Computed: follow a fixed procedure and produce an answer (halt), aka algorithm.

What can be computed? What cannot? What can be computed efficiently (and how)?

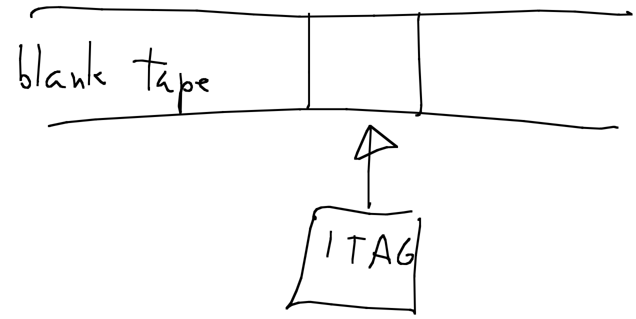
If a single question really is answerable "yes" or "no", then one of the machines, Myes or Mno, computes the answer. We just don't know which one is correct.

Any finite set of examples can be computed: just make a table and look up the answer.

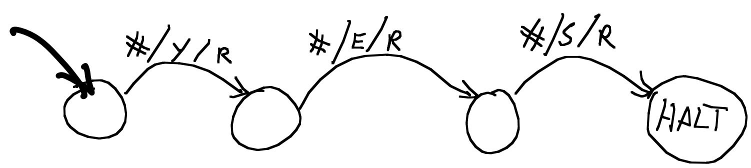
Are all programs (TMs) algorithms? No.

```
for (i = 1; i > 0; i = 1){
    j = j+1;
}
```

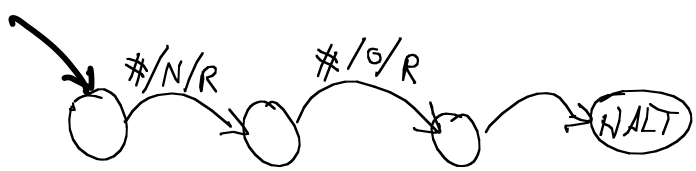
"Is There A God" machine:
Prints answer and halts.

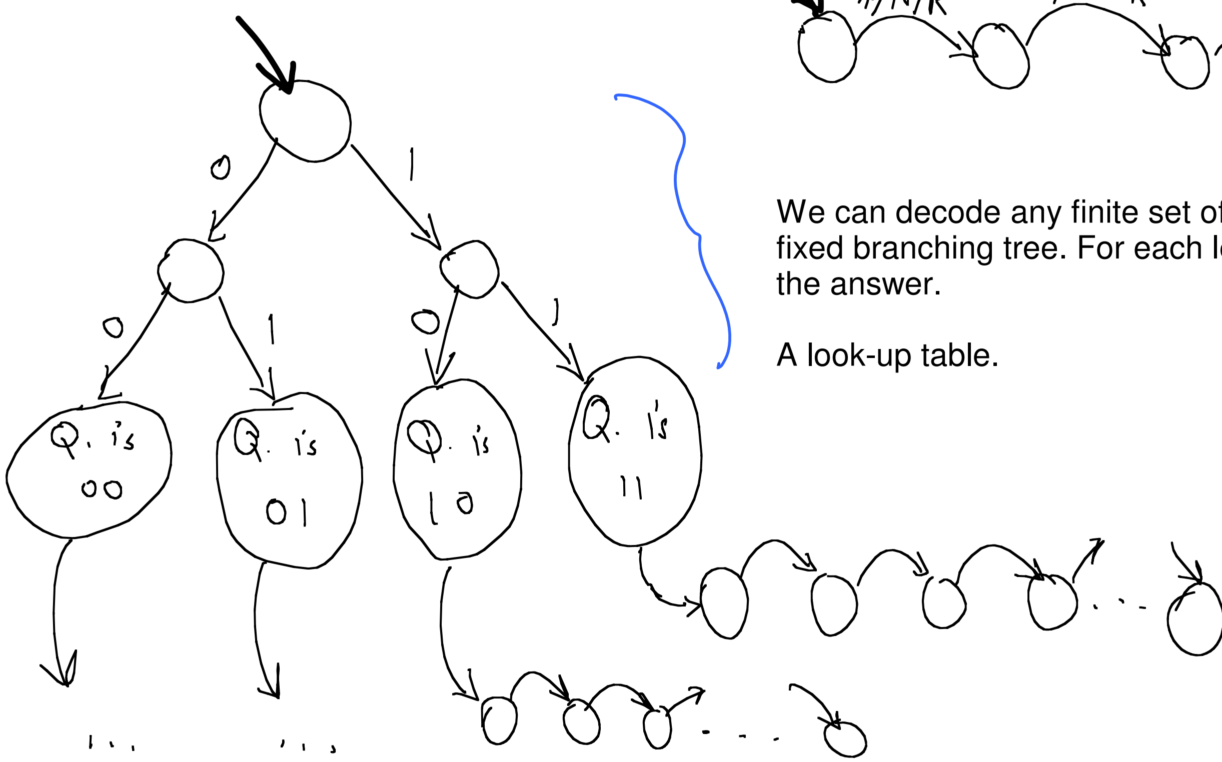


M_{yes}:



M_{NO}:





We can decode any finite set of questions using a fixed branching tree. For each leaf, we simply print the answer.

A look-up table.

prints answer for Q. = 11

Prints answer for Q. = 10

Computability (aka recursive)

Fermat's Last Theorem

Conjecture: There are no solutions to,

$$x^n + y^n = z^n$$

where n, x, y, and z are positive integers and n > 2.

Suppose we didn't know if it was true.

Proved in 1995: Frey, Ribet, Wiles, Taylor.

Suppose we asked if the question, "Is Fermat's Last Theorem true?" is computable.

$\Rightarrow$ Of course. Use either M_{yes} or M_{no}.

Supposed we asked: Is this computable?

Given some positive integer n > 2, is there a solution to,

$$x^n + y^n = z^n$$

where x, y, and z are positive integers?

If Fermat's Last Theorem is true, then M_{no} will work w/o modification. $\Rightarrow$ computable

Suppose it weren't true? That is, there are sol'ns for some n, but not all n. How would we go about it?

FLT(n)

pick next (x, y, z)
check $x^n + y^n = z^n$

?!? Will it halt? For every n?

How many questions are there? How many TMs? DIAGONALIZATION

In our encoding, we used a string of 0s and 1s to represent a TM. Symbol set is {0, 1}.

--- Each TM can be identified with an integer. (There are infinitely many machines that do the same thing.)

--- Each input tape configuration can be identified with an integer.

--- Each output tape configuration can be identified with an integer.

--- Each TM can be looked at as an integer function: given input, x, machine M produces integer M(x).
---NB M might loop forever on some inputs, if so then M is a "partial" function.

input

TM	0	1	2	3	4	...
M_0	$M_0(0)$	$M_0(1)$	$M_0(2)$	$M_0(3)$	G	...
M_1	$M_1(0)$	$M_1(1)$	$M_1(2)$			...
M_2	$M_2(0)$	$M_2(1)$				...
M_2						
⋮			⋮			

G = loops forever

← all outputs for $M_\emptyset$

← all outputs for M_1

Computable (real) numbers:
Given e, output finite number of digits of x so that the output is within e of x.
PI is such a number.

How many integer functions are there?

--- Diagonalization:
g(0) != M0(0)
g(1) != M1(1)
g(2) != M2(2)
...

---- g() is not in the list!

--- How many different ways are there to pick g()?
g(0) is any element from N - { M0(0) }
g(1) is any element from N - { M1(1) }
g(2) is any element from N - { M2(2) }
...

The g()s are so numerous proportionally, that the probability of randomly picking a TM function from a bag of integer functions is 0.

[What the heck does that really mean?]

input

TM	0	1	2	
M_0	$\neq$			...
M_1		$\neq$		...
M_2			$\neq$	...
⋮				⋱

Maybe it only means we don't know how to arrange an infinite list of TMs? We are limited in our own computing power?

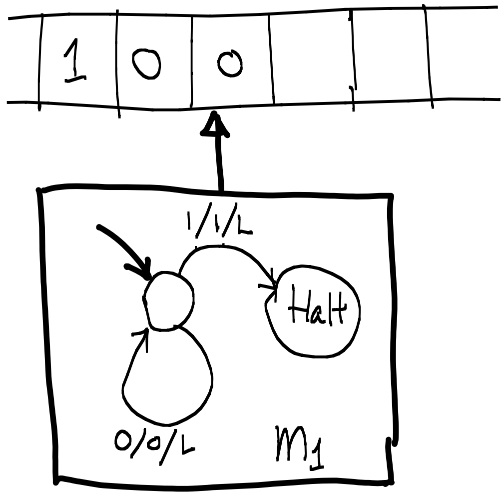
How "numerous" is "infinity to the infinity"?

As long as we are building TMs, lets see how to simplify our work.

How about combining two TMs to make a new one?

TM M_1:

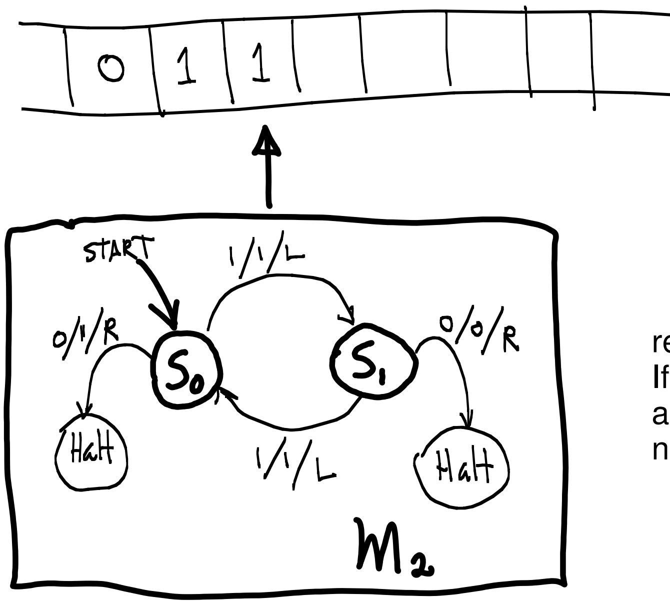
"spin left ϕ_s"



skip 0's to the left, stop at the first 1, end up to its left.

TM M_2:

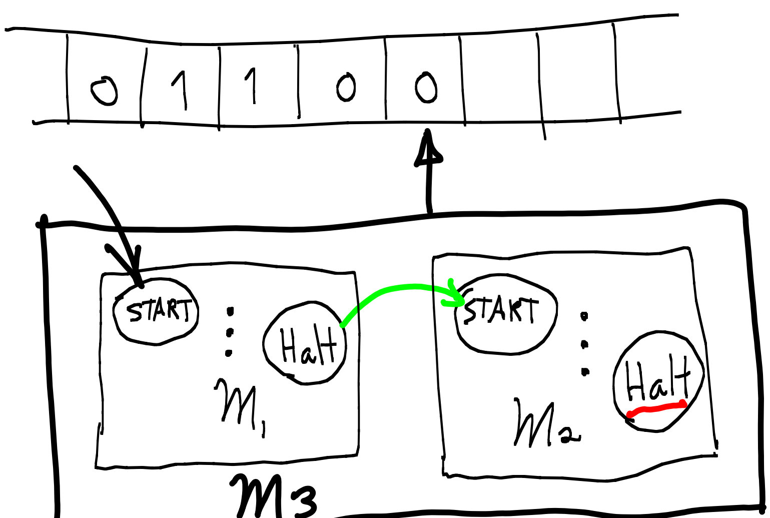
"Parity fix"



read 1's leftward until finding a 0. If even number of 1's, add another 1 on left; if odd do nothing. Halt at left end of input.

TM M_3:

"spin left ϕ_s"
then
"parity fix"



M3 starts in M1's FSM start state.

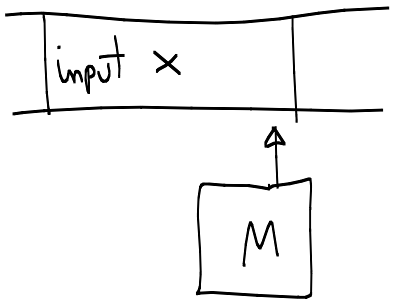
- Every M1 state transistion that goes to M1's "HALT" state is instead connected to M2's START state.
- M3's halting state is M2's "HALT" state.

Lemma: All TM's with x as input, either (1) HALT or (2) LOOP FOREVER.   (exercise: prove the lemma.)

The "Halting" integer function:

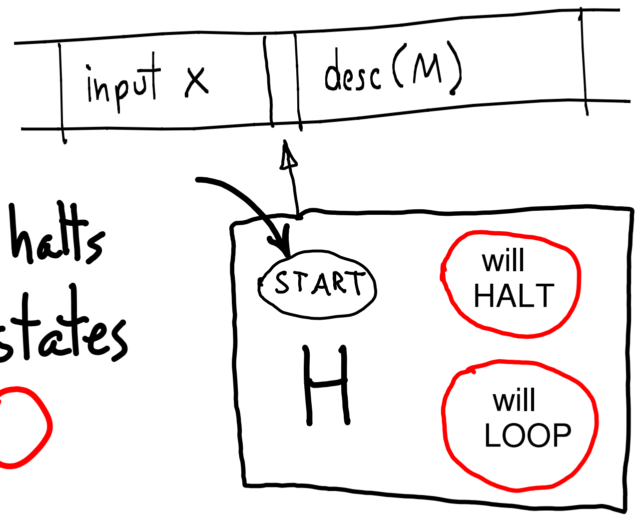
input: integer xM   (xM == an encoding of input x followed by an encoding of a TM, M.)

output: "1" if xM HALTS;   (xM == M reading x as its input.)
"0" for all other cases



Q. what will happen?

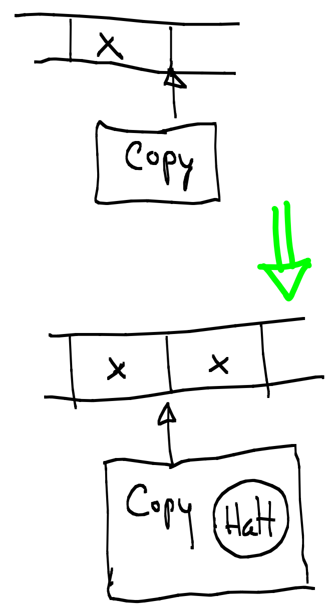
H halts in states

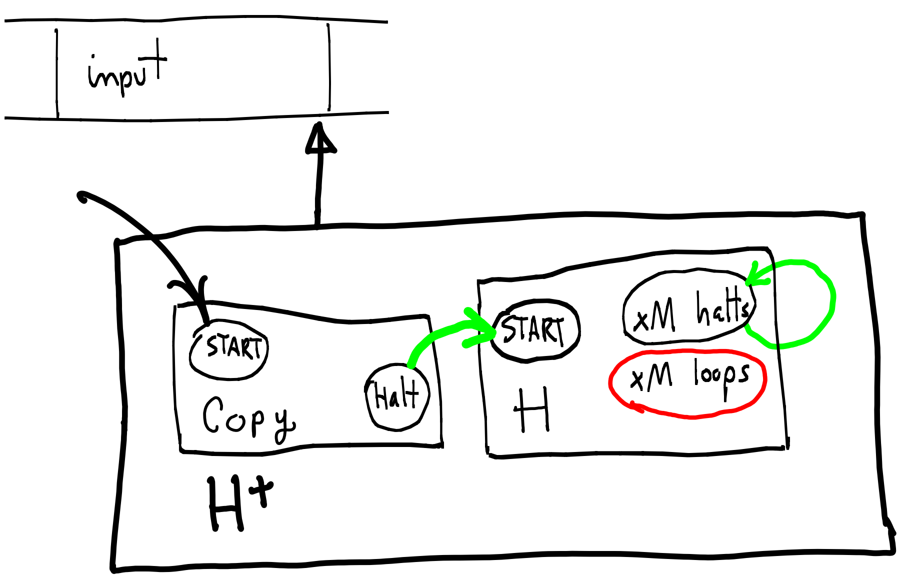


Asummption: Either (H exists) IS TRUE, or (H does not exist) IS TRUE.

Suppose (H exists) IS TRUE.

Then we can build another machine, **H+**, using H and a "Copy" TM.

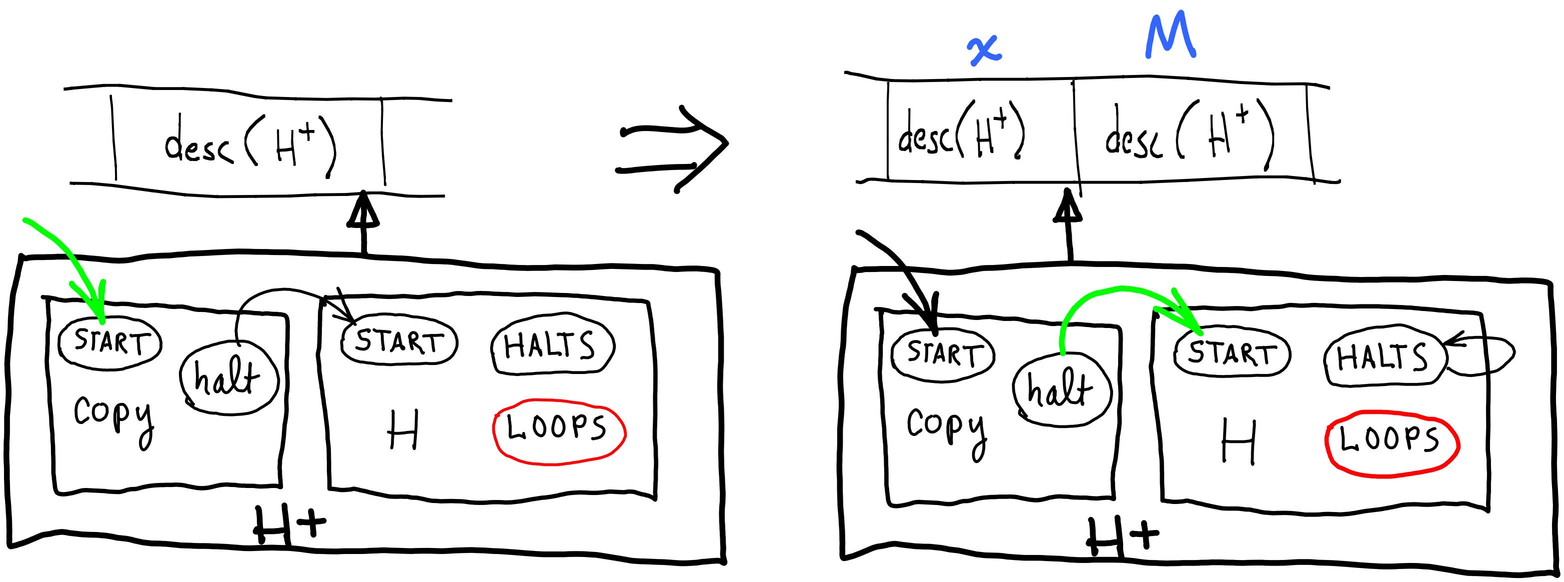




H+
1. makes a copy of its input.
2. does whatever H would do.

WHEN H+ reaches
1. "xM halts", H+ LOOPS.
2. "xM loops", H+ HALTS.

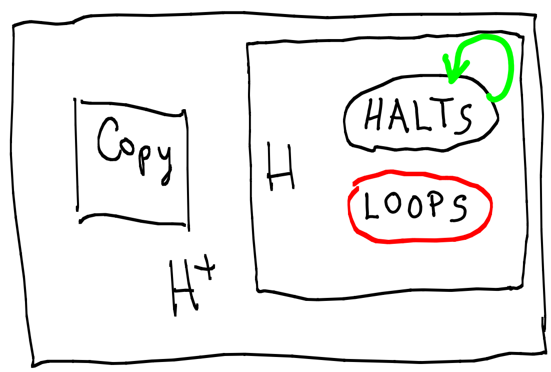
Consider putting desc(H+) on H+'s input tape. What must happen?



H+ first does exactly what Copy would do, copy its input. Next, H+ starts doing exactly what H would do.

The tape is now thought of as an input "desc(H+)", followed by a description of H+.

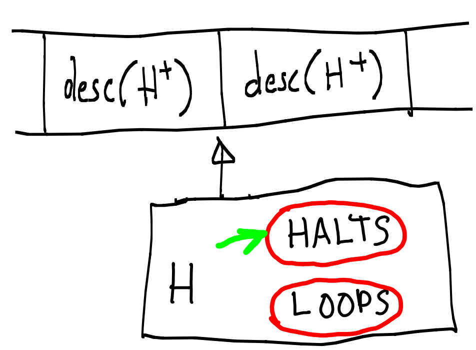
H+ WILL either (A: reach "HALTS" and loop) OR (B: reach "LOOPS" and halt).

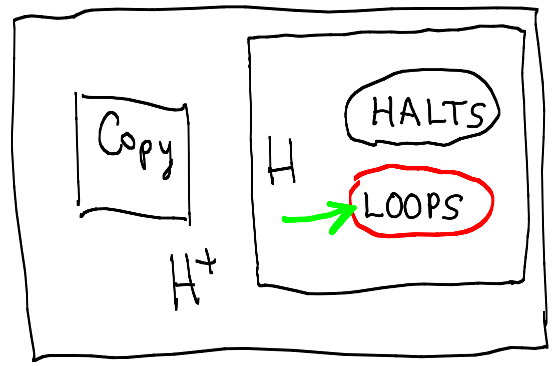


(A.) SUPPOSE desc(H+)H+ loops.

1. H+ reached HALTS.
2. Then H with input xM == desc(H+)desc(H+),
would have halted in HALTS.
3. BUT desc(H+)H+ loops.
4. Since H is correct, this cannot happen.
5. (A.) cannot happen: desc(H+)H+ cannot reach HALTS.

We assumed H is correct.
So, we supposed wrongly that H+ loops.

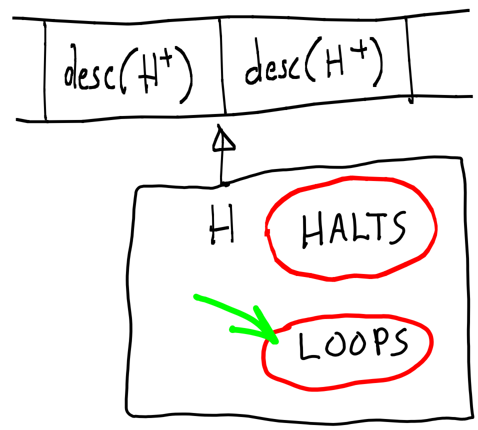




(B.) SUPPOSE desc(H+)H+ halts.

1. H+ reached LOOPS.
2. H reading desc(H+)desc(H+) must reach LOOPS.
3. BUT desc(H+)H+ halts.
4. H is correct; so, H cannot reach LOOPS.
5. (B.) cannot happen: desc(H+)H+ cannot reach LOOPS.

We assumed H is correct.
So we assumed wrongly that H+ halts.



Are we doomed?

Build something H- that partially computes the Halting Problem?

Works for some inputs, but not others?

Works for some fixed number of inputs?

Has a lookup table?

How many machines act exactly like any given description?

How many descriptions are there?

How many other things are not Turing computable? What does this say about cognition? ...???

Another Method?

```
Hnew( x, M)

print "loops forever"

1. Simulate xM for one step.
2. If  xM halted
        print "halts"
   else
        go to 1.
```

$\Rightarrow$ Is HP computable?

Bottom Line

Suppose we try to write a program $H(x, M)$.
We succeed for some special cases $\{M_1, M_{25}, M_{300}, \dots\}$
But, we always find a new M_i and have to rewrite $H(x, M)$. Also, we get it to work for $\{x_1, x_2, \dots\}$, but find a new x_i for which $x_i M_j$ loops (or halts) (if we can figure that out).

HP $\Rightarrow$ We will never be bored!

Formal Proof

Notation: "[halts]" means "H+ halts when reading its own description"; "[loops]" is to be read similarly; "==>" means, "implies", in the logical sense of material implication; "-" means logical NOT.

1. (H exists) ==> (H+ exists (is a TM)) (by properties of TM)

2. (H+ exists) ==> [halts] OR [loops] (by properties of TM)

3. (H+ exists) ==> -[loops] AND -[halts] (demonstrated above)

4. (H exists) ==> ([halts] OR [loops]) AND (-[loops] AND -[halts]) (by 1. and 2.)

5. (H exists) ==> ([halts] AND -[halts]) OR ([loops] AND -[loops]) (by AND/OR properties)

6. p ==> q EQUALS -q ==> -p (by properties of "==>")

7. -(([halts] AND -[halts]) OR ([loops] AND -[loops])) ==> -(H exists) (by 5. and 6.)

8. -(([halts] AND -[halts]) OR ([loops] AND -[loops])) (true by AND/OR properties)

9. -(H exists) (syllogism applied to 7. and 8.)