Computer Science 351

Oracle Reductions

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #14

Goals for Today:

Goals for Today:

- Introduction to ***oracle reductions*** — which will provide another (more reliable) way to prove that various languages are undecidable
- A discussion of ***closure*** of various sets of languages with respect to reducibilities, and a discussion of why this is important.

Note: In my opinion, *Introduction to the Theory of Computation* ***is not*** a good reference for this material.

Reducibilities

Definition #1: A ***reducibility*** is any binary relation $\preceq_Q$ between languages (possibly over different alphabets) such that the following properties are satisfied.

(a) $L \preceq_Q L$ for every language $L \subseteq \Sigma^\star$ (and for every alphabet Σ).

(b) For all languages $L_1 \subseteq \Sigma_1^\star$, $L_2 \subseteq \Sigma_2^\star$ and $L_3 \subseteq \Sigma_3^\star$ (and alphabets Σ_1, Σ_2 and Σ_3) if $L_1 \preceq_Q L_2$ and $L_2 \preceq_Q L_3$ then $L_1 \preceq_Q L_3$.

Please consult your CPSC 251 or MATH 271 textbook (and notes) to find the definition of a “binary relation.”

Oracle Reductions

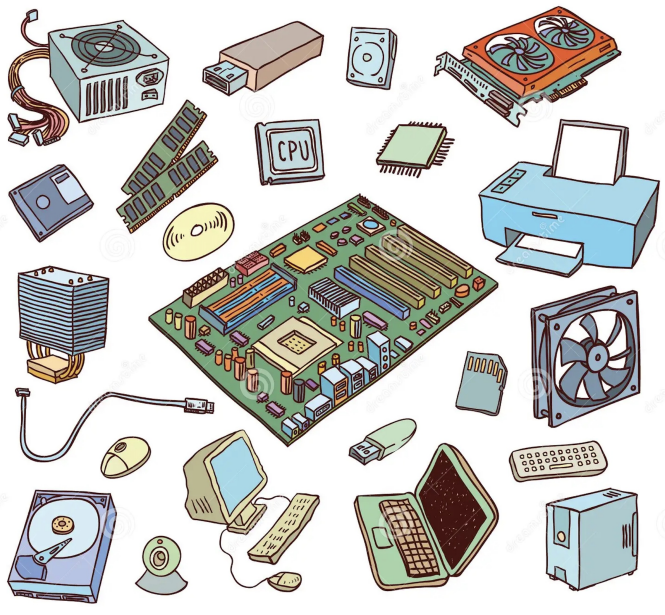
Definition #2: An ***oracle for a language $L \subseteq \Sigma^\star_L$*** is a device that is capable of reporting whether any string $\omega \in \Sigma^\star_L$ is a member of L.

- It is probably best to think of an oracle for a language $L \subseteq \Sigma^\star_L$ as being like a "black box" or ***method from a library that you can call, as a subroutine*** that can be relied on to decide membership in L.

  You do not need to write this black box (or library routine) or understand how it works — but you do get to use it.

Oracle Reductions

- It might also help to think of an oracle as being like a ***component*** in a larger system:



Each ***component*** carries out a well-defined job. You do not care how the component works, as long as it does the job it is supposed to.

Oracle Reductions

- When you are describing an "oracle reduction to a language $L \subseteq \Sigma^\star$, it will generally be sufficient to give pseudocode — for whatever problem you are trying to solve — that makes use of a separate ***Boolean method*** that decides membership in L.

  This method should receive a string $\mu \in \Sigma^\star$ as input. Its output should be *true* if $\mu \in L$ and its output should be *false* if $\mu \notin L$. That is the only information about this method that you should need to know or that you should be able to use.

  However "oracle reductions" are more formally defined (and their properties can be proved) using *oracle Turing machines*, which are defined next.

Oracle Reductions

An ***oracle Turing machine M_L with an oracle for a language $L \subseteq \Sigma_L^\star$*** is a modified deterministic multi-tape Turing machine that is allowed to query an oracle for L in a single step:

- Σ_L is a subset of the tape alphabet Γ of M_L.
- M_L has a special ***query tape*** that is initially blank.
- M_L also has three special states:
  - a ***query*** state, q_Q,
  - a ***yes*** state, q_Y,
  - a ***no*** state, q_N

  No "transitions" starting in state q_Q are defined. Instead, the behaviour or M_L when in this state is as follows.

Oracle Reductions

- Whenever M_L enters the query state and the non-blank part of the query tape stores a string $\omega \in L$ (and nothing else) then the query tape is filled with blanks and the machine moves to state q_Y using a single step. The tape head for the query tape also moves back to the leftmost cell.
- Whenever M_L enters the query state, and the query state stores something different, then the query tape is filled with blanks and the machine moves to state q_N in a single step, instead.
- M_L acts like a regular Turing machine whenever it is in a state different from q_Q.

Oracle Reductions

The definitions of

- ***accepting***, ***rejecting*** and ***looping on*** an input string,
- ***recognizing*** a language, and
- ***deciding*** a language

are all the same, for oracle Turing machines, as they are for "standard" Turing machines.

Oracle Reductions

It might seem like this kind of device is...

Magic!!!

Oracle Reductions

Well... ***in a way,*** it really is.

- It is possible to use arguments involving an oracle reduction with an oracle for L, for certain languages L, to prove that L is *undecidable* — so that there is no algorithm that can be used to decide membership in L, at all.

Oracle Reductions

Definition #3: For languages $L_1 \subseteq \Sigma_1^\star$ and $L_2 \subseteq \Sigma_2^\star$, an ***oracle reduction*** from L_1 to L_2 is an oracle Turing machine, with an oracle for L_2, that decides the language L_1.

L_1 is ***oracle-reducible*** to L_2,

$$L_1 \preceq_O L_2,$$

if there exists an oracle reduction from L_1 to L_2.

Oracle Reductions — An Example

Recall the alphabet Σ_{TM} that was introduced in Lecture #12 along with the following languages (all of which are subsets of $\Sigma^{\star}_{\text{TM}}$):

- TM+I $\subseteq \Sigma^{\star}_{\text{TM}}$ is the language of encodings of Turing machines M and input strings ω for M. As noted in Lecture #13. this language is ***decidable***.
- $\text{A}_{\text{TM}} \subseteq \Sigma^{\star}_{\text{TM}}$ is the language of encodings of Turing machines M and input strings ω for M such that M ***accepts*** ω. It was established in Lecture #12 that A_{TM} is ***recognizable*** — but it was also established in Lecture #13 that A_{TM} is ***undecidable***.

Oracle Reductions — An Example

- Let $\mathsf{NA}_{\mathsf{TM}} \subseteq \Sigma^{\star}_{\mathsf{TM}}$ be the language of encodings of Turing machines M and input strings ω for M such that M *does not* accept ω.
- It follows that that $\mathsf{A}_{\mathsf{TM}} \cup \mathsf{NA}_{\mathsf{TM}} = \mathsf{TM+I}$ and $\mathsf{A}_{\mathsf{TM}} \cap \mathsf{NA}_{\mathsf{TM}} = \emptyset$.
- Consider the algorithm — which makes use of a hypothetical method to decide membership in $\mathsf{NA}_{\mathsf{TM}}$, at line 2 — which is shown on the following slide.

Oracle Reductions — An Example

```
boolean acceptInput (μ : Σ*_TM) {
1.  if (μ ∈ TM+I)
2.    if (μ ∈ NA_TM) {
3.      reject μ
      } else {
4.      accept μ
      }
    } else {
5.    reject μ
    }
}
```

Oracle Reductions — An Example

Claim #4: Let $\mu \in A_{TM}$. Then, if the above algorithm is executed on input μ, then this input string is ***accepted***.

Proof: Let $\mu \in A_{TM}$ and consider an execution of the above algorithm on input μ.

- Since $A_{TM} \subseteq$ TM+I, $\mu \in$ TM+I and the test at at line 1 is *passed*: The algorithm's execution continues with the step at line 2.
- Since $A_{TM} \cap NA_{TM} = \emptyset$, $\mu \notin NA_{TM}$ and the test at line 2 is *failed*: The algorithm's execution continues with the step at line 4.
- The string μ is accepted when the step at line 4 is executed.

Since μ was arbitrarily chosen from A_{TM}, it follows that the algorithm accepts every string in A_{TM}, as claimed. □

Oracle Reductions — An Example

Claim #5: Let $\mu \in \Sigma^{\star}_{\mathsf{TM}}$ such that $\mu \notin \mathsf{A}_{\mathsf{TM}}$. Then, if the above algorithm is executed on input μ, then this input string is ***rejected***.

Proof: Let $\mu \in \Sigma^{\star}_{\mathsf{TM}}$ such that $\mu \notin \mathsf{A}_{\mathsf{TM}}$. Then one of the following cases holds.

(a) $\mu \in$ TM+I but $\mu \notin \mathsf{A}_{\mathsf{TM}}$.

(b) $\mu \notin$ TM+I.

The cases are considered separately on the following slides.

Oracle Reductions — An Example

(a) *Case:* $\mu \in$ TM+I but $\mu \notin A_{TM}$. Since $A_{TM} \cup NA_{TM} =$ TM+I it follows that $\mu \in NA_{TM}$. Consider an execution of the above algorithm on input μ:

- Since $\mu \in$ TM+I the test at line 1 is *passed* when checked, so that the execution of the algorithm continues with the test at line 2.
- Since $\mu \in NA_{TM}$ the test at line 2 is also *passed*, so that the execution of the algorithm continues with the step at line 3.
- The input string μ is *rejected* when the step at line 3 is executed.

Thus the string μ is rejected in this case.

Oracle Reductions — An Example

(b) *Case:* $\mu \notin$ TM+I. Consider an execution of the above algorithm on input μ.

- Since $\mu \notin$ TM+I the test at line 1 is *failed* when checked, so that the execution of the algorithm continues with the step at line 5.
- The input string μ is *rejected* when the step at line 5 is executed.

Thus the string μ is rejected in this case.

Since μ is rejected in every case, and μ was an arbitrarily chosen string in $\Sigma^{\star}_{\mathsf{TM}}$ such that $\mu \notin \mathsf{A}_{\mathsf{TM}}$, this establishes the claim. □

Oracle Reductions — An Example

Claim #6: $A_{TM} \preceq_O NA_{TM}$.

Proof: Consider an oracle Turing machine, with an oracle for NA_{TM}, that begins by making a second copy of the input string $\mu \in \Sigma^{\star}_{TM}$ on another tape, moving both tape heads back to the leftmost cells of the tape.

The oracle Turing machine could then implement the algorithm that is given above:

- Since the language TM+I is decidable, step #1 can be carried out using a multii-Tape Turing machine whose execution on an input string $\mu \in \Sigma^{\star}_{TM}$, ending in one state if $\mu \in$ TM+I and ending in a different state if $\mu \notin$ TM+I.

Oracle Reductions — An Example

- If $\mu \in$ TM+I (so that the step at line 2 would be reached) then the Turing machine's oracle for $\mathrm{NA}_{\mathrm{TM}}$ could be used to carry out this step: The "other tape" that is mentioned, above, should simply be the query tape — and the state that the Turing machine should move to, after deciding that $\mu \in$ TM, above, should be the Turing machine's query state.

  The steps at lines 3 and 4 could be implemented by having all transitions from the oracle Turing machine's "yes" state going to its "rejecting" state, and by having all transitions from the oracle Turing machine's "no" state going to its "accepting state".

Oracle Reductions — An Example

- It is sufficient to have the state that the Turing machine moves to, after deciding that $\mu \notin$ TM+I, be its "rejecting" state, in order to implement the step at line 5.
- Thus the above algorithm can be implemented using an oracle Turing machine with an oracle for $\mathrm{NA}_{\mathrm{TM}}$.
- It follows by Claims #4 and #5, above, that the algorithm decides membership in A_{TM}. Thus the oracle Turing machine that implements it decides the language A_{TM}, so that $\mathrm{A}_{\mathrm{TM}} \preceq_{\mathrm{O}} \mathrm{NA}_{\mathrm{TM}}$, as claimed. □

Oracle Reductions

Establishing an Oracle Reduction

Let $L_1 \subseteq \Sigma_1^\star$ and let $L_2 \subseteq \Sigma_2^\star$. The following ***process*** can be used to prove that $L_1 \preceq_O L_2$:

1. Describe an ***algorithm*** that has a string in $\Sigma_1^\star$ as its input and that uses a hypothetical "subroutine" that decides membership in the language L_2.
2. Prove that (if the subroutine for membership in L_2 is correct) this algorithm correctly decides membership in L_1. That is, it halts and ***accepts*** every input string $\mu \in L_1$ and it halts and ***rejects*** every input string $\mu \in \Sigma_1^\star$
3. Adding implementation details as needed, show that the algorithm can be implemented as an oracle Turing machine with an oracle for L_2.
4. Note that this oracle Turing machine decides L_1 — establishing that $L_1 \preceq_O L_2$.

"Oracle Reductions" Comprise a Reducibility

Note that "oracle reduction" comprise a *binary relation* between the set of languages and itself.

Claim #7: The set of oracle reductions forms a ***reducibility***.

- This means that $L \preceq_{\mathrm{O}} L$, for every language $L \subseteq \Sigma^\star$ (for every alphabet Σ) and that, for all languages $L_1 \subseteq \Sigma_1^\star$, $L_2 \subseteq \Sigma_2^\star$, and $L_3 \subseteq \Sigma_3^\star$ (for alphabets Σ_1, Σ_2 and Σ_3), if $L_1 \preceq_{\mathrm{O}} L_2$ and $L_2 \preceq_{\mathrm{O}} L_3$ then $L_1 \preceq_{\mathrm{O}} L_3$.
- A proof of Claim #7 is included in a supplemental document for this lecture.

Closure of the Set of Decidable Languages

Claim #8: Let $L_1 \subseteq \Sigma_1^\star$ and let $L_2 \subseteq \Sigma_2^\star$ (for alphabets Σ_1 and Σ_2). If $L_1 \preceq_O L_2$ and L_2 is decidable then L_1 is decidable too.

In other words, the set of decidable languages is ***closed*** under oracle reductions.

- A proof of Claim #8 is also included in a supplemental document for this lecture.

Closure of the Set of Decidable Languages

Corollary #9: Let $L_1 \subseteq \Sigma_1^\star$ and let $L_2 \subseteq \Sigma_2^\star$ (for alphabets Σ_1 and Σ_2). If $L_1 \preceq_O L_2$ and L_1 is ***undecidable*** then L_2 is undecidable too.

- This result — and its consequences — is the main reason why ***oracle reductions*** are part of this course.

Closure of the Set of Decidable Languages

Example, Continued:

Claim #10: The language NA_{TM} is undecidable.

Proof:

- It follows by Claim #6, above, that $A_{TM} \preceq_O NA_{TM}$.
- It was shown in Lecture #13 that the language A_{TM} is undecidable.
- It now follows by Corollary #9, above, that the language NA_{TM} is undecidable too. □

Closure of the Set of Decidable Languages

Another process to prove that a language $L \subseteq \Sigma^\star$ is undecidable:

- Choose another language $\widehat{L} \subseteq \widehat{\Sigma}^\star$ (over some alphabet $\widehat{\Sigma}$) such that $\widehat{L}$ is undecidable.
- Prove that $\widehat{L} \preceq_O L$.
- Conclude, by Corollary #9, above, that L must be undecidable too.

Non-Closure of the Set of Recognizable Languages

Claim #11: There exist languages $L \subseteq \Sigma^\star$ and $\widehat{L} \subseteq \widehat{\Sigma}^\star$ (over alphabets Σ and $\widehat{\Sigma}$) such that

- L is ***not*** recognizable,
- $\widehat{L}$ *is* recognizable, and
- $L \preceq_{\mathrm{O}} \widehat{L}$.

The the set of recognizable languages is ***not*** closed under oracle reductions.

- A supplemental document for this lecture includes a proof of this claim.
- However — if you recall and understand material from Lecture #13 and the beginning of Lecture #14 — you might be able to prove this, on your own!

Non-Closure of the Set of Recognizable Languages

- This implies that oracle reductions ***cannot*** be used to prove that languages are *unrecognizable*.
- Another kind of a reduction — “many-one reductions” — that *can* be used for this will be introduced in Lecture #15.

Oracle Reductions are Also Called "Turing Reductions"

Oracle reductions also called a ***Turing reductions*** and the notation $\preceq_T$ is sometimes used instead of $\preceq_O$.

- As this might suggest, Turing's work on computability included the development and use of these reductions.