ldots, \ldots$.

F_M halts under M iff some prefix of b_M is the computation word of a halting computation of M on Λ.

contents $(x_i) = f^{n_i}(0)$

See section §4-2 of Manna.

while program schemes.

$$W ::= x := \text{term} \mid W; W \mid \underline{\text{if}}\ qff\ \underline{\text{then}}\ W\ \underline{\text{else}}\ W\ \underline{\text{fi}} \mid \underline{\text{while}}\ qff\ \underline{\text{do}}\ W\ \underline{\text{od}}.$$

structured operational semantics (SOS) for while programs:

Given M define a relation "$\longrightarrow$" (call it the one-step relation) between while program configurations. A while program configuration is a pair

$$(\underbrace{W}_{\text{while program}}, \underbrace{\vartheta_v}_{\text{state over } M}).$$

the program here will be the rest of the program to be executed.

Axioms for if:

$$(\underline{\text{if}}\ p\ \underline{\text{then}}\ w_1\ \underline{\text{else}}\ w_2\ \underline{\text{fi}}, \vartheta_1)$$

$$\longrightarrow (w_1, \vartheta_1) \quad \text{if } (M, \vartheta_1) \models p$$

$$(\underline{\text{if}}\ p\ \underline{\text{then}}\ w_1\ \underline{\text{else}}\ w_2\ \underline{\text{fi}}, \vartheta_1)$$

$$\longrightarrow (w_2, \vartheta_1) \quad \text{if } (M, \vartheta_1) \not\models p$$

Deductive rule for $w_1; w_2$:

$$\frac{(W_1 ; \vartheta_1) \to (W_1' ; \vartheta_2)}{(W_1 ; W_2 , \vartheta_1) \to (W_1' ; W_2 , \vartheta_2)}$$

Notice this rule has nothing to do with the model, while the previous axiom depends on the model.

Axiom: $(x := t , \vartheta) \to (\text{stop} , \vartheta [x \mapsto (t)_\vartheta])$

Axiom: $(\text{stop} ; W , \vartheta) \to (W , \vartheta)$

If $\vartheta \not\models p$ then
$(\underline{\text{while}}\ p\ \underline{\text{do}}\ W\ \underline{\text{od}} , \vartheta) \to (\text{stop} , \vartheta)$

If $\vartheta \models p$ then

$$\frac{(W , \vartheta) \to (W' , \vartheta')}{(\underline{\text{while}}\ p\ \underline{\text{do}}\ W\ \underline{\text{od}} , \vartheta) \to (W' ; \underline{\text{while}}\ p\ \underline{\text{do}}\ W\ \underline{\text{od}} , \vartheta')}$$

Alternatively could just have an axiom for while:

$(\underline{\text{while}}\ p\ \underline{\text{do}}\ W\ \underline{\text{od}} , \vartheta)$

$\to (\text{if } p \text{ then } W ; \underline{\text{while}}\ p\ \underline{\text{do}}\ W\ \underline{\text{od}}$
$\quad \text{else stop} , \vartheta)$

Define

$[\![W]\!]_m (\vartheta)$
(W a while program, m a model, ϑ a state)
$= \vartheta'$

s.t.

$(W , \vartheta) \xrightarrow{*} (\text{stop} , \vartheta')$

The function $[\![W]\!]_m$ from states to states is a partial function, undefined if stop is never reached.

Need to prove by induction on the definition of → that every configuration moves to at most one other configuration, so the above is really a (partial) function (we can't have two different $\mathcal{I}$'s which are reachable from $\mathcal{I}$).

Can prove things about programs by an induction which is based on the structure of the program rather than on its execution.

6.044

11/30/87

Lecture # 31

* Equivalence of flowcharts and while programs.

* Dynamic Logic ;

[W] A .

6.044

12/02/87 Lecture # 32

(Floyd)-Hoare Logic.

Non-determinism:

Define: $[\![x := ?]\!]_m (\vartheta_v) = \{ \vartheta_v [x \mapsto d] \mid d \in m_D \}$

so $[x := ?]A \equiv \forall x.\, A$.

Dynamic logic generalizes the behaviour of the quantifier: $[W]A$ means "after W, A holds". Called a modality.

Partial correctness assertions are subset of dynamic logic. Hoare writes a (first-order) partial correctness assertion as

$$\{A\} W \{B\}$$

(if input satisfies A then after W, if W halts then outputs satisfy B). Partial because W may not halt.

W halts $\equiv$ $\neg$(W diverges)

$\equiv$ $[W]$ false.

$\langle W \rangle A$: it is possible to do W and halt in a state in which A holds.

Define: $\langle W \rangle A$ to be $\neg [W] \neg A$

(similar to $\exists x\, A \equiv \neg \forall x \neg A$)

Note $\langle x := ? \rangle A \equiv \exists x.\, A$.

$\{A\} W \{B\}$ means $A \supset [W] B$

Assignment axiom:

$$\{A^t_x\}\ x := t\ \{A\}$$

Lemma: Th $\vdash_{HL}$ {A} W {B} implies Th $\models$ {A} W {B}
(rules and the axioms are sound)
(HL = Hoare logic).

Stop axiom:

$$\{A\}\ \text{stop}\ \{A\}$$

These are the only two axioms, we now give the inference rules of HL.

(1)

$$\frac{\{A\} W_1 \{C\},\quad \{C\} W_2 \{B\}}{\{A\} W_1 ; W_2 \{B\}}$$

Note: It is not true that if {A} W_1 {C} and {C} W_2 {B} hold in some particular interpretation $\mathcal{I}$, then {A} W_1 ; W_2 {B} holds in $\mathcal{I}$. We are only saying that if the antecedents are valid so is the conclusion.

This is the sequencing rule.

(2) Condition rule

$$\frac{\{A \wedge q\} W_1 \{B\},\quad \{A \wedge \neg q\} W_2 \{B\}}{\{A\}\ \text{if } q \text{ then } W_1 \text{ else } W_2\ \{B\}}$$

(3) Invariance rule

$$\frac{\{A \wedge q\} W \{A\}}{\{A\}\ \text{while } q \text{ do } W \text{ od } \{A \wedge \neg q\}}$$

assuming $\{A \wedge q\} W \{A\}$, it is easy to prove that

$$\{A\} \text{ while } q \text{ do } W \text{ od } \{A \wedge \neg q\}.$$

Clearly $\neg q$ must be true at the end of the while. Can show that A also holds by induction on the number of times you go through the loop.

(4) Rule of consequence: if $A \supset C$, $D \supset B \in Th$ then

$$\frac{\{C\} W \{D\}}{\{A\} W \{B\}}$$

Theorem: (Construction theory completeness of Hoare logic).

Let Σ^* abbreviate $(\Sigma^*, \cdot, =, a, b, \Lambda)$ for $\Sigma = \{a, b\}$. Let

$$Th_\Sigma = Th(\Sigma^*) = \{A \mid \Sigma^* \models A \text{ and } A \text{ is a 1-st order wff}\}.$$

Restrict while programs and wffs to signature of Σ^*. Then

$$\Sigma^* \models \{A\} W \{B\} \quad \text{iff} \quad Th_\Sigma \vdash_{HL} \{A\} W \{B\}$$

Proof sketch: right to left is the soundness lemma stated above. We do the other direction by induction on definition of W.

Base: $\models \{A\}\ x := t\ \{B\}$

$\models A \supset [x := t] B$

Check that $[x := t] B \equiv B^t_x$

so get

$\models A \supset B^t_x$

so in particular

$$A \supset B^t_x \in Th_\Sigma$$

therefore

$$\vdash \{A\}\ x := t\ \{B\}$$

by assignment assignment and rule of Consequence :

$$\{B^t_x\}\ x := t\ \{B\} \qquad \text{(axiom)}$$

$$A \supset B^t_x\ ,\quad B \supset B \in Th_\Sigma$$

so by rule of consequence

$$\{A\}\ x := t\ \{B\}\ .$$

6.044 Lecture # 33
12/4/87

Invariance rule:

$$\frac{\{A \wedge q\}\ W\ \{A\}}{\{A\}\ \text{while } q \text{ do } W \text{ od}\ \{A \wedge \neg q\}}$$

Theorem: $\Sigma^* \models \{B\} W' \{C\}$ implies $Th_{\Sigma^*} \vdash_{HL} \{B\} W' \{C\}$
(the iff holds because rules are sound, but this is the main theorem).

Proof by induction on W'; last time we did the base case of assignment statement.

Case: $W' ::=$ while q do W od.

Lemma: (Expressiveness of Σ^* w.r.t. while programs)
(Σ^*: concatenation and $=$). For every W and first-order wff A, there is a first-order wff, call it $\widehat{[W]A}$, s.t.

$$\Sigma^* \models \widehat{[W]A} \equiv [W]A$$

(i.e. over Σ^* every formula of dynamic logic is equivalent to a first-order formula).

This is not true in general.

Proof: Say W has variables $\vec{x}_n = x_1, \ldots, x_n$. (i.e. these are the formal identifiers that appear in W). Then
$\{(\vec{u}_n, \vec{v}_n) \mid W$ started with $\vec{x}_n$ containing $\vec{u}_n$ halts with $\vec{x}_n$ containing $\vec{v}_n\}$
(the $\vec{u}_n, \vec{v}_n \in (\Sigma^*)^n$). This is a partial function. Claim above set is r.e. (Clear)

But we know that every r.e. set $R \subseteq (\Sigma^*)^{2n}$ is definable by a first-order wff over the signature of Σ^*:

$$A_R(\vec{u}_n, \vec{v}_n) :\quad \exists z .\ z \text{ is a halting computation word of } M_R \text{ on input } \vec{u}_n, \vec{v}_n$$

(where M_R accepts R). So define

$$\widehat{[W]A} \ \equiv_{df}\ \exists \vec{v}_n .\ A_R(\vec{x}_n, \vec{v}_n) \wedge A^{\vec{v}_n}_{\vec{x}_n}$$

($\vec{x}_n$ = all variables occurring free in W and A) ▨

Let A be the formula

$$A \equiv_{df} \widehat{[W']C}$$

The following is a validity of dynamic logic:

$$\models ([W']C \wedge q) \supset [W][W']C$$

so

$$\models (A \wedge q) \supset [W]A$$

this can be written

$$\models \{A \wedge q\}\ W\ \{A\}$$

By induction hypothesis

$$Th_{\Sigma^*} \vdash_{HL} \{A \wedge q\} W \{A\}.$$

Now by the invariance rule

$$Th_{\Sigma^*} \vdash_{HL} \{A\}\ \underbrace{\text{while } q \text{ do } W \text{ od}}_{W'}\ \{A \wedge \neg q\}$$

Hence if

$$\underset{(\text{II})}{B \supset A}, \quad \underset{(\text{III})}{(A \wedge \neg q) \supset C} \ \in Th_{\Sigma^*}$$

then by rule of consequence

$$Th_{\Sigma^*} \vdash_{HL} \{B\}\ W'\ \{C\}.$$

(II) $B \supset A$ iff $B \supset [W']C$

iff $\{B\} W' \{C\}$

so since

$\Sigma^* \models \{B\} W' \{C\}$ (by hypothesis),

we get

$B \supset A \in Th_{\Sigma^*}$.

(III) $(A \wedge \neg q) \supset C$ iff $([W']C \wedge \neg q) \supset C$

after doing W', C holds and q is false. so W' didn't do anything. so clearly C holds.

so

$\models ([W']C \wedge \neg q) \supset C$

(i.e. this is a validity of dynamic logic), so in particular

$([W']C \wedge \neg q) \supset C \in Th_{\Sigma^*}$.

Do the sequencing case yourself; it is easier than this. Need intermediate expression which will follow from Expressiveness Lemma ☐

"Resolution" - theorem proving.

Resolution is a method for theorem proving. Resolution takes the idea of Herbrand's procedure and tries to make it more efficient. Offers a dovetailed way to do the enumeration of ground instances and truth table checking together.

Resolution with unification accomplishes this dovetailing.

Resolution wants the formula in conjunctive clausal form,

$$\underbrace{(e_1 \vee e_2 \vee e_3)}_{\text{clause}} \wedge (e_1' \vee e_2') \wedge \ldots$$

where the e_i, e_i' are literals. The formula is represented (in the program data-structure) as a set of sets,

$$\{ \{e_1, e_2, e_3\}, \{e_1', e_2'\}, \ldots \}.$$

Every universal formula is equivalent to a formula in clausal normal form. There might be exponential blowup in converting to c.n.f but usually we just assume the formula is given in c.n.f.

Basic method: Pick a pair of clauses and apply the resolution rule to them.

Resolution rule: $\{stuff, L\}, \{stuff', \neg L\}$
(pair with a complementary pair of literal occurrences). Form the new clause
$stuff \cup stuff'$.

This last is called the resolvent; write
{stuff, L}, {stuff, L'} → stuff ∪ stuff'

$$(S_1 \vee L) \wedge (S_2 \vee \neg L)$$
$$\equiv (S_1 \vee L) \wedge (S_2 \vee \neg L) \wedge (S_1 \vee S_2)$$

Method: keep resolving till you get down to ∅.

$$\{\{L\}, \{\neg L\}\} \to \{\ \}$$

If original formula was unsatisfiable there will be a way to derive the false clause.

Now remember the literals are actually formed from atomic formulas.

Unification:
example: $P(x)$, $\neg P(f(y))$
make them look like L, $\neg L$ by letting $x = f(0)$.
so don't go down all the way to ground instances; unify instead.

$$q(x, f(y)) \qquad \neg q(g(z), f(z))$$

need
$$\begin{cases} x = g(z) \\ y = z \end{cases}$$

so
$$\begin{cases} x = g(w) \\ y = w \\ z = w \end{cases}$$

works; get
$$q(g(w), f(w)), \quad \neg q(g(w), f(w))$$

single resolution:

$\{L\}, \{stuff, \neg L\} \rightarrow \{stuff\}$

In this case, throw in {stuff}, but also throw out {stuff, ¬L} since stuff ⊆ {stuff, ¬L}

Heuristic: If $c_1 \subseteq c_2$ throw out c_1.

Logic program: { {logic clause}, {logic clause}, ... }
where
logic clause: clause with at most one positive literal.

$$\neg p \vee \neg q \vee \neg r \vee s \equiv (p \wedge q \wedge r) \rightarrow s$$

to meet goal s, set of goals p, q, r and try to meet them. Logic clauses are also called Horn clauses. If a set of Horn clauses is inconsistent then <u>single</u>-resolution (+ unification) reaches { }.

6.044 Lecture # 35
12/9/87

* about the final
* where to go next: MIT courses in this area.

RECURSIVE FUNCTION SCHEMES:

$F(x, y) \Leftarrow$ if $p(y)$ then $g(F(F(y,x),x),y)$
else y

i.e. definition of function symbol F is a term which contains occurrences of F. Note F is a function variable, g function constant, p predicate constant.

Given m, can evaluate $F(\underline{a}, \underline{b})$ where m says what $\underline{a}$, $\underline{b}$ mean.

Inference rules and axioms for $\rightarrow$ (evaluates in one step)

if $\underline{\text{true}}$ then t_1 else $t_2 \rightarrow t_1$

$$\frac{q_1 \rightarrow q_2}{(\text{if } q_1 \text{ then } \ldots) \rightarrow (\text{if } q_2 \text{ then } \ldots)}$$

(q_1, q_2 qfree)

$F(t_1, t_2) \rightarrow$ if $p(t_2)$ then $g(F(F(t_2,t_1),t_2))$
else t_2

Final Exam

Instructions. Do all 8 problems; a total of 200 points is allocated as shown on each problem. This exam is *open book*—you may appeal to any results from the text, handouts, or lectures. In doing a problem, you may also assume the results of *any preceding* problem (or problem part) on this exam. You have three hours. Good luck.

Problem 1 [20 points]. [Diagonalization] A language $R \subseteq \{\mathtt{a},\mathtt{b}\}^*$ is said to *separate* a pair of languages A and B iff $A \subseteq R$ and $B \subseteq \overline{R}$. Let $d(M) \in \{\mathtt{a},\mathtt{b}\}^*$ denote the code of a Turing machine M as in class notes. Let

$$K_{\mathtt{a}} = \{M \mid M \text{ on input } d(M) \text{ outputs } \mathtt{a}\}$$
$$K_{\mathtt{b}} = \{M \mid M \text{ on input } d(M) \text{ outputs } \mathtt{b}\}\ .$$

Prove that there is no recursive set R separating $K_{\mathtt{a}}$ and $K_{\mathtt{b}}$. (*Hint*: Consider the machine M which computes the function

$$f(x) = \begin{cases} \mathtt{b} & \text{if } x \in R \\ \mathtt{a} & \text{if } x \notin R.\ ) \end{cases}$$

Problem 2 [25 points]. [Post Correspondence Problem] A Post system $\{(\alpha_1,\beta_1),\ldots,(\alpha_k,\beta_k)\}$ (cf. Manna, §1-5.4) has an ***infinite solution*** iff there is an infinite sequence of integers $i_1, i_2, \ldots$ $(1 \le i_j \le k$ for all $j \ge 1)$ such that

$$\alpha_{i_1}\alpha_{i_2}\cdots = \beta_{i_1}\beta_{i_2}\cdots$$

For example, the Post system with one pair $\{(\mathtt{a},\mathtt{aa})\}$ has no solution in the ordinary (finite) sense, but does have an infinite solution. Let

$$IPCP ::= \{S \mid S \text{ is a Post system with an infinite solution}\}.$$

(a) [5 points] Briefly explain why, in Manna's reduction of the halting problem for Post machines to PCP, if the machine diverges then the Post system Manna constructs has an infinite solution.

(b) [20 points] Show that IPCP is not r.e. (*Hint*: Slightly modify Manna's construction to reduce the *complement* of the halting problem for Post machines to IPCP.)

Problem 3 [25 points]. [Semigroup Word Problems] Consider semigroup terms and interpretations over the alphabet $\{\mathtt{a},\mathtt{b}\}$, i.e., words in $\{\mathtt{a},\mathtt{b}\}^+$ (cf. Handout 15). The *core* of a semigroup interpretation $\mathcal{I}$ is $\{\mathcal{I}(u) \in S \mid u \in \{\mathtt{a},\mathtt{b}\}^+\}$. A set of semigroup equations is *degenerate* iff the core of every interpetation which satisfies the equations has exactly one element. Let

$$DG = \{\mathcal{E} \mid \mathcal{E} \text{ is a finite, degenerate set of semigroup equations}\}$$

Prove that DG is r.e. (*Hint*: Use Completeness.)

Sequence logic is an extension of first-order logic in which there are two kinds of variables: individual (first-order) variables $x, y, z, \ldots$ which refer as usual to elements of the domain, and *sequence variables*, $X, Y, Z, \ldots$ which refer to finite sequences of elements of the domain; there will also be first-order terms t and sequence terms T. The definitions are precisely what you might expect, but to avoid doubts we now spell them out in more detail.

For any first-order signature, the terms and wffs of sequence logic (*s-wff's*) are defined by the following grammar using with three additional symbols $\doteq, \bullet$, mkseq.

$$\begin{aligned} t &::= \quad \ldots\text{first-order terms}\ldots \\ T &::= \quad T{\bullet}T \mid \mathrm{mkseq}(t) \mid X \\ \mathrm{atf} &::= \quad \ldots\text{first-order atomic formulas}\ldots \mid T \doteq T \\ A &::= \quad \mathrm{atf} \mid \neg A \mid A \wedge A \mid \forall x.\, A \mid \forall X.\, A \end{aligned}$$

A *model* in sequence logic is simply a first-order model. Valuation functions, however, assign values to sequence variables as well as to individual variables. Namely, let

$$D^* = \{\langle d_1, \ldots, d_n\rangle \mid d_1, \ldots, d_n \in D, n \geq 0\}$$

be the set of finite sequences of elements from a set D. Then

Definition 1. A *sequence valuation function* over a model $\mathcal{M}$ with domain D is a function I_v which assigns an element of D to each individual variable and an element of D^* to each sequence variable, i.e., $I_v(x) \in D$ and $I_v(X) \in D^*$. A *sequence interpretation* $\mathcal{I}$ is a pair $(\mathcal{M}, I_v)$ where $\mathcal{M}$ is a model and I_v is a sequence valuation function over $\mathcal{M}$.

The meaning of sequence terms in a sequence interpretation $\mathcal{I} = ((D, I_c), I_v)$ is defined as in first-order logic with the additional feature that $\bullet$ is interpreted as concatenation, $\cdot$, of sequences and mkseq is interpreted as the function which promotes an element to the sequence of length one consisting of just this element. That is,

- $(t)_\mathcal{I}$ is defined exactly as in first-order logic.

- $(X)_{\mathcal{I}} = I_v(X)$.
- $(T_1{\cdot}T_2)_{\mathcal{I}} = (T_1)_{\mathcal{I}} \cdot (T_2)_{\mathcal{I}}$.
- $(\text{mkseq}(t))_{\mathcal{I}} = \langle (t)_{\mathcal{I}} \rangle$

Satisfaction over a sequence interpretation $\mathcal{I}$ is then defined as in first-order logic with the addition that the symbol $\doteq$ is interpreted as equality of sequences:

- $\mathcal{I} \models A$ for a first-order atf A is defined exactly as in first-order logic.
- $\mathcal{I} \models T_1 \doteq T_2$ iff $(T_1)_{\mathcal{I}} = (T_2)_{\mathcal{I}}$.
- $\mathcal{I} \models \neg A$, $\mathcal{I} \models A \wedge B$, and $\mathcal{I} \models \forall x.A$ are defined exactly as in first-order logic.
- $\mathcal{I} \models \forall X.A$ iff for all $d^* \in D^*$ it is the case that $\mathcal{I}[X \mapsto d^*] \models A$.

Abusing notation, we write $\langle t \rangle$ instead of mkseq(t) in s-wff's. We also use the other logical connectives ($\supset$, etc.) and quantifiers $\exists x$ and $\exists X$ which can be expressed in terms of the connectives above in the usual way. For example, the s-wff

$$\forall x_1, x_2, X_1, X_2.(\langle x_1 \rangle{\cdot}X_1 \doteq \langle x_2 \rangle{\cdot}X_2) \supset (x_1 = x_2) \wedge (X_1 \doteq X_2)$$

asserts that every sequence which has a first element has a unique first and rest. This s-wff is valid in all models.

Problem 4 [20 points]. [Simple wff's and Compactness]

(a) [3 points] Write an s-wff whose only free variable is X and whose meaning is that X is the empty sequence $\Lambda \in D^*$. (Remember that no constant denoting Λ appears in s-wff's.)

(b) [3 points] Write an s-wff whose only free variables are x and X, and whose meaning is that x occurs in X.

(c) [4 points] Write an s-wff which is valid in precisely those models with finite (non-empty) domain.

(d) [10 points] State precisely and explain the conclusion that sequence logic does not satisfy the Compactness Property.

Problem 5 [25 points]. [Incompleteness]

Let Th$(\{\mathsf{a}, \mathsf{b}\}^*)$ be the first-order wff's valid over the structure $(\{\mathsf{a}, \mathsf{b}\}^*, \mathsf{a}, \mathsf{b}, \cdot)$ of strings of a's and b's under concatenation. Let *s-Valid* be the set of valid s-wff's.

(a) [20 points] Show that Th$(\{\mathsf{a}, \mathsf{b}\}^*) \leq_m$ s-Valid.

(b) [5 points] Conclude a "Gödel's Incompleteness" Theorem for sequence logic.

Sequence while program schemes (swps's) are while program schemes extended to include sequence variables. That is, swps's are while program schemes with sequence assignment statements of the form $X := T$ in addition to ordinary assignment statements of the form $x := t$. Tests in swps's are quantifier-free *s-wffs*. We also allow sequence assignment statements of the form $X := \epsilon$ whose effect is defined to be setting X to the empty sequence, $\Lambda \in D^*$. For example, if X is a sequence of a's, then the following swps has the effect of setting Y equal to X and x equal to a or b depending on the parity of the length of X.

```
Y := ε; x := a;
while Y ≠̇ X do
    Y := ⟨a⟩·Y;
    if Y ≐ X then x := b else Y := ⟨a⟩·Y fi
od
```

Problem 6 [25 points]. [Induction on Terms and While Program Schemes] This problem explains why the "extra" sequence assignment statement $X := \epsilon$ is needed in swps's. A sequence interpretation $\mathcal{I}$ is Λ-*free* iff $\mathcal{I}_v(X) \neq \Lambda$ for all X.

(a) [10 points] Prove that if $\mathcal{I}$ is Λ-free, then $(T)_{\mathcal{I}} \neq \Lambda$ for all sequence terms T.

(b) [15 points] Conclude that there is no swps *without the constant* ϵ which halts with X equal to Λ in all interpretations.

Problem 7 [30 points]. [Hoare Logic] Let W be the swps given above. Use the axioms and rules of Hoare logic, extended to allow s-wff's as pre- and post-conditions, to prove the partial correctness assertion

$$\{\text{true}\}W\{\exists Z.(X \doteq Z{\cdot}Z) \equiv (x = a)\}$$

Problem 8 [30 points]. [While Schemes and Computability] Let $\mathcal{Z}_n$ be the integers mod n under addition, and let $\mathcal{I}_0$ be the valuation under which all first-order variables are zero and all sequence variables are Λ. In this problem we consider swps's whose only first-order symbols are $0, 1, +, =$. Define spectrum(W) for a swps W to be

$$\{n > 1 \mid W \text{ halts started in } (\mathcal{Z}_n, \mathcal{I}_0)\}\ .$$

(a) [5 points] Explain why spectrum(W) is r.e. for every swps W.

(b) [10 points] Exhibit a swps which, for *all* $n > 1$, halts under interpretation $(\mathcal{Z}_n, \mathcal{I}_0)$ with variable X equal to a sequence of n zeroes.

(c) [15 points] Sketch an argument demonstrating that *every* r.e. subset of $\{n \mid n > 1\}$ is the spectrum of some swps.