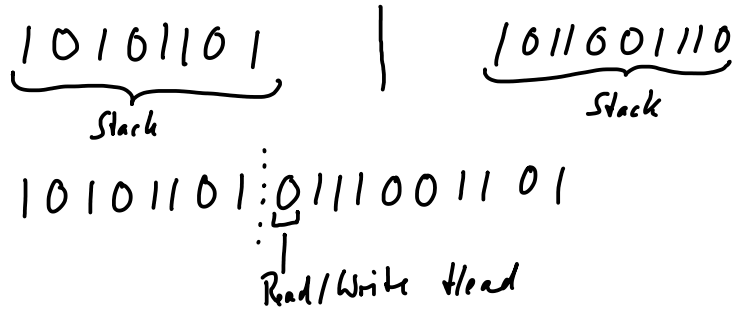
Termination of Erlang Code cannot be decided for the following reasons. We argument, that the corresponding aspect of Erlang can be used to model a Turing Machine.

- Erlang provides data structures, which can encode a Turing Tape. A list can be of arbitrary size and can represent the tape, at least the used finite part of the tape. The same holds for other data structures, like tuples. They can also be used to realize list stuctures and hence the tape.

- Erlang provides numbers. Allowing two counters allows the encoding of a Turing machine.



- Erlang provides an arbitrary number of processes. Each cell on the Turing Tape can be represented by a seperate process, which holds the value and its two neighbors and allows the modification of the cell.

- Erlang provides two (we will use three) processes in combination with recursion. Each process holds a stack, which is realized by the runtime stack. And the tape can again be seen as two stacks.

- Erlang provides a mailbox of arbitrary sice. This can also be used to encode the Turing Tape.

$$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{r, l, n\}$$

State, Tape alphabet

Encode $\delta(q, a) = (q', b, r)$ by

```
delta(SL,SR,q) ->
    A = pop(SR),
    case A of
        a -> push(SL,b),
             delta(SL,SR,q');
        b -> ...
        ...
    end.
```

Encode $\delta(q, a) = (q', b, l)$ by

```
delta(SL,SR,q) ->
    A = pop(SR),
    case A of
        a -> push(SR,b),
             X = pop(SL),
             push(SR,X),
             delta(SL,SR,q');
        b -> ...
        ...
    end.
```

Encode $\delta(q, a) = (q', b, n)$ by

```
delta(SL,SR,q) ->
    A = pop(SR),
    case A of
        a -> push(SR,b),
             delta(SL,SR,q');
        b -> ...
        ...
```

This code is executed as a third process here. It can be combined with one of the two stack processes, but this is just technical.
In the initialization of the coordinator process, the two stacks are spawned and the initial input is written to the right stack.

There are two possible cases, in which the TM and hence our program should terminate:

- if $q \in F$ then, the the function delta terminates and return accept, independed of possible transitions.
- if no transition is for q and the actual symbol and $q \notin F$, the fucntion delta terminates and returns the value reject.

At the end, the starter function can furthermore print the symbols on the stack SR up to the first blank as the output of the TM.