Score Description Library

Reference Manual

Pedro J. Morales.
English translation by Luis Rodríguez and Pedro J. Morales.

July 25, 2007

Contents

1 Introduction

Usually, Computer Music compositions are formally specified by means of programming language algorithms. Owing to the fact that *Nyquist* is actually an extension of Lisp, this language can be used for both sound synthesis and algorithmic composition.

In addition, Nyquist can be also employed for rendering music that is described as a note sequence, for instance, a score in traditional music notation. However, in this case, every note parameter has to be specified which makes difficult to compose music confortably.

The *Score Description Library* (SDL) is aimed at facilitating the translation from traditional score notation to *Nyquist* source code and also allowing a fine control over the performance and synthesis parameters.

2 Description

2.1 SDL score example

A SDL score is a quoted list in which the notes along with the pitch, duration attributes and other arguments related to timing and synthesis parameters, are specified.

```
1  ; Begin of BWV 1069
2
3  (setf *my-sdl-score*
4    '((TF 1)
5    (INIT-INSTR "i1" sinte) (INSTRUMENT "i1") (PWL-POINT :mm 100) (ATTR :idur 0.1)
6    2 (:c4 1)  :d4 (:e4 2) :c4 :g4 :e4 :a4 (:g4 1) :f4 (:g4 4)
7    (:a4 1) :c5 :f4 :a4 :g4 :f4 :e4 :g4 :f4 :a4 :d4 :f4 :e4 :d4 :c4 :d4 :e4 :f4 (:g4 2)
8    :a3 :c4 :d4 :f4
9    :g3 :b3 :c4 :e4 (:f3 1) :e4 :d4 :c4 :g3 :d4 :c4 :b3 (ATTR :idur 1) (LABEL :t1)
10   (:c4 4)))
```

- `TF` stands for *Time Factor*. All the durations will be multiplied by this factor (default value is 1).
- `INIT-INSTR` declares an instrument to be used in synthesis. The first argument *"i1"* is the instrument name in the SDL score. The second one is the *Nyquist* function name which defines the instrument. This function must be defined independently from the score by means of an expression `(defun sinte ...)`
- `INSTRUMENT` causes that all the notes from now on are synthesized by the instrument *"i1"* until a new instrument is specified.
- `PWL-POINT` defines a piece wise linear function in order to set the time-variable parameters. For instance, `:mm` takes a value of `100` when the instruction is called.
- `ATTR` sets the value of a constant parameter. The parameter value is not changed until a new `ATTR` instruction is reached. For example, the `:idur` parameter takes a value of *0.1* in every note until a new value is specified by a new `ATTR`.
- `2` defines a 2 beat rest. Two different time types can be considered: a *score time* measured in beat and a *physical time* measured in second. A quarter has a duration of 4 beats. The physical time is computed according to the score time, *tempo* and Time Factor `TF`.
- `(:c4 1)` represents a note given by a C4 pitch and a 1 beat duration.(i.e, a sixteenth). Only pitch and duration are specified. Pitch can be specified by using an alternative syntax.

Duration can be any *Lisp* expression. The rest of attributes the needed for synthesis are provided by `ATTR` y `PWL-POINT` instructions.

- `:d4` is a D4 pitch note and inherited duration of 1 beat. Rests do not alter the default duration. Changes in duration are explicitly set by a note attribute.

- `LABEL` sets a time label aimed at synchronizing score sections. `LABEL` is related to the score time. Coincidence among `LABEL` references depends on the score *tempi* map. Therefore, the user is responsible for controlling this issue.

2.2 SDL score processing

A *SDL* score is processed by using the `sdl->score` function. For example:

```
(load ‘‘sdl’’)
(sdl->score *my-sdl-score*)

=> ((0.3 0.15 (SINTE :PITCH 60 :MM 100 :IDUR 0.1))
    (0.45 0.15 (SINTE :PITCH 62 :MM 100 :IDUR 0.1))
    (0.6 0.3 (SINTE :PITCH 64 :MM 100 :IDUR 0.1))
    (0.9 0.3 (SINTE :PITCH 60 :MM 100 :IDUR 0.1))
    (1.2 0.3 (SINTE :PITCH 67 :MM 100 :IDUR 0.1))
    ...
)
```

As can be noticed, an *Xmusic* score is obtained as a result.

Every *Xmusic* score event has three elements. The first one indicates the note starting time. The second one is the stretching factor and the third one is a call to the synthesis function. For instance, the first event:

```
(0.3 0.15 (SINTE :PITCH 60 :MM 100 :IDUR 0.1))
```

starts at `0.3` seconds; the stretching factor is `0.15` and the synthesis function call is `(sinte :pitch 60 :idur 0.1)`.

`:mm` argument is not a synthesis parameter but a control for *tempo*. Thus, `sinte` implementation must not include any argument with this name.

The strecthing factor multiplies the note duration by the stretching value. For instance, for a sytnthesis function call returning a 1 second note and a stretching factor equals to `0.15` the overall note duration would be `0.15`.

One of the most remarkable features of Nyquist is the *Behavioral Abstraction* which constitutes a framework for addressing context-dependent transformations that includes stretching.

Every synthesis function has a default behavior that properly works in most of the situations. Nevertheless, this default behavior may not be appropriated in some cases. Therefore, for instance, the amplitude envelope attack of a sound may be changed by different strecthing factor values which can be unsuitable for a harpsichord-like sound.

To fix this problem, the synthesis function can be defined so that the attack time is constant and independent form the stretching factor [1].

Alternatively, all the stretching factors can be set to `1` and a specific parameter can be used to indicate the specific duration of any event. `:idur` parameter is used for this purpose. This way, sound duration can be higher or lower allowing for several articulation modes from *legato* to

[1] Extensive knowledge of Behavioral Abstraction is required to perform this task

stacatto. The time-control parameters can not be called `:dur` since this name is a *Xmusic* reserved keyword.

`sdl:normalize-score-duration` is used to set the stretching factors to 1 as is shown in the following example:

```
1
2  (sdl:normalize-score-duration (sdl->score *my-sdl-score*))
3
4  => ((0.3 1 (SINTE :PITCH 60 :MM 100 :IDUR 0.1))
5      (0.45 1 (SINTE :PITCH 62 :MM 100 :IDUR 0.1))
6      (0.6 1 (SINTE :PITCH 64 :MM 100 :IDUR 0.1))
7      (0.9 1 (SINTE :PITCH 60 :MM 100 :IDUR 0.1))
8      (1.2 1 (SINTE :PITCH 67 :MM 100 :IDUR 0.1))
9      ...
10    )
```

2.3 SDL score rendering

Once a *SDL* score has been translated to *Xmusic* format, a *Nyquist behavior* can be obtained by using the `timed-seq` function [2]

```
1  (play (timed-seq (sdl:normalize-score-duration (sdl->score *my-sdl-score*))))
```

2.4 Synchronization

When dealing with complex scores it is suitable to split them into smaller parts. Temporal references must be established in order to synchronize all the parts involved. These temporal reference can be set by using a `LABEL` instruction. `sdl->timelabels` function is used to obtain the temporal references list. This data is registered as a property list of a symbol created on-the-fly. For example:

```
1  (setf *my-time-labels* (sdl->timelabels *my-sdl-score*))
2  (print (symbol-plist *my-time-labels*))
3
4  => (:T1 64)
```

`AT-LABEL` instruction allow us to match the temporal label which has been previously specified by `LABEL` and another time value from a different score. The temporal references list must be sent, as an argument, to the `sdl->score` function as can be seen in the following example:

```
1  (setf *my-time-labels* (sdl->timelabels *my-sdl-score*))
2
3  (setf *voz-2*
4    '((TF 1)
5    (INIT-INSTR "i1" sinte) (INSTRUMENT "i1") (PWL-POINT :mm 100) (ATTR :idur 0.1)
6    (AT-LABEL :t1)
7    2 (:g4 1) :a4 (:b4 2) :g4))
8
9  (sdl->score *voz-2* *my-time-labels*)
10
11 => ((9.9 0.15 (SINTE :PITCH 67 :MM 100 :IDUR 0.1))
12     (10.05 0.15 (SINTE :PITCH 69 :MM 100 :IDUR 0.1))
13     (10.2 0.3 (SINTE :PITCH 71 :MM 100 :IDUR 0.1))
14     (10.5 0.3 (SINTE :PITCH 67 :MM 100 :IDUR 0.1)))
```

[2] It is also possible to use the `score-play` function in order to directly render the score

Warnings

Users should notice that time labels are actually score times (measured in beats), whereas in *Xmusic* scores the time is specified in seconds, which in turn are mapped from the score time, the `TF` parameter and the *tempi* map. For that reason, time labels always coincide in score time, but they have to use the same `TF` parameter and *tempi* map to achieve a coincidence in physical time.

3 Reference guide

3.1 Overall SDL score structure

A *SDL* score is a quoted list containing notes, rests or other elements related to synthesis and performance. The following elements have to be present in any *SDL* score:

- One instrument initialization `INIT-INSTR`.
- One instrument assignment `INSTRUMENT`.
- One tempo specification `:mm`, by means of either `PWL-POINT` or `ATTR`.
- One note, at least.

Examples:

```
(setf *minimal-sco-1* '((INIT-INSTR "i1" xx) (INSTRUMENT "i1") (PWL-POINT :mm 100)
                        (:c4 2)))
(setf *minimal-sco-2* '((INIT-INSTR "i2" yy) (INSTRUMENT "i2") (ATTR :mm 100) :c4))
```

Function `sdl->score` returns an error if the score does not have this basic data.

3.2 Pitch specification

Pitches can be given by using:

- A number. The standard MIDI pitch scale has been adopted here (60 = C4). No tempered pitches are allowed when using floating point numbers (FLONUM).
- The standard *Nyquist* symbols. Example: `cs4` = C4 sharp = 61.
- The standard *lambda Music* symbols[3] Pitches are represented by symbols starting with a colon. Sharps and flats are notated by `#` and `b`. Examples: `:c4 :C4 :c#4 :C#4 :cb4 :Cb4`
- Quoted symbols. Examples: `'c4 'cs4 'c#4 'cb4`
- Strings. Examples: ``c4'' ``cs4'' ``c#4'' ``cb4''

Example:

```
(setf *pitches* (list 60 60.0 c4 'c4 C4 'C4 cs4 df4 :c4 :c#4 :cb4 :df4
                      "c4" "cs4" "cb4" "c#4"))
(mapcar #'sdl:pitch-lex *pitches*)

=> (60 60 60 60 60 60 61 61 60 61 59 62 60 61 59 61)
```

[3] A library for music composition developed by Pedro Morales in *Common Lisp*

3.3 Durations

Durations are measured in beats. A quarter worth 4 beats; a half, 8 beats, and so on. Fractional durations are allowed.

Durations can be given by:

- A number.
- A *Lisp* expression. Example: `(/ 4.0 3.0)` represents a third of a quarter, that is, an eighth triplet.

Physical time and score time

Physical time (measured in seconds) is computed after score time (measured in beats), the *time factor*, `TF` and the *tempi* map (given by the parameter `:mm`).

Global Time Factor

It is given by the score instruction `TF`. Example: `(TF 2.0)` multiplies by 2.0 all the durations in the score, whereas `(TF 0.5)` does it by 0.5.

This instruction must appear only once in the score. Its default value is 1.0.

Tempi

Tempi values are given by the parameter `:mm` whose value is set by means of the instructions `ATTR` or `PWL-POINT`.

3.4 Notes

There are two alternatives for specifying notes:

- By using a two-element list (pitch and duration). Example: `(:c4 4)` is a quarter `C4`.
- By the pitch only. Duration is taken from the last note.

The default duration can be changed by means of the `DUR` instruction.

3.5 Rests

Can be noted by:

- A number representing the amount of beats.
- `DELTA` or `PAU` instructions[4], indicating the duration by a number or by a *Lisp* expression.

Example:

```
1 (setf *sco3*
2   '((TF 1.0)                    ; Overall Time Factor
3     (INIT-INSTR ``i1'' xx)      ; Preamble.  instr. init.
4     (INSTRUMENT ``i1'')         ; instr. assign.
5     (ATTR :mm 60)               ; metronome MM=60. 1 quarter = 1 sec.
6     (:c4 4)                     ; c4 quarter
7     4                           ; quarter rest
8     :d4                         ; d4. duration = quarter (inherited)
```

[4]Both instructions are equivalent.

```
    (PAU 8)                     ; half rest
    (:e4 (/ 4.0 3.0))           ; e4. eighth triplet
    :e4                         ;
    :e4                         ;
    (DELTA (* 2 4))             ; half rest
    (:f4 (* 2 1.5))             ; f4 duration = 3 = dotted eighth
    (DUR 4)                     ; current duration = 4 (quarter)
    :g4                         ; g4 duration = quarter
    (DUR (/ 4.0 3.0))           ; current duration = triplet eighth
    :a4 :a4 :a4                 ; a4 triplets
))
```

3.6 Chords

Polyphonic music can be achieved in *SDL* by gathering several scores. Nevertheless, *SDL* predefined instructions can be more appropriated in simple cases.

`CH` produces simultaneous notes, having the current duration. Example:

```
(setf *sco4*
  '((TF 1.0)                    ; Overall Time Factor
    (INIT-INSTR ``i1'' xx)      ; preamble. init. instr.
    (INSTRUMENT ``i1'')         ; assign. instr.
    (ATTR :mm 60)               ; metronome MM=60. 1 quarter = 1 sec.
    (:c4 4)                     ; c4 quarter
    4                           ; quarter rest
    (DUR 8)                     ; current duration = 8 = half
    (CH :c4 :e4 :g4)            ; three-note chord. duration = half
                                ;
```

`CH1` produces one note without increasing the time. Time increments can be indicated by the `DELTA` instruction. This way independent duration chords can be built for each note. Example:

```
(setf *sco5*
  '((TF 1.0)                    ; Overall Time Factor
    (INIT-INSTR ``i1'' xx)      ; preamble. Init. instr.
    (INSTRUMENT ``i1'')         ; assign. instr.
    (ATTR :mm 60)               ; metronome MM=60. 1 quarter = 1 sec.
    (:c4 4)                     ; c4 quarter
    4                           ; quarter rest
    (CH1 :c4 4)                 ; CHORD BEGINNING. c4 quarter
    (CH1 :e4 8)                 ; e4 half. SIMULTANEOUS BEGINNING  c4
    (:g4 4)                     ; g4 quarter. SIMULTANEOUS BEGINNING. c4, e4
    (:c5 4)                     ; c5 quarter. Starts when g4 ends
                                ;           e4 keeps playing because of the half
                                ;           duration
    (CH1 :c4 8)                 ; Arpeggiated chord BEGINNING
    (DELTA 0.2)                 ; time increases 0.2 beats
    (CH1 :e4 (- 8 0.2))         ; e4 starts 0.2 beats after c4
                                ;    ends at the same time
    (DELTA 0.2)                 ; time increases 0.2 beats
    (CH1 :g4 (- 8 0.2 0.2))     ; g4. another arpeggio note
    (:c5 (- 8 0.2 0.2 0.2))     ; c5 final arpeggio note
    (:g3 4)                     ; g3 quarter after arpeggio
))                              ;   (total arpeggio duration = 8)
```

3.7 Instruments

In every score notes must be tied to a synthesis instrument. `INIT-INSTR` is the instruction used for mapping the score instrument name to a *Nyquist* function that implements the instrument itself.

The `INSTRUMENT` instruction assigns the instrument name to the notes. The current instrument can be set by a new `INSTRUMENT` instruction. Example:

```
(setf *sco6*
  '((TF 1.0)
    (INIT-INSTR ``i1'' flute)    ; assign i1 instrument score name
                                 ; to the flute function
    (INSTRUMENT ``i1'')          ; current instr. = i1
    (ATTR :mm 60)
    (:c4 4)                      ; rendered by flute
    4
    (INIT-INSTR ``i2'' clarinet) ; assign i2 instrument score name
                                 ; to the clarinet function
    :d4                          ; rendered by flute
    (INSTRUMENT ``i2)            ; current instr. = i2
    :e4                          ; rendered by clarinet
))
```

3.8 Attributes

In *SDL* scores, parameters for synthesis functions in *Nyquist* are defined by `ATTR` and `PWL-POINT` instructions.

- `ATTR` has two arguments. The first one is the parameter name (starting with a colon). The second one is used just to set the parameter value. The current parameter value can be changed by another `ATTR` instruction.
- `PWL-POINT` behaves like the `ATTR` instruction. The only difference is that a linear interpolation is performed between two consecutive `PWL-POINT` instructions.

It is not allowed to define the same attribute value by using both `ATTR` and `PWL-POINT` instructions.

```
(setf *sco7*
  '((TF 1.0)
    (INIT-INSTR ``i1'' flute)    ; maps i1 to flute
    (INSTRUMENT ``i1'')          ; current instr. i1 (flute)
    (:c4 4)                      ; mm = 100; rel = 4; decay = 3.2
    (ATTR :decay 3.2)            ;
    (:d4 4)                      ; mm = 100; rel = 4; decay = 5.1
    (ATTR :decay 5.1)
    (:e4 4)                      ; mm = 100; rel = 4; decay = 5.1
    (PWL-POINT :rel 4.0)
    (PWL-POINT :mm 100)
    (:f4 4)                      ; mm = 100; rel = 4; decay = 5.1
    (:g4 4)                      ; mm = 100; rel = 5; decay = 5.1
    (:a4 4)                      ; mm = 100; rel = 6; decay = 5.1
    (PWL-POINT :rel 7.0)
    (:b4 4)                      ; mm = 100; rel = 7; decay = 5.1
    (:c5 4)                      ; mm = 100; rel = 7; decay = 5.1
    (:d5 4)                      ; mm = 100; rel = 7; decay = 5.1
    (PWL-POINT :rel 7.0)
    (:e5 4)                      ; mm = 100; rel = 7; decay = 5.1
))
```

3.9 Tempi

`ATTR` can be used in case of sharp tempo changes, whereas `PWL-POINT` performs gradual tempo changes. Example:

```
(setf *sco-8*
  '((TF 1)
    (INIT-INSTR "i1" flute) (INSTRUMENT "i1")
    (:c4 4)
    (PWL-POINT :mm 60)
    :d4 :e4 :f4 :g4
    (PWL-POINT :mm 60)
    :e5 :d5 :c5 :b4 :c5
    (PWL-POINT :mm 30)))

(score-print (sdl->score *voz-8*))

=> ((0 1 (FLUTE :PITCH 60 :MM 60))
    (1 1 (FLUTE :PITCH 62 :MM 60))
    (2 1 (FLUTE :PITCH 64 :MM 60))
    (3 1 (FLUTE :PITCH 65 :MM 60))
    (4 1 (FLUTE :PITCH 67 :MM 60))
    (5 1 (FLUTE :PITCH 76 :MM 60))
    (6 1.11111 (FLUTE :PITCH 74 :MM 54)) ; ritardando
    (7.11111 1.25 (FLUTE :PITCH 72 :MM 48))
    (8.36111 1.42857 (FLUTE :PITCH 71 :MM 42))
    (9.78968 1.66667 (FLUTE :PITCH 72 :MM 36))
)
```

3.10 Macros

A basic macro implementation is available in *SDL* which allows for a specification of repetitive structures such as tremeloes, etc. The `MAC` instruction is used for this purpose. The first argument is the name of a *Lisp* function and the rest ones represents the arguments for this function. Example:

```
;  Nyquist function that repeats an event n times

(defun sdl-repeat (n quoted-event)
  (let (result)
    (dotimes (i n (apply #'append result))
      (push quoted-event result))))

; macro sdl-repeat called from a score
; the sequence :f4 :g4 is repeated 4 times

(setf *score*
   '((TF 1.0)
     (INIT-INSTR "i1" flute2)(INSTRUMENT "i1")(ATTR :mm 60)
     (:e4 4) (MAC sdl-repeat 4 (:f4 :g4))))

(sdl->score *score*))

=> ((0 1 (FLUTE2 :PITCH 64 :MM 60))
    (1 1 (FLUTE2 :PITCH 65 :MM 60)) ; f4     repite 4 veces f4 g4
    (2 1 (FLUTE2 :PITCH 67 :MM 60)) ; g4
    (3 1 (FLUTE2 :PITCH 65 :MM 60)) ; f4
    (4 1 (FLUTE2 :PITCH 67 :MM 60)) ; g4
```

```
    (5 1 (FLUTE2 :PITCH 65 :MM 60)) ; f4
    (6 1 (FLUTE2 :PITCH 67 :MM 60)) ; g4
    (7 1 (FLUTE2 :PITCH 65 :MM 60)) ; f4
    (8 1 (FLUTE2 :PITCH 67 :MM 60)) ; g4
)
```

Macros can be also used to control the parameter values through a *Lisp* function instead of specifying them note by note. Example:

```
; pitches controlled by logistic equation

(setf *logistica* 0.3)
(setf *k* 3.97)

(defun logistica ()
  (setf *logistica* (* *k* (- 1.0 *logistica*) *logistica*)))

(defun call-logistica (n dur pitch-min pitch-interval)
  (let (result)
     (dotimes (i n (reverse result))
       (push (list ( pitch-min (round (* (logistica) pitch-interval))) dur)
             result))))

(setf *score2*
   '((TF 1.0)
     (INIT-INSTR "i1" flute2)(INSTRUMENT "i1")(ATTR :mm 60)
     (:e4 4) (MAC call-logistica 5 4 30 15)))

(score-print (sdl->score *score2*))

=> ((0 1 (FLUTE2 :PITCH 64 :MM 60))
    (1 1 (FLUTE2 :PITCH 43 :MM 60))
    (2 1 (FLUTE2 :PITCH 38 :MM 60))
    (3 1 (FLUTE2 :PITCH 45 :MM 60))
    (4 1 (FLUTE2 :PITCH 31 :MM 60))
    (5 1 (FLUTE2 :PITCH 34 :MM 60))
)
```

3.11 SDL Functions

`FUN` instruction allows for defining score events through *Lisp* functions.

Warning: The inclusion of attribute `:mm` is mandatory if the event to be generated is a note. Example:

```
(load "xm")

(setf *my-durations* (make-heap (list 4 6 8)))

(defun heap-durations (inicio pitch)
  (let ((dur (next *my-durations*)))
    (list inicio  dur (list 'sinte-fun :pitch pitch :mm 60  :idur dur))))

(setf *score3*
   '((TF 1.0)
     (INIT-INSTR "i1" flute2)(INSTRUMENT "i1")(ATTR :mm 60)
     (:e4 4)
```

```
     (FUN #'heap-durations 1 :c4)
     (FUN #'heap-durations 2 :d4)
     (FUN #'heap-durations 3 :e4)
     (FUN #'heap-durations 4 :c4)
     (FUN #'heap-durations 7 :d4)))

(sdl->score *score3*)

=> ((0 1 (FLUTE2 :PITCH 64 :MM 60))
    (0.25 1.5 (SINTE-FUN :PITCH :C4 :MM 60 :IDUR 6))
    (0.5 1 (SINTE-FUN :PITCH :D4 :MM 60 :IDUR 4))
    (0.75 2 (SINTE-FUN :PITCH :E4 :MM 60 :IDUR 8))
    (1 1.5 (SINTE-FUN :PITCH :C4 :MM 60 :IDUR 6))
    (1.75 1 (SINTE-FUN :PITCH :D4 :MM 60 :IDUR 4))
)
```

4 Reference summary

4.1 SDL format for score instructions

`(TF args:  time-factor)`
Overall Time Factor. All the durations in the score are multiplied by this factor.

`(TIME-IN-SECONDS)`
No arguments needed. Durations are measured in seconds when tempo is set to 60.

`(DUR lisp-expr)`
Sets the value of the current duration. Any *Lisp* expression can be used as argument.

`(DELTA lisp-expr)`
Increases the time value.

`(PAU lisp-expr)`
The same as `DELTA`.

`(INIT-INSTR string function-name)`
Initializes an instrument. It maps the score name instrument `string` to the synthesis function `function-name`.

`(INSTRUMENT string)`
Sets the current instrument to `string`. The notes following this instruction are rendered by the instrument `string` until a new instrument is set.

`(ATTR :symbol lisp-expr)`
Sets `:symbol` value to `lisp-expr`. The first argument must start with a colon. The value for this attribute is kept until a new `ATTR` instruction is reached.

`(PWL-POINT :symbol lisp-expr)`
Behaves like `ATTR`. The only difference is that a linear interpolation is performed between two consecutive `PWL-POINT` instructions.

`(CH &rest pitches)`

Produces a chord containing the pitches given by its argument and the current duration.

`(CH1 pitch duration)`
Produces a note whose pitch and duration are given by the arguments. Time is not increased, so that the next event starts at the same time.

`(FUN #'function-name &rest args)`
Calls the function `function-name` sending the arguments in the `args` list. The returning value must be an event to be added to an Xmusic score. This event has to be processed by the *Nyquist* function `timed-seq`. Hence, the event must follow the format `(start-time stretching-factor synthesis-function-call)`

`(MAC macro-name &rest args)`
Calls the function `macro-name` sending the arguments in the `args` list. The returning value must be *SDL* code which replaces the call to the macro.

`number`
This is actually a rest of `number` beat duration.

`symbol`
A note whose pitch is given by `symbol` and with the current duration.

`(pitch dur)`
Note specified by pitch and dur arguments.

4.2 SDL library functions

`(sdl->score score-data &optional time-marks)`
Produces an *Xmusic* score consisting of a `(onset-time stretch-factor synthesis-function)` format event list.
`score-data` is a *SDL* score. `time-marks` is a symbol whose property list contains the time labels to be referenced from `score-data`.

`(sdl->timelabels score-data &optional time-marks)`
Returns a symbol whose property-list contains the time labels in `time-marks` added to `score-data` time labels. Sinchronicity can be ensured by using time labels.
`(sdl:normalize-score-duration score)`
`score` is an *Xmusic* score. This function sets all the event stretching factors to 1.0. This is intended for making synthesis parameters independent from notes duration.

5 lambda Music compatibility

lambda Music is a library developed in Common Lisp and intended for MIDI rendering. Many scores from *lambda Music* can be converted to *SDL* by introducing just minor changes.

6 Final remarks

This library is just an attempt to facilitate the music transcription from traditional notation to synthesis in *Nyquist*. Currently it is under development and therefore some features have to be im-

proved. For instance, the inconsistency between physical and score time. In addition, an extended implementation of the macros should be considered.