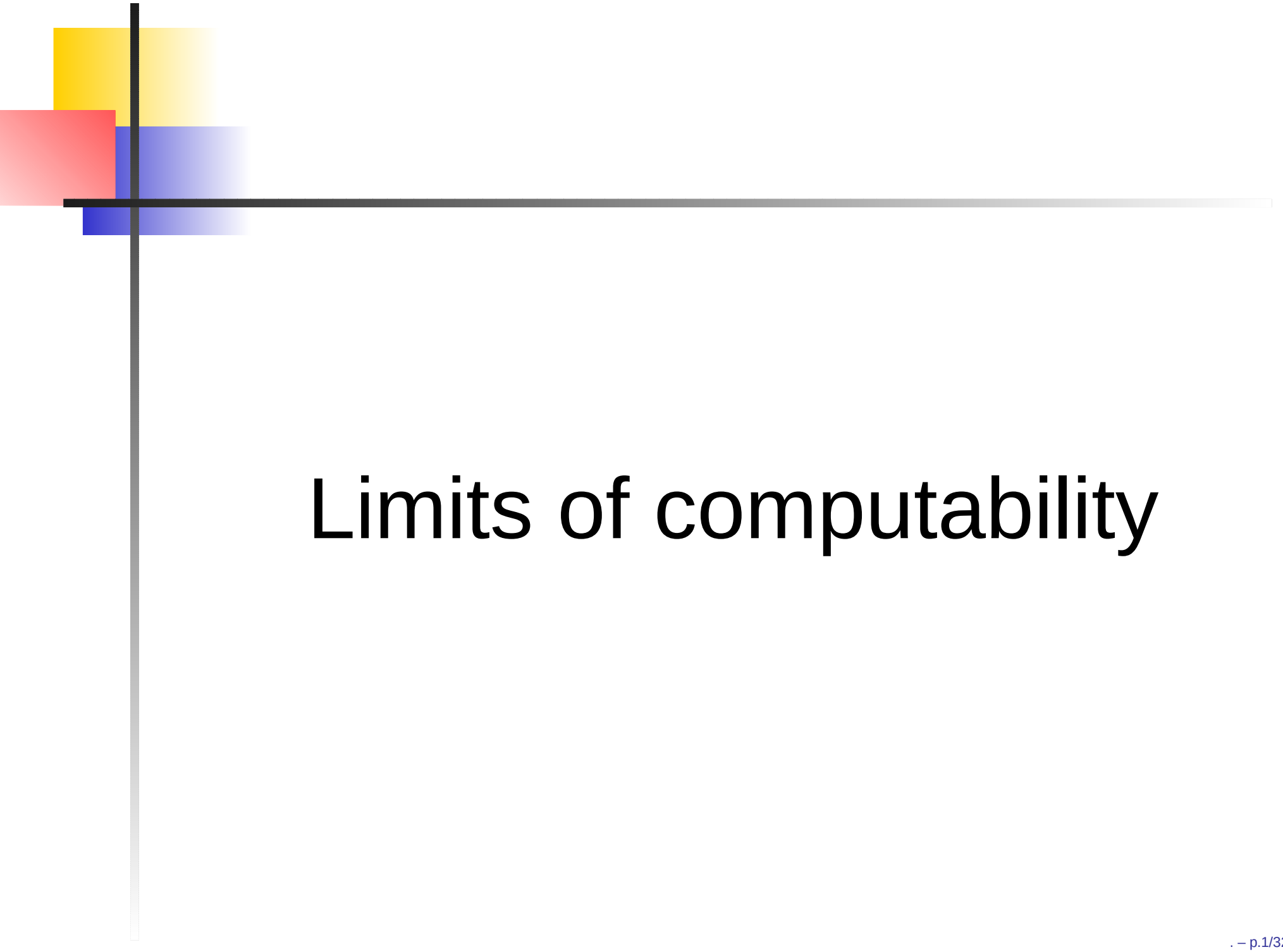


Limits of computability

Does this program exist?

Is it possible to write a Java method of the form

```
public boolean test(String program,String input) {
   // some code here
}
```

such that `test` returns

- `true` if the string `program` is the code of a Java method that prints “Hello world!” when called with `input`, and
- `false` otherwise?

Does this program exist?

For example, if the string `program` is

```
public void abc(String s) { if( s == "abc" ) { print
  "Hello world!"; } else { print "Whatever"; } }
```

then

- `test` called with `program` and `input` “abc” yields `true`;
- `test` called with `program` and any other `input` yields `false`.

A helper function

To see if `test` exists, we shall consider a funny program on the next slide. First, we need a helper function: let

`enumerate`

be an enumeration for the set

$\{(x, y, z, n) : x, y, z, n$ are positive integers and $n \geq 3\}$.

This can could be written in Java as a method

```
int[] enumerate(int i) { ... }
```

(Remember Cantor's Zig-Zag!)

maybeHelloWorld

```
String maybeHelloWorld(String s) {
    int x, y, z, n, i;

    i = 1;

    while( true ) {
        (x,y,z,n) = enumerate( i );
        if( exp(x,n) + exp(y,n) == exp(z,n) ) {
            print("Hello world!");
            break;
        }
        i = i + 1;
    }
}
```

`maybeHelloWorld`

The program `maybeHelloWorld` does the following:

- If there are positive integers x, y, z, n such that $n \geq 3$ and

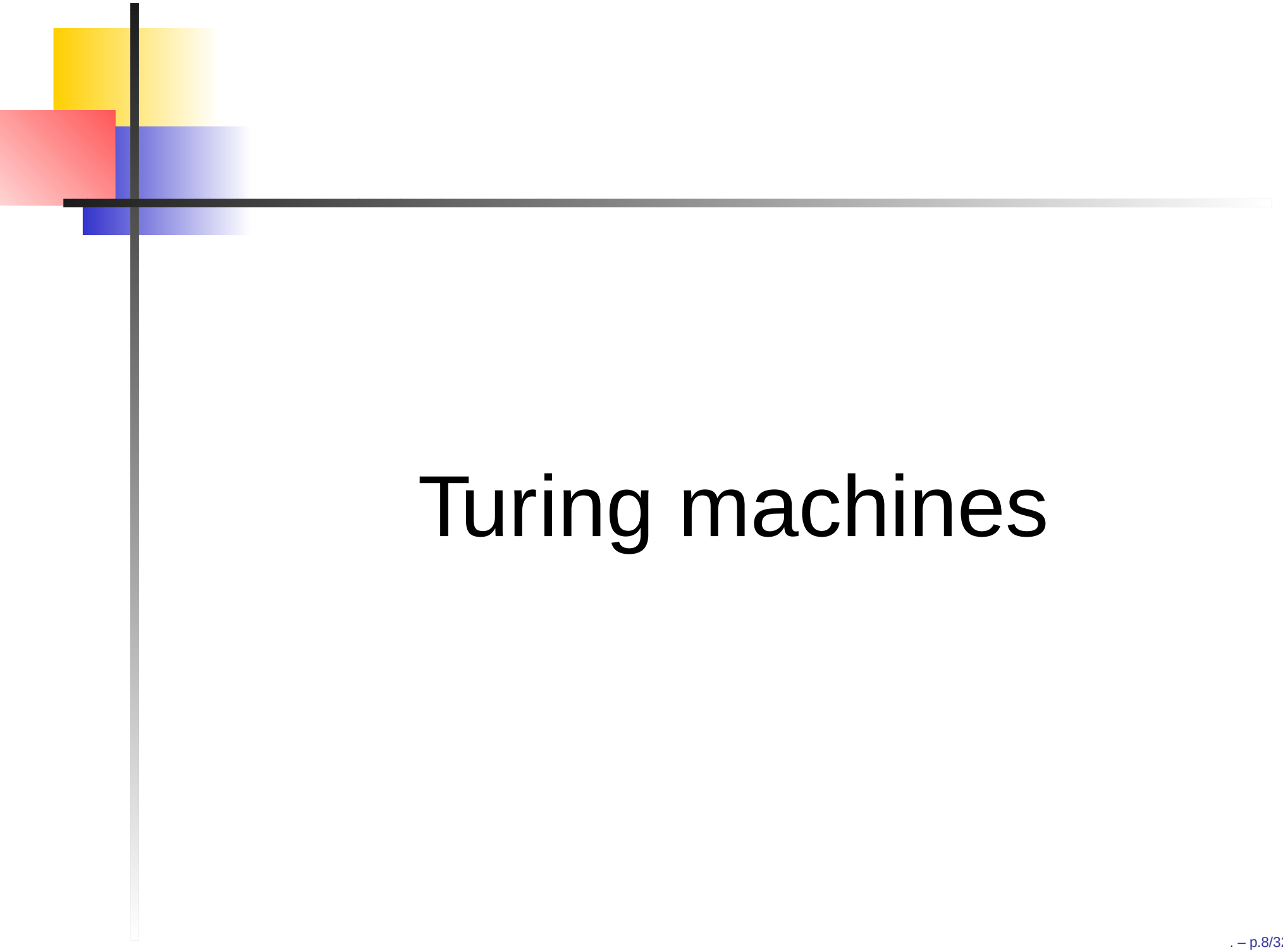
$$x^n + y^n = z^n,$$

  the it prints “Hello World!”.
- Otherwise, it goes into an infinite loop.
- What happens if we call `test` on `maybeHelloWorld` (and any `input`)?

test2

```
public void test2(String program) {
  if(test(program,program)==false) {
     print("Hello world!");
  } else {
     print("Whatever");
  }
}
```

What happens if we call `test2` with its own code as argument?



Turing machines

Motivation

- Real-life programming languages are too big and messy to make precise claims about computability.
- To address this issue, we shall use **idealized** notions of computation.
- The first such notion we shall study is that of **Turing machine**.
- Introduced by Alan Turing in the 1930's before programming languages even existed.

Computable functions

A function is defined to be

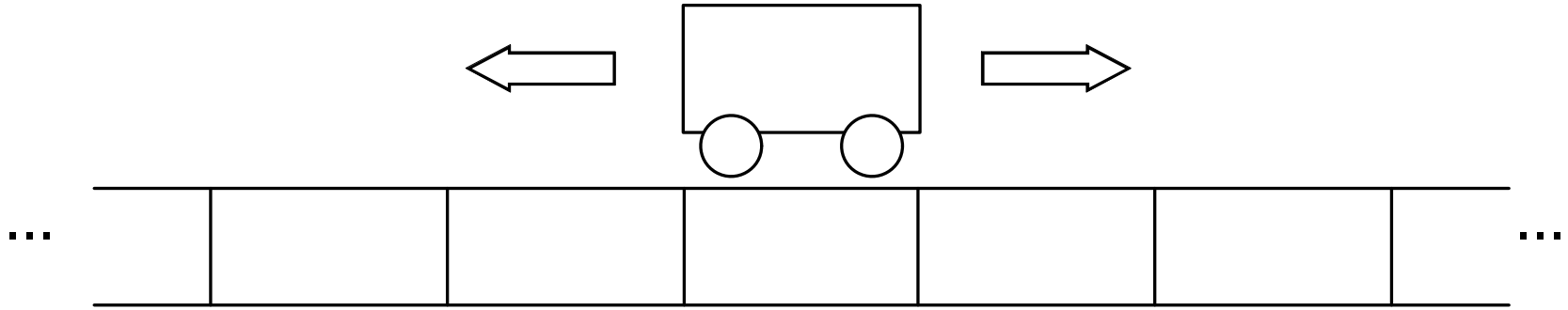
- **effectively computable** if there is a finite list of instructions by following which one could in principle compute its value for any given argument.
- **Turing computable** if it is computable by a Turing machine.

Turing’s thesis

- It will be obvious that “Turing-computable” implies “effectively computable”.
- The converse is called “Turing’s thesis”.
- Turing’s thesis cannot be proved, but we shall accumulate evidence during this course.

Turing machine

movable "tape head" (can read and write)



infinite tape consisting of "squares"

Configurations of a Turing Machine

- At any time, the tape head is in one of a finite number of states, and it is scanning one particular square.
- Each tape square is blank (written as 0) or contains a stroke (written as 1).
- We require that at any given time there are only finitely many 1's.

Representation of configurations

- Tapes are represented by binary strings, e.g. 1101101.
- There are supposed to be infinitely many 0's on the left and right.
- Configurations are represented e.g. like 11_q01101. The state (here: q) is written as a subscript on the scanned symbol.

Actions of the tape head

- W_0: write 0 into current square
- W_1: write 1 into current square
- L: move one square to the left
- R: move one square to the right

Formal definition of a Turing machine

Definition. A **Turing machine** (TM) is given by:

- A set Q of states.
- A **tape alphabet** Σ (we shall use $\Sigma = \{0, 1\}$).
- A transition function $\delta : Q \times \Sigma \rightharpoonup Q \times \{W_0, W_1, L, R\}$. The value of $\delta(q, a)$, if defined, is a pair (q', X) consisting of a next state q' and an action X.
- An initial state q_0.

Note the similarity with DFA.

How a TM operates

Let q be the state of the head, and a the **scanned symbol** (i.e. the symbol underneath). Then

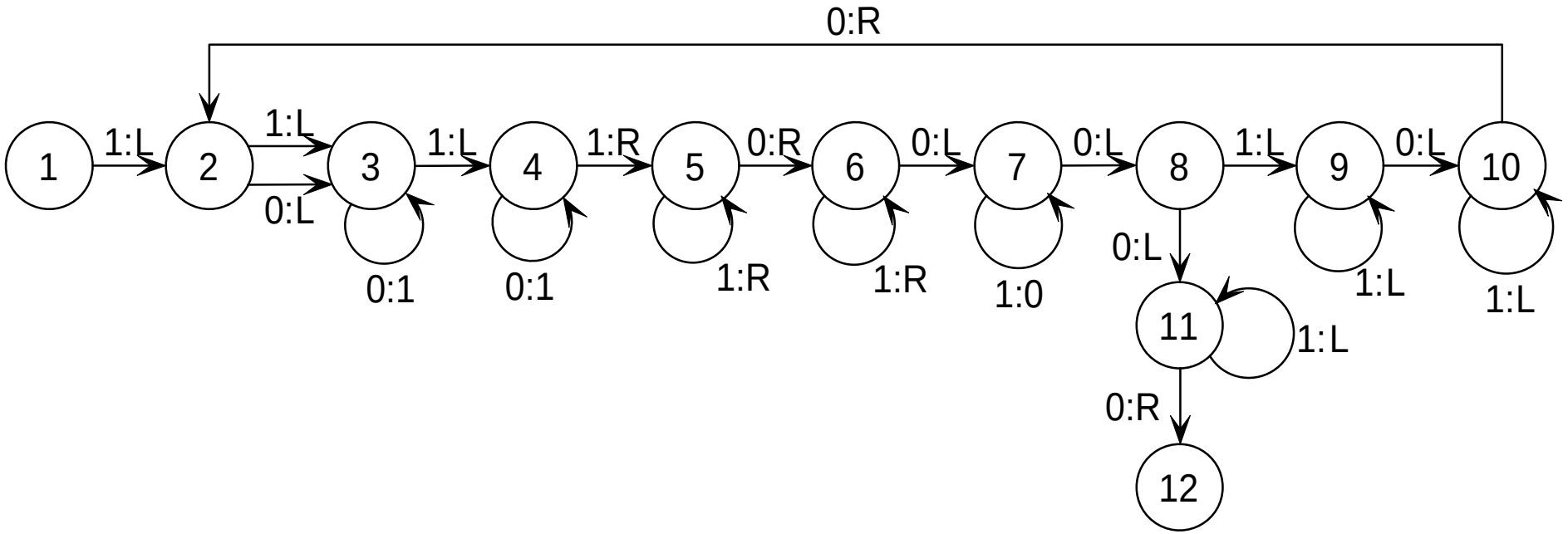
- If $\delta(q, a)$ is undefined, the machine **halts**.
- Otherwise, let (q', X) be $\delta(q, a)$. The head executes action X and goes to state q'.
- If X is W_0, the head writes 0.
- If X is W_1, the head writes 1.
- If X is L, the head moves left
- If X is R, the head moves right.

Transition table

A Turing machine can be presented as a **transition table**, for example:

	0	1
q_0	$W_1 q_0$	$L q_1$
q_1	$W_1 q_1$	$L q_2$
q_2	$W_1 q_2$	

Example: doubling the number of 1s



Some functions computable by TM's

- The machine for doubling strokes can be seen as an implementation of the function that sends the positive integer x to $2 * x$.
- There is also trivial machine for addition: it needs to do nothing, because the number of strokes after the computation is deemed to be the result.
- There is also a Turing machine for multiplication $y * x$ (see Boolos/Burgess/Jeffrey, somewhat messy).

Functions computable by TM's

- We shall gather strong evidence that every effectively computable function is computable by a TM (i.e. that Turing's Thesis holds).
- We shall focus on functions that take k natural numbers to a natural number, for some $k \geq 1$:

$$f : N^k \longrightarrow N.$$

- To state what it means for such an f to be Turing-computable, we need some definitions.

Representing natural numbers

- The argument given to the machine is only a **representation** of a number: string of digits.
- Decimal system, e.g. 1969.
- Binary system: e.g. 11110110001.
- Roman numerals: e.g. MCMLXIX.
- **Monadic** or **tally** notation: e.g. the number five is represented by five strokes: 11111.
- The representation turns out to be inessential; for Turing machines, we choose monadic notation.

Standard initial configurations

Definition. A **standard initial configuration** for arguments $x_1, \ldots, x_k$ is a configuration such that

- there are k blocks of strokes, separated by single blanks;
- the i-th block consists of x_i strokes;
- the head is in the start state (state 1) and scans the leftmost stroke. (E.g. $1_{q_0}1101111011$ is the standard initial configuration for arguments $3, 4, 2$.)

Halting configurations

Definition. A **halting configuration** is a configuration that allows no further transition (i.e. if q is the current state and a is the scanned symbol, then $\delta(q, a)$ is undefined).

Standard final configurations

Definition. A **standard final configuration** for result y is a configuration that consists of one block of y strokes, such that the head scans the leftmost stroke and is in a halting configuration.

Definition of Turing computability

Definition. A function $f : N^k \to N$ is **computed by a Turing machine** M if

- whenever $f(x_1, \ldots, x_k) = y$, then M takes the standard initial configuration for $x_1, \ldots, x_k$ to a standard final configuration with y strokes on the tape;
- whenever $f(y_1, \ldots, x_k)$ is undefined, M never halts, or halts with a non-standard final conf.

A function $f : N^k \longrightarrow N$ is called **Turing-computable** if there is a Turing machine that computes it.

Exercise

Design a TM that will do the following. Given a tape containing a block of 1's and otherwise blank, if the machine is started the leftmost 1 on the tape, it will eventually halt scanning the right-most stroke on the tape, having neither printed nor erased anything.

Exercise

Design a TM that will do the following. Given a tape containing a block of 1's, followed by a 0, followed by another block of 1's, and otherwise blank, if the machine is started the leftmost 1 on the tape, it will eventually halt scanning the right-most stroke on the tape, having neither printed nor erased anything.

Exercise

Design a TM that, started scanning the leftmost 1 of an unbroken block of 1's on an otherwise blank tape, adds another 1 to the block and halts, scanning the leftmost 1.

Exercise

Design a TM that, started scanning the leftmost 1 of an unbroken block of 1's on an otherwise blank tape, eventually halts, scanning a square on an otherwise blank tape, where the square contains a blank or a 1 depending on whether there were an even or an odd number of strokes in the original block.

Hint: recall the parity-checker DFA.

Exercise

Design a TM that will do the following. Given a tape containing a block of n strokes, followed by a blank, followed by a block of m strokes, and otherwise blank, if the machine is started scanning the leftmost 1 on the tape, it will eventually halt leaving a block of $m - n$ strokes if $m \geq n$ or $n - m$ strokes if $n \geq m$, scanning the leftmost stroke on the tape if any stroke is left.

Generalized TM's

One could allow

- more than two tape symbols,
- replace the tape by a rectangular grid,
- use several heads, several tapes, etc. . .

Turing's thesis implies that no generalization will enlarge the class of functions computable. This bold claim turns out to be true for the cases above.