

מודלים חישוביים  
**Computational Models**

6. Reductions

מינוח

- **Recursive**
  - $B \in R$
  - B is recursive/computable/decidable
  - רקורסיבית/חשיבה/כריעה
- **Recursively enumerable**
  - $B \in r.e.$
  - B is recursively enumerable/r.e./partial recursive/semi-computable/semi-decidable
  - חשיבה חלקית/כריעה למחצה/ניתן למניה רקורסיבית

Some Decision Problems

- $HALT(f,x)$   $f(x) \neq \perp$
- $HALT_0(f)$   $f() \neq \perp$
- $HALT^*(f)$   $\forall x.[f(x) \neq \perp]$
- $LOOP(f)$   $\exists x.[f(x)=\perp]$
- $EMPTY(f)$   $\forall x.[f(x)=\perp]$

A  $\propto$  B: Examples

- $HALT \propto HALT_0$   $halt(f,x) = halt_0(\lambda.f(x))$
- $LOOP \propto HALT^*$   $loop(f) = \neg halt^*(f)$
- $HALT^* \propto LOOP$
- $HALT_0 \propto LOOP$   $halt_0(f) = \neg loop(\lambda y.f(y))$

## Example of reduction

Equivalence of input-free programs ( $EQV_0$ ) is undecidable

$$eqv_0(g,h) = g \equiv h$$

Reduction:  $HALT_0 \leq EQV$

$$halt_0(f) := \neg eqv_0(f, \text{loopy})$$

## Rice's Theorem

*Every non-trivial problem regarding the (partial) function computed by a program is undecidable!*

*In other words, no "interesting" problem can be solved completely.*

## בעיות מעניינות

נאמר שבעיית הכרעה היא "מעניינת" אם:

1. השאילתות הן תכניות
2. היא איננה טריביאלית
3. היא סמנטית
4. היא אינה תלויה רק במספר הקלטים

## Trivial Problems

A decision problem is *trivial* if the answer is always 'yes' or always 'no' regardless of the query. For example,

$$TRIV(f) = F$$

$$TRIV(f) = \forall g. f(g) \equiv [g() = \perp]$$

$$TRIV(f) = [f() = \perp \vee f() \neq \perp]$$

## Semantic Problem

A *semantic* decision problem is a problem  $P$  with a *program* as input and for which the answer depends *only* on the partial function it computes:

$$f \equiv g \Rightarrow P(f) = P(g)$$

## Problems

- Is the size of this program less than 1000?
- Is this a sorting program?
- Is this quicksort?
- Is this general sorting program linear?
- Is this program as efficient as can be?
- Does this program solve the halting problem?
- Are these two homeworks “similar”?
- Is  $f(x) > x$  whenever  $f(x)$  halts?

## Semantic Problems: Examples

$f: \mathbf{D} \rightarrow \mathbf{R}$

(A program defining a *partial* function)

$$\text{EMPTY}(f) = f(\mathbf{D}) = \emptyset$$

$$\text{CONST}(f) = |f(\mathbf{D})| = 1$$

$$\text{INF}(f) = |f^{-1}(\mathbf{R})| = \infty$$

$$\text{HALT}^*(f) = f^{-1}(\mathbf{R}) = \mathbf{D}$$

## Semantic Problems: Types

Type a:  $P(\text{loopy}) = \mathbf{F}$

$$\text{ex. } |f(\mathbf{D})| = 1 \quad (\text{CONST})$$

$$\text{ex. } |f^{-1}(\mathbf{R})| = \infty \quad (\text{INF})$$

Type b:  $P(\text{loopy}) = \mathbf{T}$

$$\text{ex. } f(\mathbf{D}) = \emptyset \quad (\text{EMPTY})$$

$$\text{ex. } |f(\mathbf{D})| \leq 1 \quad (\text{NOT MULTIVALUED})$$

### Rice's Theorem: Proof (a)

- Type a:  $P(\text{loopy}) = F$
- Let  $\text{yea}$  be a program for which  $P(\text{yea})=T$
- Reduction:  $\text{HALT}_0 \approx P$

$\text{halt}_0(f) :=$

$$P \left( \lambda y. \begin{cases} \text{yea}(y) & f()=f() \\ \text{yea}(y) & \text{o/w} \end{cases} \right)$$

where  $\text{yea} = \lambda \dots$

### Rice's Theorem: Proof (b)

- Type b:  $P(\text{loopy}) = T$
- Let  $\text{nay}$  be a program for which  $P(\text{nay})=F$
- Reduction:  $\text{HALT}_0 \approx P$

$\text{halt}_0(f) :=$

$$\neg P \left( \lambda y. \begin{cases} \text{nay}(y) & f()=f() \\ \text{nay}(y) & \text{o/w} \end{cases} \right)$$

where  $\text{nay} = \lambda \dots$

### Representations

In languages that do not allow programs as parameters, one needs to encode the programs:  
 $I(\langle f \rangle, x) = \langle f(x) \rangle$ .

For example, if the language allows only numbers, one could use an enumeration of programs:

$I(n, x)$  computes  $f_n(x)$ , where  $f_n$  is the  $n$ th program in the language.

For TM,  $\langle M \rangle$  is some string encoding of  $M$ .

### Computation

A *computation* for program  $f$  and input  $\underline{x}$  is a (finite) list of terms (*configurations*)

$$[f(\underline{x}), t_1, t_2, \dots]$$

such that

- $t_{i+1}$  is obtained from  $t_i$  by
  - replacing next function call with its body, or
  - by applying a builtin function to its arguments,
- ends with an “answer”.

## Example of Computation: 1!

```

      [ 1!,
        if 1=0 then 1 else 1·(1-1)!,
          if F then 1 else 1·(1-1)!,
            1·(1-1)!,
          1·(if (1-1)=0 then 1 else (1-1)·((1-1)-1)!),
            1·(if 0=0 then 1 else (1-1)·((1-1)-1)!),
              1·(if T then 1 else (1-1)·((1-1)-1)!),
                1·1
              1 ]
  
```

## Checkers

A *checker* is a program that checks whether a list represents a valid computation:

Check( $f, \underline{x}, c$ ) =

“ $c$  is a halting computation of  $f(\underline{x})$ ”

## Checker $\Rightarrow$ Interpreter

$I(f, x) :=$

last(seq(minst( $\lambda n.$ Check( $f, x, \text{seq}(n)$ ), 0)))

where

seq enumerates sequences of terms

last gives last element of sequence

minst( $p, 0$ ) gives first  $n$  such that  $p(n)$

minst( $p, n$ ) = if  $p(n)$  then  $n$  else minst( $p, n+1$ )

## Undecidability Results

It is undecidable if two *halting* single-input programs ever compute the same value:

$\text{halt}_0(f) := \text{same}_h(\lambda c. \text{Check}(f, c), \lambda c. T)$

It is undecidable if two *halting* single-input programs always compute the same value:

$\text{halt}(f, x) := \neg \text{eqv}_h(\lambda n. \text{Stop}(f, x, n), \lambda n. F)$

## Partial Evaluator

A *partial evaluator* is a program  $X$  that takes as input a program  $f$  (in one language) and a partial input  $x$  and returns a function  $\underline{f}$  (in another language), such that

$$\underline{f} = X(f, x) = \lambda y. f(x, y)$$

A partial evaluator can be very useful if it simplifies the computation by specializing it for the given input value  $x$ .

## Uncomputability

- A programming language  $P$  in which one can write an interpreter (and other basic things) must include non-terminating programs.

Consider  $c(n) := I(P, n) + 1$

- A language with non-terminating programs (and with standard operations) must have an undecidable halting problem.

Consider  $c(f) := \text{if } \text{halt}_P(f, f) \text{ then } \text{loop}_P() \text{ else } T$

## Totality

Any programming language in which one can only write halting programs cannot compute all computable functions.

In particular, if one has basic operations (like function composition), then one cannot write an interpreter within that language.

## Primitive Recursion

A functional program is *primitive recursive* if it is constructed from

- basic arithmetic on natural numbers
- function composition
- recursive definitions of the form

$$f(n, x, \dots, z) = \begin{cases} g(x, \dots, z) & n=0 \\ h(f(n-1, x, \dots, z), n-1, x, \dots, z) & \text{o/w} \end{cases}$$

## Oracles

An oracle is a magic predicate that answers undecidable questions (like *halt*), and may be used to extend the class of (Turing machine or C) programs.

## Recursive Enumerability

- A set  $L$  is *semi-decidable* if  $\lambda x.(x \in L)$  is partially computable (answers T for all  $x \in L$ )
- A set  $L$  ( $\neq \emptyset$ ) is *recursively enumerable* (r.e.) if there's an enumeration algorithm  $\lambda: \mathbb{N} \rightarrow L$  (onto, but not necessarily 1-1)
- There's a program (in C) that prints all  $x \in L$

## Semi-Decidability (2 versions)

Let  $L$  be some set. The following are equivalent:

- There's a program  $P$  such that
$$P(x) = T \text{ iff } x \in L$$
- There's a program  $P'$  such that
$$P'(x) = T \text{ if } x \in L \text{ and}$$
$$P'(x) = \perp \text{ if } x \notin L$$

## Semi-decidable Problems

TM  $M$  halts on empty tape

- Change all  $M$ 's halting states to accepting states

Program  $f$  halts on  $x$

- Let  $\text{halt}(f,x) := [f(x)=f(x)]$
- If  $f(x)$  halts, then this program answers T
- If it doesn't, this program never answers anything

## RE Example

- Input: *partial*  $f: \mathbb{N} \rightarrow \mathbb{N}$ ,  $y \in \mathbb{N}$
- Question: Is there an  $x$  such that  $f(x)=y$ ?

Invert( $f, y$ ) := In( $f, 0, 0, 0, y$ ) where

*In( $f, x, n, m, y$ ) := Step( $f, 0, 0$ )= $y$  or Step( $f, 1, 0$ )= $y$  or  
Step( $f, 0, 1$ )= $y$  or Step( $f, 2, 0$ )= $y$  or Step( $f, 1, 1$ )= $y$  or ...*

$$\text{In}(f, x, n, m, y) := \begin{cases} \text{T} & \text{Step}(f, x, n) = y \\ \text{In}(f, x-1, n+1, m, y) & n < m \\ \text{In}(f, m+1, 0, m+1, y) & \text{o/w} \end{cases}$$

## Non-r.e. Example

Problem: SELF

Input: Scheme program

$X$ : Scheme  $\rightarrow$  Scheme

Question:  $X(X) = \perp$  ?

Suppose *Self* semi-decides this problem.

$\text{Self}(\text{Self}) = \text{T} \Rightarrow \text{SELF}(\text{Self}) = \text{F}$

$\text{Self}(\text{Self}) = \perp \Rightarrow \text{SELF}(\text{Self}) = \text{T}$

## Enumerability $\Rightarrow$ Semidecidable

Suppose  $L = \{\lambda(0), \lambda(1), \dots\}$ .

Define

$L(x) := L'(x, 0)$

where

$$L'(x, n) := \begin{cases} \text{T} & x = \lambda(n) \\ L'(x, n+1) & \text{o/w} \end{cases}$$

## Semidecidable $\Rightarrow$ Enumerability

Suppose we have a program for

$L(x) = \text{if } x \in L \text{ then T else } \perp$

And an enumerator  $w_i$  for  $w_0, w_1, w_2, \dots$

We can use a “stepper”

$L^n(x) = \text{“}L(x) \text{ returns T in within } n \text{ steps”}$

Then we can search:

$\lambda(n) = \text{argument } w_i \text{ of } n\text{th T in the sequence}$

$L^0(w_0), L^1(w_0), L^0(w_1), L^2(w_0), L^1(w_1), L^0(w_2), \dots$

## Semidecidable $\Rightarrow$ Enumerability

$\lambda(n) := \lambda'(n,0,0,0)$  where

$$\lambda'(n,i,x,m) := \begin{cases} w(x) & \text{Step}(L,w(x),i) \wedge n=0 \\ \lambda'(n-1,i,x,m) & \text{Step}(L,w(x),i) \\ \lambda'(n,i+1,x-1,m) & i < m \\ \lambda'(n,i,m+1,m+1) & \text{o/w} \end{cases}$$

## Complement

- If B is a problem, its *complement*  $\bar{B}$  is the set of all elements not in B

$$\bar{B}(x) \Leftrightarrow \neg B(x)$$

$$x \in \bar{B} \Leftrightarrow x \notin B$$

- Problem B is *co-r.e.* if its complement is r.e.

$$B \in \text{co-r.e.} \Leftrightarrow \bar{B} \in \text{r.e.}$$

## Theorems

- A decision problem is recursive *iff* its complement is

$$B \in \text{R} \Leftrightarrow \bar{B} \in \text{R}$$

- A decision problem is recursive *iff* it and its complement are both recursively-enumerable

$$B \in \text{R} \Leftrightarrow B, \bar{B} \in \text{r.e.}$$

$$B \in \text{R} \Leftrightarrow \bar{\bar{B}} \in \text{R}$$

$$\bar{\bar{B}} \propto B \quad \bar{\bar{\bar{B}}} = B$$

$$\bar{\bar{B}}(x) := \neg B(x)$$

## $B, \bar{B} \in \text{r.e.} \Rightarrow B \in \text{R}$

Suppose  $B_T$  tests for yes-B and  $B_F$  tests for not-B.  
Run  $B_F$  and  $B_T$  “in parallel”.

$B(x) := B'(0,x)$  where

$$B'(n,x) := \begin{cases} T & \text{Step}(B_T, x, n) = T \\ F & \text{Step}(B_F, x, n) = T \\ B'(n+1, x) & \text{o/w} \end{cases}$$

## Reductions

- General (Turing) reduction:  $A \leq B$

$$A(x,y) := k[B(g(x,y)), B(h(x,y))]$$

- Mapping (many-one) reduction:  $A \leq_m B$

$$A(x,y) := B(g(x,y))$$

## Mapping Reduction

$$A \leq_m B \\ A(x,y) := B(g(x,y))$$

- To justify, show:
  - $g$  is computable
  - $A(x,y) = T$  iff  $B(g(x,y)) = T$
- If  $B$  is r.e., then so is  $A$
- If  $A$  is not r.e., then neither is  $B$

## Mapping Reduction Example

$$\text{Halt} \leq_m \text{Halt}_0 \\ \text{halt}(f,x) := \text{halt}_0(\lambda.f(x))$$

- Correctness:
  - $\lambda.f(x)$  is computable given  $f,x$
  - $\text{Halt}(f,x) = T \Leftrightarrow \text{Halt}_0(\lambda.f(x)) = T$
- Conclusion: Since  $\text{Halt}_0$  is r.e., so is  $\text{Halt}$

## Recursion Theorem

For every program  $f: Haskell \rightarrow Haskell$  there exists a single-argument function  $c$  such that

$$c \equiv f(c).$$

Proof: Let  $c = k(k)$  where  $k = \lambda w.f(w(w))$

$$c \equiv k(k) \equiv (\lambda w.f(w(w)))(k) \equiv f(k(k)) \equiv f(c)$$

\*Haskell is a purely functional language using lazy evaluation.

## Why Haskell?

- *Haskell* uses **normal-order** evaluation, not applicative order.
- When evaluating  $f(e)$ ,  $f$  is evaluated first, not  $e$ .
- For applicative order, more complicated:

Let  $c := h(k)$  where

$$k(w) := f(h(w))$$

$$h(y) := \lambda z.y(y)(z)$$

## Recursion

Have  $c_f = k(k)$  where  $k = \lambda w.f(w(w))$

Suppose  $f$  is the *body* of a recursive definition:

$$f(g) = \lambda y. \text{if } A(y) \text{ then } D(y) \text{ else } E(g,y)$$

Then  $c_f$  is the recursive function itself:

$$c_f(y) \Rightarrow k(k)(y) \Rightarrow (\lambda w.f(w(w)))(k)(y) \Rightarrow f(k(k))(y)$$

$$\Rightarrow (\lambda y. \text{if } A(y) \text{ then } D(y) \text{ else } E(k(k),y))(y)$$

$$\Rightarrow (\text{if } A(y) \text{ then } D(y) \text{ else } E(k(k),y)) \Rightarrow$$

- $D(y)$
- $E(c_f,y)$

## Example: Factorial

- $f(g) = \lambda x. \text{if } x=0 \text{ then } 1 \text{ else } x.g(x-1)$
- $c(f) \Rightarrow \lambda z.f(c)(z)$  where  $c=...$
- $c(0) \Rightarrow f(c)(0) \Rightarrow \text{if } 0=0 \text{ then } 1 \text{ else } \dots \Rightarrow 1$
- $c(1) \Rightarrow f(c)(1) \Rightarrow \text{if } 0=1 \dots 1.c(0) \Rightarrow 1.1 \Rightarrow 1$
- $c(2) \Rightarrow f(c)(2) \Rightarrow \text{if } 0=2 \dots 2.c(1) \Rightarrow 2.1 \Rightarrow 2$

## How to show decidability

- By sketching a decision procedure
- By showing that the problem is “finite”
- By reduction to known decidable problems
- By showing that both it and its complement are semi-decidable

## Finite Problems

- Problems with only a finite number of different queries are always decidable.
- Problems with only a finite number of queries for which the answer is *yes* are always decidable.
- Problems with only a finite number of queries for which the answer is *no* are always decidable.

## How to show undecidability

- By *diagonalization* (as we did for Busy Beaver and Halting Problem)
- By general *reduction* from known undecidable problem (like HALT, HALT<sub>0</sub> or HALT\*)
- By applying *Rice's Theorem* for semantic problems
- By imitating the proof of Rice's Theorem

## How to show semi-decidability

- By sketching a semi-decision procedure (like for the Halting Problem)
- By reduction to a known semi-decidable problems (like HALT<sub>0</sub>)

### **How to show non-r.e.**

- By “diagonalization” (as for  $\text{HALT}^*$ )
- By mapping reduction from known non-r.e. problem (like  $\text{HALT}^*$ )
- Show that:
  - Its complement is r.e.
  - It is not decidable