Undecidability

*Lila Kari*

University of Waterloo

(with thanks to Anna Lubiw and Karen Lemone)
Resolution for predicate logic

• **Input**: set of clauses $S = \{C_1, C_2, \ldots, C_n\}$
• **Repeat**, trying to get $\{\}$
• Choose two clauses, one with $P(...)$ and one with $\text{not } P(...)$
• If these can be unified, then resolve and call the resolvent $C$
• If $C = \{\}$ then output “empty clause”
• Else add $C$ to $S$.

Lila Kari, University of Waterloo
Resolution

• **Theorem.** Resolution is a sound and complete system of formal deduction.

  * (soundness) If the output of the procedure is “empty clause”, then \( S \) is unsatisfiable

  * (completeness) If \( S \) unsatisfiable, then some sequence of choices will lead to an output of “empty clause”.

Resolution

• **Theorem.** Resolution is a sound and complete system of formal deduction.

  * (soundness) If the output of the procedure is “empty clause”, then $S$ is unsatisfiable.
  * (completeness) If $S$ unsatisfiable, then some sequence of choices will lead to an output of “empty clause”.

*Recall that a conclusion is a logical consequence of some premises (semantically), iff the set $S$ of clauses obtained from all premises and the negation of conclusion is unsatisfiable.*
Remarks

- The previous procedure, which connects the unsatisfiability of $S$ with obtaining the empty clause from $S$, by resolution, is not an algorithm.
- We have not said how to make choices (of what clauses to unify, or resolve).
- There is no point at which we can conclude “satisfiable.”
Algorithms

- There are problems that cannot be solved by computer programs (i.e. algorithms) even assuming unlimited time and space.
Algorithms

• There are problems that cannot be solved by computer programs (i.e. algorithms), even assuming unlimited time and space.

• What is an algorithm?

• The following are equivalent:
  * C programs, Java programs, Python programs, etc.
  * Turing machines
  * High level pseudo-code

• We can use any of these definitions as our definition of algorithm.
Algorithms

• We say that an algorithm “solves” a problem if, for any input, the algorithm produces the correct output.

• E.g., an algorithm to decide if a formula in the language of predicate logic is (universally) valid, it must output the correct answer (yes/no) for every such input formula.

Lila Kari, University of Waterloo
• **Q:** Is there an algorithm to do the following:
  **Input:** Set of 1st order predicate clauses
  **Output:** Is the set satisfiable, yes or no?
• **Q:** Is there an algorithm to do the following:
  **Input:** Set of 1st order predicate clauses
  **Output:** Is the set satisfiable, yes or no?

• **A:** **No,** there is no such algorithm.
• Q: Is there an algorithm to do the following:
  Input: Set of 1st order predicate clauses
  Output: Is the set satisfiable, yes or no?
• A: No, there is no such algorithm.
• Q: Is there an algorithm to do the following:
  Input: A formula in 1\textsuperscript{st} order predicate logic
  Output: Is the formula (universally) valid, yes or no?
• **Q:** Is there an algorithm to do the following: 
  **Input:** Set of 1st order predicate clauses 
  **Output:** Is the set satisfiable, yes or no? 
  
  • **A:** No, there is no such algorithm. 

• **Q:** Is there an algorithm to do the following: 
  **Input:** A formula in 1st order predicate logic 
  **Output:** Is the formula (universally) valid, yes or no? 

  • **A:** **No,** there is no such algorithm.

Lila Kari, University of Waterloo
Undecidability

- A decision problem has yes/no answers
- A decision problem that has no algorithm is called undecidable
Some undecidable problems

- **Halting Problem**: Given a program \( P \) (e.g. in Scheme or Python) and input \( x \), does \( P \) halt on input \( x \)?
Some undecidable problems

- **Halting Problem**: Given a program $P$ (e.g. in Scheme or Python) and input $x$, does $P$ halt on input $x$?
- **Program Verification**: Given a specification of inputs and corresponding outputs, and given a program $P$, does $P$ meet the specifications?
Some undecidable problems

• **Halting Problem**: Given a program $P$ (e.g. in Scheme or Python) and input $x$, does $P$ halt on input $x$?

• **Program Verification**: Given a specification of inputs and corresponding outputs, and given a program $P$, does $P$ meet the specifications?

• **Program Equivalence**: Given two programs, do they produce the same output for every input?
Halting Problem Examples

Input: integer \( x \)

While \( x \) not equal to 1

\[
x := x - 2
\]

End

Halts if \( x \) is an odd positive integer, otherwise loops forever.

Lila Kari, University of Waterloo
``3x+1” Problem

Input: natural number $x$
While $x$ not equal to 1
    if $x$ is even then $x := x/2$
    else $x := 3x+1$

Lila Kari, University of Waterloo
``3x+1`` Problem

Input: natural number \( x \)

While \( x \) not equal to 1

if \( x \) is even then \( x := x/2 \)

else \( x := 3x+1 \)

Does this halt on all inputs? No one knows.

Lila Kari, University of Waterloo
``3x+1” Problem

Input: natural number $x$
While $x$ not equal to 1
    if $x$ is even then $x:= x/2$
    else $x:= 3x+1$

Does this halt on all inputs? No one knows.

The problem: Suppose for some $x$, we run the program for 2 weeks (months, years) and it has not halted yet. We still cannot tell if it will halt tomorrow, or go on forever.

Lila Kari, University of Waterloo
Halting Problem for Turing Machines (TM)

TM = Universally accepted model of computation/algorithm/program

* Tape (cells)
* Read/write head & CPU
* States (of the CPU): $q_i$
* Input symbols (on tape): $s_j$
* Rewriting rules $q_i s_j \rightarrow s_k L q_n$
  $q_i s_j \rightarrow s_k R q_n$
* Start state $q_0$
* Accepting (final) states $q_f$
Turing Machines in action

- Turing machine simulation with JFLAP

https://www.youtube.com/watch?v=IkYhfk4X47c

Lila Kari, University of Waterloo
Undecidability of the Halting Problem

• **Halting Problem**: Does there exist a program (TM) with:

  **Input:** A program $P$ and an input $I$
  **Output:** “yes” if the program $P$ halts on input $I$ and “no” otherwise

• **Answer:** **NO!**
Proof idea (by contradiction)

• Assume such a TM exists, call it $H(P, I)$ where $P$ is a program and $I$ is an input

• $H(P, I)$ outputs:
  • “halt” (Y) if the program $P$ halts on input $I$, and
  • “loops forever” (N) if the program $P$ never stops on input $I$
Proof idea (contd.)

• We can feed a program $P$ any input
• What happens if we give a program $P$ its own encoding $P$ as an input?
• In other words, what happens if we call $H(P, P)$?
Proof (by contradiction)

Step 1: Construct a program $K(P)$ such that:

- If $H(P, P)$ outputs “halt”, then $K(P)$ goes into an infinite loop, e.g., printing “ha” at each iteration
- If $H(P,P)$ outputs “loops forever”, then $K(P)$ halts
Input

Program $P$

$P$ as program

Program $H(P, I)$

$P$ as input

Output $H(P, P)$

Program $K(P)$

If $H(P, P) = \text{``halts,''}$ then loop forever

If $H(P, P) = \text{``loops forever,''}$ then halt
Step 2: Call $K(K)$
Step 2: Call $K(K)$. We have *two* possibilities:

- **If $K$ halts on input $K$, by definition of $H$, when $K$ is input to $H(K,K)$ then $H(K, K)$ outputs “halt”**. However, by construction of $K(K)$, which calls $H(K,K)$, if $H(K,K)$ outputs “halt”, then $K(K)$ loops forever.

- **If $K$ loops forever on input $K$, by definition of $H$, when $K$ is input to $H(K,K)$ then $H(K,K)$ outputs “loops forever”**. However, by construction of $K(K)$, which calls $H(K,K)$, if $H(K,K)$ outputs “loops forever”, then $K(K)$ halts.
Step 2: Call $K(K)$. We have two possibilities:

* If $K$ halts on $K$ then $H(K, K)$ outputs “halt”, which means $K$ loops forever on $K$.

* If $K$ loops forever on input $K$, then $H(K, K)$ outputs “loops forever”, which means $K$ halts on $K$.

CONTRADICTION!
This contradiction implies that such a program (Turing machine) \( H(P,I) \) that outputs “Y” if \( P \) halts on input \( I \), and outputs “No” if \( P \) does not halt on input \( I \), does not exist.

The Halting Problem is **undecidable**!
Historical Remarks

The Halting Problem was proved undecidable by Alan Turing in 1936

Lila Kari, University of Waterloo
Lila Kari, University of Waterloo
Proving undecidability

• To show that a new problem $B$ is undecidable we use the concept of reducibility.

• Intuitively, a problem $A$ is reducible to (reduces to, is reduced to) problem $B$ if an algorithm for solving problem $B$ (if it existed) could also be used as a subroutine for solving $A$.

• We write $A \leq B$. 
Proving the undecidability of a new problem $B$

• If we can transform every instance of a known undecidable problem $A$ into an instance of the new problem $B$
• Then the new problem $B$ is at least “as hard as” the known undecidable problem $A$, that is,
  • $A \leq B$
• Hence, if we assume we could solve $B$, this means we could solve $A$ – a contradiction
• Thus, $B$ is undecidable
Problem $A$ is reduced to problem $B$

If we can solve problem $B$ then we can solve problem $A$
Problem $A$ is reduced to problem $B$

If $B$ is decidable then $A$ is decidable

If $A$ is undecidable then $B$ is undecidable

Karen Lemone, WPI
Example
the halting problem
is reduced to
the blank-tape halting problem
The blank-tape halting problem

Input: Turing Machine $M$

Question: Does $M$ halt when started with a blank tape?
Theorem: The blank-tape halting problem is undecidable.

Proof: Reduce the halting problem to the blank-tape halting problem.
Suppose we have a decider for the blank-tape halting problem:

- **M** halts on blank tape
- **M** doesn’t halt on blank tape
We want to build a decider for the halting problem:

\[ M \rightarrow \text{halting problem decider} \rightarrow \begin{cases} \text{YES} \quad M \text{ halts on } w \\ \text{NO} \quad M \text{ doesn't halt on } w \end{cases} \]
We want to reduce the halting problem to the blank-tape halting problem:

\[ M \rightarrow M_w \]

Blank-tape halting problem decider

Karen Lemone, WPI
We need to convert one problem instance to the other problem instance.

Halting problem decider

Convert Inputs?

Blank-tape halting problem decider

Karen Lemone, WPI
Construct a new machine $M_w$

- When started on blank tape, writes $w$
- Then continues execution like $M$

$M_w$

<table>
<thead>
<tr>
<th>step 1</th>
<th>step 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>if blank tape</td>
<td>execute $M$</td>
</tr>
<tr>
<td>then write $w$</td>
<td>with input $w$</td>
</tr>
</tbody>
</table>

Karen Lemone, WPI
$M$ halts on input string $w$ if and only if $M_w$ halts when started with blank tape.
Halting problem decider

\[ M \xrightarrow{\text{Generate}} M_w \xrightarrow{M_w} \text{blank-tape halting problem decider} \]

Karen Lemone, WPI
We reduced the halting problem to the blank-tape halting problem.

Since the halting problem is undecidable, the blank-tape halting problem is undecidable.

END OF PROOF

Karen Lemone, WPI
Example:

the halting problem

is reduced to

the state-entry problem
The state-entry problem

Inputs:

- Turing Machine $M$
- State $q$
- String $w$

Question: Does $M$ enter state $q$ on input $w$?
Theorem:
The state-entry problem is undecidable

Proof: Reduce the halting problem to the state-entry problem
Suppose we have a Decider for the state-entry algorithm:

\[ M \rightarrow \text{state-entry problem decider} \]

- YES: \( M \) enters \( q \)
- NO: \( M \) doesn’t enter \( q \)

Karen Lemone, WPI
We want to build a decider for the halting problem:

\[ M \rightarrow \text{Halting problem decider} \rightarrow \begin{cases} \text{YES} & M \text{ halts on } w \\ \text{NO} & M \text{ doesn't halt on } w \end{cases} \]
We want to reduce the halting problem to the state-entry problem:

**Halting problem decider**

\[ M \rightarrow M' \]
\[ w \rightarrow q \]

State-entry problem decider

\[ \text{YES} \rightarrow \text{YES} \]
\[ \text{NO} \rightarrow \text{NO} \]

Karen Lemone, WPI
We need to convert one problem instance to the other problem instance.
Convert $M$ to $M'$:

- Add new state $q$
- From any halting state of $M$ add transitions to $q$

Karen Lemone, WPI
$M$ halts on input $w$ if and only if $M'$ halts on state $q$ on input $w$.
We reduced the halting problem to the state-entry problem.

Since the halting problem is undecidable, the state-entry problem is undecidable.

END OF PROOF

Karen Lemone, WPI
Summary of Undecidable Problems

Halting Problem:
Does machine $M$ halt on input $w$?

Membership problem:
Does machine $M$ accept string $w$?
Blank-tape halting problem:

Does machine $M$ halt when starting on blank tape?

State-entry Problem:

Does machine $M$ enter state $q$ on input $w$?
A fun undecidability example

• Tile System $T = \text{Finite set of tiles, unlimited supply of each tile type (with given glues on edges)}$


• A tiling (assignment of tiles to points on the integer grid) is valid if adjacent edges of neighbouring tiles have the same glue.
Classical "Tiling Problem"

- Can any square, of any size, be tiled using only the available tile types, without violating the glue-matching rule?

Yes

No

Harel, D. *Computers Ltd.* 2000

Lila Kari, University of Waterloo
Classical “Tiling Problem”

“Given a tile system $T$, does there exist a valid tiling of the plane with tiles from $T$?”

**Theorem:** The Tiling Problem is undecidable. (there does not exist an algorithm for solving it)

[Berger66], [Robinson71]
Undecidability of the Tiling Problem

• The Tiling Problem is undecidable
• Proof Idea: Simulate a TM with tiles
• For each Turing Machine rule
  \[ q_i s_j \rightarrow s_k \text{ L } q_n \quad \text{or} \quad q_i s_j \rightarrow s_k \text{ R } q_n \]

construct tiles that have those rules encoded in the glues/colours on their edges

Lila Kari, University of Waterloo
Alphabet, Action \( (q_i s_j \rightarrow s_k R q_n) \), Merging, and Starting Tiles
Simulation of TM computations by valid tilings
Simulation of TM computations by valid tilings

$$q_00 \rightarrow X \cdot R \cdot q_1$$
Simulation of TM computations by valid tilings

\[
q_10 \rightarrow 0 \ R \ q_1
\]

\[
q_00 \rightarrow X \ R \ q_1
\]
Simulation of TM computations by valid tilings

$q_0 0 \rightarrow X R q_1$

$q_1 0 \rightarrow 0 R q_1$

$q_1 1 \rightarrow Y L q_2$

Lila Kari, University of Waterloo
The Tiling Problem and the Halting Problem

- The tile system admits a valid tiling of the plane if and only if the computation of Turing Machine never halts when started on a blank tape.

- Since the Halting Problem on a Blank Tape for Turing Machines is undecidable, the Tiling Problem is also undecidable.
Sometimes We Cannot Do It!

The uncomputable (undecidable)

The computable (decidable)

Harel, D. Computers Ltd. 2000
Lila Kari, University of Waterloo
Credits

• Text on slides 2-13 modified from Anna Lubiw, CS245 W15
• Slides on reducibility, Karen Lemone
  http://web.cs.wpi.edu/~kal/

Lila Kari, University of Waterloo