Griffith University

3515ICT Theory of Computation

Undecidability

(Based loosely on slides by Harald Søndergaard of
The University of Melbourne)
**Theorem.** Some languages are not Turing-recognisable, and hence not decidable.

*Proof.* First, the set of all Turing machines is countably infinite. Choose some encoding of Turing machines, *cf.* the previous encoding of DFAs. There are only finitely-many TMs of any given length $n$. So we can list all the TMs of length 1, then all the TMs of length 2, and so on.

Second, the set of all languages (over any nonempty alphabet $\Sigma$) is not countable. Let $s_1, s_2, \ldots$ be a listing of all strings in $\Sigma^*$. (Again, we could list the strings in order of increasing length.)

For the sake of contradiction, suppose the set of language over $\Sigma$ *is* countable. Let $L_1, L_2, \ldots$ be a listing of all languages over $\Sigma$. *I.e.*, each $L_i$ is a some subset of the $s_i$. 
Define the specific language

\[ L = \{ s_i \in \Sigma^* \mid s_i \not\in L_i \}. \]

Then, if \( L = L_k \), for some \( k \geq 1 \), either \( s_k \in L \) or \( s_k \not\in L \), and each case leads to a contradiction.

I.e., \( L \) is not one of the \( L_i \), so no listing of the language over \( \Sigma \) is possible, and hence the set of languages over \( \Sigma \) is not countable. (This proof method is called \textit{diagonalisation}.)

Hence, there are more languages than there are Turing machines to recognise them, so some languages are not Turing-recognisable.

What is an example of such a language?
We now prove that a particular problem is undecidable, i.e., we prove that a particular language is not Turing-decidable.

In particular, we prove that it is undecidable to determine whether a given Turing machine accepts a given input string. (The corresponding problems are decidable for DFAs and PDAs.) That is, the language

\[ A_{TM} = \{ (M, w) \mid M \text{ is a TM and } M \text{ accepts } w \} \]

is undecidable.

The main difference from the case of \( A_{CFG} \), e.g., is that a Turing machine may fail to halt.
**Theorem.** The language

\[ A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \} \]

is undecidable.

**Proof.** Suppose (for the sake of contradiction) that \( A_{TM} \) is decidable, and is decided by a TM \( H \):

\[ H\langle M, w \rangle = \begin{cases} 
   \text{accept} & \text{if } M \text{ accepts } w \\
   \text{reject} & \text{if } M \text{ does not accept } w 
\end{cases} \]

Using \( H \) we can construct a Turing machine \( D \) which decides whether a given machine \( M \) accepts its own encoding \( \langle M \rangle \):

1. Input is \( \langle M \rangle \), where \( M \) is some Turing machine.
2. Run \( H \) on \( \langle M, \langle M \rangle \rangle \).
3. If \( H \) accepts, \( \text{reject} \). If \( H \) rejects, \( \text{accept} \).
In summary:

\[
D\langle M \rangle = \begin{cases} 
    \text{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\
    \text{reject} & \text{if } M \text{ accepts } \langle M \rangle 
\end{cases}
\]

But no machine can satisfy that specification without leading to a contradiction!

Consider \(D\)'s behaviour on its own encoding:

\[
D\langle D \rangle = \begin{cases} 
    \text{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\
    \text{reject} & \text{if } D \text{ accepts } \langle D \rangle 
\end{cases}
\]

Hence neither \(D\) nor \(H\) can exist.

Sipser shows that this proof is really just another use of diagonalisation.
Note that

\[ A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \} \]

is Turing-recognisable.

The reason is that it is possible to construct a universal Turing machine \( U \) which is able to simulate any Turing machine.

On input \( \langle M, w \rangle \), \( U \) simulates \( M \) on input \( w \).

If \( M \) enters its accept state, \( U \) accepts.

If \( M \) enters its reject state, \( U \) rejects.

If \( M \) never halts, neither does \( U \).
The set of Turing-recognisable languages is closed under the regular operations, and intersection.

The set of decidable languages are closed under the same operations, \textit{and also under complement}.

**Theorem.** A language $L$ is decidable iff both $L$ and $\overline{L}$ are Turing-recognisable.

**Proof.** If $L$ is decidable, clearly $L$ and also $\overline{L}$ are recognisable.

Assume both $L$ and $\overline{L}$ are recognisable. That is, there are recognisers $M_1$ and $M_2$ for $L$ and $\overline{L}$, respectively.

A Turing machine $M$ can then take input $w$ and run $M_1$ and $M_2$ on $w$ in parallel. If $M_1$ accepts, so does $M$. If $M_2$ accepts, $M$ rejects.

Note that at least one of $M_1$ and $M_2$ is guaranteed to eventually accept.

Hence $M$ decides $L.$
This gives us an example of a language which is not Turing-recognisable: \( \overline{A_{TM}} \).

We know that \( A_{TM} \) is recognisable.

If \( \overline{A_{TM}} \) were also Turing-recognisable, then \( A_{TM} \) would be decidable.

But we have shown that it isn’t.

Hence, \( \overline{A_{TM}} \) is not Turing-recognisable.

Remember that \( \overline{A_{TM}} = \{ \langle M, w \rangle \mid M \text{ is a Turing machine and } M \text{ does not accept } w \} \). (\( M \) could not accept \( w \) either by rejecting or by looping indefinitely.)
Informally, problem $\mathcal{P}_1$ is reducible to problem $\mathcal{P}_2$ if an algorithm for solving $\mathcal{P}_2$ can be used to solve $\mathcal{P}_1$. (Note the direction.)

Formally, let $\mathcal{P}_1$ and $\mathcal{P}_2$ be decision problems. Then $\mathcal{P}_1$ is reducible to $\mathcal{P}_2$ iff there is a TM $M$ that transforms every instance $p_1$ of $\mathcal{P}_1$ to an instance $p_2$ of $\mathcal{P}_2$ such that $p_1$ and $p_2$ have the same answer (yes or no). Equivalently, $\mathcal{P}_1$ is reducible to $\mathcal{P}_2$ iff there is a TM $M$ that transforms any TM $M_2$ that solves problem $\mathcal{P}_2$ to a TM $M_1$ that solves problem $\mathcal{P}_1$. Then,

- $\mathcal{P}_1$ reducible to $\mathcal{P}_2$ and $\mathcal{P}_2$ decidable
  \[ \Rightarrow \mathcal{P}_1 \text{ decidable.} \]
- $\mathcal{P}_1$ reducible to $\mathcal{P}_2$ and $\mathcal{P}_1$ undecidable
  \[ \Rightarrow \mathcal{P}_2 \text{ undecidable.} \]

So reducibility is useful for proving both decidability and undecidability results.
**The Halting Problem is Undecidable**

**Theorem.** $\text{HALT}_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \}$ is undecidable.

**Proof.** We show $A_{TM}$ is reducible to $\text{HALT}_{TM}$.

Suppose we have a TM $R$ that decides $\text{HALT}_{TM}$. Then we can construct a TM $S$ that decides $A_{TM}$ as follows:

1. Run TM $R$ on input $\langle M, w \rangle$,
2. If $R$ rejects (i.e., TM $M$ does not halt on $w$), reject.
3. If $R$ accepts (i.e., TM $M$ halts on $w$), simulate $M$ on $w$ until it halts.
4. If $M$ accepts, accept, otherwise reject.

This decides $A_{TM}$. But $A_{TM}$ was undecidable, so $\text{HALT}_{TM}$ must also be undecidable.
An Alternative Proof

Proof. Suppose TM $H$ decides $\text{HALT}_{TM}$, i.e.,

\[
H(M, w) = \begin{cases} 
\text{accept} & \text{if } M(w) \text{ halts} \\
\text{reject} & \text{if } M(w) \text{ loops}
\end{cases}
\]

Then we can modify TM $H$ to TM $J$, so that:

\[
J(M, w) = \begin{cases} 
\text{loops} & \text{if } M(w) \text{ halts} \\
\text{accepts} & \text{if } M(w) \text{ loops}
\end{cases}
\]

Next we apply TM $J$ to a TM $M$ and $\langle M \rangle$:

\[
K(M) = J(M, \langle M \rangle) = \begin{cases} 
\text{loops} & \text{if } M(\langle M \rangle) \text{ halts} \\
\text{accepts} & \text{if } M(\langle M \rangle) \text{ loops}
\end{cases}
\]

Finally we apply TM $K$ to its own encoding $\langle K \rangle$:

\[
K(K) = \begin{cases} 
\text{loops} & \text{if } K(K) \text{ halts} \\
\text{accepts} & \text{if } K(K) \text{ loops}
\end{cases}
\]

Contradiction! So decider $H$ cannot exist.
Theorem. \( E_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset \} \) is undecidable.

Proof. We show \( A_{TM} \) is reducible to \( E_{TM} \).

First, given \( \langle M, w \rangle \), a Turing machine can modify the encoding of \( M \), to transform \( M \) into \( M' \), which recognises \( L(M) \cap \{w\} \).

This is what the new machine \( M' \) does on input \( x \):

1. If \( x \neq w \), reject.

2. If \( x = w \), run \( M \) on \( w \) and accept if \( M \) accepts.

(Note how \( w \) has been “hard-wired” into \( M' \): \( M' \) is like \( M \), but it has extra states to compare its input with \( w \).)
TM Emptiness Is Undecidable (cont.)

Now, suppose TM $R$ decides $E_{TM}$. Then we can construct the following decider for $A_{TM}$:

1. From input $\langle M, w \rangle$, construct $\langle M' \rangle$.

2. Run $R$ on $\langle M' \rangle$.

3. If $R$ rejects (i.e., $L(M') \neq \emptyset$, so $w \in L(M') \subseteq L(M)$), accept;
   if $R$ accepts (i.e., $L(M') = \emptyset$, so $w \notin L(M)$), reject.

As no such decider for $A_{TM}$ can exist, $E_{TM}$ must be undecidable.
Theorem. $R_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular} \}$ is undecidable.

Proof. We show that $A_{TM}$ is reducible to $R_{TM}$.

First note that, given $\langle M, w \rangle$, a Turing machine can modify $\langle M \rangle$ into $\langle M' \rangle$, where

$$L(M') = \begin{cases} 
\Sigma^* & \text{if } M \text{ accepts } w \\
\{ 0^n1^n \mid n \geq 0 \} & \text{otherwise}
\end{cases}$$

Here is what the new machine $M'$ does on input $x$:

1. If $x$ has the form $0^n1^n$, accept.
2. Otherwise, run $M$ on $w$ and accept if $M$ does.

Again, $w$ has been hard-wired into $M'$. 
The point of this construction is that $M'$ recognises a regular language ($\Sigma^*$) if $M$ accepts $w$ and a nonregular language ($\{0^n1^n \mid n \geq 0\}$) otherwise.

Suppose TM $R$ decides $R_{TM}$. Then we can construct the following decider for $A_{TM}$:

1. From input $\langle M, w \rangle$, construct $\langle M' \rangle$.
2. Run $R$ on $\langle M' \rangle$.
3. If $R$ accepts, accept; if $R$ rejects, reject.

Again, as no such decider for $A_{TM}$ can exist, $R_{TM}$ must be undecidable.
TM Equality is Undecidable

**Theorem.** $EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \}$ is undecidable.

**Proof.** We show $E_{TM}$ is reducible to $EQ_{TM}$.

Assume that $R$ decides $EQ_{TM}$. Here is a decider for $E_{TM}$:

1. Input is $\langle M \rangle$.

2. Construct a Turing machine $M_\emptyset$ that rejects all input.

3. Run $R$ on $\langle M, M_\emptyset \rangle$.

4. If $R$ accepts, accept; if it rejects, reject.

But we already saw that $E_{TM}$ is undecidable. Hence, $EQ_{TM}$ is undecidable.

This is getting repetitious!
Rice’s Theorem. Every nontrivial semantic property of Turing machines is undecidable!

A property $\mathcal{P}$ is nontrivial if there exist TMs $M_1$ and $M_2$ s.t. $M_1$ satisfies $\mathcal{P}$ and $M_2$ does not satisfy $\mathcal{P}$.

A property $\mathcal{P}$ is semantic if, for all pairs of TMs $M_1$ and $M_2$ s.t. $L(M_1) = L(M_2)$, $M_1$ satisfies $\mathcal{P}$ iff $M_2$ satisfies $\mathcal{P}$, i.e., if $\mathcal{P}$ depends only on the language of a TM.

Because a property can be identified with a language, we can restate Rice’s Theorem as follows:

Rice’s Theorem. Every nonempty proper subset of Turing-recognisable languages is undecidable.
Proof of Rice’s Theorem

Proof. (Sipser, p.215; IALC, pp.388–390)
Let $\mathcal{P}$ be a nonempty proper subset of Turing-recognisable languages. We prove that $A_{TM}$ is reducible to $\mathcal{P}$.

Suppose $\emptyset \not\in \mathcal{P}$ (otherwise use $\overline{\mathcal{P}}$). As $\mathcal{P}$ is nonempty, let $L \in \mathcal{P}$, and let $M_L$ be a TM that recognises $L$.

Given an instance $\langle M, w \rangle$ of $A_{TM}$, we use $L$ to construct an instance $\langle M' \rangle$ of $\mathcal{P}$. TM $M'$ acts as follows on input $x$:

1. Simulate $M$ on $w$.
2. If $M$ accepts, simulate $M_L$ on $x$.
3. If $M_L$ accepts, accept.

Note that $M$, $w$ and $M_L$ have been hard-wired into $M'$.
From the definition of $M'$, it follows that

$$L(M') = \begin{cases} L(M_L) & \text{if } M \text{ accepts } w \\ \emptyset, & \text{otherwise} \end{cases}$$

As $L(M_L) = L \in \mathcal{P}$ and $\emptyset \notin \mathcal{P}$, $\langle M' \rangle \in \mathcal{P}$ iff $\langle M, w \rangle \in A_{TM}$. I.e., we have proved that $A_{TM}$ is reducible to $\mathcal{P}$. But $A_{TM}$ is not decidable, so neither is $\mathcal{P}$. □

Here is another way to think about this proof:
Suppose we have a decider $M_\mathcal{P}$ for $\mathcal{P}$. We use $M_\mathcal{P}$ to construct a decider $M_A$ for $A_{TM}$ that acts as follows on input $\langle M, w \rangle$:

1. Construct $M'$ from $M$, $w$ and $L$ as above.
2. Run $M_\mathcal{P}$ on $M'$.
3. If $M_\mathcal{P}$ accepts, accept; otherwise, reject.

By the above analysis, this construction works.
Applications of Rice’s Theorem

Rice’s Theorem can thus be used to prove the following properties of Turing machines $M$ are undecidable.

- $L(M) = \emptyset$.
- $1011 \in L(M)$.
- $L(M)$ is a finite language.
- $L(M)$ is a regular language.
- $L(M)$ is a context-free language.
- $L(M) = \Sigma^*$.
- Many other properties of $M$. 
Post’s Correspondence Problem

An instance of PCP is a finite set of “dominos” such as
\[
\left\{ \left[ \begin{array}{c} b \\ ca \end{array} \right], \left[ \begin{array}{c} a \\ ab \end{array} \right], \left[ \begin{array}{c} ca \\ a \end{array} \right], \left[ \begin{array}{c} abc \\ c \end{array} \right] \right\}
\]

Formally, an instance of PCP is a set of pairs \((a_i, b_i)\) with \(a_i, b_i \in \Sigma^+\), for \(1 \leq i \leq k\).
(There is an infinite supply of each domino.)

An instance of PCP has a solution if there exists a sequence of dominos in which the tops and bottoms “match”, \(i.e.,\) if there exists a sequence of integers \(i_1, \ldots, i_m\) s.t. \(a_{i_1} \ldots a_{i_m} = b_{i_1} \ldots b_{i_m}\).

In this case, yes:
\[
\left[ \begin{array}{c} a \\ ab \end{array} \right] \left[ \begin{array}{c} b \\ ca \end{array} \right] \left[ \begin{array}{c} ca \\ a \end{array} \right] \left[ \begin{array}{c} a \\ ab \end{array} \right] \left[ \begin{array}{c} abc \\ c \end{array} \right]
\]
How about this case?

\[
\left\{ \begin{array}{c}
\begin{bmatrix} a \\ cb \end{bmatrix}, \begin{bmatrix} bc \\ ba \end{bmatrix}, \begin{bmatrix} c \\ aa \end{bmatrix}, \begin{bmatrix} abc \\ c \end{bmatrix}
\end{array} \right\}
\]

And this?

\[
\left\{ \begin{array}{c}
\begin{bmatrix} ab \\ aba \end{bmatrix}, \begin{bmatrix} bba \\ aa \end{bmatrix}, \begin{bmatrix} aba \\ bab \end{bmatrix}
\end{array} \right\}
\]

And this?

\[
\left\{ \begin{array}{c}
\begin{bmatrix} baa \\ abaaa \end{bmatrix}, \begin{bmatrix} aaa \\ aa \end{bmatrix}
\end{array} \right\}
\]

Yes:

\[
\begin{bmatrix} aaa \\ aa \end{bmatrix} \begin{bmatrix} baa \\ abaaa \end{bmatrix} \begin{bmatrix} aaa \\ aa \end{bmatrix}
\]
**Theorem:** $PCP$ is undecidable.

The proof has complicated details, but the idea is simple.

We reduce $A_{TM}$ to $PCP$ via computation histories.

That is, for given $M$ and $w$ we construct an instance $P$ of $PCP$ such that $P$ has a solution iff $M$ accepts $w$.

A solution to $P$ will effectively *simulate* the running of $M$ on $w$.

The theorem has many useful consequences.
Theorem. $\text{AMB}_{CFG} = \{ \langle G \rangle \mid G \text{ is an ambiguous CFG} \}$ is undecidable.

Proof. (Sipser, Problem 5.21; IALC, Theorem 9.20) We show $PCP$ is reducible to $\text{AMB}_{CFG}$.

Let

$$P = \left\{ \left[ \frac{a_1}{b_1} \right], \left[ \frac{a_2}{b_2} \right], \ldots, \left[ \frac{a_k}{b_k} \right] \right\}$$

be an instance of $PCP$. Construct a CFG $G$ with the following rules

$$S \to A \mid B$$
$$A \to a_1 Ac_1 \mid \cdots \mid a_k Ac_k \mid a_1 c_1 \mid \cdots \mid a_k c_k$$
$$B \to b_1 Bc_1 \mid \cdots \mid b_k Bc_k \mid b_1 c_1 \mid \cdots \mid b_k c_k$$

where $c_1, \ldots, c_k$ are new terminal symbols.

We show that $P$ has a solution iff $G$ is ambiguous.
CFG Ambiguity is Undecidable (cont.)

First, note that $A$ (resp., $B$) generates the set of “nested parenthesis strings” over $(a_i, c_i)$ pairs (resp., $(b_i, c_i)$ pairs), and is unambiguous.

Next, suppose that $P$ has a solution $i_1, \ldots, i_m$, i.e., $a_{i_1} a_{i_2} \ldots a_{i_m} = b_{i_1} b_{i_2} \ldots b_{i_m}$. Then

$S \rightarrow A \rightarrow a_{i_1} Ac_{i_1} \Rightarrow a_{i_1} a_{i_2} Ac_{i_2} c_{i_1} \Rightarrow^*$

$a_{i_1} a_{i_2} \ldots a_{i_m} c_{i_m} \ldots c_{i_2} c_{i_1}$ and also

$S \rightarrow B \rightarrow b_{i_1} Bc_{i_1} \Rightarrow b_{i_1} b_{i_2} Bc_{i_2} c_{i_1} \Rightarrow^*$

$b_{i_1} b_{i_2} \ldots b_{i_m} c_{i_m} \ldots c_{i_2} c_{i_1}$. But these are two distinct, leftmost derivations of the same string, so $G$ is ambiguous.

Conversely, suppose $G$ is ambiguous. Because the grammars consisting of the rules for $A$ (resp., for $B$) are unambiguous, the only possible pair of distinct, leftmost derivations for the same string must start $S \rightarrow A$ and $S \rightarrow B$, and the resulting string defines a solution to $P$. 
Theorem. Let $G_1$ and $G_2$ be context-free grammars. Then the following problems are undecidable:

(a) $L(G_1) \cap L(G_2) = \emptyset$?
(b) $L(G_1) = L(G_2)$? (CFG equality)
(c) $L(G_1) = \Sigma^*$? (CFG “all”)

Proof (IALC, Theorem 9.22). Reduction from PCP. Let $P = \{(a_i, b_i) \mid 1 \leq i \leq k\}$ be an instance of PCP.

Let $L_A$ (resp., $L_B$) be the language generated from the rules for variable $A$ (resp., $B$) in the grammar $G$ above. Obviously, $L_A$ and $L_B$ are context-free. Here, the complements of $L_A$ and $L_B$, $\overline{L_A}$ and $\overline{L_B}$, are also context-free. It is difficult to construct grammars for $\overline{L_A}$ and $\overline{L_B}$, but it is straightforward (if tedious) to construct (deterministic) PDAs that recognise $\overline{L_A}$ and $\overline{L_B}$.
CFG Equality is Undecidable (cont.)

(a) Let $G_1$ be the grammar for $L_A$ and $G_2$ be the grammar for $L_B$. Then $L(G_1) \cap L(G_2)$ is the set of solutions to $P$. The intersection is thus empty iff $P$ has no solutions. (Clearly, “$P$ has no solutions” is decidable iff “$P$ has a solution” is decidable.)

(b) Let $\Sigma = \{a_1, \ldots, a_k, b_1, \ldots, b_k\}$ be the symbols in $P$, and $I = \{c_1, \ldots, c_k\}$ be the set of new terminals in $G$. Let $G_1$ be a grammar for the context-free language $\overline{L_A} \cup \overline{L_B}$. Let $G_2$ be a grammar for the regular and hence context-free language $(\Sigma \cup I)^*$. Then $L(G_1) = \overline{L_A} \cup \overline{L_B} = \overline{L_A} \cap \overline{L_B}$ is the set of all strings in $(\Sigma \cup I)^*$ that are not solutions to $P$. Thus $L(G_1) = L(G_2)$ iff $P$ has no solutions.

(c) Similarly.
From the above result, PDA equality is also undecidable. Otherwise, we could transform the CFGs to equivalent PDAs, and then use the PDA equality decision procedure.

However, deterministic PDA equality \textit{is} decidable. As before, call a language \textit{deterministic} if it is recognised by some deterministic PDA. Then the complement of a deterministic language is also deterministic.
1. The Busy Beaver Problem. Let $B(k)$ be the maximum number of 1’s a halting TM with $k$ states can print when started on an initially empty tape. Then $B(k)$ grows unimaginably fast and is not a computable function.

2. Minimal TMs. A TM $M$ is minimal if no TM equivalent to $M$ has a shorter encoding. Then

$$MIN_{TM} = \{ \langle M \rangle \mid M \text{ is a minimal TM} \}$$

is not Turing-recognisable.

3. Arithmetic Let $Th(N, +, \times)$ be the set of true first-order sentences over the natural numbers, $N$, with addition, multiplication, and equality, e.g., $\forall x \exists y(x + x = y)$. Then $Th(N, +, \times)$ is an undecidable theory.

Note. In contrast, $Th(N, +)$ and $Th(R, +, \times)$ are both decidable theories.