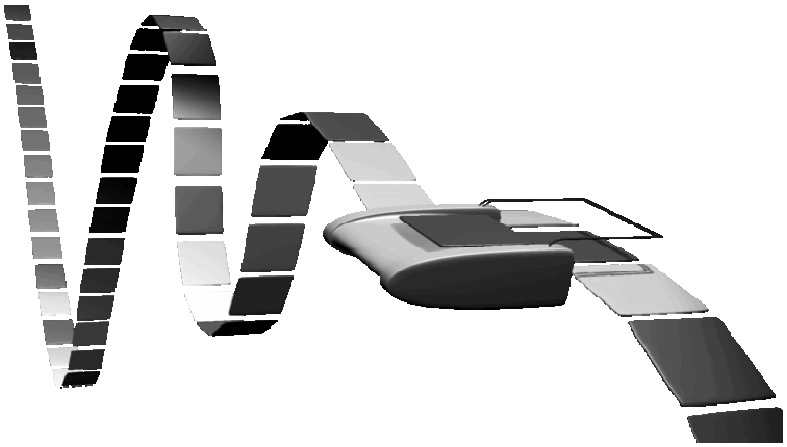
Artificial Intelligence

Lab 2

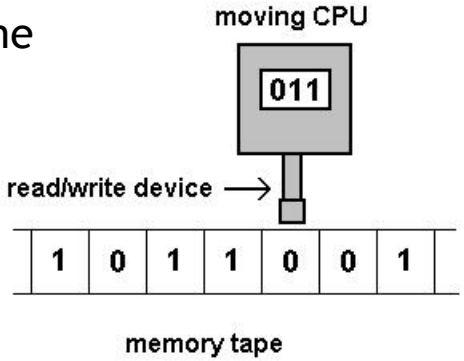
Marco Piastra

Turing Machine (A. Turing, 1937)



- An abstract model of effective computation
  - A **tape**, made up of individual **cells**
  - Each cell contains a **symbol**, from a finite **alphabet**
  - A **read/write head**, which can move in each direction - one cell at time
  - A **state register** that keeps the current **state**, from a finite set
  - A **transition table**, i.e. a set of *entries* like this:

    { <*current state*, *symbol read*> → <*next state, symbol written*, *move*> }



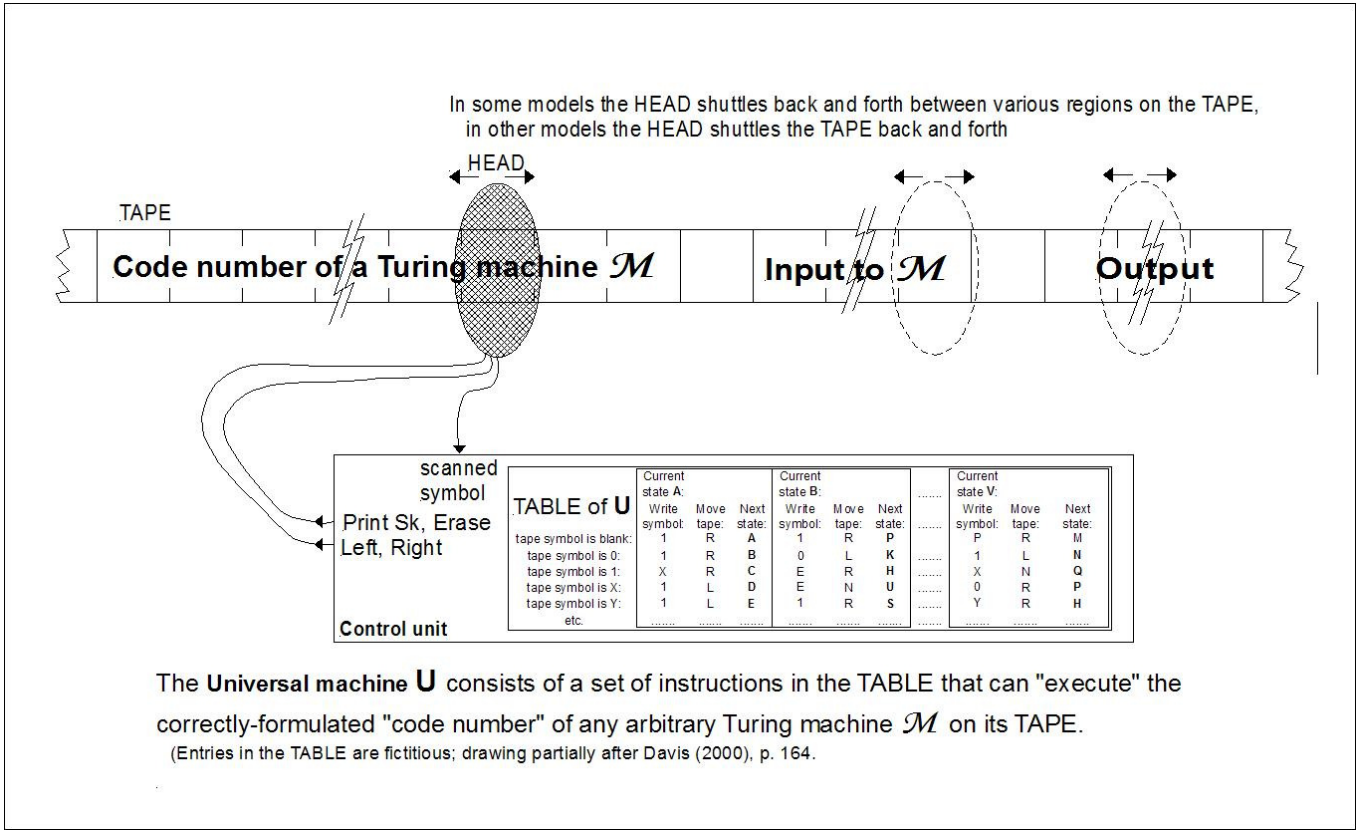
The **transition table** describes a *finite state machine*

- Each *transition* is governed by the input symbol, the current state and the corresponding entry in the transition table
- The next state is written into the state register
- The output is written to the cell
- Then the head moves (i.e. *left*, *right*, *none*)

Universal Turing Machine (A. Turing, 1937)

- As with the `basic' model, but the transition table is part of the input
  The transition table is loaded from the tape, at the beginning of computation

In its initial state, the machine contains only its `firmware' which is just sufficient for loading the table



Church-Turing Thesis

Caution: there is no such a thesis in the original writings of either author. Its formulation can be extrapolated from both. Hence the attribution (made by others)

- A possible formulation (from Wikipedia):

  "*Every 'function which would naturally be regarded as computable'* *can be computed by a Turing machine.*"

The vagueness in the above sentence gives raise to different interpretations. One of these (though not entirely equivalent) is (from Wikipedia):

"*Every 'function that could be physically computed'* *can be computed by a Turing machine.*"

Searle: "... At present, obviously, the metaphor is the digital computer."

assert

(file turingmachine.jess)

- Asserting a fact in working memory

Example:

```
(defrule event
    ?tm <- (tm (current-state ?cs&~:(eq ?cs halt))
               (current-square ?is))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns)
           (head-move ?mv))
    =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os " move " ?mv crlf)
    (modify ?tm (current-square ?os)
                (current-state ?ns))
    (assert (move ?tm ?mv))
)
```

The fact is inserted in memory

retract

(file turingmachine.jess)

- Retracting (i.e. removing) a fact from working memory

Example:

```
(defrule move-right
  (declare (salience 1))
    ?action <- (move ?tm right)
    ?tm <- (tm  (left-part $?rest-left)
                (current-square ?sym)
                (right-part ?sym-right $?rest-right))
    =>
    (modify ?tm (left-part ?sym ?rest-left)
                (current-square ?sym-right)
                (right-part ?rest-right))
    (retract ?action)
)
```

This variable binds to the fact to be retracted

The fact is retracted from memory

Constraints on variables

(file turingmachine.jess)

- The binding between variables and value can be *constrained*

Example:

Variable Constraint

```
(defrule event
    ?tm <- (tm (current-state ?cs&~:(eq ?cs halt))
               (current-square ?is))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns)
           (head-move ?mv))
    =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os " move " ?mv crlf)
    (modify ?tm (current-square ?os)
                (current-state ?ns))
    (assert (move ?tm ?mv))
)
```

General syntax:

`&:`<*constraint*> *positive form*

`&~:`<*constraint*> *negative form*

salience

(file turingmachine.jess)

- Altering salience (i.e. *priority*) of a rule

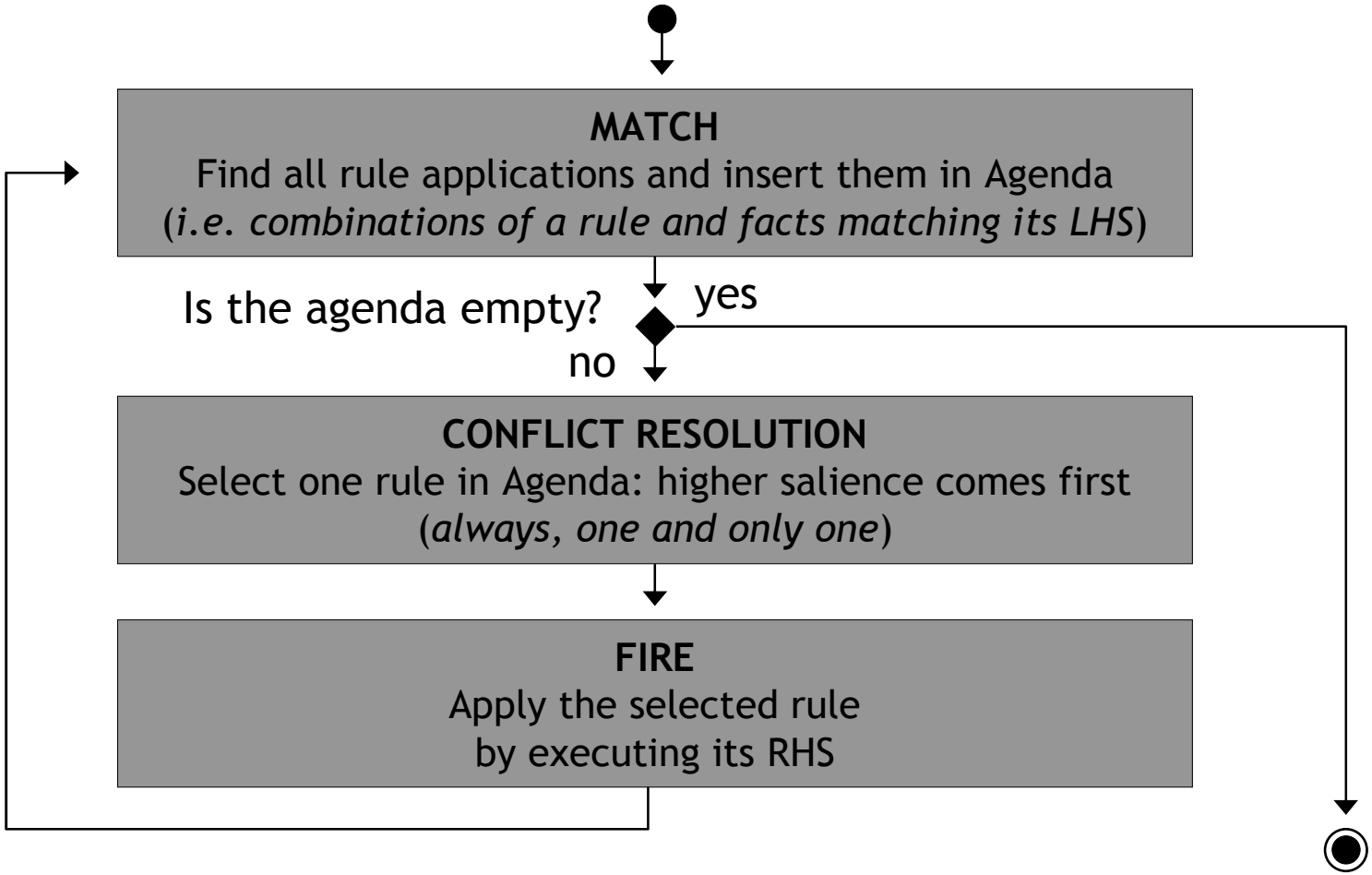
Example:

It is not a a clause, is a declaration
It is optional
By default the value of salience is 0

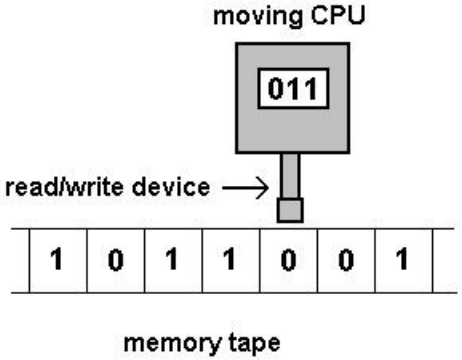
```
(defrule move-right
  (declare (salience 1))
    ?action <- (move ?tm right)
    ?tm <- (tm  (left-part $?rest-left)
                (current-square ?sym)
                (right-part ?sym-right $?rest-right))
    =>
    (modify ?tm (left-part ?sym ?rest-left)
                (current-square ?sym-right)
                (right-part ?rest-right))
    (retract ?action)
)
```

Agenda (i.e. How Jess works - in a second approximation)

- The Agenda contains all applicable combinations rule-facts



Turing Machine

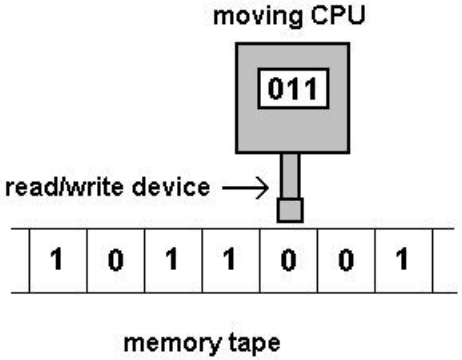


(file turingmachine.jess)

- A finite-state automaton + a read/write moving head + an infinite tape

```
(deftemplate tm
    (slot current-state)
    (multislot left-part)
    (slot current-square)
    (multislot right-part)
)
```

Turing Machine



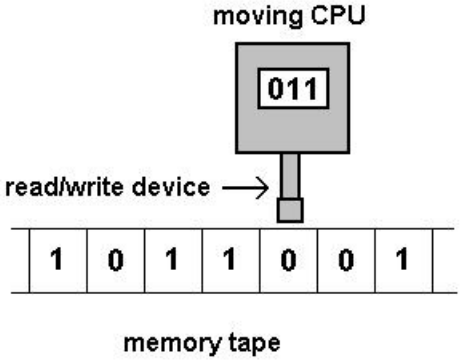
(file turingmachine.jess)

- The Turing Machine and the entries in the transition table

```
(deftemplate tm
    (slot current-state)
    (multislot left-part)
    (slot current-square)
    (multislot right-part)
)

(deftemplate event
    (slot current-state)
    (slot input-symbol)
    (slot output-symbol)
    (slot new-state)
    (slot head-move)
)
```

Turing Machine



(file turingmachine.jess)

- Questions (for you):
  a) How do the rule in this program work?
  b) How does the program simulate a infinite tape (in both directions)?
  c) Why the rules have different *salience*?
  d) What would it happen if we made all rules have *salience* 0 ?

  Try modify the file then, before **`(run)`**,
  issue the command **`(set-strategy breadth)`**

  (Extra bonus: look into the Jess documentation to discover what the last command mean)

(use the example paritychecker-tm.jess)