Decidability: Reduction Proofs

• Basic technique for proving a language is (semi)decidable is reduction

• Based on the following principle:
  – Have problem $A$ that needs to be solved
  – If there exists a problem $B$, such that $B$’s solution will enable the solution to $A$, you can solve $A$ by
    1. Solving $B$
    2. Using $B$’s solution to solve $A$
  – I.e., $B \Rightarrow A$
  – The problem of solving $A$ has been reduced to solving $B$
  – Note that failure to solve $B$ does not preclude finding a solution for $A$
  * There may be some $C$ whose solution will enable a solution to $A$

• Reduction proofs use proof by contradiction
  – Assume you want to show that $X$ has no solution
  – Suppose that $Y$ reduces to $X$
  * Then, $(Y \text{ reduces to } X) \land X \Rightarrow Y$
  – If $Y$ is chosen with the knowledge that $Y$ cannot be solved, must conclude that
  * $X$ has no solution either

• For our purposes, ”having a solution” means that a language is (semi)decidable

• Formally
  – A reduction $R$ from $L_1$ to $L_2$ consists of one or more TMs such that
    1. If there is a TM $Oracle$ that (semi)decides $L_2$, then
    2. The TMs of $R$ can be combined with $Oracle$ to (semi)decide $L_1$
  – $L_1 \leq L_2$ denotes that $L_1$ is reducible to $L_2$
  – Reduction often referred to as Turing reducibility, since it is defined in terms of TMs
Decidability: Reduction Proofs (2)

- Steps of a reduction proof

Given language $L_2$, where want to show that $L_2 \notin D$ (or $SD$)

1. Choose language $L_1$ such that
   
   $L_1 \leq L_2$
   
   $L_1 \notin D$ (or $SD$)

2. Define the reduction $R$ from $L_1$ to $L_2$
   
   - Assume Oracle exists for deciding $L_2$
   - Describe the composition $C$ of $R$ and Oracle
   - $C$ will be used to decide $L_1$

3. Demonstrate that $C$ does, in fact, decide $L_1$
   
   - Must show that
     
     (a) $R$ can be implemented as one or more TMs
     (b) $C$ is correct:
       
       i. If $x \in L_1 \Rightarrow, C(x)$ does accept
       ii. If $x \notin L_1 \Rightarrow, C(x)$ does not accept

4. Conclude that cannot (semi)decide $L_2$, since cannot (semi)decide $L_1$, and hence proposed Oracle cannot exist
Decidability: Reduction Proofs (3)

- Common error is to go in the wrong direction
  - Instead of reducing from language known to be undecidable to the one in question, mistakenly reduce from the one in question to the one known to be undecidable
  - E.g.,
    Want to show $A$ is undecidable and know that $B$ is undecidable
      * Proper way:
        1. Show that $B$ reduces to $A$ (i.e., deciding $A$ means we can decide $B$), and
        2. Assume there is a procedure for deciding $A$
        3. Show that this procedure allows us to decide $B$
           - But that would contradict what we already know about $B$
        4. Conclude that the hypothesized procedure for deciding $A$ cannot exist
      * Incorrect approach: Reduce from $A$ to $B$ (i.e., deciding $B$ means we can decide $A$),
        - The situation ultimately says nothing about $A$’s decidability
        - We know that $B$ cannot be decided, but maybe there is a $C$ that will allow us to decide $A$

- Reduction often accomplished by mapping instances of $A$ to instances of $B$
  - Then can use decision procedure of $B$ on transformed instances of $A$

- Given alphabet $\Sigma$
  - $L_1$ is mapping reducible to $L_2$, denoted $L_1 \leq_m L_2$, iff there is some computable function $f$ such that
    $$\forall x \in \Sigma^* (x \in L_1 \iff f(x) \in L_2)$$
  - Allows you to ask "Is $f(x)$ in $L_2$" rather than "Is $x$ in $L_1$"
  - If $f$ can be computed by TM $R$, then $R$ is a mapping reduction from $L_1$ to $L_2$
  - If $L_1 \leq_M L_2$, and there is a TM Oracle that decides $L_2$, then
    * Composition $C(x) = \text{Oracle}(R(x))$ will decide $L_1$
Decidability: Proofs - Theorem 21.1

• Statement:

  Language $H_\epsilon = 
  \{ < M >: \text{TM } M \text{ halts on } \epsilon \} \in SD$, but $\notin D$

• Proof ($H_\epsilon \in SD$):
  
  – Construct $T(< M >)$ as follows:
    1. Run $M$ on $\epsilon$
    2. Accept
  – If $M$ halts on $\epsilon$, $T$ accepts $M$, and therefore $T$ semidecides $M$

• Proof ($H_\epsilon \notin D$): Let $R$ be mapping reduction from $H$ to $H_\epsilon$
  
  – Construct $R(< M, w >)$ as follows
    1. Construct encoding $< M# >$ of TM $M#(x)$ that, on input $x$,
       * Erases the tape
       * Writes $w$ onto the tape
       * Runs $M$ on $w$
    2. Return $< M# >$
  – Construction details
    1. Write code to erase tape
       \[
       \begin{array}{c}
       R \rightarrow R \rightarrow \square \\
       \downarrow \\
       \end{array}
       \]
    2. for (each c in w) {
          write c;
          if (c not last character of w)
            write R;
      }
    3. write $L\square M$
Decidability: Proofs - Theorem 21.1 (2)

– Correctness of $C$
  * Suppose $Oracle$ for deciding $H_\epsilon$ exists
  * $M#$ ignores its own input and halts on everything or nothing
    1. If $M$ halts on $w$, $M#$ halts
    2. If $M$ does not halt on $w$, $M#$ does not halt
  * If $< M, w > \in H$, $M$ halts on $w$ and $M#$ halts on everything (including $\epsilon$)
    $Oracle(< M# >)$ does accept
  * If $< M, w > \notin H$, $M$ does not halt on $w$ and $M#$ halts on nothing
    $Oracle(< M# >)$ does not accept
– But then $Oracle(R(< M, w >))$ decides $H$
  * But since $H \notin D$, $Oracle$ cannot exist, and therefore $H_\epsilon \notin D$
Decidability: Proofs - Points to Emphasize

• Decidability reduction proof involves two kinds of languages
  1. Encodings of TMs
  2. Language that a TM accepts
     – This is input as a string to TM that accepts or rejects the string

• Proof involves five types of TMs
  1. Oracle - the hypothesized decision TM
  2. $R$
  3. $C$ - the composition Oracle($R$)
  4. $M#$ - whose encoding is the argument to Oracle
  5. $M$ - TM of interest, whose encoding is argument to $R$
Decidability: Proofs - Theorem 21.2

- **Statement:**
  
  Language $H_{any} =$
  
  \{< M >: \text{There exists at least one string on which TM } M \text{ halts}\} \in SD, \text{ but } \notin D$

- **Proof ($H_{any} \in SD$):**
  
  - Construct $T(< M >)$ as follows
    1. Use dovetailing to generate strings in $\Sigma^*$
    2. Run $M$ on these strings in parallel, one step at a time
    3. If $M$ accepts any string, $T$ accepts

  - Since $T$ halts and accepts whenever $M$ does, $T$ semidecides $H_{any}$

- **Proof ($H_{any} \notin D$):** Let $R$ be mapping reduction from $H$ to $H_{any}$
  
  - Construct $R(< M, w >)$ as follows
    1. Construct encoding $< M\# >$ of TM $M\#(x)$ that, on input $x$,
      (a) If $x == w$, runs $M$ on $w$
      (b) Loops otherwise
    2. Return $< M\# >$

  - **Correctness of $C$**
    * Assume *Oracle* for deciding $H_{any}$ exists
    * $M\#$ can only halt on string $w$
      1. If $< M, w > \in H, M$ halts on $w$, and $M\#$ halts
         \hspace{1cm} *Oracle*($< M\# >$) does accept
      2. If $< M, w > \notin H, M$ does not halt on $w$, and $M\#$ does not halt
         \hspace{1cm} *Oracle*($< M\# >$) does not accept

  - *Oracle*($R(< M, w >)$) decides $H$
    * But since $H \notin D$, *Oracle* cannot exist, and therefore $H_{any} \notin D$
Decidability: Proofs - Theorem 21.2 (2)

- Alternative proof ($H_{any} \notin D$):
  - Construct mapping reduction $R(<M, w>)$ from $H$ to $H_{any}$ as follows
    1. Construct encoding $<M\#>$ of TM $M\#(x)$ that, on input $x$,
      (a) Erases tape
      (b) Writes $w$ on tape
      (c) Runs $M$ on $w$
    2. Return $<M\#>$
  - Correctness of $C$
    * Assume $Oracle$ for deciding $H_{any}$ exists
    * $M\#$ can only halt on string $w$
      1. If $<M, w> \in H$, $M$ halts on $w$, $M\#$ halts on everything
         - $Oracle(<M\#>)$ does accept
      2. If $<M, w> \notin H$, $M$ does not halt on $w$, $M\#$ does not halt
         - $Oracle(<M\#>)$ does not accept
  - $Oracle(R(<M, w>))$ decides $H$
    * But since $H \notin D$, $Oracle$ cannot exist, and therefore $H_{any} \notin D$
Decidability: Proofs - Theorem 21.3

- Statement:

  Language $H_{all} =$
  $$\{<M>: \text{TM } M \text{ halts on } \Sigma^*\} \notin D$$

- Proof: Let $R$ be mapping reduction from $H_\epsilon$ to $H_{all}$
  - Construct $R(<M>)$ as follows
    1. Construct encoding $<M\#>$ of TM $M\#(x)$ that, on input $x$,
       * Erases the tape
       * Runs $M$
    2. Return $<M\#>$
  - Correctness of $C$
    * Assume Oracle for deciding $H_{all}$ exists
    * $C = Oracle(R(<M>))$ exists and is correct
      1. If $<M> \in H_\epsilon$, $M$ halts on $\epsilon$ and $M\#$ halts on everything
         $Oracle(<M\#>)$ does accept
      2. If $<M> \notin H_\epsilon$, $M$ does not halt on $\epsilon$ and $M\#$ halts on nothing
         $Oracle(<M\#>)$ does not accept
  - Then $Oracle(R(<M,w>))$ decides $H_\epsilon$
    * But since $H_\epsilon \notin D$, Oracle cannot exist, and therefore $H_{all} \notin D$
Decidability: Proofs - Theorem 21.4

• Statement:

Language $A =
\{ <M, w>: M \text{ is a TM and } w \in L(M) \} \notin D$

• Discussion:

– Would be easy to define $R(<M, w>)$ as simply \texttt{return \langle M, w \rangle};

– But it will not work

  * If $<M, w> \in H$, then $M$ halts on $w$
    
      · If it halts and accepts, $\text{Oracle}(<M, w>)$ will also accept
      
      · If it halts and does not accept, $\text{Oracle}(<M, w>)$ will also not accept

  * Therefore, cannot guarantee that $\text{Oracle}(<M, w>)$ only accepts whenever $M$ halts on $w$

• Proof: Let $R$ be mapping reduction from $H$ to $A$

  – Construct $R(<M, w>)$ as follows
    
    1. Construct encoding $<M \#>$ of TM $M\#(x)$ that, on input $x$,
       
       * Erases the tape
       
       * Writes $w$ to the tape
       
       * Runs $M$ on $w$
       
       * Accepts
    2. Return $<M \#, w>$

  – Correctness of $C$

    * Assume $\text{Oracle}$ for deciding $A$ exists
    
    * $C = \text{Oracle}(R(<M, w>))$ exists and is correct
      
      1. If $<M, w> \in H$, $M$ halts on $w$ and $M\#$ accepts everything
         
         $\text{Oracle}(<M\#, w>)$ does accept
      2. If $<M, w> \notin H$, $M$ does not halt on anything and $M\#$ halts on nothing
         
         $\text{Oracle}(<M\#, w>)$ does not accept

  – Then $\text{Oracle}(R(<M, w>))$ decides $H$

    * But since $H \notin D$, $\text{Oracle}$ cannot exist, and therefore $A \notin D$
Decidability: Proofs - Theorems 21.5, 21.6, 21.7

• Theorem 21.5
  - Statement:
    Language $A_{\text{epsilon}} = \{ <M> : \text{TM } M \text{ halts on } \epsilon \} \notin D$
  - Proof: Similar to proof of Theorem 21.1

• Theorem 21.6
  - Statement:
    Language $A_{\text{any}} = \{ <M> : \text{TM } M \text{ accepts at least one string} \} \notin D$
  - Proof: Similar to proof of Theorem 21.2

• Theorem 21.7
  - Statement:
    Language $A_{\text{all}} = \{ <M> : \text{TM } M \text{ accepts every string in } \Sigma^* \} \notin D$
  - Proof: Similar to proof of Theorem 21.3
Decidability: Proofs - Theorem 21.8

- **Statement:**
  
  Language $EqTMs = \{ <M_a, M_b>: M_a, M_b$ are TMs and $L(M_a) = L(M_b) \} \notin D$

- **Proof:** Let $R$ be mapping reduction from $A_{all}$ to $EqTMs$
  
  - Construct $R(<M>)$ as follows
    1. Construct encoding $<M#>$ of TM $M#(x)$ that, on input $x$,
        * Accepts
    2. Return $<M, M#>$
  
  - Correctness of $C$
    * Assume Oracle for deciding $EqTMs$ exists
    * $C = Oracle(R(<M>))$ exists and is correct
      1. If $<M> \in A_{all}$, $L(M) = L(M#)$
         $Oracle(<M, M#>)$ does accept
      2. If $<M> \notin A_{all}$, $L(M) \neq L(M#)$
         $Oracle(<M, M#>)$ does not accept
  
  - Then $Oracle(R(<M, w>))$ decides $A_{all}$
    * But since $A_{all} \notin D$, Oracle cannot exist, and therefore $EqTMs \notin D$
Decidability: Proofs - Theorem 21.9

• Statement:

Language \( L_2 = \{< M >: \text{TM } M \text{ accepts no even length strings}\} \notin D \)

• Discussion:

  – If use mapping reduction from \( H \) to \( L_2 \) as usual
    * Oracle will produce inverse result desired
  – Solution is to use a second TM that inverts Oracle’s output

• Proof (Not a mapping reduction): Let \( R \) be mapping reduction from \( H \) to \( L_2 \)

  – Construct \( R(< M, w >) \) as follows
    1. Construct encoding \( < M# > \) of TM \( M#(x) \) that, on input \( x \),
       * Erases the tape
       * Writes \( w \) to the tape
       * Runs \( M \) on \( w \)
       * Accepts
    2. Return \( < M# > \)
  – Let \( \neg \) be TM that inverts output of another
    * Then \( \{R, \neg\} \) reduces \( H \) to \( L_2 \)
  – Correctness of \( C \)
    * Assume Oracle for deciding \( L_2 \) exists
    * \( C = \neg \text{Oracle}(R(< M, w >)) \) exists and is correct
      1. If \( < M, w > \in H, M \text{ halts on } w \text{ and } M# \text{ accepts everything, including even length strings} \)
         \( \text{Oracle}(< M# >) \text{ does not accept, so } C \text{ does accept} \)
      2. If \( < M, w > \notin H, M \text{ does not halt on } w \text{ and } M# \text{ accepts nothing} \)
         \( \text{Oracle}(< M# >) \text{ does accept, so } C \text{ does not accept} \)
  – Then Oracle(\( R(< M, w >)) \) decides \( H \)
    * But since \( H \notin D \), Oracle cannot exist, and therefore \( L_2 \notin D \)
Decidability: ”Real” Programs

• Since ”real” programming languages are equal in power to TMs, what is undecidable wrt TMs will also be undecidable WRT real programs

• The following are not decidable:

  1. Given program $P$ and input $x$, does $P$ halt on $x$?
  2. Given program $P$, will $P$ enter an infinite loop on some input?
  3. Given program $P$ and input $x$, does $P$ ever output a 0? or anything?
  4. Given programs $P_1$ and $P_2$, is $P_1 \equiv P_2$?
  5. Given program $P$, input $x$, and variable $v$, does $P$ assign a value to $v$?
  6. Given program $P$ and code segment $S$ in $P$, does $P$ ever reach $S$ on any input?
  7. Given program $P$ and code segment $S$ in $P$, does $P$ reach $S$ on every input?

• While questions like 5, 6, and 7 are about the details of a program’s operation - and would suggest they might be decidable - they are not because they cannot be answered by inspection or by bounded simulation
Decidability: "Real" Programs - Theorem 21.12

• Statement:

Language $EqPrograms =$

\[
\{ < P_a, P_b > : P_a \text{ and } P_b \text{ are programs in programming language } PL \\
\quad \text{and } L(P_a) = L(P_b) \}
\notin D
\]

• Proof: Let $SimUM$ be simulation of Universal TM written in $PL$

  1. Construct $R(< M_a, M_b >)$ as follows
     1. Using $PL$, code $P_1(w)$ so that it invokes $SimUM(< M_a, w >)$ and returns its result
     2. Using $PL$, code $P_2(w)$ so that it invokes $SimUM(< M_b, w >)$ and returns its result
     3. Return $< P_1, P_2 >$

  2. Correctness of $C$
     * Assume $Oracle$ for deciding $EqPrograms$ exists
     * $C = Oracle(R(< M_a, M_b >))$ exists and is correct
       1. If $< M_a, M_b > \in EqTMs$, $L(M_a) = L(M_b)$
          $L(P_1) = L(P_2)$, and $Oracle(< P_1, P_2 >)$ accepts
       2. If $< M_a, M_b > \notin EqTMs$, $L(M_a) \neq L(M_b)$
          $L(P_1) \neq L(P_2)$, and $Oracle(< P_1, P_2 >)$ does not accept

  3. Then $Oracle(R(< M_a, M_b >))$ decides $EqTMs$
     * But since $EqTMs \notin D$, $Oracle$ cannot exist, and therefore $EqPrograms \notin D$
Decidability: "Real" Programs - Theorem 21.13

• Statement:

Language \( L = \{ \langle M, q \rangle : \text{TM } M \text{ reaches state } q \text{ on some input } \} \notin D \)

• Proof: Let \( R \) be mapping reduction from \( H_{\text{any}} \) to \( L \)

  – Construct \( R(\langle M \rangle) \) as follows

    1. Using \( \langle M \rangle \), encode \( \langle M# \rangle \) for TM \( M# \) that is the same as \( M \) but has a new transition that is constructed as

        * If \( M \) has transition \( ((q_1, c_1), (q_2, c_2, a)) \) where \( q_2 \neq h \) is a halting state, replace with \( ((q_1, c_1), (h, c_2, a)) \)

    2. Return \( \langle M#, h \rangle \)

  – Correctness of \( C \)

    * Assume Oracle for deciding \( L \) exists

    * \( C = \text{Oracle}(R(\langle M \rangle)) \) exists and is correct

      1. If \( \langle M \rangle \in H_{\text{any}} \), there is some string on which \( M \) halts

          · Therefore, there must be some string on which \( M# \) reached \( h \), and \( \text{Oracle}(\langle M#, h \rangle) \) does accept

      2. If \( \langle M \rangle \notin H_{\text{any}} \), there is no string on which \( M \) halts

          · Therefore, there is no string for which \( M# \) reached \( h \), and \( \text{Oracle}(\langle M#, h \rangle) \) does not accept

  – Then \( \text{Oracle}(R(\langle M \rangle)) \) decides \( H_{\text{any}} \)

    * But since \( H_{\text{any}} \notin D \), Oracle cannot exist, and therefore \( L \notin D \)
Semidecidability: Proofs - Theorems 21.15, 21.16

• Theorem 21.15
  
  – Statement:
    Language $H_{\text{any}} = \{ <M> : \text{there does not exist any string on which TM } M \text{ halts } * \} \notin SD$
  
  – Proof: Note that $\neg H_{\text{any}} = H_{\text{any}}$
    1. Theorem 21.2 proved that $H_{\text{any}} \in SD, H_{\text{any}} \notin D$
    2. If $H_{\text{any}} \in SD$, then $H_{\text{any}} \in D$ by Theorem 20.6
    3. Therefore $H_{\text{any}} \notin SD$

• Theorem 21.16
  
  – Statement:
    Language $\neg H_{\epsilon} = \{ <M> : \text{TM } M \text{ does not halt on } \epsilon \} \notin SD$
  
  – Proof: Left as exercise
Semidecidability: Proofs - Theorem 21.17

• Statement:

Language $A_{anbn} =$

\{<M>: M is a TM and $L(M) = A^nB^n, n \geq 0\} \notin SD$

• Proof (incorrect approach): Let $R$ be mapping reduction from $\neg H$ to $A_{anbn}$

  - Construct $R(<M,w>)$ as follows

    1. Construct encoding $<M#>$ of TM $M#(x)$ that, on input $x$,
       (a) Copies $x$ onto tape 2
       (b) Erases tape 1
       (c) Writes $w$ onto tape 1
       (d) Runs $M$ on $w$
       (e) Erases tape 1
       (f) *Copies $x$ onto tape 1
       (g) If $x \in A^nB^n$, accept, otherwise loop

    2. Return $<M#>$

  - Correctness of $C$

    * Assume Oracle for semideciding $A_{anbn}$ exists

    * $C = Oracle(R(<M,w>))$ exists and is correct

      1. If $M$ halts on $w$, $M#$ reaches step *, and $M#$ accepts $A^nB^n$

         $Oracle(<M#>)$ does accept

      2. If $M$ does not halt on $w$, $M#$ does nothing, and $M#$ does not accept $A^nB^n$

         $Oracle(<M#>)$ does not accept

    * But this is backwards

      • Cannot invert the result because Oracle only semidecides
      • To remedy the situation, modify $M#$ so it

        1. Accepts just $A^nB^n$ (if $M$ does not halt on $w$), or
        2. Accepts everything (if $M$ does halt on $w$)
• Proof (amended): Let $R$ be mapping reduction from $\neg H$ to $A_{anbn}$
  
  - Construct $R(<M, w>)$ as follows
    1. Construct encoding $<M\#>$ of TM $M\#(x)$ that, on input $x$,
       (a) If $x \in A^nB^n$, accept
       (b) Otherwise
          i. Erase tape 1
          ii. Write $w$ onto tape 1
          iii. *Run $M$ on $w$
          iv. Accept
    2. Return $<M\#>$
  
  - Correctness of $C$
    * Assume Oracle for semideciding $A_{anbn}$ exists
    * $C = Oracle(R(<M, w>))$ exists and is correct
      1. If $M$ does not halt on $w$, $M\#$ accepts $A^nB^n$
         · $M\#$ stalls at $*$, so accepts nothing else
         $Oracle(<M\#>)$ does accept
      2. If $M$ does halt on $w$, $M\#$ accepts everything
         $Oracle(<M\#>)$ does not accept
    - Then $Oracle(R(<M, w>))$ decides $\neg H$
      * But since $\neg H \notin SD$, Oracle cannot exist, and therefore $L_{anbn} \notin SD$
Semidecidability: Proofs - Theorem 21.18

• Statement:

Language $H_{all} =$

$\{ \langle M \rangle : \text{TM } M \text{ halts on } \Sigma^* \} \notin SD$

• Proof (incorrect approach): Let $R$ be mapping reduction from $\neg H$ to $H_{all}$

  - Construct $R(\langle M, w \rangle)$ as follows

  1. Construct encoding $\langle M\# \rangle$ of TM $M\#(x)$ that, on input $x$,

     (a) Erases the tape
     (b) Writes $w$ onto the tape
     (c) *Runs $M$ on $w$

  2. Return $\langle M\# \rangle$

  - Correctness of $C$

    * Assume Oracle for semideciding $H_{all}$ exists
    * $C = Oracle(R(\langle M, w \rangle))$ exists and is correct, and semidecides $\neg H$

      1. If $\langle M, w \rangle \in \neg H$, $M$ does not halt on $w$, and $M\#$ stalls at *, halting on nothing

         $Oracle(\langle M\# \rangle)$ does not accept

      2. If $\langle M, w \rangle \notin \neg H$, $M$ does halt on $w$, and $M\#$ halts on everything

         $Oracle(\langle M\# \rangle)$ does accept

  - But this is reverse of what is needed

    * But cannot use approach used in Theorem 21.17 - behavior would not depend on $M$’s halting on $w$
Semidecidability: Proofs - Theorem 21.18 (2)

• Proof (amended): Let $R$ be mapping reduction from $\neg H$ to $H_{all}$

  - Construct $R(<M, w>)$ as follows
    1. Construct encoding $<M\#>$ of TM $M\#(x)$ that, on input $x$,
       (a) Copies $x$ onto tape 2
       (b) Erases tape 1
       (c) Writes $w$ onto tape 1
       (d) Runs $M$ on $w$ for $|x|$ steps, or until $M$ halts
       (e) **If $M$ halts, loop
       (f) *Else halt
    2. Return $<M\#>$

  - Correctness of $C$
    * Assume Oracle for semideciding $\neg H$ exists
    * $C = Oracle(R(<M, w>))$ exists and is correct, and semidecides $\neg H$
      1. If $<M, w> \in \neg H$, $M$ does not halt on $w$
         • No matter how long $x$ is, $M$ will not halt in $|x|$ steps
         • For every $x$, $M\#$ reaches step * , and halts on everything
           $Oracle(<M\#>)$ does accept
      2. If $<M, w> \notin \neg H$, $M$ does halt on $w$
         • Requires $n$ steps
         • If $|x| < n$, $M\#$ reaches step * and halts
         • If $|x| \geq n$, $M\#$ stalls at step **
         • $M\#$ does not halt on anything
           $Oracle(<M\#>)$ does not accept
    * But since $\neg H \notin SD$, Oracle cannot exist, and therefore $H_{all} \notin SD$