Beyond regular languages

A language L over an alphabet Σ is the set of all accepted strings.

- often infinite, e.g. language defined by regular expression $(ab)*$

Recall: "Regular languages", i.e. languages generated by regular expressions, are the precisely the languages that can be recognized by FSA.

Claim: There are languages that cannot be recognized by FSA

Proof: ① Suppose L can be recognized with an FSA with m states, and contains a string w of length $|w| > m$.

Then at least one state must be visited twice ("pigeon hole principle")

② Let y be the substring corresponding to the cycle from this state to itself, and write

$$w = xyz$$

Then (i) $|xy| \leq m$

(ii) $xy^n z$ must also be in L (that's how an FSA works!)

③ Now take $L = \{0^p 1^p : p \geq 1\}$

Suppose L is recognized by an FSA with m states.

Take $p > m$.

$\Rightarrow$ there is $y = 0^n$ with $1 \leq n \leq m$ which fits ②

$\Rightarrow$ $0^{p+n} 1^p \in L$ ↯

We conclude that L cannot be recognized by an FSA.

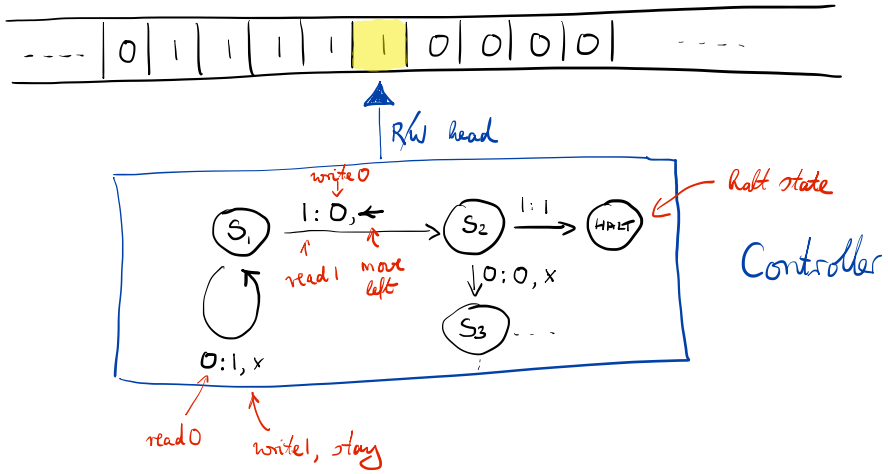
Chomsky hierarchy of languages:

Language: Regular → Context free → Context sensitive → recursively enumerable

Parser: FSA → pushdown automaton (FSA + infinite stack) → linearly bounded Turing machine → Turing machine (FSA + doubly infinite tape)

What is a Turing machine?

- Model computer: simple, but not efficient
- Doubly infinite tape as "memory", finite set of symbols (e.g. 0 and 1)
- Finite-State Transducer as "CPU + Program"
- Transducer can read from the tape, write to the tape, and move the head



Church-Turing thesis: Every effectively computable integer algorithm can be performed on a Turing machine ("universality")

Fact: "Simple language" is equivalent to a Turing machine

"Simple language" has
- variables of non-negative integer type
- Two statements: incr(x), decr(x)
- while-loops
- macros

E.g.: Macro $X \leftarrow 0$:

```
while (X):
    decr (X)
```

$X \leftarrow n$:

```
X ← 0
incr (X)
   ⋮        } n times
incr (X)
```

$Y \leftarrow X$:

```
Y ← 0
while (X):
    incr (Y)
    decr (X)
```

$Y \leftarrow Y + X$:

```
while (X)
    incr (Y)
    decr (X)
```

.....

Purpose:
- Break up problems into very simple components to prove theoretical results.
- Even though Turing machines can be built (with finite memory), they are not efficient architectures

The halting problem

Q: Is it possible to write a program that determines whether another program will halt after a finite number of steps?

Theorem (Turing, 1936): The halting problem is not solvable.

Proof (informal sketch):

① Every program can be converted into an integer number ("Gödel number"), thus used as input for another program.

(2) Now suppose there exists a program T that reads an arbitrary program P and outputs a variable $X = 1$ if P terminates,
$X = 0$ " " does not terminate.

(3) Write a program S(P) as follows:

```
call T(P)
while (X):
    {}
```

Thus: If P terminates, S does not terminate
P does not terminate, S terminates

(4) Now call S(S) ⟶ Contradiction.