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 “unsatisfiable”
- Else add $C$ to $S$.  

Lila Kari, University of Waterloo
Resolution

• This is not an algorithm, because we have not said how to make choices, and there is no point at which we decide “satisfiable”

• **Theorem.** Resolution is sound and complete.

• Equivalently:
  * (soundness) If the output of the procedure is “unsatisfiable”, then $S$ is unsatisfiable
  * (completeness) If $S$ unsatisfiable, then *some* sequence of choices will output “unsatisfiable”.


• Is there an algorithm to do the following:
  **Input:** Set of 1st order predicate clauses
  **Output:** Is the set satisfiable, yes or no?
• Is there an algorithm to do the following: 
  **Input:** Set of 1st order predicate clauses 
  **Output:** Is the set satisfiable, yes or no? 
• No, there is no such algorithm.
• Is there an algorithm to do the following: 
  **Input:** Set of 1st order predicate clauses 
  **Output:** Is the set satisfiable, yes or no?
• No, there is no such algorithm.
• 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?
• Is there an algorithm to do the following: 
   **Input:** Set of 1st order predicate clauses 
   **Output:** Is the set satisfiable, yes or no?
   • **No**, there is no such algorithm.

• 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?
   • **No**, there is no such algorithm.
• Is there an algorithm to do the following:
  **Input:** Set of 1st order predicate clauses
  **Output:** Is the set satisfiable, yes or no?
  • No, there is no such algorithm.

• 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?
  • No, there is no such algorithm.

Lila Kari, University of Waterloo
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, etc.
  * Turing machines
  * High level pseudo-code

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

Lila Kari, University of Waterloo
Algorithms

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

• E.g., an algorithm to decide if a formula is (universally) valid must output the correct answer (yes/no) for every input formula.
Undecidability

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

- **Validity**: Given a formula in 1st order predicate logic, is it valid?
- **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 $x := 3x + 1$
"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
Turing Machine (TM)

Model of computation/algorithm/program

* Tape (cells)
* Read/write head
* States $q_i$; Input symbols $s_j$
* Rewriting rules $q_i s_j \rightarrow s_k L q_n$
* Start state $q_0$
* Accepting states $q_f$
Turing Machines in action

• Turing machine simulation with JFLAP

https://www.youtube.com/watch?v=IkYhfk4X47c
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 (by contradiction)

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

• $H$ outputs “halt” (Y) or “loop forever” (N)
Proof (by contradiction)

- Assume such a TM exists, call it $H(P, I)$ where $P$ is program and $I$ is input
- $H$ outputs “halt” (Y) or “loop forever” (N)
- We can feed a program $P$ any input, including its own encoding
- What happens if we give $P$ input $P$?
Proof (by contradiction)

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

1) If $H(P, P)$ outputs “halt”, then $K(P)$ goes into an infinite loop printing “ha” at each iteration

2) If $H(P, P)$ outputs “loop forever”, then $K(P)$ halts
M.C. Escher, “Drawing Hands”

Lila Kari, University of Waterloo
Step 2: Call $K(K)$
Step 2: Call $K(K)$. We have two possibilities:

1) If $K$ halts on $K$ then $H(K,K)$ outputs “halt”, which means $K$ loops forever on $K$. 
Step 2: Call $K(K)$. We have two possibilities:

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

2) 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
Mom, look, we've made the turing machine out of a clothes line!

Awesome!

Will it stop? I'll need to hang clothes.

How can we possibly know that?
Proving undecidability

- To show that a new problem $B$ is undecidable 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 undecidability

• If we can transform *every* instance of a known undecidable problem \( A \) into an instance of the new problem \( B \), and

• Solve that

• Then the new problem \( B \) is at least “as hard as” the known undecidable problem \( A \), hence it 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:

\[ M \rightarrow \text{blank-tape halting problem decider} \]

\[ \text{YES} \rightarrow M \text{ halts on blank tape} \]

\[ \text{NO} \rightarrow M \text{ doesn't halt on blank tape} \]
We want to build a decider for the halting problem:

\[ M \rightarrow \text{halting problem decider} \rightarrow \begin{cases} \text{YES} & M \text{ halts on } w \\ \text{NO} & M \text{ doesn’t halt on } w \end{cases} \]
We want to reduce the halting problem to the blank-tape halting problem:
We need to convert one problem instance to the other problem instance.

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 $M$

\[ M_w \]

**Step 1**
- if blank tape
- then write $w$

**Step 2**
- execute $M$
- with input $w$

Karen Lemone, WPI
$M$ halts on input string $w$

if and only if

$M_w$ halts when started with blank tape
Halting problem decider

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

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
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:

- $M \rightarrow \text{state-entry problem decider}$
- $w \rightarrow \text{state-entry problem decider}$
- $q \rightarrow \text{state-entry problem decider}$

If the decider outputs YES, then $M$ enters $q$.
If the decider outputs NO, then $M$ doesn't enter $q$.

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

\[
\begin{align*}
M & \xrightarrow{\text{YES}} M \text{ halts on } w \\
& \xrightarrow{\text{NO}} M \text{ doesn't halt on } w
\end{align*}
\]
We want to reduce the halting problem to the state-entry problem:
We need to convert one problem instance to the other problem instance.

Halting problem decider

\[ M \rightarrow \text{Convert Inputs ?} \rightarrow M' \rightarrow \text{State-entry problem decider} \]

Karen Lemone, WPI
Convert $M$ to $M'$:

- Add new state $q$
- From any halting state of $M$ add transitions to $q$
\( M \) halts on input \( w \)

if and only if

\( M' \) halts on state \( q \) on input \( w \)
Halting problem decider

Generate $M'$

State-entry problem decider

$M$ → $M'$
$q$
$w$

Yes → Yes
No → No

Karen Lemone, WPI
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 $w$?
Another Example

- **Tile System** $T = \text{Finite set of tiles, unlimited supply of each “tile type”}

- **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 $T$?”

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

[Berger66], [Robinson71]
Turing Machines and Tilings

- The Tiling Problem is **undecidable**
- Proof - Simulate a TM with tiles
- For each Turing Machine rule
  \[ q_i s_j \rightarrow s_k L q_n \quad \text{or} \quad q_i s_j \rightarrow s_k R q_n \]
  construct tiles that have those rules encoded in the **glues** on their edges
Alphabet, Action ($q_i s_j \to s_k R q_n$), Merging, and Starting Tiles
Simulation of TM Computations by Valid Tilings
Simulation of TM Computations by Valid Tilings

$q_0 0 \rightarrow X R q_1$
Simulation of TM Computations by Valid Tilings

\[ q_10 \rightarrow 0 \text{ R } q_1 \]

\[ q_00 \rightarrow X \text{ R } q_1 \]
Simulation of TM Computations by Valid Tilings

$q_11 \rightarrow YL \ q_2$

$q_10 \rightarrow 0 \ R \ q_1$

$q_00 \rightarrow X \ R \ q_1$

Lila Kari, University of Waterloo
TM and the Tiling 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 computable (decidable)

The uncomputable (undecidable)
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