

Home task:

- Install and play with RACKET ([drracket](#) — RACKET IDE; [racket language documentation](#));
- Implement FlowChart interpreter on Racket,  $int_{Racket}^{FlowChart}$ , with respect to the abstract syntax from Fig. 1; FlowChart program example can be found on Fig. 3a;
- Implement Post's variant Turing Machine interpreter on FlowChart,  $int_{FlowChart}^{TM}$ , with respect to the abstract syntax from Fig. 2; TM program example can be found on Fig. 3b.

$\mathcal{X}$	$::= \{x, y, z, \dots\}$	Variable
$\mathcal{P}$	$::= \text{read } \mathcal{X}^*; \mathcal{B}^+$	Program
$\mathcal{B}$	$::= \mathcal{L} : \mathcal{A}^* \mathcal{I}$	Basic block
$\mathcal{A}$	$::= \mathcal{X} := \mathcal{E}$	Assignment
$\mathcal{I}$	$::= \text{goto } \mathcal{L}$   $\text{if } \mathcal{E} \text{ goto } \mathcal{L} \text{ goto } \mathcal{L}$   $\text{return } \mathcal{E}$	Jump
$\mathcal{E}$	$::= \mathcal{C} \mid \mathcal{X} \mid \mathcal{O} \mathcal{E}^*$	Expression
$\mathcal{C}$	$::= \text{'Value' /* a quoted value */}$	Constant
$\mathcal{O}$	$::= \text{car} \mid \text{cdr} \mid \text{cons} \mid \dots \text{ /* (any other) */}$	Operator
$\mathcal{L}$	$::= \text{/* whatever */}$	Label

Fig. 1: Abstract FlowChart syntax

$\mathcal{A}$	$::= \{0, 1, \sqcup\}$	Alphabet
$\mathcal{P}$	$::= \mathcal{I}^*$	Program
$\mathcal{I}$	$::= \text{left} \mid \text{right} \mid \text{write } \mathcal{A} \mid \text{goto } i \mid \text{if } \mathcal{A} \text{ goto } i$	Instruction
		where $i$ is instruction index

Fig. 2: Abstract Turing Machine syntax

<code>read name namelist valuelist;</code>	
<code>search: if (equal? name (car namelist)) goto found goto cont;</code>	0: <code>if 0 goto 3</code>
<code>cont : valuelist := cdr valuelist;</code>	1: <code>right</code>
<code>namelist := cdr namelist ;</code>	2: <code>goto 0</code>
<code>goto search;</code>	3: <code>write 1</code>
<code>found : return car valuelist;</code>	
(a) FlowChart program example (find name)	(b) Turing Machine program example

Fig. 3: Program examples (in some concrete syntax)

## 1 How exactly should it look like on Racket

First, you have to define a FlowChart interpreter on Racket. I.e. something like:

```
(define int (lambda (program data) (<interpreter code>)))
```

where *program* stands for FlowChart program and *data* — for a list of its input data. Note that both *program* and *data* are data, i.e. quoted values, from the Racket's point of view. In order to avoid parsing we write programs on FlowChart as their abstract syntax trees. E.g. program *find\_name* from Fig. 3a can be defined as follows

```
(define find_name
  '((read name namelist valuelist)
    (search (if (equal? name (car namelist)) found cont))
    (cont (:= valuelist (cdr valuelist))
          (:= namelist (cdr namelist))
          (goto search))
    (found (return (car valuelist)))
  ))
```

and its invocation

```
(int find_name '(y (x y z) (1 2 3))
;; the expected output is '2
```

Then, you have to define a Turing-machine interpreter on FlowChart

```
(define tm-int
  '((read Q Right)
    (init (:= Qtail Q) (:= Left '()) (goto loop))
    (loop (if (...) label-1 label-2))
    ...
  )
)
```

An example of TM-program definition:

```
(define tm-example '((0 if 0 goto 3) (1 right) (2 goto 0) (3 write 1)))
```

and its invocation

```
(int tm-int `(,tm-example (1 1 1 0 1 0 1)))
;; the expected output --- '(1 1 0 1)
```