Undecidability and Reductions

Lecture 20
April 20, 2021
Part I

TM Recap and Recursive/Decidable Languages
Turing Machine

- **DFA** with infinite tap
- One move: read, **write**, move one cell, change state

[Diagram of Turing Machine]

On a given input string $w$, a TM $M$ does one of the following:

- halt and accept $w$
- halt and reject $w$
- go into an infinite loop (not halt)
- crash in which case we think of it as rejecting $w$
Turing Machine

- **DFA** with infinite tap
- One move: read, write, move one cell, change state

On a given input string $w$ a TM $M$ does one of the following:

- halt and accept $w$
- halt and reject $w$
- go into an infinite loop (not halt)
- crash in which case we think of it as rejecting $w$
Recursive and Recursively Enumerable

**Definition**

Given TM $M$, $L(M) = \{w \in \Sigma^* \mid M$ accepts $w\}$.
We say $M$ accepts $L$.

**Caveat:** A language $L$ can be accepted by many different TMs.
Recursive and Recursively Enumerable

**Definition**

Given TM $M$, $L(M) = \{ w \in \Sigma^* | M$ accepts $w \}$. We say $M$ accepts $L$.

Caveat: A language $L$ can be accepted by many different TMs.

**Definition**

$M$ is an **algorithm** if it halts on every input and accepts/rejects.
Recursive and Recursively Enumerable

**Definition**

Given TM $M$, $L(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$. We say $M$ accepts $L$.

**Caveat:** A language $L$ can be accepted by many different TMs.

**Definition**

$M$ is an **algorithm** if it halts on every input and accepts/rejects.

**Definition**

A language $L$ is **decidable (or recursive)** if there is an algorithm $M$ such that $L = L(M)$. 

Chandra (UIUC)  CS/ECE 374  4  Spring 2021  4 / 35
Recursive and Recursively Enumerable

**Definition**
Given TM $M$, $L(M) = \{ w \in \Sigma^* \mid M$ accepts $w \}$. We say $M$ accepts $L$.

*Caveat:* A language $L$ can be accepted by many different TMs.

**Definition**
$M$ is an **algorithm** if it halts on every input and accepts/rejects.

**Definition**
A language $L$ is **decidable (or recursive)** if there is an algorithm $M$ such that $L = L(M)$.

**Definition**
A language $L$ is **recursively enumerable** if there is a TM $M$ such that $L = L(M)$.
Recursive and Recursively Enumerable

- If $L$ is recursive then $\bar{L} = \Sigma^* - L$ is also recursive
- If $L$ is recursive then $L$ is a r.e.

Question: Are r.e. languages interesting? And why?

- Technical/mathematical reasons
- Pragmatic reasons. We are used to programs that are correct, but are willing to give up on efficiency/halting.

Definition

$L$ is undecidable if there is no algorithm $M$ such that $L = L(M)$.

$L$ is not r.e. if there is no TM $M$ such that $L = L(M)$.

Recursive and Recursively Enumerable

- If $L$ is recursive then $\overline{L} = \Sigma^* - L$ is also recursive
- If $L$ is recursive then $L$ is a r.e.
- Suppose $L$ is r.e. $L = L(M)$ for some $M$.
  - If $w \in L$ then $M$ halts and accepts $w$.

Question: Are r.e. languages interesting? And why?

Technical/mathematical reasons

Pragmatic reasons. We are used to programs that are correct, but are willing to give up on efficiency/halting.

Definition

$L$ is undecidable if there is no algorithm $M$ such that $L = L(M)$.

$L$ is not r.e. if there is no TM $M$ such that $L = L(M)$. 
Recursive and Recursively Enumerable

- If $L$ is recursive then $\overline{L} = \Sigma^* - L$ is also recursive.
- If $L$ is recursive then $L$ is a r.e.
- Suppose $L$ is r.e. $L = L(M)$ for some $M$.
  - If $w \in L$ then $M$ halts and accepts $w$.
  - If $w \notin L$ then

Question: Are r.e languages interesting? And why?

Technical/mathematical reasons.

Pragmatic reasons. We are used to programs that are correct, but are willing to give up on efficiency/halting.

Definition

- $L$ is undecidable if there is no algorithm $M$ such that $L = L(M)$.
- $L$ is not r.e if there is no TM $M$ such that $L = L(M)$. 
If $L$ is recursive then $\bar{L} = \Sigma^* - L$ is also recursive.

If $L$ is recursive then $L$ is a r.e.

Suppose $L$ is r.e. $L = L(M)$ for some $M$.

- If $w \in L$ then $M$ halts and accepts $w$.
- If $w \notin L$ then $M$ may or may not halt! If $M$ halts then it rejects $w$.

Question: Are r.e languages interesting? And why?

Technical/mathematical reasons

Pragmatic reasons. We are used to programs that are correct, but are willing to give up on efficiency/halting.

Definition $L$ is undecidable if there is no algorithm $M$ such that $L = L(M)$.

$L$ is not r.e if there is no TM $M$ such that $L = L(M)$.
Recursive and Recursively Enumerable

- If $L$ is recursive then $\overline{L} = \Sigma^* - L$ is also recursive.
- If $L$ is recursive then $L$ is a r.e.
- Suppose $L$ is r.e. $L = L(M)$ for some $M$.
  - If $w \in L$ then $M$ halts and accepts $w$.
  - If $w \notin L$ then $M$ may or may not halt! If $M$ halts then it rejects $w$.

**Question:** Are r.e languages interesting? And why?
- Technical/mathematical reasons
- Pragmatic reasons. We are used to programs that are correct, but are willing to give up on efficiency/halting.
Recursive and Recursively Enumerable

- If \( L \) is recursive then \( \overline{L} = \Sigma^* - L \) is also recursive.
- If \( L \) is recursive then \( L \) is a r.e.
- Suppose \( L \) is r.e. \( L = L(M) \) for some \( M \).
  - If \( w \in L \) then \( M \) halts and accepts \( w \).
  - If \( w \notin L \) then \( M \) may or may not halt! If \( M \) halts then it rejects \( w \).

**Question:** Are r.e languages interesting? And why?

- Technical/mathematical reasons
- Pragmatic reasons. We are used to programs that are correct, but are willing to give up on efficiency/halting.

**Definition**

\( L \) is **undecidable** if there is no algorithm \( M \) such that \( L = L(M) \). \( L \) is **not r.e** if there is no TM \( M \) such that \( L = L(M) \).
Universal TM

A single TM that can simulate other TMs. Basis of modern computers. Single computer that runs many different programs.

- UTM takes as input $\langle M \rangle$ (encoding of a TM $M$) and a string $w$. Typically written as $\langle M, w \rangle$.

What is the language of UTM? Special name called Universal Language denote by $L_u$.

$L_u = \{ \langle M, w \rangle | M$ accepts $w$. $\}$.
Universal TM

A single TM that can simulate other TMs. Basis of modern computers. Single computer that runs many different programs.

- UTM takes as input $\langle M \rangle$ (encoding of a TM $M$) and a string $w$. Typically written as $\langle M, w \rangle$.
- UTM simulates $M$ on $w$.
  - If $M$ accepts $w$ then UTM accepts its input $\langle M, w \rangle$.
  - If $M$ halts and rejects $w$ then UTM rejects its input $\langle M, w \rangle$.
  - If $M$ does not halt on $w$ then UTM also does not halt on input $\langle M, w \rangle$ and hence does not accept its input.
Universal TM

A single TM that can simulate other TMs. Basis of modern computers. Single computer that runs many different programs.

- UTM takes as input $\langle M \rangle$ (encoding of a TM $M$) and a string $w$. Typically written as $\langle M, w \rangle$.
- UTM simulates $M$ on $w$.
  - If $M$ accepts $w$ then UTM accepts its input $\langle M, w \rangle$.
  - If $M$ halts and rejects $w$ then UTM rejects its input $\langle M, w \rangle$.
  - If $M$ does not halt on $w$ then UTM also does not halt on input $\langle M, w \rangle$ and hence does not accept its input.

- What is the language of UTM? Special name called Universal Language denote by $L_u$.

$$L_u = \{ \langle M, w \rangle \mid M \text{ accepts } w \}.$$
Observation

There is a fixed encoding such that every TM $M$ can be represented as a unique binary string.

Equivalently we think of a TM as simply a program which is a string.

For each string that is not a valid encoding we associate a dummy TM that does not accept any string. Why?
Encoding TMs

Observation

*There is a fixed encoding such that every TM $M$ can be represented as a unique binary string.*

Equivalently we think of a TM as simply a program which is a string.

For each string that is not a valid encoding we associate a *dummy* TM that does not accept any string. Why?

One-to-one correspondence between binary strings and TMs.

$M_i$ is the the TM associate with integer $i$
How many TMs?

One-to-one correspondence between integers and TMs.

**Proposition**

*The number of TMs is countably infinite.*
How many TMs?

One-to-one correspondence between integers and TMs.

**Proposition**

*The number of TMs is countably infinite.*

Easy but important corollaries:

- Hence, countably infinite number of r.e (hence also recursive) languages
- Number of languages is uncountably infinite! Hence there must be languages that are not r.e/recursive and hence undecidable! In fact, most languages are undecidable!
How many TMs?

One-to-one correspondence between integers and TMs.

**Proposition**

The number of TMs is countably infinite.

Easy but important corollaries:

- Hence, countably infinite number of r.e (hence also recursive) languages
- Number of languages is uncountably infinite! Hence there must be languages that are not r.e/recursive and hence undecidable! In fact, most languages are undecidable!

**Question:** Which interesting languages are undecidable/not r.e?
Part II

Undecidable Languages and Proofs via Reductions
Undecidable Languages

Counting argument shows that too many languages and too few TMs/programs hence most languages are not decidable.

What “real-world” and “natural” languages are undecidable?

**Short answer:** reasoning about general programs is difficult.

Theorem (Turing)

Following languages are undecidable.

\[ L_{\text{HALT}} = \{ \langle M \rangle \mid M \text{ halts on blank input} \} \]

\[ L_{\text{HALT}}, w = \{ \langle M, w \rangle \mid M \text{ halts on input } w \} \]

\[ L_u = \{ \langle M, w \rangle \mid M \text{ accepts } w \} \]
Undecidable Languages

Counting argument shows that too many languages and too few TMs/programs hence most languages are not decidable.

What “real-world” and “natural” languages are undecidable?

**Short answer:** reasoning about general programs is difficult.

**Theorem (Turing)**

Following languages are undecidable.

- \( L_{\text{HALT}} = \{ \langle M \rangle \mid M \text{ halts on blank input} \} \)
- \( L_{\text{HALT},w} = \{ \langle M, w \rangle \mid M \text{ halts on input } w \} \)
- \( L_u = \{ \langle M, w \rangle \mid M \text{ accepts } w \} \)

Recall that languages are problems. Jeff’s notes calls Halting problem \( \text{HALT} \) (the second version)
What else is undecidable?

Via (sometimes highly non-trivial) reductions one can show
- Essentially many questions about sufficiently general programs are undecidable
- Many problems in mathematical logic are undecidable
- Posts correspondence problem which is a string problem
- Tiling problems
- Problems in mathematics such as Diophantine equation solution (Hilbert’s 10th problem)

Undecidability connects computation to mathematics/logic and proofs
What do we want you to know?

- The core undecidable problems (\textit{HALT} and \(L_u\))
- Ability to do simple reductions that prove undecidability of program behaviour
Reductions

1. $\mathcal{R}$: Reduction $X \rightarrow Y$
2. $A_Y$: algorithm for $Y$:
3. $\implies$ New algorithm for $X$:

We write $X \leq Y$ if $X$ reduces to $Y$

Lemma

If $X \leq Y$ and $X$ is undecidable then $Y$ is undecidable.
Write a program that prints “Hello World”

```plaintext
main() {
    print(''Hello World'')
}
```
CS 125 assignment

Write a program that prints “Hello World”

```c
main() {
    print(‘‘Hello World’’)
}
```

**Question:** Can we create an autograder?
Write a program that prints “Hello World”

```c
main() {
    print(‘Hello World’)  
}
```

**Question:** Can we create an autograder? No! Why?

```c
main() {
    stealthcode()
    print(‘Hello World’)  
}
stealthcode() {
    do this
    do that
    viola
}
```
Reducing Halting to Autograder

-Halting problem: given arbitrary program foo(), does it halt?

```c
main()
{
  foo()
  print('Hello World')
}
```

Note: Reduction only needs to add a few lines of code to foo().

foobar() prints "Hello World" if and only if foo() halts!

If we had CS125 Autograder then we can solve Halting. But Halting is hard according to Turing. Hence...
Reducing Halting to Autograder

- **Halting problem:** given *arbitrary* program `foo()`, does it halt?
- **Reduction to CS125Autograder:** given `foo()` output `foobar()`

```plaintext
main() {
    foo()
    print(‘Hello World’) }
foo() {
    line 1
    line 2 ...
}
```

**Note:** Reduction only needs to add a few lines of code to `foo()`
Reducing Halting to Autograder

- **Halting problem**: given arbitrary program `foo()`, does it halt?
- **Reduction to CS125Autograder**: given `foo()` output `foobar()`

```c
main() {
    foo()
    print(‘Hello World’)  // Note: Reduction only needs to add a few lines of code to foo()
}
foo() {
    line 1
    line 2
    ...
}
```

**Note**: Reduction only needs to add a few lines of code to `foo()`

- `foobar()` prints “Hello World” **if and only if** `foo()` halts!
- If we had CS125Autograder then we can solve Halting. But Halting is hard according to Turing. Hence ...
Reducing Halting to Autograder

foo() \rightarrow \text{REDUCTION} \rightarrow \text{foobar()}

foo() \rightarrow \text{REDUCTION} \rightarrow \text{AUTOGRADER} \rightarrow \text{HALT Decider}

\text{HALT Decider}

\begin{align*}
\text{foo()} & \rightarrow \text{REDUCTION} \\
\text{AUTOGRADER} & \rightarrow \text{HALT Decider} \\
\end{align*}
Connection to proofs

Goldbach’s conjecture: Every even integer $\geq 4$ can be written as sum of two primes. Made in 1742, still open.
Goldbach’s conjecture: Every even integer $\geq 4$ can be written as sum of two primes. Made in 1742, still open.

If Halting can be solved then can solve Goldbach’s conjecture. How? Can write a program that halts if and only if conjecture is false.

```c
void golbach() {
    int n = 4
    repeat
        flag = FALSE
        for (int i = 2, i < n; i ++) do
            if (i and (n - i) are both prime)
                flag = TRUE; Break
        if (!flag) return ‘‘Goldbach’s Conjecture is False’’
        n = n + 2
    Until (TRUE)
}
```
More reduction about languages

We will show following languages about program behaviour are undecidable.

- $L_{374} = \{\langle M \rangle \mid L(M) = \{0^{374}\}\}$
- $L_{\neq \emptyset} = \{\langle M \rangle \mid L(M) \neq \emptyset\}$

- A template to show that essentially checking whether a given program’s language satisfies some non-trivial property is undecidable

Same proof technique as the one for autograder
Undecidability of $L_{374}$

**Understanding:** What is the problem of deciding $L_{374}$?

*Given an arbitrary program $\text{boo}(\text{str } w)$ does $\text{boo}()$ accept only the string $0^{374}$ and nothing else?*
Undecidability of $L_{374}$

**Understanding:** What is the problem of deciding $L_{374}$?

*Given an arbitrary program* $\text{boo}(\text{str } w)$ *does* $\text{boo}()$ *accept only the string* $0^{374}$ *and nothing else?*

Seems harder than autograder for printing “Hello World”!
Undecidability of $L_{374}$

**Understanding:** What is the problem of deciding $L_{374}$?

*Given an arbitrary program* $\texttt{boo}(\texttt{str w})$ *does* $\texttt{boo}()$ *accept only the string* $0^{374}$ *and nothing else?*

Seems harder than autograder for printing “Hello World”!

Prove that if we had a decider $\text{Decide}L_{374}$ for $L_{374}$ then we can create a decider for HALT.
Undecidability of $L_{374}$

**Understanding:** What is the problem of deciding $L_{374}$?

*Given an arbitrary program $\text{boo}(\text{str } w)$ does $\text{boo()}$ accept only the string $0^{374}$ and nothing else?*

Seems harder than autograder for printing “Hello World”!

Prove that if we had a decider $\text{Decide}L_{374}$ for $L_{374}$ then we can create a decider for HALT.

Recall: Decider for HALT takes an arbitrary program $\text{foo}()$ and needs to check if $\text{foo()}$ halts.
Undecidability of $L_{374}$

**Understanding:** What is the problem of deciding $L_{374}$?

Given an arbitrary program $\text{boo}(\text{str } w)$ does $\text{boo}()$ accept only the string $0^{374}$ and nothing else?

Seems harder than autograder for printing “Hello World”!

Prove that if we had a decider $\text{Decide}L_{374}$ for $L_{374}$ then we can create a decider for HALT.

Recall: Decider for HALT takes an arbitrary program $\text{foo}()$ and needs to check if $\text{foo}()$ halts.

Reduction should transform $\text{foo}()$ into a program $\text{fooboo}()$ such that answer to $\text{fooboo}()$ from $\text{Decide}L_{374}$ will let us know if $\text{foo}()$ halts.
Undecidability of $L_{374}$

A simple program $simpleboo(str \ w)$

```
simpleboo(str w) {
    if (w == 0^{374}) then return YES
    return NO
}
```

Easy to see that $L(simpleboo()) = \{0^{374}\}$. 
Undecidability of $L_{374}$

A simple program $\text{simpleboo}(\text{str } w)$

```cpp
simpeboo(\text{str } w) \{
    \text{if } (w = 0^{374}) \text{ then return YES}
    \text{return NO}
\}
```

Easy to see that $L(\text{simpleboo}()) = \{0^{374}\}$.

Given arbitrary program $\text{foo}()$ reduction creates $\text{fooboo}(\text{str } w)$:

```cpp
fooboo(\text{str } w) \{
    \text{foo()}
    \text{if } (w = 0^{374}) \text{ then Return YES}
    \text{return NO}
\}
```

```
\text{foo} () \{
    \text{code of foo} ... 
\}
```
Lemma

Language of \texttt{fooboo()} is \{0^{374}\} if \texttt{foo()} halts. Language of \texttt{fooboo()} is \emptyset if \texttt{foo()} does not halt.
Undecidability of $L_{374}$

**Lemma**

Language of $\text{fooboo}()$ is $\{0^{374}\}$ if $\text{foo}()$ halts. Language of $\text{fooboo}()$ is $\emptyset$ if $\text{foo}()$ does not halt.

**Corollary**

$\text{fooboo}()$ in $L_{374}$ if and only if $\text{foo}() \in L_{\text{HALT}}$.

**Corollary**

If $L_{374}$ is decidable then $L_{\text{HALT}}$ is decidable. Since $L_{\text{HALT}}$ is undecidable $L_{374}$ is undecidable.
Undecidability of $L \neq \emptyset$

**Understanding:** What is the problem of deciding $L \neq \emptyset$?

*Given an arbitrary program* $boo(str \ w)$ *does* $boo()$ *accept any string?*

Reduce from HALT: given arbitrary program $foo()$ create $fooboo()$ such that $fooboo()$ accepts some string iff $foo()$ halts.
Undecidability of $L \neq \emptyset$

**Understanding:** What is the problem of deciding $L \neq \emptyset$?

*Given an arbitrary program $\text{boo}(\text{str } w)$ does $\text{boo}()$ accept any string?*

Reduce from HALT: given arbitrary program $\text{foo}()$ create $\text{fooboo}()$ such that $\text{fooboo}()$ accepts some string iff $\text{foo}()$ halts.
Undecidability of $L \neq \emptyset$

A simple program $simpleboo(str \ w)$

```
simpeboo(str w) {
    return YES
}
```

Easy to see that $L(simpeboo()) = \Sigma^*$ and hence not empty.
Undecidability of $L \neq \emptyset$

A simple program $\text{simpleboo}(\text{str } w)$

```
simpeboo(str w) {
    return YES
}
```

Easy to see that $L(\text{simpleboo}()) = \Sigma^*$ and hence not empty.

Given arbitrary program $\text{foo}()$ reduction creates $\text{fooboo}(\text{str } w)$ as follows

```
fooboo(str w) {
    foo()
    return YES
}
foo () {
    code of foo ...
}
```
Undecidability of $L \neq \emptyset$

**Lemma**

Language of $\text{fooboo}(\cdot)$ is $\Sigma^*$ if $\text{foo}(\cdot)$ halts. Language of $\text{fooboo}(\cdot)$ is $\emptyset$ if $\text{foo}(\cdot)$ does not halt.
## Undecidability of $L \neq \emptyset$

<table>
<thead>
<tr>
<th>Lemma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language of $\text{fooboo()}$ is $\Sigma^*$ if $\text{foo()}$ halts. Language of $\text{fooboo()}$ is $\emptyset$ if $\text{foo()}$ does not halt.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Corollary</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{fooboo()}$ in $L \neq \emptyset$ if and only if $\text{foo()} \in L_{\text{HALT}}$.</td>
</tr>
</tbody>
</table>
Beyond r.e

Lemma

If $L$ is recursive then $\overline{L} = \Sigma^* - L$ is recursive.

Proof.

We have TMs $M$, $M'$ such that $L = \mathcal{L}(M)$ and $\overline{L} = \mathcal{L}(M')$.

Construct new TM $M^*$ that on input $w$ simulates both $M$ and $M'$ on $w$ in parallel. One of them has to halt and give right answer.

Corollary

Suppose $L$ is r.e but not recursive. Then $\overline{L}$ is not r.e.
Lemma

If $L$ is recursive then $\bar{L} = \Sigma^* - L$ is recursive.

Lemma

Suppose $L$ and $\bar{L}$ are both r.e. Then $L$ is recursive.
Beyond r.e

**Lemma**

If $L$ is recursive then $\overline{L} = \Sigma^* - L$ is recursive.

**Lemma**

Suppose $L$ and $\overline{L}$ are both r.e. Then $L$ is recursive.

**Proof.**

We have TMs $M, M'$ such that $L = L(M)$ and $\overline{L} = L(M')$. Construct new TM $M^*$ that on input $w$ simulates both $M$ and $M'$ on $w$ in parallel. One of them has to halt and give right answer.
Beyond r.e

**Lemma**

If $L$ is recursive then $\overline{L} = \Sigma^* - L$ is recursive.

**Lemma**

Suppose $L$ and $\overline{L}$ are both r.e. Then $L$ is recursive.

**Proof.**

We have TMs $M, M'$ such that $L = L(M)$ and $\overline{L} = L(M')$. Construct new TM $M^*$ that on input $w$ simulates both $M$ and $M'$ on $w$ in parallel. One of them has to halt and give right answer.

**Corollary**

Suppose $L$ is r.e but not recursive. Then $\overline{L}$ is not r.e.
Corollary

Suppose \( L \) is r.e but not recursive. Then \( \overline{L} \) is not r.e.

Thus \( \overline{L_{HALT}} \) and \( \overline{L_u} \) are not even r.e. What does this mean?
Beyond r.e

Corollary

Suppose \( L \) is r.e but not recursive. Then \( \overline{L} \) is not r.e.

Thus \( \overline{L_{HALT}} \) and \( \overline{L_u} \) are not even r.e. What does this mean?

What problem is \( \overline{L_{HALT}} \)? Given code/program \( \langle M \rangle \) does it not halt on blank input? How can we tell?

We can simulate \( M \) using a UTM. How long? If \( M \) halts during simulation, UTM can reject \( \langle M \rangle \). But if it does not halt after a billion steps can we stop simulation and say for sure that \( M \) will not halt? Perhaps there are other ways of figuring this out? Proof says no.
Part III

Undecidability of Halting
Turing’s Theorem

Theorem (Turing)

Following languages are undecidable.

- \( L_{HALT} = \{\langle M \rangle \mid M \text{ halts on blank input} \} \)
- \( L_{HALT,w} = \{\langle M, w \rangle \mid M \text{ halts on input } w \} \)
- \( L_u = \{\langle M, w \rangle \mid M \text{ accepts } w \} \)

Exercise: Prove that the above languages can be reduced to each other.
Theorem (Turing)

Following languages are undecidable.

- \( L_{HALT} = \{ \langle M \rangle \mid M \text{ halts on blank input} \} \)
- \( L_{HALT,w} = \{ \langle M, w \rangle \mid M \text{ halts on input } w \} \)
- \( L_u = \{ \langle M, w \rangle \mid M \text{ accepts } w \} \)

Exercise: Prove that the above languages can be reduced to each other.

Two proofs

- A two step one based on Cantor’s diagonalization
- A slick one but essentially the same idea in a different fashion
Diagonalization based proof

TM's can be put in 1-1 correspondence with integers: $M_i$ is $i$'th TM

**Definition**

$L_d = \{ \langle i \rangle \mid M_i \text{ does not accept } \langle i \rangle \}$. Same as

$L_d = \{ \langle M_i \rangle \mid M_i \text{ does not accept } \langle i \rangle \}$. 
### Understanding $L_d$

<table>
<thead>
<tr>
<th></th>
<th>$w_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$w_5$</th>
<th>$w_6$</th>
<th>$w_7$</th>
<th>$w_8$</th>
<th>$w_9$</th>
<th>$...$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$M_1$</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$M_2$</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$M_3$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>$...$</td>
</tr>
<tr>
<td>$M_4$</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$M_5$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$M_6$</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>$...$</td>
</tr>
<tr>
<td>$M_7$</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>$...$</td>
</tr>
<tr>
<td>$M_8$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>$...$</td>
</tr>
<tr>
<td>$M_9$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>$...$</td>
</tr>
<tr>
<td>$...$</td>
<td>$...$</td>
<td>$...$</td>
<td>$...$</td>
<td>$...$</td>
<td>$...$</td>
<td>$...$</td>
<td>$...$</td>
<td>$...$</td>
<td>$...$</td>
<td>$...$</td>
<td>$...$</td>
</tr>
</tbody>
</table>
Consider for each $i$, whether or not $M_i$ accepts $w_i$.

<table>
<thead>
<tr>
<th></th>
<th>$w_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$w_5$</th>
<th>$w_6$</th>
<th>$w_7$</th>
<th>$w_8$</th>
<th>$w_9$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_1$</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_2$</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_3$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>...</td>
</tr>
<tr>
<td>$M_4$</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_5$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_6$</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>...</td>
</tr>
<tr>
<td>$M_7$</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>...</td>
</tr>
<tr>
<td>$M_8$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_9$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Understanding $L_d$

<table>
<thead>
<tr>
<th></th>
<th>$w_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$w_5$</th>
<th>$w_6$</th>
<th>$w_7$</th>
<th>$w_8$</th>
<th>$w_9$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_1$</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_2$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_3$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>...</td>
</tr>
<tr>
<td>$M_4$</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_5$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_6$</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>...</td>
</tr>
<tr>
<td>$M_7$</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>...</td>
</tr>
<tr>
<td>$M_8$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_9$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
$L_d$ is not r.e

$L_d = \{ \langle i \rangle \mid M_i \text{ does not accept } \langle i \rangle \}.$

**Theorem**

$L_d$ is not r.e.
**$L_d$ is not r.e.**

$L_d = \{ \langle i \rangle \mid M_i \text{ does not accept } \langle i \rangle \}$.

**Theorem**

$L_d$ is not r.e.

Proof by contradiction. Suppose it is. Then there is some $i^*$ such that $L_d = L(M_{i^*})$. 

Does $\langle i^* \rangle \in L_d$?

If yes then $M_{i^*}$ accepts $\langle i^* \rangle$ since $L_d = L(M_{i^*})$. But this is a contradiction since $i^* \notin L_d$ by definition of $L_d$.

If no then $M_{i^*}$ does not accept $\langle i^* \rangle$ since $L_d = L(M_{i^*})$. But this is a contradiction since $i^* \in L_d$ by definition of $L_d$.

Thus we obtain a contradiction in both cases which implies that $L_d$ is not r.e.
**Theorem**

\( L_d \) is not r.e.

Proof by contradiction. Suppose it is. Then there is some \( i^* \) such that \( L_d = L(M_{i^*}) \). Does \( \langle i^* \rangle \in L_d \)?
$L_d$ is not r.e.

$L_d = \{\langle i \rangle \mid M_i \text{ does not accept } \langle i \rangle\}.$

**Theorem**

$L_d$ is not r.e.

Proof by contradiction. Suppose it is. Then there is some $i^*$ such that $L_d = L(M_{i^*})$. Does $\langle i^* \rangle \in L_d$?

- If yes then $M_{i^*}$ accepts $\langle i^* \rangle$ since $L_d = L(M_{i^*})$. But this is a contradiction since $i^* \notin L_d$ by definition of $L_d$. 

Theorem

$L_d$ is not r.e.

Proof by contradiction. Suppose it is. Then there is some $i^*$ such that $L_d = L(M_{i^*})$. Does $\langle i^* \rangle \in L_d$?

- If yes then $M_{i^*}$ accepts $\langle i^* \rangle$ since $L_d = L(M_{i^*})$. But this is a contradiction since $i^* \not\in L_d$ by definition of $L_d$.

- If no then $M_{i^*}$ does not accept $\langle i^* \rangle$ since $L_d = L(M_{i^*})$. But this is a contradiction since $i^* \in L_d$ by definition of $L_d$. 

Therefore, $L_d$ is not r.e.
**$L_d$ is not r.e.**

$L_d = \{ \langle i \rangle \mid M_i \text{ does not accept } \langle i \rangle \}$.

**Theorem**

$L_d$ is not r.e.

Proof by contradiction. Suppose it is. Then there is some $i^*$ such that $L_d = L(M_{i^*})$. Does $\langle i^* \rangle \in L_d$?

- If yes then $M_{i^*}$ accepts $\langle i^* \rangle$ since $L_d = L(M_{i^*})$. But this is a contradiction since $i^* \not\in L_d$ by definition of $L_d$.

- If no then $M_{i^*}$ does not accept $\langle i^* \rangle$ since $L_d = L(M_{i^*})$. But this is a contradiction since $i^* \in L_d$ by definition of $L_d$.

Thus we obtain a contradiction in both cases which implies that $L_d$ is not r.e.
Lemma

$L_d \leq \bar{L}_u$. That is, if there is an algorithm for $\bar{L}_u$ then there is an algorithm for $L_d$. Equivalently, if there is an algorithm for $L_u$ then there is an algorithm for $L_d$. 

Corollary

$L_{\text{HALT}}$ is undecidable.
$L_d$ is not r.e implies $L_u$ is not decidable

Lemma

$L_d \leq \overline{L}_u$. That is, if there is an algorithm for $\overline{L}_u$ then there is an algorithm for $L_d$. Equivalently, if there is an algorithm for $L_u$ then there is an algorithm for $L_d$.

Algorithm for $L_d$ from an algorithm for $L_u$:

- Given $\langle i \rangle$ we simply feed $\langle M_i, i \rangle$ to algorithm for $L_u$
- If algorithm for $L_u$ says NO return YES Else return NO

Corollary

$L_u$ is undecidable.

Corollary

$\text{HALT}$ is undecidable.
$L_d$ is not r.e implies $L_u$ is not decidable

**Lemma**

$L_d \leq \bar{L}_u$. That is, if there is an algorithm for $\bar{L}_u$ then there is an algorithm for $L_d$. Equivalently, if there is an algorithm for $L_u$ then there is an algorithm for $L_d$.

Algorithm for $L_d$ from an algorithm for $L_u$:

- Given $\langle i \rangle$ we simply feed $\langle M_i, i \rangle$ to algorithm for $L_u$
- If algorithm for $L_u$ says NO return YES Else return NO

**Corollary**

$L_u$ is undecidable.

**Corollary**

$L_{HALT}$ is undecidable.
The Big Picture

UNDECIDABLE

R. E.

RECURSIVE

EXP

NP

NPC

P

NPC

CFLs

Regular

not even accepted by a TM