Reducibility

Robert M. Keller
Harvey Mudd College
30 March 2005

What is this?

- Reducibility is one of the key ideas in computability and algorithms.

- It deals with the question of whether one problem can be solved or not, based on the solvability of another problem.

Why should we care?

- Oh, so maybe there's more than one unsolvable problem.

- What is the significance to me, a
  - programmer
  - software engineer
  - game developer
  - computer designer
  - mathematician
  - politician

  ?

Related, Almost-Interchangeable Terms

- **Uncomputable**: A function that can't be computed.
- **Unsolvable**: The problem of finding an algorithm for an uncomputable function.
- **Undecidable**: A language for which there is no algorithm that determines acceptance of strings in the language.

Computability deals with algorithms

- To say a problem is unsolvable does not refer to specific single **instances** of computing a function's value or a string's acceptance.

- Rather it refers to the non-existence of an **algorithm** for computing the function or accepting the language.

Unsolvability is Forever

- Unsolvability does not simply mean we **don't know** how to solve a problem at this point in time.

- It means that there will **never** be a way to solve the problem.

Famous Unsolvable Problems: Hilbert's Tenth Problem

10. Determination of the solvability of a Diophantine equation. Given a diophantine equation [e.g. $7x^2-5xy-3y^2+2x+4y-11=0$] with any number of unknown quantities and with rational integral numerical coefficients:

*To devise a **process** according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.*

Specific Instances

- Many specific instances of diophantine equations have solutions and we can show this.
- Many don’t, and we can prove this for some of them.
- The problem is we have no way of telling which is which, in general.

A Much Simpler-Looking Problem: Post's Correspondence Problem

- Create an algorithm that, with a list of pairs of strings as input $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, will determine whether or not there is a pair-wise concatenation (with repetitions allowed) in which both sides are equal:

$$x_{i1}x_{i2}\ldots x_{ir} = y_{i1}y_{i2}\ldots y_{ir}$$

where $\{i1, \ldots, ir\} \subseteq \{1, 2, \ldots, n\}$.

Specific Instances

- Many specific instances of the PCP have solutions and we can show this.
- Many don’t, and we can prove this for some of them.
- The problem is we have no way of telling which is which, in general.

Review of An Unsolvable Problem

- Previously we showed a specific partial function for which there is no algorithm:

  the divergence-testing partial function (dtp)

The Divergence-Testing Partial Function

- Recall that <M> means an encoding of TM M.
- Consider the partial function

$$\text{dtp}(\langle M \rangle) = \begin{cases} 1 \text{ if M diverges on input } \langle M \rangle \\ \text{undefined otherwise} \end{cases}$$

- (Note: "divergent" and undefined result are the same thing.)

- By the way, we often use this functional notation when we really mean that dtp **accepts** the input, lieu of literally giving result 1.

Specific Instances

- Pick your favorite Turing machine M.
- One reason it's probably your favorite is that you can establish the value of

  dtp(<M>)

  through your own reasoning power.
- This does not mean that you have an algorithm for computing dtp.

Specific Instances

- Consider the decimal expansion of π:

  3.14159265…

- Define p(x) = 1 if 10 5's occur consecutively in the expansion and 0 otherwise.

- (yes, p ignores x entirely)

- Is p computable?

A Related Problem: The Halting Problem

- Consider the partial function

$$\text{halts}(\langle M \rangle) = \begin{cases} 1 \text{ if } M \text{ halts on input } \langle M \rangle \\ 0 \text{ otherwise} \end{cases}$$

- It is no big surprise that this function isn't computable.
- However, we have in mind a particular way of showing this fact.

Shorthand Notation

- Instead of saying “M diverges on input x”

  we’ll just say “M(x) diverges”, etc.

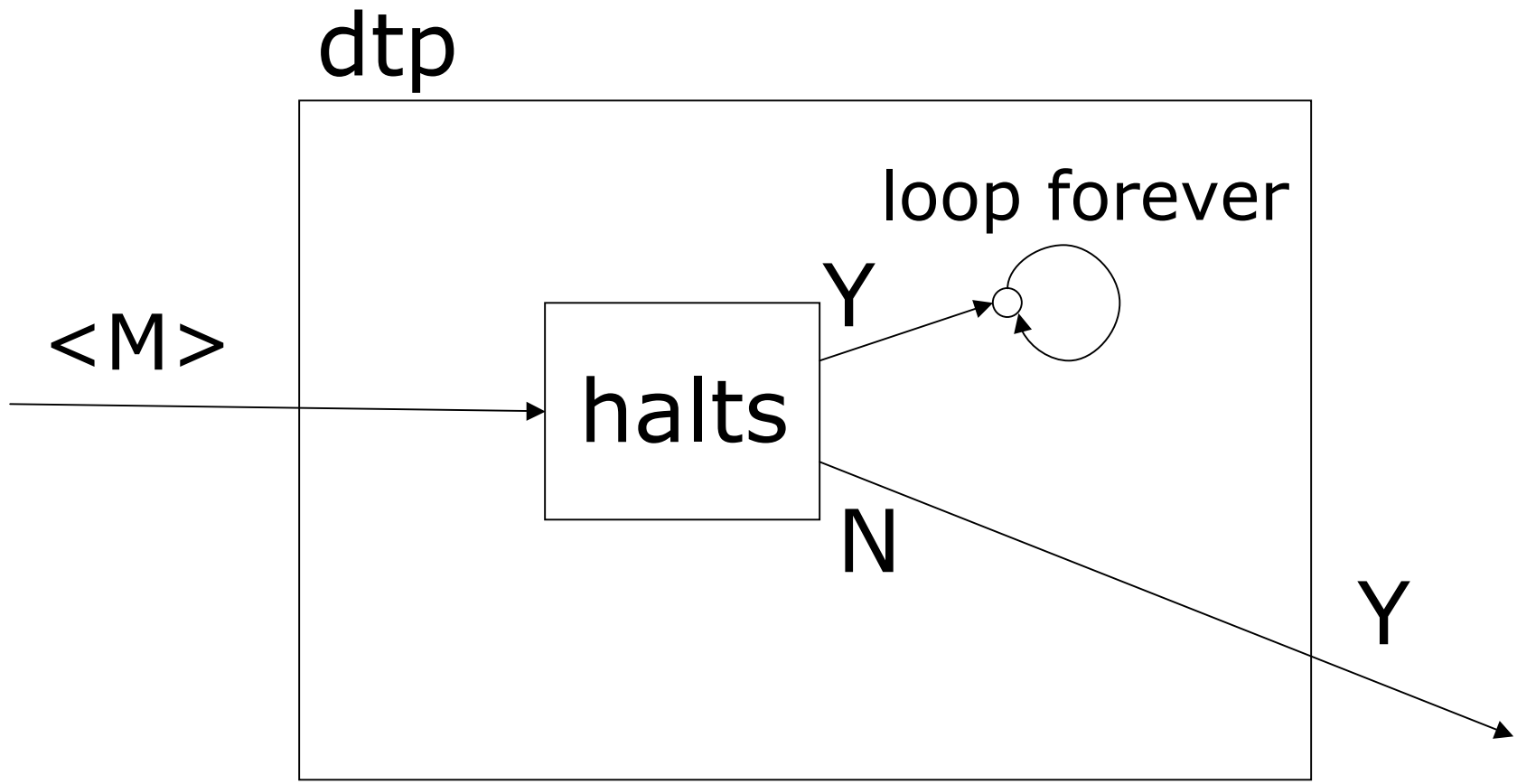
  as in “M(<M>) diverges”,

  and similarly for “halts” in place of “diverges”.

Modifying an Algorithm

- **If** we could implement halts, **then** we could also implement dtp.
- The next slide shows how.
- However, we already showed that we **can't** implement dtp.
- Therefore we can't implement halts either.
- (We are using contrapositive reasoning: Prove $\neg D \rightarrow \neg H$ by proving $H \rightarrow D$.)

Implementing dtp using halts



Same Idea Using Function Notation

$$\text{halts}(\langle M \rangle) = \begin{cases} 1 \text{ if } M(\langle M \rangle) \text{ halts} \\ 0 \text{ if } M(\langle M \rangle) \text{ diverges} \end{cases}$$

$$\text{dtp}(\langle M \rangle) = \begin{cases} 1 \text{ if } M(\langle M \rangle) \text{ diverges} \\ \text{undefined otherwise} \end{cases}$$

Thus we have an implementation:

$$\text{dtp}(\langle M \rangle) = \begin{cases} 1 \text{ if halts}(\langle M \rangle) = 0 \\ \text{undefined otherwise} \end{cases}$$

Reduction

- We can summarize our argument as follows:

  "The divergence testing problem **effectively reduces** to the halting problem."

  If the halting problem were solvable, so would the divergence testing problem be.

  The divergence testing problem isn't solvable, so neither is the halting problem.

Effective?

- The word “effective” means that the connection between the two problems is **algorithmic** in nature.
- All we used in the proof that halts is uncomputable (in addition to dtp) was an equality test and a forced loop, two things we can easily do in an algorithm.
- Often we will omit the word “effective” but it is still implied.

Notation for Reduction

- If problem A effectively reduces to problem B, then write:

  $A \leq B$

- In some sense, B is "at least as hard" as A, maybe harder.
- Note that this relation is reflexive and transitive.
- If $A \leq B$ but not $B \leq A$, then write $A < B$.
- If $A \leq B$ and $B \leq A$, this does not imply that A and B are the same problem, just that they are **inter-reducible** or "equally computable".

Another Example

- Define

$$\text{halts2}(\langle M \rangle, x) = \begin{cases} 1 \text{ if TM } M \text{ halts on } x \\ 0 \text{ otherwise} \end{cases}$$

- Is there an algorithm for halts2?
- Why or why not?

Yet Another Example

- Define

$$\text{halts2p}(\langle M \rangle, x) = \begin{cases} 1 \text{ if TM M halts on x} \\ \\ \text{undefined otherwise} \end{cases}$$

- Is there an algorithm for halts2p?
- Why or why not?

A More Subtle Example

- Define

$$\text{haltsbt}(\langle M \rangle) = \begin{cases} 1 \text{ if TM M halts on an all-blank tape} \\ 0 \text{ otherwise} \end{cases}$$

- Is there an algorithm for haltsbt?

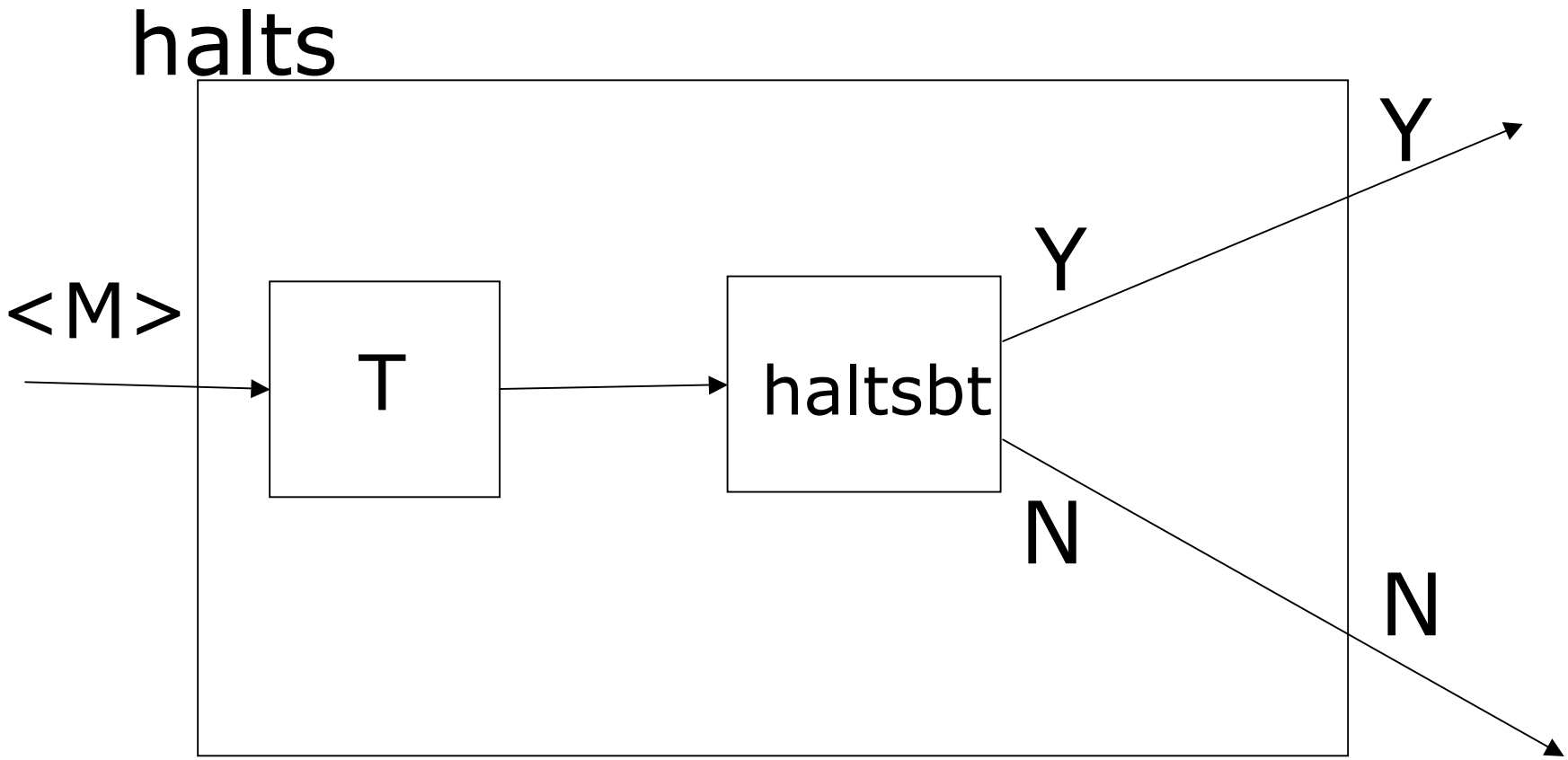
Wrong-Direction Reduction

- It is clear that haltsbt $\leq$ halts2. (Why?)
- However, this fact does **not** help establish that haltsbt is computable, because halts2 **isn't** computable.
- Also, it doesn't help establish that haltsbt isn't computable either. For that we'd need an unsolvable problem P for which P $\leq$ haltsbt.

Transforming a Function's Argument

- We'll show that halts $\leq$ haltsbt.
- The nuance here is that in implementing halts using haltsbt, we will use a device of **transforming the *argument*** to halts.
- The next page shows the setup we'd like to achieve.

Reducing halts to haltsbt



What is T?

- Recall that

$$\text{halts}(\langle M \rangle) = \begin{cases} 1 \text{ if } M(\langle M \rangle) \text{ halts} \\ 0 \text{ if } M(\langle M \rangle) \text{ diverges} \end{cases}$$

- whereas

$$\text{haltsbt}(\langle M \rangle) = \begin{cases} 1 \text{ if } M(\text{blank}) \text{ halts} \\ 0 \text{ if } M(\text{blank}) \text{ diverges} \end{cases}$$

- we want T such that

$$\text{halts}(\langle M \rangle) = \text{haltsbt}(T(\langle M \rangle))$$

What is T?

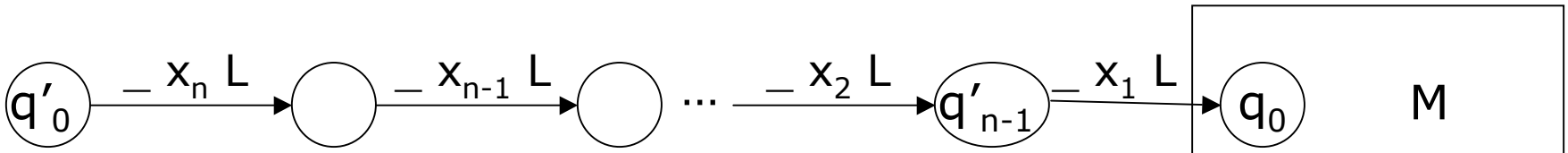
- T must modify <M> in such a way that
  - T(<M>) halts on a blank tape iff
  - M halts on <M>
- Here's how T does its job:
- With <M> as input, T constructs the description of a new machine M' that works as follows:
  - M' writes the description <M> on its tape, from right to left.
  - Then M' behaves exactly as M would have, from that point on.
- So M' halts on a blank tape iff M halts on <M>.
- T(<M>) = the description <M'> as constructed.

Notes about T

- The machine M’ constructed is dependent on M in two ways:
  - M’ writes the description of M on its tape.
  - M’ eventually behaves like M.
- T produces the description of M’ without itself executing any part of M. T is impervious to what M does.
- The transformation done by T is effective.

What is Actually Required in M'?

- M' has to write a finite string on the tape.
- If the string is length n, this can be done by adding n new states, with appropriate left-moving transitions at the front of the transitions for M.
- Suppose the description of M is $x_1\, x_2 \ldots x_n$, where $x_i \in \Gamma$ (the tape alphabet), _ is the blank symbol, and q_0 is the initial state of M. Then M' looks like:



new states added

q'_0 is the initial state of M'

The haltsSome problem

- Define

$$\text{haltsSome}(\langle M \rangle) = \begin{cases} 1 \text{ if M halts on } \textit{some} \text{ tape} \\ 0 \text{ otherwise} \end{cases}$$

- What is your intuition about this function?
- What kind of reduction would you seek, if any?

The haltsSomeP problem

- Define

$$\text{haltsSomeP}(\langle M \rangle) = \begin{cases} 1 \text{ if M halts on } \textit{some} \text{ tape} \\ \text{undefined otherwise} \end{cases}$$

- What is your intuition about this function?
- What kind of reduction would you seek, if any?

The haltsAll problem

- Define

$$\text{haltsAll}(\langle M \rangle) = \begin{cases} 1 \text{ if M halts on } \textit{all} \text{ tapes} \\ 0 \text{ otherwise} \end{cases}$$

- What is your intuition about this function?
- What kind of reduction would you seek, if any?

The haltsAllP problem

- Define

$$\text{haltsAllP}(\langle M \rangle) = \begin{cases} 1 \text{ if M halts on } \textit{all} \text{ tapes} \\ \text{undefined otherwise} \end{cases}$$

- What is your intuition about this function?
- What kind of reduction would you seek, if any?