Programming Paradigms Assignment 1

Author: Martin Raunkjær Anderse
AAU study number: 20136030
AAU email adress: marand13@student.aau.dk

Status of program and non-completed tasks

The program is partially complete. All constructors with (almost all) validation, predicates and selectors are done. As are the *scale*, *transpose* and *re-instrument* functions and the transformation function that transforms *music-elements* into *note-abs-time-with-duration*.

Requirement 6 (*monophonic?*) and 7 (*degree-of-polyphony?*) has not been completed due to lack of time. Although I would argue that requirement 6 is trivially solved by using requirement 7, as in the statement below, so with a little good faith, I am only one requirement short :)

```
(define (monophonic? music-element)
  (= (degree-of-polyphony? music-element) 1))
```

Scheme system details

The program is done in Racket, with no additional functions imported besides the music library specified for the task.

External program parts and inspiration

No external libraries has been used. As for inspiration, the program draws heavily from the *racket guide*[1]. Especially the info and examples on *structs* has been useful, as well as the info on *pattern matching*.

Non-completed tasks

As mentioned, the *monophonic?* and *degree-of-polyphony?* functions are incomplete. Furthermore, a number of small shortcomings are present various places in the program, listed below:

- line 47: unfinished validation of the list input to parallel and sequential music elements. It should be checked if the input is a list and if each item is a music element.

[1]https://docs.racket-lang.org/guide/index.html

- line 116: In the *get music duration* function. If there is a long pause as the last element a sequence, this is not taken into consideration when calculating the duration of an element, so some wrong results could show here (it has no effect on the generated list of absolute time nodes fortunately).
-

Relevant remarks

Although the program works well, there are some obvious areas of improvement, that could be alleviated if more time were available.

tail-recursion
Non of the functions use tail recursion. This could be implemented for sparing some stack memory.

more higher-order functions
Higher order functions are used a few places in the programs, in the selector and predicate function sections, but there are a lot of places where it could be implemented with great gain, such as in the *scale*, *transpose* and *re-instrument* functions, which look almost exactly the same. Unfortunately there was not time for it. I look forward to discussing it at the exam :)