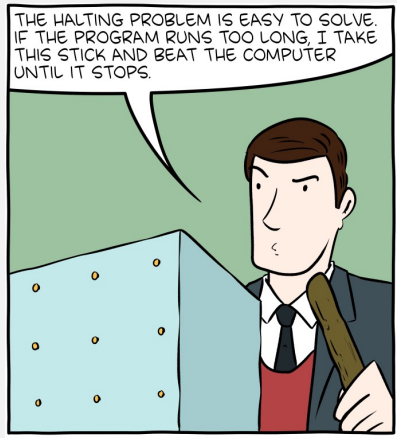
Chapter 5: Reducibility
+ Register Machines I



What if Alan Turing had been an engineer?

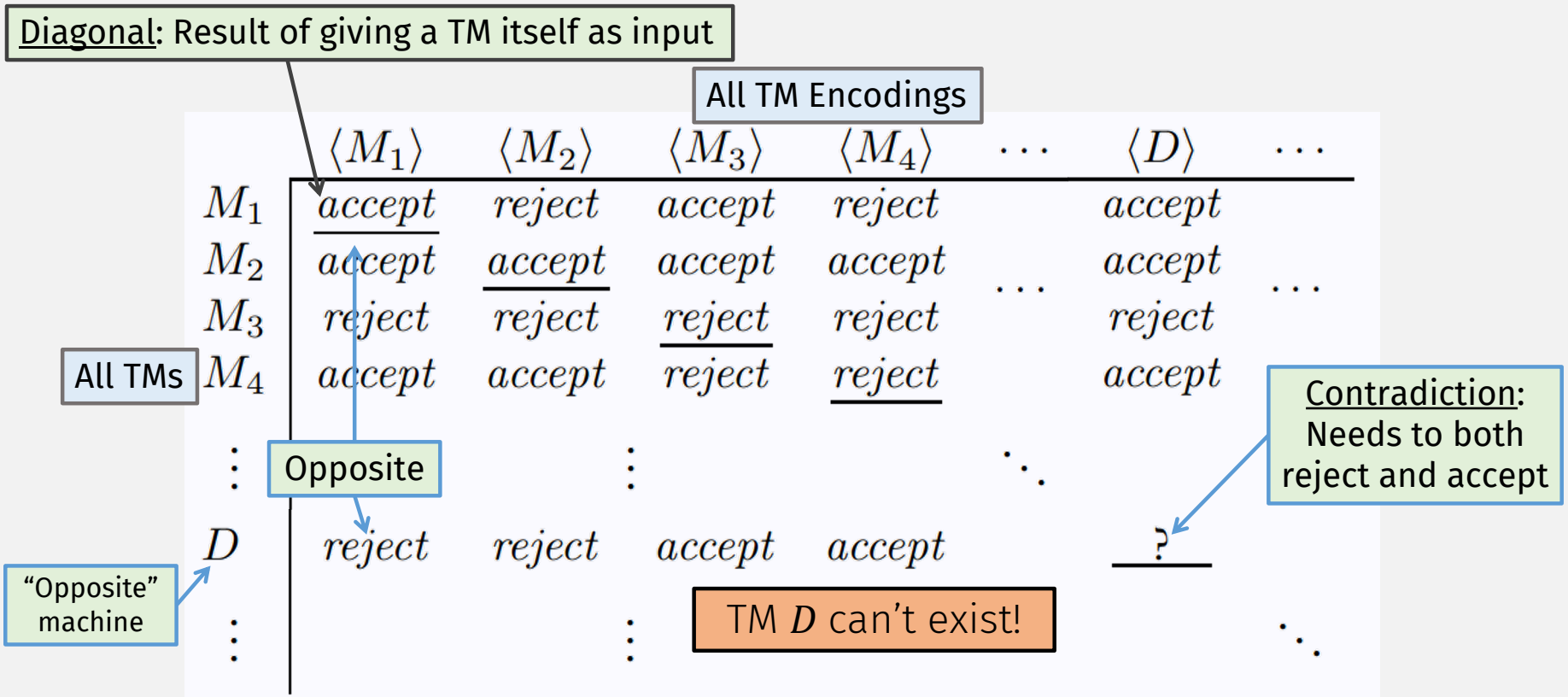
Announcements

- HW8/9 out Frida
- Autograder w finish implementing it all still.
- Meanw owdsou uts / ouputs on Piazza?
  - Easier to ually check w/JFLAP, my imp
  - Good in-between, also happy to steal for actual tests

Announcements

- HW8/9 out Fri sat – Due Wed right before break
- Thank earlier class, just make HW7 due on Thursday @ 10 before exam
- HW7 input output files slow, behind. I will add up _some_
  - Use JFLAP files posted + examples I give
  - Crowdsource examples and I/O examples as a way to prep for exam.
    - I bless this!
- No Reg Machines on this Exam :/// But on optional final
- Friday I'll record lecture; wed we'll do some exam review
- Autograder rety count for exam – 7 tries
- Exam Friday 8am to Sat 10pm (should not be 2x as long, just more flexible timing)
- Open extra NOT FOR CREDIT dropboxes for HW467

Last time: Diagonalization of TMs



Last time: A_{TM} is undecidable

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

Proof, by contradiction.

- Assume A_{TM} is decidable. Then there exists a decider:

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

- If H exists, then we can create D:

"Opposite" machine

D = "On input $\langle M\rangle$, where M is a TM:

1. Run H on input $\langle M, \langle M\rangle\rangle$. ← Result of giving a TM itself as input
2. Output the opposite of what H outputs. That is, if H accepts, *reject*; and if H rejects, *accept*."

Last time: A_{TM} is undecidable

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

Proof, by contradiction.

- Assume A_{TM} is decidable. Then there exists a decider:

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

- If H exists, then we can create D:

D = "On input $\langle M\rangle$, where M is a TM:

1. Run H on input $\langle M, \langle M\rangle\rangle$.
2. Output the opposite of what H outputs. That is, if H accepts, *reject*; and if H rejects, *accept*."

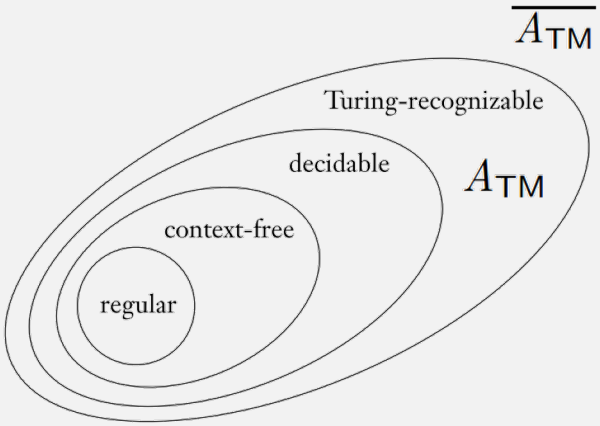
- But D does not exist! Therefore we have a contradiction!

Last time: Unrecognizability

- We've proved:

  A_{TM} is Turing-recognizable

  A_{TM} is undecidable



- And:

THEOREM 4.22

A language is decidable iff it is Turing-recognizable and co-Turing-recognizable.

- So:

  $\overline{A_{TM}}$ is not Turing-recognizable

Proving Undecidability

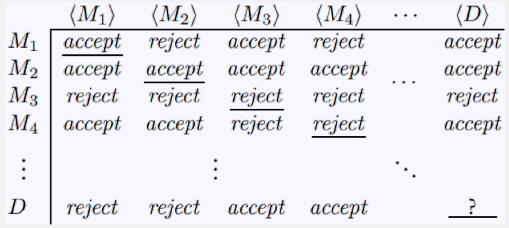
- It's easy to prove things are decidable
  - Construct a decider!

- It's more difficult to prove things are undecidable
  - We have proved only one: $A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$
  - And it was *complicated!*

- Fortunately, the first one is the hardest!
  - Now we use A_{TM} to more easily prove other languages undecidable!

Today: Easier Undecidability Proofs!

- We proved $A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ undecidable by …
- … showing that its decider could be used to implement an impossible "D" decider.
- In other words, we **reduced** A_{TM} to the "D" problem.
  - That was hard (needed to invent diagonalization)



	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\cdots$	$\langle D\rangle$
M_1	*accept*	*reject*	*accept*	*reject*		*accept*
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*
M_3	*reject*	*reject*	*reject*	*reject*		*reject*
M_4	*accept*	*accept*	*reject*	*reject*		*accept*
$\vdots$			$\vdots$		$\ddots$	
D	*reject*	*reject*	*accept*	*accept*		?

- But now we can reduce problems to A_{TM}: much easier!

The Halting Problem

$$HALT_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Thm: $HALT_{\text{TM}}$ is undecidable

Proof, by contradiction:

- Assume $HALT_{\text{TM}}$ has *decider* R; use it to create decider for A_{TM}:

S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, *reject*. ← This means M loops on input w
3. If R accepts, simulate M on w until it halts. ← This step always halts
4. If M has accepted, *accept*; if M has rejected, *reject*."

The Halting Problem

$$HALT_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Thm: $HALT_{\text{TM}}$ is undecidable

Proof, by contradiction:

- Assume $HALT_{\text{TM}}$ has *decider* R; use it to create decider for A_{TM}:

S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

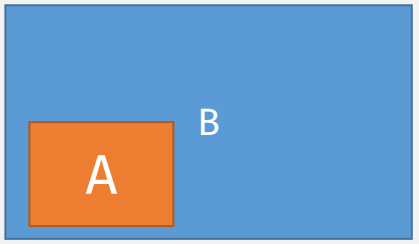
1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*."

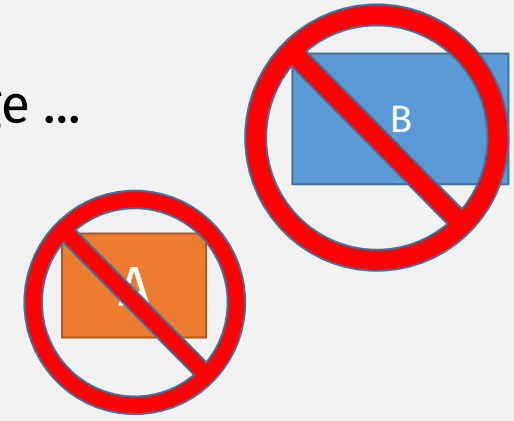
- But A_{TM} is undecidable!
  - I.e., this decider that we just created cannot exist! So $HALT_{\text{TM}}$ is undecidable

Easier Undecidability Proofs

In general, to prove the undecidability of a language:

- Use proof by contradiction:
- Assume the language is decidable,
- Show that its decider can be used to create a decider for …
- … a known undecidable language …
- … which doesn't have a decider!





Summary: Languages About Machines

- $A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$ — Decidable
- $A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ — Decidable
- $A_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ — **Undecidable**
- $E_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$ — Decidable
- $E_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$ — Decidable
- today $E_{\text{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$ — **Undecidable**
- $EQ_{\text{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ — Decidable
- $EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$ — **Undecidable**
- today $EQ_{\text{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$ — **Undecidable**

Reducibility: Modifying the TM

$E_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

Thm: E_{TM} is undecidable

Proof, by contradiction:

- Assume E_{TM} has *decider* R; use to create A_{TM} *decider*:

S = "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

First, construct M_1

- Run R on input $\langle M \rangle$ $\boxed{1}$ ← **Note:** M is only an arg; we never actually run M!
- If R accepts, *reject* (because it means $\langle M \rangle$ doesn't accept w)
- if R rejects, then *accept* ($\langle M \rangle$ accepts w)

- Idea: Wrap $\langle M \rangle$ in a new TM that can *only* accept w:

M_1 = "On input x:
1. If $x \neq w$, *reject*.
2. If $x = w$, run M on input w and *accept* if M does."

Reducibility: Modifying the TM

$E_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

Thm: E_{TM} is undecidable

Proof, by contradiction:

- Assume E_{TM} has *decider* R; use to create A_{TM} *decider*:

$S =$ "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

First, construct M_1

- Run R on input $\langle M_1 \rangle$
- If R accepts, *reject* (because it means $\langle M \rangle$ doesn't accept w)
- if R rejects, then *accept* ($\langle M \rangle$ accepts w)

- Idea: Wrap $\langle M \rangle$ in a new TM that can *only* accept w:

$M_1 =$ "On input x:
1. If $x \neq w$, *reject*.
2. If $x = w$, run M on input w and *accept* if M does."

Reduce to something else: EQ_{TM} is undecidable

$$EQ_{\mathsf{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

Proof, by contradiction:

$$E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$$

- Assume EQ_{TM} has *decider* R; use to create ~~A_{TM}~~ *decider*:

$S =$ "On input $\langle M\rangle$, where M is a TM:

1. Run R on input $\langle M, M_1\rangle$, where M_1 is a TM that rejects all inputs.
2. If R accepts, *accept*; if R rejects, *reject*."

Reduce to something else: EQ_{TM} is undecidable

$$EQ_{\mathsf{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

Proof, by contradiction:

$$E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$$

- Assume EQ_{TM} has *decider* R; use to create ~~A_{TM}~~ *decider*:

S = ~~"On input $\langle M\rangle$, where M is a TM:~~

1. Run R on input $\langle M, M_1\rangle$, where M_1 is a TM that rejects all inputs.
2. If R accepts, *accept*; if R rejects, *reject*."

- But E_{TM} is undecidable!

Summary

- $A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$ — Decidable
- $A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ — Decidable
- $A_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ — **Undecidable**
- $E_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$ — Decidable
- $E_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$ — Decidable
- today $E_{\text{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$ — **Undecidable**
- $EQ_{\text{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ — Decidable
- $EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$ — **Undecidable**
- today $EQ_{\text{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$ — **Undecidable**

We can't compute <u>anything</u> about Turing Machines, i.e., about programs!

Also Undecidable …

- $REGULAR_{\text{TM}}$ = {<M> | M is a TM and $L(M)$ is a regular language}
- $CONTEXTFREE_{\text{TM}}$ = {<M> | M is a TM and $L(M)$ is a CFL}
- $DECIDABLE_{\text{TM}}$ = {<M> | M is a TM and $L(M)$ is a decidable language}
- $FINITE_{\text{TM}}$ = {<M> | M is a TM and $L(M)$ is a finite language}
- …

Rice's Theorem

- $ANYTHING_{\text{TM}}$ = {<M> | M is a TM and "something something" about $L(M)$}

Rice's Theorem: Every non-trivial extensional property of programs is *undecidable*.

- Extensional equivalence - equivalence modulo function they compute – behavior
- Non-trivial – besides things true of all and no programs
- Extensional – not about the implementation, but the behavior

Think "asking about the function, not the machine"

Register Machines I

Text Register Machines

- Updated, more modern primitive model of computation
- Contains a finite number of registers
- Some computational unit
- (Text) register machines hold program text

Registers – hold data. Queue style input

R1:					
R2:					
R3:					

Simple Syntax, Simple Instruction Set

- Programs consist entirely of 1 and # symbols
- These are the text of programs, and also the data in registers
- “Code is data”

Instruction list

Add 1 to Rn	Add # to Rn	Go forward n	Go backward n	Cases on Rn
1^n#	1^n##	1^n###	1^n####	1^n#####

Cases

- What, if any, can the first value in a register _be_?
- If Rn is empty, we go to the very next instruction.
- If the first symbol of Rn is 1, we delete that symbol and go to the second instruction after the case instruction.
- If the first symbol of Rn is #, we delete that symbol and go to the third instruction after the case instruction.

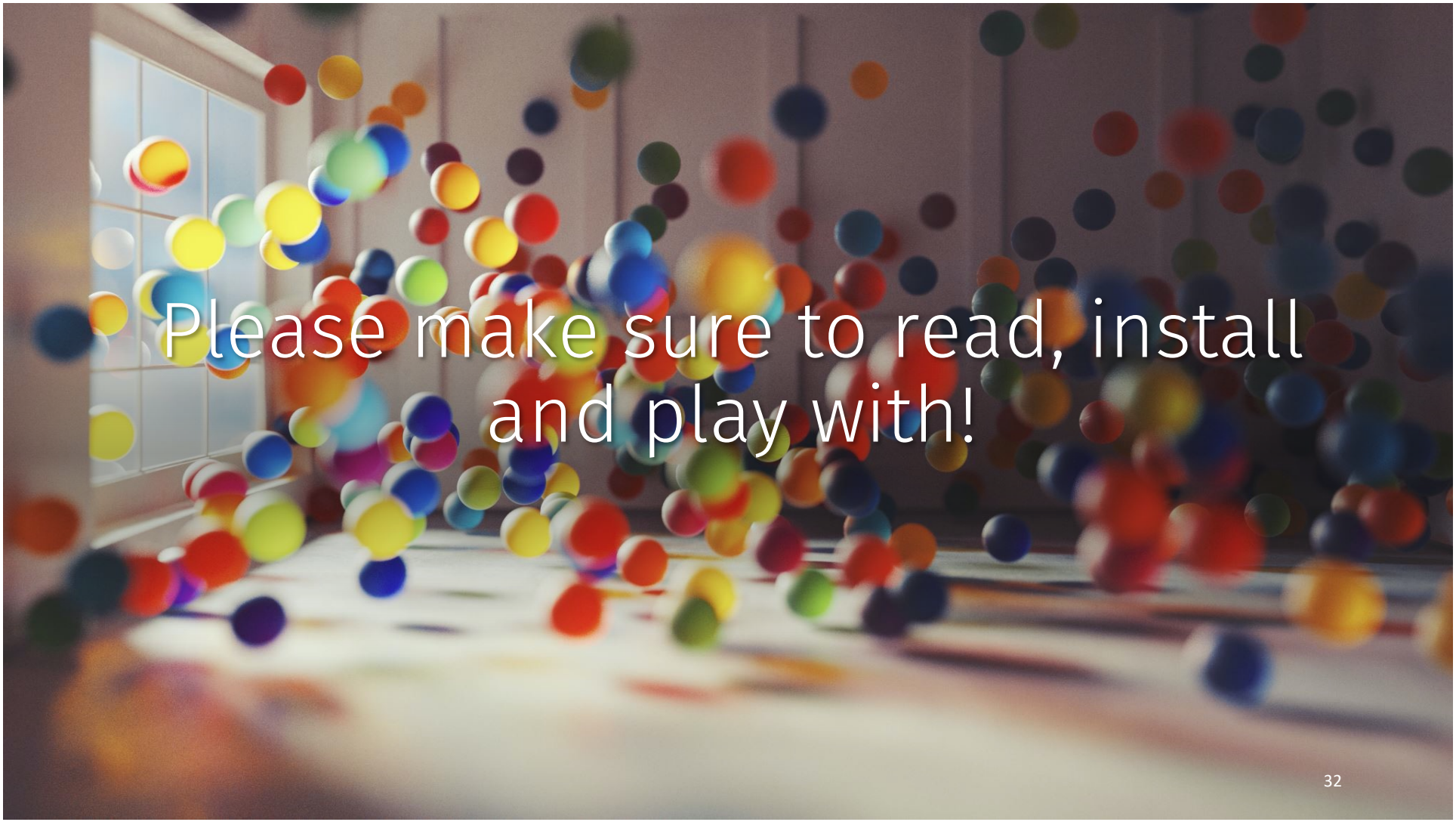
What does this program do?

Add 1 to Rn	Add # to Rn	Go forward n	Go backward n	Cases on Rn
1^n#	1^n##	1^n###	1^n####	1^n#####

11#####	Cases on R2
111111###	Go forward 6: we're done!
111###	Go forward 3
1##	Add # to R1
1111####	Go backward 4 (to the top)
1#	Add 1 to R1
111111####	Go backward 6 (to the top)

“Halt” vs. “Stops improperly”

- A program halts if control transfers right below the last instruction of the program p
- A program stops improperly if control ever moves before the program begins or more than 1 instruction after the end
- (Why?)


Please make sure to read, install
and play with!