qparse

Generated by Doxygen 1.8.15

Chapter 1

src/general

Misc classes:

- ini : project constants and parameters. Some are read from a `.ini` file, based on the C++ header only `version` of the library `inih`.
- QPconfig : verification of compile variables.
- Rational numbers.
- tracing : based on the library `spdlog` of Gabi Melman.

Chapter 2

src/intput

Facilities for reading and writing the data given in input to the quantization by parsing algorithm, i.e.

- the grammar (`WTA`)
- sequence of timestamped musical events (`InputSegment`).

The grammar can be read from a text file describing transition rules and some options (weight type, maximum number of grace notes). For details on the syntax of transition rules and options, see ../../README.md "top readme".

A grammar can can also be saved to a file.

The input segment can be read from files in two formats:

- text format.
- MIDI file input. based on the library `Midifile` of Craig Stuart Sapp.

The input segment (updated with quantized dates after quantization) can be exported (written) as a MIDI file.

Chapter 3

src/output

structures for the representation of the output of the parsing procedure and conversion into music transcription results (i.e. music notation).

Chapter 4

src/parsers

A parser class defines a running environement for transcription by parsing for a given input. It assembles elements from the table directory for the construction of a table used to compute a tree from a grammar and some input.

Each parser class may contain a demo fonction to be called in a target.

The following parsers have been implemented:

- Inputless1best : compute the 1-best tree of a WTA. no input segment.
- Inputlesskbest : compute the k best trees of a WTA. no input segment.
- 1bar1bestSIP : computing the 1-best tree in a given WTA language for the transcription of a given input segment. If the WTA trees represent 1 bar, this scenario is transcription of the whole segment as a single bar.
- 1barkbestSKIP : computing the k best trees in a given WTA language for the transcription of a given input segment.
- Multibar1bestSIPBU : 1-best parsing with SIP pointers. Process input as multiple bars, where a sequence of bars is represented by a binary tree (meta-run), constructed in a bottom-up fashion:

```
                             [bars 1-n]
                              /     \
                  [bars 1-(n-1)]   bar m
                ...
      [bars 1-2]   bar3
       /      \
  [bars 1]    bar2
  /    \
[ ]   bar1
```

Every node corresponds to a SIP pointer. The nodes p1, p2 immediately below a node p represent a binary run (p1, p2) in the table entry for p. The pointers in [ ] correspond to several bars (meta pointers), they do have a non-WTA state. Every other pointer correspond to a single bar, with the initial WTA state (and contain a best run for that bar).

It is assumed that the number of bars is known and the bar length is fixed (tempo does not vary from bar to bar).

This parser can be used for online (bar by bar) transcription.

- Multibar1bestSIPflat : same as above but the sequence of bars is represented by a tuple (flat tree), constructed from left to right.

```
 [bars 1-n]
  /      \
bar1 ... bar m
```

Every node correspond to a SIP pointer. The top note correspond to the whole segment (meta pointer for all bars) Every node below correspond to a single bar, with the initial WTA state (and contain a best run for that bar). This parser cannot be used for online transcription.

This parser is very inefficient with constraint solving (in Table) for pre, post values - need to store an exponential number of partial runs. The BU version is more efficient - with more compact representation of partial runs.

Chapter 5

sources of the qparse library

for rhythm quantization by k-best parsing algorithms based on weighted tree automata.

The sources are organized into the following sub-directories.

- targets : main functions for producing various command line utilities.
- general : initialization, tracing.
- weight : several possible domains for weight values for tree automata.
- schemata : class of weighted tree automata used for parsing.
- segment : classes for abstract representation of data in input processed by parsing.
- table : parse tables.
- parsers : various transcription scenarii, by parsing of a given `input`, assembling elements from the `table` directory.
- input : reading from and writing to files the data given in input to the quantization by parsing algorithm (schema and segment).
- score model : abstract model used to produce scores in various formats from parse trees.
- output : representation of the output of the parsing procedure and conversion into music transcription results.

Chapter 6

src/schemata

Classes related to rhythm `grammars` used in input for quantization by parsing.

A `grammar` (or weighted tree automaton, `WTA`) associates to every tree with `labeled` leaves a unique weight value, in one of the domain defined in directory `weight` (a unique `weight` domain is fixed for a `grammar`).

A `grammar` is defined by a list of `transition` rules, were each transition rule is defined by a target `state`, a body (sequence of states) and a `weight`. An initial `state` is distinguished in every `grammar`.

For reading and saving a `grammar` from a text file, describing transition rules and options, see ../input/README.md "src/input/README".

Chapter 7

src/scoremodel

An abstract model used to produce scores in various format from parsing results (*i.e.* from trees).

Chapter 8

src/segment

Classes for the abstract representation of data in input processed by parsing.

The first category of classes are used for the representation of performances in input: sequences of timestamped musical events.

- `Pitch` : MIDI and name/accident/octave pitch.
- `MusEvent` : musical events (without timestamps); it can be a pitched note or a rest. No time information (onset or duration).
- `Point` : musical event extended with real-time date (in seconds).
  - a point is marked either as `on` of `off` (similarly to note-on / note-off midi messages).
  - a point can be linked to a matching point (according to the MIDI on-off pairing).
  - a point `p` marked `on` and linked to another point `p'` (`on` or `off`) has a duration = `date(p') - date(p)` (this quantity must be positive or null).
  - any other point has an unspecified duration.
- `MusPoint` : Point extended with musical-time date and durations (expressed in fraction of bars).
- `InputSegment` sequence of musical points events, ordered by real-time dates. Constructors for empty input segmemnt and for inserting new points (inservtion respects the date order). For import/export from MIDI files, see ../input/README.md "dir input/".

The second category of classes represent time intervals, and tools for the alignement of input events to these intervals (for quantization). Every interval has real-time and musical-time bound.

- `Interval` : time interval with realtime bounds (in seconds) and musical bounds (in fraction of bars).
- `AlignedInterval` : `Interval` extended with with computed alignment of `InputSegment` points inside the left- and right-bounds: points resp. inside the first half and second half of interval.
- `IntervalTree` : the above organized hierarchicaly in a tree of nested intervals.
- `IntervalHeap` : table for storage of aligned intervals to avoid recomputation of alignments.

Chapter 9

src/table

A table stores the result of parsing wrt a given grammar. It defines the associations to `keys` of `records`, where every `records` contains some `runs`, and can be requested for a k-best `run`.

`Keys` : Several classes of `keys` are implemented (named `Ptr*`), made of different components, amongst:

- state
- rank (kth best number)
- interval bounds and alignments
- pre and post values.

A `key` can have unknown components, and in this case it is called partial. Partial `keys` can be completed symbolic constraint solving techniques (see below).

`Runs` : A `run` is either a tuple of `keys` (inner `run`) or a `label` (terminal = leaf `run`). A `key` in a `run` defines a pointer to a sub-`run`: it is the k-best `run` (according to the rank) for the `key` defined by other components of the pointer. Therefore, one can reconstruct a tree given a `run` and a `table`, and a `table` associates best trees (with `labels` in leaves) to `keys`.

An inner `run` is partial when some of its `key` is unknown or when its weight is not computed (from the weights of sub-`runs`). We complete a `run` by updating the `keys` and weight from left to right.

`Records` : A `record` stores (complete) `runs`. Bests `runs` can be accessed from a record.

`Table` : A `table` associates a `record` (hence best `runs`) to a (complete) `key`. One can add a `run` to a `key` in a given `table`. More precisely:

- one can add a complete `run` to a complete `key` : the `run` is then just added to the associated `record`.
- one can add a partial `run` to a `key`. In that case the `run` is completed first.
- one can add a complete `run` to a partial `key` : the `key` is then completed according to the values in the `run`. Some example can be found in notes.

Chapter 10

/src/targets

definition of command line utilities.

Every `target` is defined in a .cpp file with a main function. The commandline options are handled using the GNU `getopt` function.

- `quant` Transcription of an input given by a text or a MIDI file wrt a given automaton (stored in a file and called schema). Various output possible, including MEI score file and quantized MIDI file.
- `equiv` Enumeration of rhythm trees in a given schema language equivalent to a given sequence of (quantized) durations (Inter-Onset-Intervals).
- `schemas` Utilities for the construction of schema and computation of weights.
- `midiutils` Utilities for reading MIDI files and conversion to text format.

Chapter 11

src/weight

Definition of several domains and operations for the weights of automata. Each of them is defined as a semiring, with

- a weight domain,
- a binary operator `add` (associative, commutative),
- a neutral element `zero` for `plus`,
- a binary operator `mult` (associative),
- a neutral element `one` for mult,

such that `zero` is an absorbing element for `mult` and `mult` distributes over `plus`.

- `Weight` is a structure of polymorphic weight. i.e. a wrapper (envelop) containing a pointer to a `LetterWeight`. A `Weight` with an empty pointer is called unknown.
- a `LetterWeight` is the definition of a semiring. We have implemented the following `LetterWeights`:
- `FloatWeight` : scalar weight values
  - domain : floating point numbers
  - operator `add` is +
  - `zero` is 0.0
  - operator `mult` is $*$
  - `one` is 1.0
- `TropicalWeight` : tropical algebra: scalar weight values are non-negative floating point numbers
  - domain : positive or null double + infinity
  - operator `add` is min
  - `zero` is infinity
  - operator `mult` is +
  - `one` is 0
- `ViterbiWeight` : scalar weight values are probabilities of the best derivations
  - domain : positive or null rational numbers in [0, 1]
  - operator `add` is max

  - zero is 0
  - operator mult is $*$
  - one is 1

- Distance : a particular case of TropicalWeight which can be constructed from an interval of an input segment, and corresponds to the distance of alignement of the points on the left and right bounds of the interval.

- PerformanceModel : a particular case of ViterbiWeight which can be constructed from an interval of an input segment, and defines a probability of alignment of the points on the left and right bounds of the interval, following a Gaussian distribution.

- CountingWeight : a algebra of weight for counting the number of applications of automata transitions rules on a given corpus. Useful for computation of Viterbi Weight values from corpus.
  - domain :
    - vectors of fixed dim k $>$ 0
    - + FAIL = stuck (0 run in state s for 1 tree)
    - + ERROR = ambiguity in grammar (2 runs for 1 tree)
  - operator add : for all x, y vectors dim k
    - x + y = ERROR
    - ERROR is absorbing for +
  - zero = FAIL
  - operator mult : for all x, y vectors dim k
    - x . y = component-wise sum
    - x . FAIL = FAIL . x = FAIL
    - FAIL . FAIL = FAIL
    - is ERROR absorbing for .
  - one = null vector of dim k

Chapter 12

Todo List

Member Atable< P >::bestTree (Run< P > *p)=0

TBR param p

Member ComboState::ComboState (const ComboState &, pre_t rp=0, pre_t rr=0)

TBR

Member DurationList::DurationList (std::string)

TBR only for testing.

Class EventLabel

TBR (NOT USED)

Class InnerLabel

TBR (NOT USED)

Class InputSegment

do the same think with musical time duration.

suppr. samplestosec

suppr. member _res (resolution)

Member InputSegment::quantizu (Atable< P > *table, const P &p, size_t b=0)

TBR (replaced by quantize)

TBR

Member InputSegmentMIDI::export_midifile (std::string, Rational)

TBR mv export to segment/InputSegment* classes

Member InputSegmentMIDI::export_midifile (MidiFile &midifile, std::string midiout, Rational beatperbar)

TBR mv export to segment/InputSegment* classes

Member InputSegmentMIDI::export_midifile_mono (MidiFile &midifile, std::string midiout, Rational beatperbar)

TBR mv export to segment/InputSegment* classes

Member InputSegmentMIDI::InputSegmentMIDI (const std::string filename, bool mono=true, bool norest=false, int tracknb=1)

TBR

Class IntervalHasher

TBR

Class Label

TBR the class Label is not used (except for static members)

Class MusPoint

redefine musical time duration as realtime duration, with links.

replace _mduration by mduration computed from linked point's date

Class Pitch

extend conversions to MIDIcent (import OM)

Member Point::_onoff

TBR

Member Point::_rduration

TBR (added for backward compatibility)

Member Point::rduration () const

TBR (only for backward compability)

Member PreState::PreState (const PreState &)

TBR

Class Run< P >

suppr. null runs

Member SKIPpointer::SKIPpointer (Environment ∗env, pre_t pre=0, pre_t post=0, bool bar=false, size_t k=1)

TBR deprecated

Member SKpointer::SKpointer (WTA ∗a, Environment ∗env, pre_t pre=0, pre_t post=0, Rational mlen=Rational(1), size_t k=1)

TBR deprecated (replace by specific constructor)

Member Spointer::Spointer (WTA ∗a, Environment ∗env, pre_t pre=0, pre_t post=0, Rational mlen=Rational(1), size_t k=1)

TBR deprecated (replace by specific constructor)

Member TRACE_LEVEL

TBR

Member TropicalWeight::make (double v) const

TBR : stricly positive

Member ViterbiWeight::invert ()

TBR

Member Weight::invert ()

TBR : replace by div with const rhs

Member WTA::abstract (bool flag=false)

TBR unused

Member WTA::add (state_t, const Transition &, bool initial=false)

suppr. flag initial

Member WTA::add (state_t, bool initial=false)

suppr. flag initial

Member WTA::initials

SUPPR

Member WTA::isInitial (state_t) const

TBR

Member WTAFile::WTAFile (const std::string filename, bool count_flag=false, bool penalty_flag=true, bool stochastic_flag=false)

TBR

Chapter 13

Module Documentation

13.1 Input module

The `input` module contains utilities for reading from and writing to files the data given in input to the quantization by parsing algorithm (schema and segment).

Classes

- class InputSegmentMIDI

  import an InputSegment from a MIDI file.
- class InputSegmentSerial

  serialization of an input segment in a text file.
- class WTAFile

  wrapper for constructing WTA with various flags for weight type.

13.1.1 Detailed Description

The `input` module contains utilities for reading from and writing to files the data given in input to the quantization by parsing algorithm (schema and segment).

13.2 Output module

The `output` module contains representations for the output of the parsing procedure and conversion into music transcription results.

Namespaces

- ScoreModel

Classes

- class DurationList

  list of rational durations to label nodes of WTA Runs for Kbest enum.
- struct std::hash< DurationList >
- class DurationTree

  a tree container for duration lists. to avoid recomputation of division of duration lists.
- class Label

  labels for nodes of output Rhythm Trees.
- class InnerLabel

  label for inner node. contains only arity (more info later?)
- class EventLabel
- class MEI
- class OMRhythmTree
- class Onsets

  sequence of onsets used for merge of duration lists.
- class PointedRhythmTree
- class QDate

  quantified onset values expressed in number of samples.
- class Position

  position in a RT.
- class RhythmTree

  Rhythm Trees.
- class SerialLabel

  static functions for serializable int encoding of input and output leaf symbols containing the following info:
- class ValueList

  list of rational durations as components of value states.
- struct std::hash< ValueList >

Macros

- #define **RT_PAR_OPEN** '('
- #define **RT_PAR_CLOSE** ')'
- #define **RT_SEP** ','

Typedefs

- typedef size_t label_t

  type for concrete and abstract labels

Enumerations

- enum **LabelKind** { **EVENT**, **TIE**, **INNER** }

Functions

- std::ostream & **operator**<< (std::ostream &o, const DurationList &l)
- bool **operator==** (const DurationList &lhs, const DurationList &rhs)
- bool **operator!=** (const DurationList &lhs, const DurationList &rhs)
- std::ostream & **operator**<< (std::ostream &o, const DurationTree &t)
- std::ostream & **operator**<< (std::ostream &o, const OMRhythmTree &t)
- const Onsets operator+ (const Onsets &lhs, const Onsets &rhs)

  ordered merge
- std::ostream & **operator**<< (std::ostream &o, const QDate &rhs)
- std::ostream & **operator**<< (std::ostream &o, const Position &pos)
- std::ostream & **operator**<< (std::ostream &o, const RhythmTree &t)
- Position **operator+** (const Position &p, const size_t &i)
- std::ostream & **operator**<< (std::ostream &o, const ValueList &l)
- bool **operator==** (const ValueList &lhs, const ValueList &rhs)
- bool **operator!=** (const ValueList &lhs, const ValueList &rhs)
- string RhythmTree::APTED () const

  format for Tree Edit Distance Salzburg library.
- DurationList::DurationList ()

  empty duration list.
- **DurationList::DurationList** (const DurationList &)
- DurationList & **DurationList::operator=** (const DurationList &)
- DurationList::DurationList (const DurationList &l, Rational q)

  copy of duration list l where all elements are multiplied by given Ratio q.
- DurationList::DurationList (std::string)

  read duration list from file.
- bool **DurationList::empty** () const
- bool **DurationList::complete** () const
- bool **DurationList::unit** () const
- bool DurationList::single_continuation () const

  one (non null) continuation and no event in the main list.
- bool DurationList::single_event () const

  no continuation and only one event in the main list.
- bool DurationList::event () const

  no continuation and some grace notes (dur=0) + one event (dur>0) in the main list.
- size_t DurationList::nbgn () const

  number of grace note must be an event()
- Rational DurationList::length () const

  sum of the elements of the duration list (including continuation)
- void DurationList::add (Rational)

  add the event at the end of the main list.
- void DurationList::addcont (Rational)

  push a continuation value.
- DurationList & DurationList::operator+= (const DurationList &rhs)

  concatenation.
- void DurationList::normalize ()

  divide by the number of lists summed.
- **DurationTree::DurationTree** (const DurationList &d)

- DurationTree ∗ **DurationTree::sub** (size_t, size_t)
- **Label::Label** (int a=0)
- size_t **Label::arity** () const
- static size_t **Label::nbGraceNotes** (label_t)
- static size_t **Label::nbEvents** (label_t)
- static bool **Label::continuation** (label_t)
- static bool **Label::abstract** (label_t)
- static bool **Label::abstract** (label_t a, label_t n)
- static bool **Label::leqabstract** (label_t a, label_t n)
- **InnerLabel::InnerLabel** (unsigned int)
- **EventLabel::EventLabel** (unsigned int n=0)
- size_t **EventLabel::nbGraceNotes** () const
- void **EventLabel::addGraceNotes** (unsigned int)
- void **EventLabel::pushEvent** (Event ∗)
- string RhythmTree::lily (int depth, bool tie=false) const

  LilyPond format.
- string RhythmTree::lilydot (int depth)

  LilyPond format with dots.
- string RhythmTree::lilydot (int depth, bool tie, bool dot, bool ignore_first, bool ignore_second)

  LilyPond format with dots.
- MEI::MEI ()
- void MEI::createFromScore (const ScoreModel::Score &s)
- void MEI::createScoreDef (const ScoreModel::Score &s)
- std::pair< string, int > MEI::chooseClef (std::pair< Pitch, Pitch > range)
- void MEI::writeInFile (const string fname)
- static Note ∗ **MEI::makeNote** (const ScoreModel::Note ∗noteEvent)
- static Tie ∗ **MEI::makeTie** (const ScoreModel::Tie ∗tie)
- static TupletSpan ∗ **MEI::makeTupletSpan** (const ScoreModel::Tuplet ∗tuplet)
- MEI::∼MEI ()
- **OMRhythmTree::OMRhythmTree** (Rational lab, bool tied=false)
- **OMRhythmTree::OMRhythmTree** (const RhythmTree ∗, Rational dur=Rational(1))
- size_t **OMRhythmTree::size** () const
- OMRhythmTree ∗ **OMRhythmTree::child** (size_t) const
- void **OMRhythmTree::add** (OMRhythmTree ∗)
- string **OMRhythmTree::to_string** () const
- Onsets::Onsets (const DurationList &)

  the list of onsets defined by the given duration list (IOI's) the first onset is 0.
- DurationList Onsets::ioi () const

  the list of IOI associated to this list of onsets.
- **PointedRhythmTree::PointedRhythmTree** (label_t lab)
- **PointedRhythmTree::PointedRhythmTree** (const RhythmTree ∗, const InputSegment ∗, size_t i=0)
- **QDate::QDate** (size_t blocs, size_t rel)
- **QDate::QDate** (const QDate &)
- virtual QDate & **QDate::operator=** (const QDate &)
- virtual QDate ∗ **QDate::clone** () const
- Rational QDate::absolute (size_t res) const

  quantified date as rational value.
- void **QDate::print** (std::ostream &) const
- void QDate::print (std::ostream &, size_t) const

  fractional print using resolution value.
- Position::Position ()

  empty sequence = root position
- **Position::Position** (const Position &)

- bool **Position::empty** () const
- size_t **Position::length** () const
- void Position::operator+= (size_t i)

  concatenate given int to this position
- void **Position::print** (std::ostream &o) const
- RhythmTree::RhythmTree ()

  empty inner tree (not terminal)
- RhythmTree::RhythmTree (label_t lab)

  single leaf rhythm tree (terminal tree)
- RhythmTree::RhythmTree (const string &)

  extract RT from string description
- bool RhythmTree::terminal () const

  single node tree
- label_t RhythmTree::label () const

  label for terminal node
- bool RhythmTree::continuation () const

  label of terminal node is a continuation
- bool RhythmTree::single_event () const

  label of terminal node is a single event (1 note / rest, no grace note).
- size_t RhythmTree::nbgn () const

  number of grace notes in this terminal node.
- size_t RhythmTree::arity () const

  arity of root node (0 for terminal tree)
- RhythmTree ∗ RhythmTree::child (size_t i) const

  return the ith child of this tree
- void RhythmTree::add (RhythmTree ∗)

  add a subtree.
- bool RhythmTree::reducible () const

  this tree contains a subtree of the form.
- bool RhythmTree::tail_redex () const

  inner and the children list is of the form.
- bool RhythmTree::tail_reducible () const

  inner and one of the children 1..a is reducible.
- bool RhythmTree::tie () const

  return whether this tree is a continuation (a leaf).
- bool RhythmTree::tied () const

  return whether the leftmost innermost leaf is a tie (continuation).
- bool RhythmTree::binary () const

  return whether this tree is binary.
- bool RhythmTree::second_tied () const

  return whether this tree is binary and the second child is tied.
- bool RhythmTree::dot_after () const

  return whether this tree is binary and the left son is a dot (continuation after the dotted note).
- bool RhythmTree::dot_before () const

  return whether this tree is binary and the right son is a dot (continuation before the dotted note).
- string **RhythmTree::to_string** () const
- static size_t SerialLabel::nbEvents (label_t)

  number of note + grace notes encoded in given leaf label
- static pre_t SerialLabel::pre (label_t)

  return the pre value of the given leaf label
- static pre_t SerialLabel::post (label_t)

return the post value of the given leaf label

- static size_t SerialLabel::nbGraceNotes (label_t)

  return the number of grace node encoded in given leaf label
- static bool SerialLabel::continuation (label_t)

  the given leaf label is a continuation (no event, no grace note)
- static label_t SerialLabel::serialize (pre_t pre, pre_t post, size_t nb)

  return the leaf label encoding the given
- **ValueList::ValueList** (Rational)
- **ValueList::ValueList** (std::string)
- **ValueList::ValueList** (const DurationList &)
- **ValueList::ValueList** (const ValueList &)
- ValueList & **ValueList::operator=** (const ValueList &)
- bool **ValueList::empty** () const
- bool **ValueList::complete** () const
- bool **ValueList::single_continuation** () const
- bool **ValueList::event** () const
- bool **ValueList::single_event** () const
- void **ValueList::add** (Rational)
- void **ValueList::addcont** (Rational)
- Rational **ValueList::front** () const
- Rational **ValueList::pop** ()
- Rational **ValueList::popcont** ()
- void **ValueList::popcont** (Rational)

Variables

- static bool RhythmTree::dot_flag = false

  global variable set if a dot is added in lilydot.

13.2.1 Detailed Description

The `output` module contains representations for the output of the parsing procedure and conversion into music transcription results.

MEI interface. Can be used to output MEI document from a transcription result

Author

Philippe Rigaux

13.2.2 Function Documentation

13.2.2.1 APTED()

```
string RhythmTree::APTED ( ) const
```

format for Tree Edit Distance Salzburg library.

RT output format for Tree Edit Distance library APTED algorithm of M. Pawlik and N. Augsten `http://tree-edit-distance.dbresearch.uni-salzburg.at`.

13.2.2.2 DurationList() [1/2]

```
DurationList::DurationList (
             const DurationList & l,
             Rational q )
```

copy of duration list l where all elements are multiplied by given Ratio q.

Parameters

l	suration list to copy and update
q	given ratio for update

13.2.2.3 DurationList() [2/2]

```
DurationList::DurationList (
             std::string filename )
```

read duration list from file.

one ratio per line if the first line is negative ratio, it is a continuation all other line must contain positive or null ratios.

Warning

> the file must not be empty.

Todo TBR only for testing.

13.2.2.4 add() [1/2]

```
void DurationList::add (
             Rational q )
```

add the event at the end of the main list.

Warning

> fail if event cannot be added (makes sum $>$ 1).
> this list must not have have been summed with others.

13.2.2.5 addcont()

```
void DurationList::addcont (
             Rational q )
```

push a continuation value.

Warning

fail if cont cannot be added (makes sum $>$ 1).
this list must not have been summed with others.

13.2.2.6 operator+=() [1/2]

```
DurationList & DurationList::operator+= (
             const DurationList & rhs )
```

concatenation.

Parameters

rhs	duration list to concatenate, must not be empty, and must not be the summation of several duration lists.

13.2.2.7 lily()

```
string RhythmTree::lily (
             int depth,
             bool tie = false ) const
```

LilyPond format.

Lilypond output for RT http://lilypond.org.

13.2.2.8 MEI()

```
MEI::MEI ( )
```

Main constructor

13.2.2.9 createFromScore()

```
void MEI::createFromScore (
             const ScoreModel::Score & s )
```

Check with eth Spiritual example: case of a

13.2.2.10 createScoreDef()

```
void MEI::createScoreDef (
             const ScoreModel::Score & s )
```

Create the score definition part

13.2.2.11 chooseClef()

```
std::pair< string, int > MEI::chooseClef (
             std::pair< Pitch, Pitch > range )
```

Choose a clef based on range

13.2.2.12 writeInFile()

```
void MEI::writeInFile (
             const string fname )
```

Save in file

13.2.2.13 ∼MEI()

```
MEI::∼MEI ( )
```

Destructor

13.2.2.14 Onsets()

```
Onsets::Onsets (
             const DurationList & d )
```

the list of onsets defined by the given duration list (IOI's) the first onset is 0.

Warning

a continuation in duration list will be treated like other events

13.2.2.15 operator+=() [2/2]

```
void Position::operator+= (
             size_t i )
```

concatenate given int to this position

Parameters

i	int must be positive

13.2.2.16 RhythmTree()

```
RhythmTree::RhythmTree ( )
```

empty inner tree (not terminal)

Warning

> the child list must be completed with add

13.2.2.17 label()

```
label_t RhythmTree::label ( ) const
```

label for terminal node

Warning

> this tree must be terminal

13.2.2.18 continuation()

```
bool RhythmTree::continuation ( ) const
```

label of terminal node is a continuation

Warning

> this tree must be terminal

13.2.2.19 single_event()

```
bool RhythmTree::single_event ( ) const
```

label of terminal node is a single event (1 note / rest, no grace note).

Warning

> this tree must be terminal

13.2.2.20 nbgn()

```
size_t RhythmTree::nbgn ( ) const
```

number of grace notes in this terminal node.

Warning

this tree must be terminal

13.2.2.21 child()

```
RhythmTree * RhythmTree::child (
             size_t i ) const
```

return the ith child of this tree

Warning

this tree must be inner (not terminal)

13.2.2.22 add() [2/2]

```
void RhythmTree::add (
             RhythmTree * t )
```

add a subtree.

Warning

this tree must not be terminal

13.2.2.23 reducible()

```
bool RhythmTree::reducible ( ) const
```

this tree contains a subtree of the form.

```
p(n, o,...,o)
```

or

```
p(o,...,o)
```

13.2.2.24 tail_redex()

```
bool RhythmTree::tail_redex ( ) const  [protected]
```

inner and the children list is of the form.
`(_, o,...,o)`

13.2.2.25 serialize()

```
label_t SerialLabel::serialize (
            pre_t pre,
            pre_t post,
            size_t nb )  [static]
```

return the leaf label encoding the given

Parameters

pre	value in 0..MAX_GRACE
post	value in 0..MAX_GRACE
nb	number of events

13.3 Schemata module

The `schemata` module contains classes of weighted tree automata used for parsing.

Namespaces

- State

  States.

Classes

- class ComboState

  tmp state structure for construction of ComboWTA from a WTA (base schema) and an input segment casted into state_t after construction
- struct ComboStateHasher
- class ComboWTA

  WTA combo: A special kind of WTA for quantization constructed from.
- class CountingWTA

  copy of WTA dedicated to corpus statistics.
- class PreState

  tmp state structure for construction of PreWTA from a WTA (base schema) casted aka state_t after construction
- class PreWTA

  extension of WTA where states are associated pre and post values.
- class AONode

  AND-OR alternating nested lists used by Adrien in RQ.
- class ANode
- class ONode
- struct ds_transition

  dag schema
- class dagSchema

  dag whose edges are labeled by arity values two distinguished nodes:
- class Transition

  a Transition is defined by a sequence of antecedent states (body) the weight must be not null (null weight means a missing transition).
- class ValueState
- struct ValueStateHasher
- class ValueWTA

  Value WTA is a special kind of WTA associated to an initial WTA (schema) and a rhythmic value (DurationList).
- class TransitionList
- class WTA

  class of schemas = weighted tree automata = weighted CFG.
- class DepthMarking

  marking of states of a WTA with informations on the depth of their occurences initialized with a WTA, can be interrogated afterwards

Typedefs

- typedef std::unordered_map< ComboState, state_t, ComboStateHasher > **Combomap**
- typedef std::set< std::pair< state_t, Transition & >, bool(∗)(std::pair< state_t, Transition & >, std::pair< state_t, Transition & >)> OTransitionTable

  transtition table ordered by transition's ids
- typedef long **state_t**
- typedef std::vector< state_t >::iterator **Transition_iterator**
- typedef std::vector< state_t >::const_iterator **Transition_const_iterator**
- typedef std::unordered_map< ValueState, state_t, ValueStateHasher > **Valuemap**
- typedef std::list< Transition >::iterator **TransitionList_iterator**
- typedef std::list< Transition >::const_iterator **TransitionList_const_iterator**

Functions

- std::ostream & **operator**<< (std::ostream &o, const ComboState &cs)
- bool **trcomp** (std::pair< state_t, Transition &> lhs, std::pair< state_t, Transition &> rhs)
- std::ostream & **operator**<< (std::ostream &o, const CountingWTA &a)
- std::ostream & **operator**<< (std::ostream &o, const PreState &ps)
- std::ostream & **operator**<< (std::ostream &o, const PreWTA &a)
- std::ostream & **operator**<< (std::ostream &o, const Transition &t)
- std::ostream & **operator**<< (std::ostream &o, const ValueState &vs)
- size_t **gcd** (size_t a, size_t b)
- size_t **lcm** (size_t a, size_t b)
- std::ostream & **operator**<< (std::ostream &o, const WTA &a)
- **ComboState::ComboState** (const InputSegment ∗s, IntervalHeap ∗)
- **ComboState::ComboState** (state_t, IntervalTree ∗, pre_t rp=0, pre_t rr=0)
- ComboState::ComboState (const ComboState &, pre_t rp=0, pre_t rr=0)
- bool **ComboState::compatible** (label_t label) const
- bool **ComboState::operator==** (const ComboState &s) const
- bool ComboState::operator< (const ComboState &s) const

  lexicographic comparison on hash value (array[5])
- state_t ComboWTA::initial (pre_t pre=0, pre_t post=0) const

  state representing the whole segment.
- ComboWTA::ComboWTA (const InputSegment ∗, size_t bloc, const WTA &, pre_t pre=0)

  construction from input segment and WTA (base schema) with given max pre value and bloc number (in input segment, for alignement).
- CountingWTA::CountingWTA ()

  default initializer for cython
- CountingWTA::CountingWTA (const WTA &a)

  copy base WTA reset weight values to counting weights (unit vectors)
- void CountingWTA::resetCounting (size_t dim)

  the weight of this WTA are replaced by "CountingWeight" unit vector of length dim (one unit per transition)
- virtual Weight CountingWTA::eval (const RhythmTree &t) const

  special version of eval for CountingWeight with feedback in case of fail
- Weight **CountingWTA::evalCountingVerbose** (const RhythmTree &, state_t, Position) const
- **PreState::PreState** (state_t, pre_t pre=0, pre_t post=0)
- PreState::PreState (const PreState &)
- bool **PreState::operator==** (const PreState &s) const
- bool PreState::operator< (const PreState &s) const

  lexicographic comparison on hash value (array[5])
- state_t PreState::serialize ()

return a state value unically associated to this PreState

- bool **PreState::compatible** (label_t label) const
- static bool PreState::compatible_post (state_t, const AlignedInterval ∗)

  compatible(s, al) the serialized state value s is compatible with the content of the alignment al (sub-segment of initial input corr. to an interval)
- PreWTA::PreWTA (const WTA &)

  construction from WTA (base schema)
- static pre_t PreWTA::pre (state_t)

  access to original components of new PreWTA states
- static pre_t **PreWTA::post** (state_t)
- static state_t **PreWTA::state** (state_t)
- virtual state_t PreWTA::initial (pre_t pre=0, pre_t post=0) const

  initial(pre, port) returns state representing the whole segment, with pre points of the previous segment aligned to the left and post points of the current segment aligned to the right (i.e. to the left of the next segment)
- bool **State::isMeta** (state_t)
- bool **State::isWTA** (state_t)
- bool **State::isLabel** (state_t)
- state_t State::MetaState (size_t barnb)

  Meta state corresponding to bar nb barnb.
- void **ds_transition::rename** (unsigned int s, unsigned int u)
- void ds_transition::shift (unsigned int n)

  increase source and target state by n
- void ds_transition::shift0 (unsigned int n)

  increase source and target state by n, if they are not 0
- dagSchema::dagSchema (const ANode &)

  translation of AND-OR alternating nested lists into dag-schemas
- **dagSchema::dagSchema** (const ONode &)
- void dagSchema::add (const ds_transition &dst)
- Transition::Transition ()

  transition with unknown weight and empty body.
- Transition::Transition (const Weight &)

  Transition(w) creates a transition with weight a copy of w and empty body.
- Transition::Transition (LetterWeight ∗)

  Transition(lw) creates a transition with weight a wrapper of the letter lw (must be non null)
- Transition::Transition (std::vector$<$ state_t $>$, const Weight &)

  Transition(v, w) creates a transition with weight a copy of w and body a copy of the vector v.
- Transition::Transition (std::vector$<$ state_t $>$, LetterWeight ∗)

  Transition(v, lw) creates a transition with weight a wrapper of the letter lw (must be non null) and body a copy of the vector v.
- Transition::Transition (state_t, const Weight &)

  Transition(s, w) creates a transition with weight a copy of w and body (of size 1) the singleton (s) (terminal symbol).
- Transition::Transition (state_t, LetterWeight ∗)

  Transition(s, lw) creates a transition with weight a wrapper of the letter lw (must be non null) and body (of size 1) the singleton (s) (terminal symbol).
- bool **Transition::inner** () const
- bool **Transition::terminal** () const
- label_t Transition::label () const
- void Transition::scalar (double)

  modify weight of transition.
- void **Transition::invert** ()
- size_t Transition::size () const

  size of body.

- size_t **Transition::arity** () const
- state_t Transition::at (size_t i) const

  at(i) returns the ith state in the body.
- void Transition::push (state_t)

  add given state at the end of the body of this transition.
- bool Transition::member (state_t) const

  whether the given state belongs to the body of this transition.
- bool Transition::allin (const std::set$<$ state_t $>$ &) const

  every state of the body is in the given set.
- bool Transition::nonein (const std::set$<$ state_t $>$ &) const

  no state of the body is in the given set.
- **ValueState::ValueState** (state_t, DurationTree $*$)
- bool **ValueState::compatible** (label_t label) const
- bool **ValueState::operator==** (const ValueState &s) const
- ValueWTA::ValueWTA (const DurationList &, const WTA &)

  construction from given initial list and WTA (base schema)
- bool TransitionList::empty () const

  zero transition
- size_t TransitionList::size () const

  number of transitions.
- void **TransitionList::add** (const Transition &)
- void **TransitionList::clear** ()
- void **TransitionList::remove** (TransitionList_iterator)
- void TransitionList::remove (state_t)

  remove all transitions of length $>$ 1 in the list containing the given state do not remove length 1 transitions to terminal symbols
- WTA::WTA ()

  nullary constructor for cython
- WTA::WTA (Weight seed, pre_t pre=0, pre_t post=0)

  empty automaton
- size_t WTA::size () const

  number of states
- bool **WTA::empty** () const
- bool WTA::isRegistered (state_t) const

  the state is present in the automaton
- bool WTA::isInitial (state_t) const

  the state is an initial state
- TransitionList & WTA::add (state_t, bool initial=false)

  add(s, i) register state s if s was already registered, return a reference to its transition list. otherwise, create state s with an empty transition list and returns a reference to it. moreover s is set as initial if i = true.
- TransitionList & WTA::add (state_t, const Transition &, bool initial=false)

  add(s, t) add a transition with head s and with body/weight described in t if s was not registered, it is registered the transition t is added to the transition list of s and a reference to this transition list is returned moreover s is set as initial if i = true.
- void WTA::remove (state_t)

  remove the entry for given state s in the table of the table i.e. all transitions targeted to s, and all the transitions with s in their body. if s was in the initial set, it is also removed from this set. s must be registered.
- TransitionList_const_iterator WTA::begin (state_t) const

  begin(s) returns an iterator pointing to the first transition with head state s. s must be registered. not for modifying transition list of s. use add(...) methods for this.
- TransitionList_const_iterator WTA::end (state_t) const

  begin(s) returns an iterator pointing to the past-the-end transition with head state s. s must be registered. not for modifying transition list of s. use add(...) methods for this.

- size_t WTA::countStates () const

  number of states
- size_t WTA::countTransitions () const

  number of transition
- size_t WTA::countAll () const

  number of symbols (state occurences)
- size_t WTA::oftarget (state_t) const

  oftarget(s) return the number of transitions of head state s. s must be registered.
- size_t **WTA::resolution** () const
- std::set< state_t > WTA::step (const std::set< state_t > &)

  step(s) returns the set of states reachable in one transition step by this WTA from the given state set s. all the states in the set s must be registered.
- std::set< state_t > WTA::allStates () const

  returns the set of all states occuring in wta (in head or body)
- std::set< state_t > WTA::emptyStates () const

  returns the set of all non-inhabited (zero weight) states in wta
- bool WTA::isClean () const

  the WTA has no empty states
- void WTA::clean ()

  remove states not inhabited and transitions containing these states
- void WTA::abstract (bool flag=false)

  abstract the leaf label values in domain [0..MAX_GRACE] every value > MAX_GRACE is casted to MAX_GRACE the weights are summed accordingly
- void WTA::CountingtoStochastic ()

  cast weights in all transitions.
- void WTA::CountingtoPenalty ()

  cast weights in all transitions.
- void WTA::PenaltytoCounting ()

  cast weights in all transitions.
- void WTA::StochastictoPenalty ()

  cast weights in all transitions.
- bool WTA::hasWeightType (std::string code) const

  return wether the weights in transition have the type of the code (code of the letter weight if there is one or "UNKN↩ OWN" otherwise).
- virtual Weight WTA::weight_zero () const

  return the 0 value in the weight domain in this WTA
- virtual Weight WTA::weight_one () const

  return the 1 value in the weight domain in this WTA
- virtual Weight WTA::eval (const RhythmTree &t) const

  evaluate the weight of the tree t for WTA in initial state
- virtual Weight **WTA::eval** (const RhythmTree &t, state_t s) const
- void WTA::print (std::ostream &) const

  print sizes to output stream
- **DepthMarking::DepthMarking** (const WTA &)
- int DepthMarking::depth (state_t) const

  return depth mark if given state marked return -1 otherwise
- bool DepthMarking::multiple (state_t) const

  return true if the given state can occur at multiple depths return false otherwise or if state not marked
- int DepthMarking::mark (state_t, int)

  mark state using given depth and return new mark value can be the given depth or a greater depth with which the state had been already marked.

Variables

- static bool(∗ CountingWTA::_trcomp_ptr)(std::pair< state_t, Transition &>, std::pair< state_t, Transition &>) = &trcomp

  pointer to comparison functionå

13.3.1 Detailed Description

The `schemata` module contains classes of weighted tree automata used for parsing.

13.3.2 Function Documentation

13.3.2.1 ComboState()

```
ComboState::ComboState (
            const ComboState & cs,
            pre_t rp = 0,
            pre_t rr = 0 )
```

Todo TBR

13.3.2.2 initial()

```
state_t ComboWTA::initial (
            pre_t pre = 0,
            pre_t post = 0 ) const  [virtual]
```

state representing the whole segment.

Parameters

pre	points of the previous segment aligned to the left
post	points of the current segment aligned to the right (i.e. to the left of the next segment).

Reimplemented from WTA.

13.3.2.3 PreState()

```
PreState::PreState (
            const PreState & ps )
```

Todo TBR

13.3.2.4 add() [1/3]

```
void dagSchema::add (
            const ds_transition & dst )
```

Warning

for testing. do not use

13.3.2.5 Transition() [1/3]

```
Transition::Transition (
            const Weight & w )
```

Transition(w) creates a transition with weight a copy of w and empty body.

Warning

the letter weight in the envelop w is cloned

13.3.2.6 Transition() [2/3]

```
Transition::Transition (
            std::vector< state_t > v,
            const Weight & w )
```

Transition(v, w) creates a transition with weight a copy of w and body a copy of the vector v.

Warning

the letter weight in the envelop w is cloned.

13.3.2.7 Transition() [3/3]

```
Transition::Transition (
            state_t s,
            const Weight & w )
```

Transition(s, w) creates a transition with weight a copy of w and body (of size 1) the singleton (s) (terminal symbol).

Warning

the letter weight in the envelop w is cloned.

13.3.2.8 label()

```
label_t Transition::label ( ) const
```

Warning

this transition must be terminal

13.3.2.9 at()

```
state_t Transition::at (
            size_t i ) const
```

at(i) returns the ith state in the body.

Parameters

i	must be an index of the body.

13.3.2.10 empty()

```
bool TransitionList::empty ( ) const
```

zero transition

Returns

an empty transition

13.3.2.11 size()

```
size_t TransitionList::size ( ) const
```

number of transitions.

Returns

the number of transitions in this WTA

13.3.2.12 isInitial()

```
bool WTA::isInitial (
            state_t s ) const
```

the state is an initial state

Todo TBR

13.3.2.13 add() [2/3]

```
TransitionList & WTA::add (
            state_t s,
            bool initial = false )
```

add(s, i) register state s if s was already registered, return a reference to its transition list. otherwise, create state s with an empty transition list and returns a reference to it. moreover s is set as initial if i = true.

Todo suppr. flag initial

13.3.2.14 add() [3/3]

```
TransitionList & WTA::add (
            state_t s,
            const Transition & t,
            bool initial = false )
```

add(s, t) add a transition with head s and with body/weight described in t if s was not registered, it is registered the transition t is added to the transition list of s and a reference to this transition list is returned moreover s is set as initial if i = true.

Todo suppr. flag initial

13.3.2.15 abstract()

```
void WTA::abstract (
            bool flag = false )
```

abstract the leaf label values in domain [0..MAX_GRACE] every value $>$ MAX_GRACE is casted to MAX_GRACE the weights are summed accordingly

leaf labels in domain of Label (not SerialLabel).

Todo TBR unused

13.3.2.16 CountingtoStochastic()

```
void WTA::CountingtoStochastic ( )
```

cast weights in all transitions.

Warning

> this WTA must have Weight Type "FloatWeight". this WTA is casted into Weight Type "ViterbiWeight" divide by sum for target state

13.3.2.17 CountingtoPenalty()

```
void WTA::CountingtoPenalty ( )
```

cast weights in all transitions.

Warning

> this WTA must have Weight Type "FloatWeight". this WTA is casted into Weight Type "TropicalWeight" composition of CountingtoStochastic and StochastictoPenalty

13.3.2.18 PenaltytoCounting()

```
void WTA::PenaltytoCounting ( )
```

cast weights in all transitions.

Warning

> this WTA must have Weight Type "TropicalWeight". this WTA is casted into Weight Type "FloatWeight" inverse

13.3.2.19 StochastictoPenalty()

```
void WTA::StochastictoPenalty ( )
```

cast weights in all transitions.

Warning

> this WTA must have Weight Type "ViterbiWeight". this WTA is casted into Weight Type "TropicalWeight" -ln

13.4 Segment module

The `segment` module contains classes for abstract representation of data in input processed by parsing.

Classes

- class AlignedInterval

  Extension of Interval with computed alignment of InputSegment points onto left- and right-bounds.
- class Record< P >

  abstract class describing the basic functionalities of a record.
- class Environment

  wrapper abstract class embedding a standard input environment for parsing algos.
- class Atable< P >

  abstract interface to parse table
- class Run< P >

  a run is a compact representation of parse trees as a tuple of pointers to subruns.
- class InputSegment

  intermediate representation for input performance data (sequence of timestamped events).
- class InputSegmentMono

  conversion of InputSegment to remove overlapping notes.
- class InputSegmentNogap
- class Interval

  an Interval in an input segment with realtime bounds (seconds) and musical bounds (fraction of bars).
- struct IntervalHasher

  hash function for using interval as key in a unordered map.
- struct PointedIntervalEq
- struct PointedIntervalHash
- class IntervalHeap

  table for storage of aligned intervals to avoid recomputation of alignments.
- class IntervalTree

  extension of Aligned Interval to define a tree of nested Alignements with sharing using hash table to store all alignment constructed.
- class MusEvent

  input events
- class RestEvent
- class NoteEvent
- class MusPoint

  Point extended with mutable musical time date and duration (expressed in fraction of bars).
- class Pitch

  internal representation of a pitch value.
- class Point

  timestamped event.
- struct SpiralPoint

  Elaine Chew's spiral of fifths.
- struct NoteName

Typedefs

- typedef std::unordered_set< IntervalTree ∗, PointedIntervalHash, PointedIntervalEq > **IntervalSet**

Functions

- std::ostream & **operator**<< (std::ostream &o, const AlignedInterval &p)
- std::ostream & **operator**<< (std::ostream &o, const InputSegment &s)
- std::ostream & **operator**<< (std::ostream &o, const Interval &p)
- std::ostream & **operator**<< (std::ostream &o, const IntervalTree &p)
- std::ostream & **operator**<< (std::ostream &o, const MusEvent &rhs)
- std::ostream & **operator**<< (std::ostream &o, const Pitch &p)
- std::ostream & **operator**<< (std::ostream &o, const Point &rhs)
- bool **operator==** (const SpiralPoint &lhs, const SpiralPoint &rhs)
- bool **operator!=** (const SpiralPoint &lhs, const SpiralPoint &rhs)
- std::ostream & **operator**<< (std::ostream &o, const SpiralPoint &rhs)
- bool **operator==** (const NoteName &lhs, const NoteName &rhs)
- bool **operator!=** (const NoteName &lhs, const NoteName &rhs)
- std::ostream & **operator**<< (std::ostream &o, const NoteName &p)
- AlignedInterval::AlignedInterval (const InputSegment ∗s, Rational mend=Rational(1), bool f_align=false)

  Interval covering the whole length of the given input segment with given musical time length (number of bars).
- AlignedInterval::AlignedInterval (const InputSegment ∗s, Rational mbeg, Rational mend, double rbeg, double rend, size_t first, bool f_align=false)

  aligned interval with musical-time bounds [mbegin, mbegin+mdur[and real-time bounds [rbegin, rbegin+rdur[for the input segment s.
- AlignedInterval::AlignedInterval (const AlignedInterval &)

  copy.
- virtual AlignedInterval & **AlignedInterval::operator=** (const AlignedInterval &)
- virtual bool **AlignedInterval::operator==** (const AlignedInterval &) const
- bool AlignedInterval::aligned () const

  this interval has been aligned.
- size_t AlignedInterval::align (const InputSegment ∗s, size_t b)

  set the alignment parameters, starting from index b of input segment point and return the next index of point in input segment to be processed (first index at right of this interval) or the size of input segment (total # points) if end of segment is reached.
- size_t AlignedInterval::align (const InputSegment ∗s)

  same as previous but uses _seg_first instead of argument b.
- size_t AlignedInterval::rewind (const InputSegment ∗s, size_t b)

  compute only the value of the next point (the first element of input segment after the right bound of this interval) starting from index b of input segment point.
- size_t AlignedInterval::rewind (const InputSegment ∗)

  same as previous but uses _seg_first instead of arg. b.
- Environment::Environment (InputSegment ∗s=NULL)
- InputSegment::InputSegment (double b=0, double e=0)

  constructs an empty input segment (no events)
- **InputSegment::InputSegment** (const InputSegment &)
- InputSegment::InputSegment (const InputSegment &s, double b, double e)

  copy and resize.
- size_t InputSegment::size () const

  number of non-floating points in segment.
- std::vector< MusPoint >::iterator InputSegment::begin ()

  iterators to the segment's contents.
- std::vector< MusPoint >::iterator **InputSegment::end** ()
- std::vector< MusPoint >::const_iterator **InputSegment::cbegin** () const
- std::vector< MusPoint >::const_iterator **InputSegment::cend** () const
- bool **InputSegment::check_index** (long i) const
- void InputSegment::link (long i, long j)

the event of index i is linked to the event of index j.

- long InputSegment::add_back (MusEvent ∗e, double rdate, double rdur, bool on, long link=MUSPOINTREF_NULL, Rational mdate=MUSTIME_UNKNOWN, Rational mduration=MUSTIME_UNKNOWN)

  insert new timestamped muspoint created from the parameters, at the end of the segment.
- long **InputSegment::add_back** (const MusPoint &)
- long InputSegment::add_floating (MusEvent ∗e, double rdate, double rdur, bool on, long link=MUSPOINTREF_NULL, Rational mdate=MUSTIME_UNKNOWN, Rational mduration=MUSTIME_UNKNOWN)

  create new timestamped muspoint from the parameters, and add the the heap of floating points (not in segment).
- long **InputSegment::add_floating** (const MusPoint &)
- const MusPoint & InputSegment::point (long i) const

  return a ref to the point of index i.
- MusPoint & InputSegment::ncpoint (long i)

  same as point but not const.
- MusEvent ∗ InputSegment::event (long i) const

  return the event of the point of index i.
- double InputSegment::rdate (long i) const

  return the real-time date (in seconds) of the point of index i
- double InputSegment::rduration (const MusPoint &p) const

  return the real-time duration (in seconds) of the given point.
- double InputSegment::rduration (long i) const

  return the real-time duration (in seconds) of the point of index i.
- Rational & InputSegment::mdate (long i)

  return a reference to the musical-time date (in fraction of bar) of the point of index i.
- Rational & InputSegment::mduration (long i)

  return a reference to the musical-time duration (in fraction of bar) of the point of index i.
- void InputSegment::close (double e)

  set end date.
- void InputSegment::respell (int k=0)

  pitch spelling. unwindowed.
- void InputSegment::respell (Rational ws, int k=0)

  pitch spelling with a sliding window of given musical duration.
- void InputSegment::print (std::ostream &) const

  print size to output stream.
- template<class P >
  void InputSegment::quantize (Atable< P > ∗table, const P &p)

  set the musical time date and duration of events in this given input segment, according to the best run for p in given table.
- template<class P >
  size_t InputSegment::quantizu (Atable< P > ∗table, const P &p, size_t b=0)

  set the musical time date and duration of events in this given input segment, according to the best run for p in given table, starting from point number b in interval.
- InputSegmentMono::InputSegmentMono (const InputSegment &s)

  transform the given input segment into a monophonic input segment (no two notes in the same time).
- InputSegmentNogap::InputSegmentNogap (const InputSegment &s, bool norest=true)

  transform the given input segment into a new input segment without gaps.
- Interval::Interval (const InputSegment ∗s, Rational mend=Rational(1))

  top interval constructed from an input segment.
- Interval::Interval (const InputSegment ∗s, Rational mbeg, Rational mend, double rbeg, double rend)

  build an interval with musical-time bounds [mbegin, mbegin+mdur[and real-time bounds [rbegin, rbegin+rdur[for the input segment s.
- Interval::Interval (const Interval &)

  copy.

- Interval::Interval (Interval ∗)

  used for copy of downcasted IntervalTree.
- virtual Interval & **Interval::operator=** (const Interval &)
- virtual bool Interval::operator== (const Interval &) const

  for using Interval as key in map.
- bool **Interval::insideBar** () const
- bool **IntervalHeap::empty** () const
- size_t **IntervalHeap::size** () const
- IntervalTree ∗const IntervalHeap::make (const InputSegment ∗s, Rational mend, double rext=0)

  find or create (and push) a top interval of real-time duration covering the whole length of the given input segment s (root of interval tree) + the given extension.
- IntervalTree ∗const IntervalHeap::make (const InputSegment ∗s, Rational mbeg, Rational mend, double rbeg, double rend, size_t first, IntervalTree ∗p, IntervalTree ∗ps)

  get interval from heap, build it if not present.
- IntervalTree::IntervalTree (const InputSegment ∗s, Rational mend=Rational(1))

  top interval (root of interval tree).
- IntervalTree::IntervalTree (const InputSegment ∗s, Rational mbeg, Rational mend, double rbeg, double rend, size_t first, IntervalTree ∗p=NULL, IntervalTree ∗ps=NULL)

  build an interval tree with musical-time bounds [mbegin, mbegin+mdur[and real-time bounds [rbegin, rbegin+rdur[for the input segment s.
- IntervalTree ∗ IntervalTree::top (const InputSegment ∗s, IntervalHeap ∗h, Rational mend=Rational(1))

  top interval (root of interval tree) covering the whole length of the given input segment s.
- IntervalTree ∗ IntervalTree::split (const InputSegment ∗, IntervalHeap ∗, double rdur, Rational mdur, size_t i)

  return a sub interval.
- IntervalTree ∗ IntervalTree::split_back (const InputSegment ∗, IntervalHeap ∗, double rdur, Rational mdur, size_t i)

  return a sub interval.
- IntervalTree ∗ IntervalTree::sub (const InputSegment ∗, IntervalHeap ∗, size_t a, size_t i)

  return a the i-1th sub-interval of the division of this interval in n equal parts. the sub-interval returned is aligned.
- **MusEvent::MusEvent** (int nb=EVENTNB_UNKNOWN)
- **MusEvent::MusEvent** (const MusEvent &)
- **RestEvent::RestEvent** (int nb=EVENTNB_UNKNOWN)
- **RestEvent::RestEvent** (const RestEvent &)
- virtual MusEvent ∗ **RestEvent::clone** () const
- virtual void **RestEvent::print** (std::ostream &o) const
- NoteEvent::NoteEvent (unsigned int vel=MusEvent::UNDEF_VELOCITY, int nb=EVENTNB_UNKNOWN)

  unpitched note (drums).
- NoteEvent::NoteEvent (Pitch p, unsigned int vel=MusEvent::UNDEF_VELOCITY, int nb=EVENTNB_UNKNOWN)

  pitched note.
- NoteEvent::NoteEvent (unsigned int p, unsigned int vel=MusEvent::UNDEF_VELOCITY, int nb=EVENTNB_UNKNOWN)

  pitched note with MIDI pitch in 0..127.
- **NoteEvent::NoteEvent** (const NoteEvent &)
- virtual MusEvent ∗ **NoteEvent::clone** () const
- virtual void **NoteEvent::print** (std::ostream &o) const
- **MusPoint::MusPoint** (MusEvent ∗e, double rdate, double rdur, bool on, long link=MUSPOINTREF_NULL, Rational mdate=MUSTIME_UNKNOWN, Rational mduration=MUSTIME_UNKNOWN)
- MusPoint::MusPoint (const Point &p, Rational mdate=MUSTIME_UNKNOWN, Rational mduration=MUSTIME_UNKNOWN)

  copy of point.
- MusPoint::MusPoint (const MusPoint &)

  event (if any) is cloned.
- MusPoint & MusPoint::operator= (const MusPoint &)

  event (if any) is cloned.
- bool **MusPoint::operator==** (const Point &) const

- virtual void **MusPoint::print** (std::ostream &o) const
- Pitch::Pitch ()

  undef pitch value.
- Pitch::Pitch (char name, float alt=0.0, int oct=0)

  construct pitch from name+alteration+octave.
- Pitch::Pitch (unsigned int pitch, PitchUnit u=MIDI)

  construct note from MIDI pitch
- **Pitch::Pitch** (const Pitch &)
- Pitch & **Pitch::operator=** (const Pitch &)
- bool **Pitch::operator==** (const Pitch &) const
- Point::Point (MusEvent ∗e, double rdate, double rdur, bool on, long link=MUSPOINTREF_NULL)

  timestamped monophonic or polyphonic event.
- Point::Point (const Point &)
- Point::∼Point ()
- virtual Point & Point::operator= (const Point &)
- virtual bool **Point::operator==** (const Point &) const
- virtual void **Point::print** (std::ostream &o) const
- **SpiralPoint::SpiralPoint** (double, double, double)
- **SpiralPoint::SpiralPoint** (const SpiralPoint &rhs)
- SpiralPoint & **SpiralPoint::operator=** (const SpiralPoint &)
- bool SpiralPoint::isnormal () const
- void **SpiralPoint::operator+=** (const SpiralPoint &rhs)
- void **SpiralPoint::operator-=** (const SpiralPoint &rhs)
- void **SpiralPoint::operator∗=** (double a)
- double SpiralPoint::distance (const SpiralPoint &rhs) const
- NoteName::NoteName (char n, float alt, int id)

  notename object from name, alteration and index.
- **NoteName::NoteName** (const NoteName &rhs)
- NoteName & **NoteName::operator=** (const NoteName &rhs)
- static const NoteName & NoteName::ofkey (int k)

  ref to a NoteName in table synonyms. ∗/
- static const NoteName & NoteName::closest (unsigned int pitch, const SpiralPoint &p)

  note name (ref in table synonyms) corresponding to given midi pitch and closest to given point.

Variables

- static const unsigned int **MusEvent::UNDEF_VELOCITY** = 128
- static const unsigned int **Pitch::UNDEF_MIDICENT** = 12800
- static const char **Pitch::UNDEF_NOTE_NAME** = 'X'
- static const int **Pitch::UNDEF_NOTE_OCTAVE** = 128
- static const float **Pitch::UNDEF_NOTE_ALTERATION** = 11
- static const int **NoteName::UNDEF_NOTE_INDEX** = 99
- static const double NoteName::h = 1.0

  z distance between two successive points of the spiral (one fifth apart).
- static const double NoteName::r = std::sqrt(7.5) ∗ h

  radius of the cylinder in which the spiral is embedded.
- static const NoteName **NoteName::synonyms** [12][3]

13.4.1 Detailed Description

The `segment` module contains classes for abstract representation of data in input processed by parsing.

13.4.2 Function Documentation

13.4.2.1 AlignedInterval() [1/2]

```
AlignedInterval::AlignedInterval (
            const InputSegment * s,
            Rational mend = Rational(1),
            bool f_align = false )
```

Interval covering the whole length of the given input segment with given musical time length (number of bars).

Parameters

s	given input segment
mend	given musical time length
f_align	flag says wether alignement must be computed for the interval.

13.4.2.2 AlignedInterval() [2/2]

```
AlignedInterval::AlignedInterval (
            const InputSegment * s,
            Rational mbeg,
            Rational mend,
            double rbeg,
            double rend,
            size_t first,
            bool f_align = false )  [protected]
```

aligned interval with musical-time bounds [mbegin, mbegin+mdur[and real-time bounds [rbegin, rbegin+rdur[for the input segment s.

Parameters

first	must be the first element of input segment after the beginning of this interval.
f_align	flag says wether alignement must be computed for the interval.

13.4.2.3 align()

```
size_t AlignedInterval::align (
            const InputSegment * s,
            size_t b )
```

set the alignment parameters, starting from index b of input segment point and return the next index of point in input segment to be processed (first index at right of this interval) or the size of input segment (total # points) if end of segment is reached.

Parameters

s	input segment processed
b	must be the index of a segment's point. it must be after the left bound of the interval (begin). it can be out of the interval, i.e. after the right bound (end). _seg_first is replaced by b.

Warning

> The realtime begin date of this interval can be out of the input segment bounds.
> The realtime end date of this interval can be out of the input segment bounds. In the later case, alignement is done like the input segment is padded with empty space up to the end of this interval.

13.4.2.4 rewind()

```
size_t AlignedInterval::rewind (
             const InputSegment * s,
             size_t b )
```

compute only the value of the next point (the first element of input segment after the right bound of this interval) starting from index b of input segment point.

Parameters

b	same preconditions on b as for align.

13.4.2.5 Environment()

```
Environment::Environment (
             InputSegment * s = NULL )
```

Parameters

s	input segment can be : • NULL : e.g. for simple enumeration of the given wta • non-NULL: e.g. for quantization of the points of given input segment using a given wta

13.4.2.6 InputSegment() [1/2]

```
InputSegment::InputSegment (
             double b = 0,
             double e = 0 )
```

constructs an empty input segment (no events)

Parameters

b	start date (in seconds)
e	end date (in seconds)

13.4.2.7 InputSegment() [2/2]

```
InputSegment::InputSegment (
            const InputSegment & s,
            double b,
            double e )
```

copy and resize.

Parameters

s	input segment to copy.
b	new start date (in seconds)
e	new end date (in seconds)

Warning

copy only the events inside the new bounds.

13.4.2.8 link()

```
void InputSegment::link (
            long i,
            long j ) [protected]
```

the event of index i is linked to the event of index j.

Parameters

j	must be a valid index,
i	must be a valid and not NULL index,

Warning

both i and j can be in heap (negative index).
the point at i must not be linked (NULL link index).
the realtime date of i must be <= realtime date of j (if not NULL).

13.4.2.9 add_back()

```
long InputSegment::add_back (
            MusEvent * e,
            double rdate,
            double rdur,
            bool on,
            long link = MUSPOINTREF_NULL,
            Rational mdate = MUSTIME_UNKNOWN,
            Rational mduration = MUSTIME_UNKNOWN )
```

insert new timestamped muspoint created from the parameters, at the end of the segment.

Warning

The realtime dateof the point must be after the current last point of this segment.

Returns

the index of the inserted point (can be used as link).

13.4.2.10 add_floating()

```
long InputSegment::add_floating (
            MusEvent * e,
            double rdate,
            double rdur,
            bool on,
            long link = MUSPOINTREF_NULL,
            Rational mdate = MUSTIME_UNKNOWN,
            Rational mduration = MUSTIME_UNKNOWN )
```

create new timestamped muspoint from the parameters, and add the the heap of floating points (not in segment).

(allocated and freed by this segment)

Returns

the index of the new point (can be used as link).

13.4.2.11 point()

```
const MusPoint & InputSegment::point (
            long i ) const
```

return a ref to the point of index i.

- ith point in this input segment if 0 <= i < input segment size
- or the -i-1th floating point if heap size <= i < 0.

Parameters

i	must be in the above range of values.

13.4.2.12 respell() [1/2]

```
void InputSegment::respell (
            int k = 0 )
```

pitch spelling. unwindowed.

Warning

this segment must have been quantized.

13.4.2.13 respell() [2/2]

```
void InputSegment::respell (
            Rational ws,
            int k = 0 )
```

pitch spelling with a sliding window of given musical duration.

Warning

this segment must have been quantized.

13.4.2.14 quantize()

```
template<class P >
void InputSegment::quantize (
            Atable< P > * table,
            const P & p )
```

set the musical time date and duration of events in this given input segment, according to the best run for p in given table.

Warning

ptr type P must have interval.
all the musical dates and durations of events in this segment will be changed.

13.4.2.15 quantizu()

```
template<class P >
size_t InputSegment::quantizu (
            Atable< P > * table,
            const P & p,
            size_t b = 0 )
```

set the musical time date and duration of events in this given input segment, according to the best run for p in given table, starting from point number b in interval.

Returns

the next point of input segment with musical date and duration yet unset after processing p.

Warning

ptr type P must have interval.
all the musical date of events must be unknown in seg.
all the musical durations of events must be unknown in seg.

Todo TBR (replaced by quantize)

Todo TBR

13.4.2.16 InputSegmentMono()

```
InputSegmentMono::InputSegmentMono (
            const InputSegment & s )
```

transform the given input segment into a monophonic input segment (no two notes in the same time).

by moving note-off events

13.4.2.17 InputSegmentNogap()

```
InputSegmentNogap::InputSegmentNogap (
            const InputSegment & s,
            bool norest = true )
```

transform the given input segment into a new input segment without gaps.

by prolongations of some notes (option norest = true) or insertion of rests events (option norest = false)

13.4.2.18 Interval() [1/2]

```
Interval::Interval (
            const InputSegment * s,
            Rational mend = Rational(1) )
```

top interval constructed from an input segment.

Interval covering the whole length of the given input segment s with given musical time length (number of bars)

13.4.2.19 Interval() [2/2]

```
Interval::Interval (
            const InputSegment * s,
            Rational mbeg,
            Rational mend,
            double rbeg,
            double rend )  [protected]
```

build an interval with musical-time bounds [mbegin, mbegin+mdur[and real-time bounds [rbegin, rbegin+rdur[for the input segment s.

Warning

not aligned.

13.4.2.20 make() [1/2]

```
IntervalTree *const IntervalHeap::make (
            const InputSegment * s,
            Rational mend,
            double rext = 0 )
```

find or create (and push) a top interval of real-time duration covering the whole length of the given input segment s (root of interval tree) + the given extension.

- inside-bar interval (musical time duration of 1 bar) if flag bar is true
- multiple interval if flag bar is false (default).

Warning

not aligned.

13.4.2.21 make() [2/2]

```
IntervalTree *const IntervalHeap::make (
            const InputSegment * s,
            Rational mbeg,
            Rational mend,
            double rbeg,
            double rend,
            size_t first,
            IntervalTree * p,
            IntervalTree * ps )
```

get interval from heap, build it if not present.

Warning

not aligned (when built).

13.4.2.22 IntervalTree() [1/2]

```
IntervalTree::IntervalTree (
            const InputSegment * s,
            Rational mend = Rational(1) )  [protected]
```

top interval (root of interval tree).

covering the whole length of the given input segment s inside-bar interval of musical time duration of 1 bar if flag bar is true multi-bar interval if flag bar is false.

Warning

the interval tree created is not registered to an interval heap.
not aligned.

13.4.2.23 IntervalTree() [2/2]

```
IntervalTree::IntervalTree (
            const InputSegment * s,
            Rational mbeg,
            Rational mend,
            double rbeg,
            double rend,
            size_t first,
            IntervalTree * p = NULL,
            IntervalTree * ps = NULL )  [protected]
```

build an interval tree with musical-time bounds [mbegin, mbegin+mdur[and real-time bounds [rbegin, rbegin+rdur[for the input segment s.

Parameters

p	pointer to the parent.
ps	pointer to the previous sibling.

Warning

not aligned - must be aligned afterwards.
use only internaly construction of recursive paths.

13.4.2.24 top()

```
IntervalTree ∗ IntervalTree::top (
            const InputSegment ∗ s,
            IntervalHeap ∗ h,
            Rational mend = Rational(1) )
```

top interval (root of interval tree) covering the whole length of the given input segment s.

inside-bar interval of musical time duration of 1 bar if flag bar is true. multi-bar interval if flag bar is false.

13.4.2.25 split()

```
IntervalTree ∗ IntervalTree::split (
            const InputSegment ∗ s,
            IntervalHeap ∗ ih,
            double rdur,
            Rational mdur,
            size_t i )
```

return a sub interval.

- if i = 1 first sub-interval starting at same point as this interval of realtime duration rdur of musical duration mdur bar. it not is aligned.
- if i = 2 second sub-interval (rest) starting at this interval realtime start + rdur and this interval musical time start + mdur of realtime duration this realtime duration - rdur. if the real starting date is out of this interval, then the real duration of the returned second sub-interval is zero. the musical starting date must be inside this interval. it is not aligned.

Parameters

rdur	must be strictly positive.
mdur	must be strictly positive.

13.4.2.26 split_back()

```
IntervalTree ∗ IntervalTree::split_back (
             const InputSegment ∗ s,
             IntervalHeap ∗ ih,
             double rdur,
             Rational mdur,
             size_t i )
```

return a sub interval.

- if i = 1 first sub-interval starts at same point as this interval of realtime duration : duration of this interval - rdur of musical duration : musical duration of this interval - mdur bars. if the starting date is out of the input segment, then the real duration of the returned first sub-interval is zero. it is not aligned.
- if i = 2 second sub-interval (rest) starts at this interval realtime end - rdur and this interval musical time end - mdur of realtime duration rdur. it is not aligned.

Parameters

rdur	must be strictly positive.
mdur	must be strictly positive.

13.4.2.27 sub()

```
IntervalTree ∗ IntervalTree::sub (
             const InputSegment ∗ s,
             IntervalHeap ∗ ih,
             size_t a,
             size_t i )
```

return a the i-1th sub-interval of the division of this interval in n equal parts. the sub-interval returned is aligned.

Parameters

a	must be $>$ 1
i	must be smaller than a.

Warning

this interval must be aligned.

13.4.2.28 MusPoint()

```
MusPoint::MusPoint (
             const Point & p,
             Rational mdate = MUSTIME_UNKNOWN,
             Rational mduration = MUSTIME_UNKNOWN )
```

copy of point.

extended with given onset and duration values (in fraction of bars)

13.4.2.29 Pitch() [1/2]

```
Pitch::Pitch (
            char name,
            float alt = 0.0,
            int oct = 0 )
```

construct pitch from name+alteration+octave.

Parameters

name	see table NAMES in constant.h
alt	in [-2, 2] where 1.0 is half tone
oct	in -10..10

13.4.2.30 Pitch() [2/2]

```
Pitch::Pitch (
            unsigned int pitch,
            PitchUnit u = MIDI )
```

construct note from MIDI pitch

Parameters

pitch	in 0..127

13.4.2.31 Point()

```
Point::Point (
            const Point & p )
```

Warning

event (if any) is cloned.

13.4.2.32 ∼Point()

```
Point::∼Point ( )
```

Warning

event is deallocated and matcher (linked) also.

13.4.2.33 operator=()

```
Point & Point::operator= (
             const Point & p )  [virtual]
```

Warning

event (if any) is cloned.

13.4.2.34 isnormal()

```
bool SpiralPoint::isnormal ( ) const
```

Returns

wether coordinate are not NAN. ∗/

13.4.2.35 distance()

```
double SpiralPoint::distance (
             const SpiralPoint & rhs ) const
```

Returns

Euclidian distance to given point.

13.4.2.36 NoteName()

```
NoteName::NoteName (
             char n,
             float alt,
             int id )
```

notename object from name, alteration and index.

Parameters

n	must be between 'A' and 'G'
alt	must be between -2.0 and 2.0
id	must be between -15 and 19

13.4.2.37 closest()

```
const NoteName & NoteName::closest (
            unsigned int pitch,
            const SpiralPoint & p )  [static]
```

note name (ref in table synonyms) corresponding to given midi pitch and closest to given point.

Parameters

p	point in spiral
pitch	must be in 0..128

13.4.3 Variable Documentation

13.4.3.1 synonyms

```
const NoteName NoteName::synonyms  [static]
```

Initial value:

```
=
{
    { NoteName('B',  1.0,  12), NoteName('C',  0.0,   0),      NoteName('D', -2.0, -12) },
    { NoteName('C',  1.0,   7), NoteName('D', -1.0,  -5),      NoteName('B',  2.0,  19) },
    { NoteName('C',  2.0,  14), NoteName('D',  0.0,   2),      NoteName('E', -2.0, -10) },
    { NoteName('D',  1.0,   9), NoteName('E', -1.0,  -3),      NoteName('F', -2.0, -15) },
    { NoteName('D',  2.0,  16), NoteName('E',  0.0,   4),      NoteName('F', -1.0,  -8) },
    { NoteName('E',  1.0,  11), NoteName('F',  0.0,  -1),      NoteName('G',  2.0,  15) },
    { NoteName('E',  2.0,  18), NoteName('F',  1.0,   6),      NoteName('G', -1.0,  -6) },
    { NoteName('F',  2.0,  13), NoteName('G',  0.0,   1),      NoteName('A', -2.0, -11) },
    { NoteName('G',  1.0,   8), NoteName('A', -1.0,  -4),      NoteName() },
    { NoteName('G',  2.0,  15), NoteName('A',  0.0,   3),      NoteName('B', -2.0,  -9) },
    { NoteName('A',  1.0,  10), NoteName('B', -1.0,  -2),      NoteName('C',  2.0,  14) },
    { NoteName('A',  2.0,  17), NoteName('B',  0.0,   5),      NoteName('C', -1.0,  -7) }
}
```

13.5 Table module

The `table` module contains classes for parse tables and their content.

Classes

- class Parser< P >
- class Atable< P >

  abstract interface to parse table
- class Brecord< P >

  record associated to Ptr for one-best procedures.
- class Krecord< P >

  record associated to Ptr for k-best procedures.
- class Pointer

  abstract class defining a signature for a class of pointer to best runs.
- class Spointer

  key in a parse table.
- struct SpointerHasher
- class SIpointer
- struct SIpointerHasher

  hash function for using as key in a table. rank is ignoreds : same as SpointerHasher
- class SIPpointer

  key in a parse table. pointer to a (best) run for 1-best parsing for WTA and input segment.
- struct SIPpointerHasher

  hash function for using as key in a table rank is ignoreds : same as SpointerHasher
- class SKpointer

  pointer to a (best) run. for k-best parsing with standard WTA a SKpointer contains
- struct SKpointerHasher

  hash function for using as key in a table rank is ignoreds : same as SpointerHasher
- class SKIPpointer
- struct SKIPpointerHasher

  hash function for using as key in a table.
- class Record< P >

  abstract class describing the basic functionalities of a record.
- class Run< P >

  a run is a compact representation of parse trees as a tuple of pointers to subruns.
- class Table< P, R, H >

  parse table.

Macros

- #define **PTR_LPAR** '('
- #define **PTR_RPAR** ')'

Typedefs

- template<class P >
  using **RunCompare** = std::function< bool(const Run< P > ∗, const Run< P > ∗)>
- template<class P , class R , class H >
  using **MapRecord** = std::unordered_map< P, R, H >
- template<class P , class H >
  using **MapInstances** = std::unordered_multimap< P, P, H >

Functions

- std::ostream & **operator**<< (std::ostream &o, const Spointer &p)
- std::ostream & **operator**<< (std::ostream &o, const SIpointer &p)
- std::ostream & **operator**<< (std::ostream &o, const SIPpointer &p)
- std::ostream & **operator**<< (std::ostream &o, const SKpointer &p)
- std::ostream & **operator**<< (std::ostream &o, const SKIPpointer &p)
- virtual Weight Pointer::terminalWeight (const InputSegment ∗, const Transition &) const

  return the weight for a terminal Run associated to the given Transition. The transition must be terminal. This pointer must be compatible with the Transition. input segment can be NULL.
- virtual Weight Pointer::innerWeight (const Transition &) const

  return the initial weight for an inner Run associated to the given Transition. the weight will have to be multiplied with all the weights of subruns. the transition must be inner. this pointer must be divisible.
- Spointer::Spointer ()

  specific
- Spointer::Spointer (label_t)

  specific
- Spointer::Spointer (WTA ∗a, Environment ∗env, pre_t pre=0, pre_t post=0, Rational mlen=Rational(1), size_t k=1)

  top ptr (head of the main Run).
- Spointer::Spointer (Environment ∗env, const Spointer &p, size_t a, size_t i, state_t s)

  sub-pointer or instance as leaf.
- Spointer::Spointer (const Spointer &)

  copy.
- Spointer::Spointer (const Spointer &p0, const Spointer &p1)

  next sibling.
- Spointer::Spointer (const Spointer &p, const Spointer &p0, const Spointer &p1)

  instance as parent.
- virtual Spointer & Spointer::operator= (const Spointer &)
- virtual bool Spointer::operator== (const Spointer &) const

  for use as key in a unorered_multimap.
- virtual bool Spointer::operator< (const Spointer &) const

  for use as key in a multimap.
- virtual bool Spointer::instance (const Spointer &p) const
- virtual bool Spointer::subsume (const Spointer &p) const
- virtual bool Spointer::complete () const

  the pointer is complete i.e. all fields are set
- virtual bool Spointer::dummy () const

  return whether this pointer is a dummy pointer i.e. it was constructed with P() default false.
- virtual label_t Spointer::label (const Transition &t) const

  return a concrete label value corresponding to this pointer when considered as a leaf position, using the label of the given transition. the given transition must be terminal.
- virtual bool Spointer::divisible () const
- SIpointer::SIpointer ()

  dummy ptr
- SIpointer::SIpointer (label_t)

  fake ptr for terminal run, contains only a label symbol it is considered as complete see description in Ptr.hpp
- SIpointer::SIpointer (Environment ∗env, state_t s, Rational mdur=Rational(1), double rext=0)

  class specific top ptr (covering the whole input segment + given extension in realtime, of given musical duration.
- SIpointer::SIpointer (Environment ∗, const SIpointer &p, double rdur, Rational mdur, bool position, size_t i, state_t s)

  split ptr p in 2 parts.

- SIpointer::SIpointer (Environment ∗, const SIpointer &p, size_t a, size_t i, state_t s)

  sub-pointer or instance as leaf.
- SIpointer::SIpointer (const SIpointer &)

  copy.
- SIpointer::SIpointer (const SIpointer &p, const SIpointer &p0, const SIpointer &p1)

  instance as parent.
- SIpointer::SIpointer (const SIpointer &p0, const SIpointer &p1)

  instance as next sibling.
- virtual SIpointer & SIpointer::operator= (const SIpointer &)
- bool **SIpointer::equal_node** (const SIpointer &) const
- virtual bool SIpointer::operator== (const SIpointer &) const

  for use as key in a unordered_multimap.
- virtual bool **SIpointer::operator!=** (const SIpointer &) const
- virtual bool SIpointer::operator< (const SIpointer &) const

  for use as key in a multimap.
- virtual bool SIpointer::instance (const SIpointer &p) const
- virtual bool SIpointer::subsume (const SIpointer &p) const
- virtual bool SIpointer::complete () const
- virtual label_t SIpointer::label (const Transition &t) const
- virtual bool SIpointer::divisible () const
- virtual bool SIpointer::compatible (const label_t, bool abstract=true) const
- virtual bool SIpointer::dummy () const
- virtual Weight SIpointer::terminalWeight (const InputSegment ∗, const Transition &) const
- SIPpointer::SIPpointer (pre_t pre=PP_UNKNOWN, pre_t post=PP_UNKNOWN)

  dummy ptr.
- SIPpointer::SIPpointer (label_t)

  fake ptr for terminal run, contains only a label symbol. it is considered as complete
- SIPpointer::SIPpointer (Environment ∗env, state_t s, pre_t pre=0, pre_t post=0, Rational mdur=Rational(1), double rext=0)

  class specific top ptr (covering the whole input segment
- SIPpointer::SIPpointer (Environment ∗, const SIPpointer &p, double rdur, Rational mdur, bool position, size↩_t i, state_t s)

  split ptr p in 2 parts.
- SIPpointer::SIPpointer (Environment ∗, const SIPpointer &p, size_t a, size_t i, state_t s)

  sub-pointer or instance as leaf.
- SIPpointer::SIPpointer (const SIPpointer &)

  copy.
- SIPpointer::SIPpointer (const SIPpointer &p, const SIPpointer &p0, const SIPpointer &p1)

  instance as parent.
- SIPpointer::SIPpointer (const SIPpointer &p0, const SIPpointer &p1)

  instance as next sibling.
- virtual SIPpointer & SIPpointer::operator= (const SIPpointer &)
- virtual bool SIPpointer::operator== (const SIPpointer &) const

  for use as key in a unordered_multimap.
- virtual bool **SIPpointer::operator!=** (const SIPpointer &) const
- virtual bool SIPpointer::operator< (const SIPpointer &) const

  for use as key in a multimap.
- virtual bool SIPpointer::instance (const SIPpointer &p) const
- virtual bool SIPpointer::subsume (const SIPpointer &p) const
- virtual bool SIPpointer::complete () const
- label_t SIPpointer::label (const Transition &t) const
- virtual bool SIPpointer::compatible (const label_t, bool abstract=true) const

- virtual bool SIPpointer::dummy () const
- virtual Weight SIPpointer::terminalWeight (const InputSegment ∗s, const Transition &t) const
- SKpointer::SKpointer ()

  specific
- SKpointer::SKpointer (label_t, size_t k=1)

  specific
- SKpointer::SKpointer (WTA ∗a, Environment ∗env, pre_t pre=0, pre_t post=0, Rational mlen=Rational(1), size_t k=1)

  top ptr.
- SKpointer::SKpointer (Environment ∗, const SKpointer &p, size_t a, size_t i, state_t s)

  sub-pointer or instance as leaf.
- SKpointer::SKpointer (const SKpointer &)

  copy.
- SKpointer::SKpointer (const SKpointer &p0, const SKpointer &p1)

  next sibling.
- SKpointer::SKpointer (const SKpointer &p, const SKpointer &p0, const SKpointer &p1)

  instance as parent.
- virtual SKpointer & SKpointer::operator= (const SKpointer &)
- virtual bool SKpointer::operator== (const SKpointer &) const
- virtual bool SKpointer::instance (const SKpointer &p) const
- virtual bool SKpointer::subsume (const SKpointer &p) const
- virtual void **SKpointer::incr** ()
- SKIPpointer::SKIPpointer ()

  dummy ptr.
- SKIPpointer::SKIPpointer (label_t, size_t k=1)

  specific fake ptr for terminal run, contains only a label symbol. it is considered as complete
- SKIPpointer::SKIPpointer (Environment ∗env, pre_t pre=0, pre_t post=0, bool bar=false, size_t k=1)
- SKIPpointer::SKIPpointer (Environment ∗env, state_t s, pre_t pre=0, pre_t post=0, Rational mdur=Rational(1), size_t k=1)

  class specific top ptr (covering the whole input segment.
- SKIPpointer::SKIPpointer (Environment ∗env, const SKIPpointer &p, size_t a, size_t i, state_t s)

  sub-pointer or instance as leaf.
- SKIPpointer::SKIPpointer (const SKIPpointer &)

  copy.
- SKIPpointer::SKIPpointer (const SKIPpointer &p0, const SKIPpointer &p1)

  next sibling.
- SKIPpointer::SKIPpointer (const SKIPpointer &p, const SKIPpointer &p0, const SKIPpointer &p1)

  instance as parent.
- virtual SKIPpointer & SKIPpointer::operator= (const SKIPpointer &)
- virtual bool SKIPpointer::operator== (const SKIPpointer &) const
- virtual bool SKIPpointer::instance (const SKIPpointer &p) const
- virtual bool SKIPpointer::subsume (const SKIPpointer &p) const
- virtual void **SKIPpointer::incr** ()

Variables

- template<class P >
  RunCompare< P > weightMin

  one ordering for k-best to select the min weight Run where partial run is considered to be the lowest.
- template<class P >
  RunCompare< P > weightMax

  one ordering for k-best to select the max weight run where partial run is considered to be the highest

13.5.1 Detailed Description

The `table` module contains classes for parse tables and their content.

13.5.2 Function Documentation

13.5.2.1 Spointer() [1/4]

```
Spointer::Spointer (
            WTA * a,
            Environment * env,
            pre_t pre = 0,
            pre_t post = 0,
            Rational mlen = Rational(1),
            size_t k = 1 )
```

top ptr (head of the main Run).

See also

description in Ptr.hpp

Parameters

bar	must be true
k	must be 1

Todo TBR deprecated (replace by specific constructor)

13.5.2.2 Spointer() [2/4]

```
Spointer::Spointer (
            Environment * env,
            const Spointer & p,
            size_t a,
            size_t i,
            state_t s )
```

sub-pointer or instance as leaf.

See also

description in Ptr.hpp

Parameters

p	must have a wta state
a	
i	if a>0 and i=0, construct a copy of p. if a>0 and 0<i<=a, construct a copy a ptr with state s.

13.5.2.3 Spointer() [3/4]

```
Spointer::Spointer (
            const Spointer & p0,
            const Spointer & p1 )
```

next sibling.

See also

description in Ptr.hpp

Warning

should not be called since p1 must be partial.

13.5.2.4 Spointer() [4/4]

```
Spointer::Spointer (
            const Spointer & p,
            const Spointer & p0,
            const Spointer & p1 )
```

instance as parent.

See also

description in Ptr.hpp

Warning

should not be called since p must be partial.

13.5.2.5 operator=() [1/5]

```
Spointer & Spointer::operator= (
             const Spointer & p )  [virtual]
```

See also

description in Ptr.hpp

13.5.2.6 operator==() [1/5]

```
bool Spointer::operator== (
             const Spointer & p ) const  [virtual]
```

for use as key in a unorered_multimap.

See also

description in Ptr.hpp

13.5.2.7 operator<() [1/3]

```
bool Spointer::operator< (
             const Spointer & p ) const  [virtual]
```

for use as key in a multimap.

See also

description in Ptr.hpp

13.5.2.8 instance() [1/5]

```
bool Spointer::instance (
             const Spointer & p ) const  [virtual]
```

See also

description in Ptr.hpp

13.5.2.9 subsume() [1/5]

```
bool Spointer::subsume (
            const Spointer & p ) const  [virtual]
```

See also

description in Ptr.hpp

13.5.2.10 divisible() [1/2]

```
bool Spointer::divisible ( ) const  [virtual]
```

Warning

this pointer must have a WTA state always return true in that case

Reimplemented from Pointer.

Reimplemented in SIpointer.

13.5.2.11 SIpointer() [1/6]

```
SIpointer::SIpointer ( )
```

dummy ptr

See also

description in Ptr.hpp

13.5.2.12 SIpointer() [2/6]

```
SIpointer::SIpointer (
            Environment * env,
            state_t s,
            Rational mdur = Rational(1),
            double rext = 0 )
```

class specific top ptr (covering the whole input segment + given extension in realtime, of given musical duration.

Parameters

env	must contain an input segment and interval heap.

13.5.2.13 SIpointer() [3/6]

```
SIpointer::SIpointer (
            Environment * env,
            const SIpointer & p,
            double rdur,
            Rational mdur,
            bool position,
            size_t i,
            state_t s )
```

split ptr p in 2 parts.

if position = 0, first part has (real-time/musical-time) durations rdur/mdur

if position = 1, second part has (real-time/musical-time) durations rdur/mdur construct part number i (1 or 2)

Parameters

env	must contain an input segment and interval heap
rdur	must be strictly positive.
mdur	must be strictly positive.
i	must be 1 or 2.
s	can be WTA state or Meta state.

13.5.2.14 SIpointer() [4/6]

```
SIpointer::SIpointer (
            Environment * env,
            const SIpointer & p,
            size_t a,
            size_t i,
            state_t s )
```

sub-pointer or instance as leaf.

See also

description in Ptr.hpp

13.5.2.15 SIpointer() [5/6]

```
SIpointer::SIpointer (
             const SIpointer & p,
             const SIpointer & p0,
             const SIpointer & p1 )
```

instance as parent.

See also

description in Ptr.hpp

13.5.2.16 SIpointer() [6/6]

```
SIpointer::SIpointer (
             const SIpointer & p0,
             const SIpointer & p1 )
```

instance as next sibling.

See also

description in Ptr.hpp

13.5.2.17 operator=() [2/5]

```
SIpointer & SIpointer::operator= (
             const SIpointer & p )  [virtual]
```

See also

description in Ptr.hpp

13.5.2.18 operator==() [2/5]

```
bool SIpointer::operator== (
             const SIpointer & p ) const  [virtual]
```

for use as key in a unordered_multimap.

See also

description in Ptr.hpp

13.5.2.19 operator<() [2/3]

```
bool SIpointer::operator< (
             const SIpointer & p ) const  [virtual]
```

for use as key in a multimap.

See also

description in Ptr.hpp

13.5.2.20 instance() [2/5]

```
bool SIpointer::instance (
             const SIpointer & p ) const  [virtual]
```

See also

description in Ptr.hpp

13.5.2.21 subsume() [2/5]

```
bool SIpointer::subsume (
             const SIpointer & p ) const  [virtual]
```

See also

description in Ptr.hpp

13.5.2.22 complete() [1/2]

```
bool SIpointer::complete ( ) const  [virtual]
```

See also

description in Ptr.hpp

Reimplemented from Spointer.

Reimplemented in SIPpointer.

13.5.2.23 label() [1/2]

```
label_t SIpointer::label (
             const Transition & t ) const  [virtual]
```

See also

description in Ptr.hpp the _pre value must be known _node must be set

Reimplemented from Spointer.

Reimplemented in SIPpointer.

13.5.2.24 divisible() [2/2]

```
bool SIpointer::divisible ( ) const  [virtual]
```

See also

description in Ptr.hpp

if this pointer has a WTA state: it is not worth descending when this pointer corresponds to an input sub-segment not inhabited.

if this pointer has a Meta state: it is not worth descending when this ptr corresponds to an empty segment.

Reimplemented from Spointer.

13.5.2.25 compatible() [1/2]

```
bool SIpointer::compatible (
             const label_t label,
             bool abstract = true ) const  [virtual]
```

See also

description in Ptr.hpp

Reimplemented from Pointer.

Reimplemented in SIPpointer.

13.5.2.26 dummy() [1/2]

```
bool SIpointer::dummy ( ) const  [virtual]
```

See also

description in Ptr.hpp

Reimplemented from Spointer.

Reimplemented in SIPpointer.

13.5.2.27 terminalWeight() [1/2]

```
Weight SIpointer::terminalWeight (
             const InputSegment * s,
             const Transition & tr ) const  [virtual]
```

See also

description in Ptr.hpp

Warning

input segment must not be NULL.

Reimplemented from Pointer.

Reimplemented in SIPpointer.

13.5.2.28 SIPpointer() [1/7]

```
SIPpointer::SIPpointer (
             pre_t pre = PP_UNKNOWN,
             pre_t post = PP_UNKNOWN )
```

dummy ptr.

See also

description in Ptr.hpp

13.5.2.29 SIPpointer() [2/7]

```
SIPpointer::SIPpointer (
             label_t s )
```

fake ptr for terminal run, contains only a label symbol. it is considered as complete

See also

description in Ptr.hpp

13.5.2.30 SIPpointer() [3/7]

```
SIPpointer::SIPpointer (
             Environment * env,
             state_t s,
             pre_t pre = 0,
             pre_t post = 0,
             Rational mdur = Rational(1),
             double rext = 0 )
```

class specific top ptr (covering the whole input segment

- given extension in realtime.

Parameters

env	must contain an input segment and interval heap

13.5.2.31 SIPpointer() [4/7]

```
SIPpointer::SIPpointer (
             Environment * env,
             const SIPpointer & p,
             double rdur,
             Rational mdur,
             bool position,
             size_t i,
             state_t s )
```

split ptr p in 2 parts.

if position = 0, first part has (real-time/musical-time) durations rdur/mdur

if position = 1, second part has (real-time/musical-time) durations rdur/mdur

construct part number i (1 or 2)

Parameters

env	must contain an input segment and interval heap
rdur	must be strictly positive.
mdur	must be strictly positive.
i	must be 1 or 2.
s	(state) can be WTA or Meta.

13.5.2.32 SIPpointer() [5/7]

```
SIPpointer::SIPpointer (
            Environment * env,
            const SIPpointer & p,
            size_t a,
            size_t i,
            state_t s )
```

sub-pointer or instance as leaf.

See also

description in Ptr.hpp

13.5.2.33 SIPpointer() [6/7]

```
SIPpointer::SIPpointer (
            const SIPpointer & p,
            const SIPpointer & p0,
            const SIPpointer & p1 )
```

instance as parent.

See also

description in Ptr.hpp

13.5.2.34 SIPpointer() [7/7]

```
SIPpointer::SIPpointer (
            const SIPpointer & p0,
            const SIPpointer & p1 )
```

instance as next sibling.

See also

description in Ptr.hpp

13.5.2.35 operator=() [3/5]

```
SIPpointer & SIPpointer::operator= (
             const SIPpointer & p )  [virtual]
```

See also

description in Ptr.hpp

13.5.2.36 operator==() [3/5]

```
bool SIPpointer::operator== (
             const SIPpointer & p ) const  [virtual]
```

for use as key in a unordered_multimap.

See also

description in Ptr.hpp

13.5.2.37 operator<() [3/3]

```
bool SIPpointer::operator< (
             const SIPpointer & p ) const  [virtual]
```

for use as key in a multimap.

See also

description in Ptr.hpp

13.5.2.38 instance() [3/5]

```
bool SIPpointer::instance (
             const SIPpointer & p ) const  [virtual]
```

See also

description in Ptr.hpp

13.5.2.39 subsume() [3/5]

```
bool SIPpointer::subsume (
            const SIPpointer & p ) const  [virtual]
```

See also

description in Ptr.hpp

13.5.2.40 complete() [2/2]

```
bool SIPpointer::complete ( ) const  [virtual]
```

See also

description in Ptr.hpp

Reimplemented from SIpointer.

13.5.2.41 label() [2/2]

```
label_t SIPpointer::label (
            const Transition & t ) const  [virtual]
```

See also

description in Ptr.hpp

Warning

the _pre value must be known
_node must be set

Reimplemented from SIpointer.

13.5.2.42 compatible() [2/2]

```
bool SIPpointer::compatible (
            const label_t label,
            bool abstract = true ) const  [virtual]
```

See also

description in Ptr.hpp

Reimplemented from SIpointer.

13.5.2.43 dummy() [2/2]

```
bool SIPpointer::dummy ( ) const  [virtual]
```

See also

description in Ptr.hpp

Reimplemented from SIpointer.

13.5.2.44 terminalWeight() [2/2]

```
Weight SIPpointer::terminalWeight (
             const InputSegment * s,
             const Transition & t ) const  [virtual]
```

See also

description in Ptr.hpp

Parameters

s	input segment must not be NULL.

Reimplemented from SIpointer.

13.5.2.45 SKpointer() [1/4]

```
SKpointer::SKpointer (
             WTA * a,
             Environment * env,
             pre_t pre = 0,
             pre_t post = 0,
             Rational mlen = Rational(1),
             size_t k = 1 )
```

top ptr.

See also

description in Ptr.hpp

Parameters

bar	must be true

Todo TBR deprecated (replace by specific constructor)

13.5.2.46 SKpointer() [2/4]

```
SKpointer::SKpointer (
            Environment * env,
            const SKpointer & p,
            size_t a,
            size_t i,
            state_t s )
```

sub-pointer or instance as leaf.

See also

description in Ptr.hpp

Warning

no default duration for ambiguity reasons.

13.5.2.47 SKpointer() [3/4]

```
SKpointer::SKpointer (
            const SKpointer & p0,
            const SKpointer & p1 )
```

next sibling.

See also

description in Ptr.hpp

Warning

should not be called since p1 must be partial

13.5.2.48 SKpointer() [4/4]

```
SKpointer::SKpointer (
             const SKpointer & p,
             const SKpointer & p0,
             const SKpointer & p1 )
```

instance as parent.

See also

description in Ptr.hpp

Warning

should not be called since p must be partial

13.5.2.49 operator=() [4/5]

```
SKpointer & SKpointer::operator= (
             const SKpointer & p )  [virtual]
```

See also

description in Ptr.hpp

13.5.2.50 operator==() [4/5]

```
bool SKpointer::operator== (
             const SKpointer & p ) const  [virtual]
```

See also

description in Ptr.hpp

13.5.2.51 instance() [4/5]

```
bool SKpointer::instance (
             const SKpointer & p ) const  [virtual]
```

See also

description in Ptr.hpp

13.5.2.52 subsume() [4/5]

```
bool SKpointer::subsume (
             const SKpointer & p ) const  [virtual]
```

See also

description in Ptr.hpp

13.5.2.53 SKIPpointer() [1/7]

```
SKIPpointer::SKIPpointer ( )
```

dummy ptr.

See also

description in Ptr.hpp

13.5.2.54 SKIPpointer() [2/7]

```
SKIPpointer::SKIPpointer (
             label_t s,
             size_t k = 1 )
```

specific fake ptr for terminal run, contains only a label symbol. it is considered as complete

See also

description in Ptr.hpp

13.5.2.55 SKIPpointer() [3/7]

```
SKIPpointer::SKIPpointer (
             Environment * env,
             pre_t pre = 0,
             pre_t post = 0,
             bool bar = false,
             size_t k = 1 )
```

Todo TBR deprecated

13.5.2.56 SKIPpointer() [4/7]

```
SKIPpointer::SKIPpointer (
             Environment * env,
             state_t s,
             pre_t pre = 0,
             pre_t post = 0,
             Rational mdur = Rational(1),
             size_t k = 1 )
```

class specific top ptr (covering the whole input segment.

Warning

env must contain an input segment and interval heap.

13.5.2.57 SKIPpointer() [5/7]

```
SKIPpointer::SKIPpointer (
             Environment * env,
             const SKIPpointer & p,
             size_t a,
             size_t i,
             state_t s )
```

sub-pointer or instance as leaf.

See also

description in Ptr.hpp

Warning

no default duration for ambiguity reasons

13.5.2.58 SKIPpointer() [6/7]

```
SKIPpointer::SKIPpointer (
             const SKIPpointer & p0,
             const SKIPpointer & p1 )
```

next sibling.

See also

description in Ptr.hpp

Warning

should not be called since p1 must be partial

13.5.2.59 SKIPpointer() [7/7]

```
SKIPpointer::SKIPpointer (
             const SKIPpointer & p,
             const SKIPpointer & p0,
             const SKIPpointer & p1 )
```

instance as parent.

See also

description in Ptr.hpp

Warning

should not be called since p must be partial

13.5.2.60 operator=() [5/5]

```
SKIPpointer & SKIPpointer::operator= (
             const SKIPpointer & p )  [virtual]
```

See also

description in Ptr.hpp

13.5.2.61 operator==() [5/5]

```
bool SKIPpointer::operator== (
             const SKIPpointer & p ) const  [virtual]
```

See also

description in Ptr.hpp

13.5.2.62 instance() [5/5]

```
bool SKIPpointer::instance (
             const SKIPpointer & p ) const  [virtual]
```

See also

description in Ptr.hpp

13.5.2.63 subsume() [5/5]

```
bool SKIPpointer::subsume (
             const SKIPpointer & p ) const  [virtual]
```

See also

description in Ptr.hpp

13.5.3 Variable Documentation

13.5.3.1 weightMin

```
template<class P >
RunCompare<P> weightMin
```

Initial value:

```
=
[](const Run<P>* lhs, const Run<P>* rhs)
{
    assert (lhs);
    assert (rhs);
    if (rhs->partial())
    {
        return false;
    }
    else
    {
        if (lhs->partial()) return true;
        return (lhs->weight > rhs->weight);
    }
}
```

one ordering for k-best to select the min weight Run where partial run is considered to be the lowest.

13.5.3.2 weightMax

```
template<class P >
RunCompare<P> weightMax
```

Initial value:

```
=
[](const Run<P>* lhs, const Run<P>* rhs)
{
    assert (lhs);
    assert (rhs);
    if (lhs->partial())
    {
        return false;


    }
    else
    {
        if (rhs->partial()) return true;
        else return (lhs->weight < rhs->weight);
    }
}
```

one ordering for k-best to select the max weight run where partial run is considered to be the highest

13.6 General module

The general module contains reusable tools and utilities, initialization of constants, and tracing functions.

Namespaces

- patch

  trace levels:

Classes

- class Rational

  class of rational numbers
- class std::hash< Rational >

Macros

- #define **PP_UNKNOWN** -1
- #define **PP_KNOWN**(x) (x >= 0)
- #define **TRACE_ON**
- #define **DEBUG_ON**
- #define _TRACE_CAND 1

  addition of candidates
- #define _TRACE_BEST

  addition of best runs
- #define _TRACE_TBL

  initialization and construction of tables
- #define **ERROR**(...) console->error(__VA_ARGS__)
- #define **WARN**(...) console->warn(__VA_ARGS__)
- #define **INFO**(...) console->info(__VA_ARGS__)
- #define **TRACE**(...) console->trace(__VA_ARGS__)
- #define **DEBUG**(...) console->debug(__VA_ARGS__)
- #define **TRACE_CAND**(...) TRACE(__VA_ARGS__)
- #define **TRACE_BEST**(...) TRACE(__VA_ARGS__)
- #define **TRACE_TBL**(...) TRACE(__VA_ARGS__)

Typedefs

- typedef long pre_t

  type for pre post values in Runs

Enumerations

- enum WeightDom { **UNDEF**, WeightDom::PENALTY, WeightDom::STOCHASTIC, WeightDom::COUNTING }

  weight types

Functions

- double **duration** (clock_t start)
- int read_config (const std::string filename)

  read the constant and optimisation flag values in a config file INI file, see https://en.wikipedia.↵org/wiki/INI_file return 0 if reading the values succeded -1 in case of file open error or a number of line in case of parse error in .ini file.
- std::ostream & **operator**<< (std::ostream &o, const WeightDom &t)
- long virtual_memory_size ()

  Here we check that the compile flags are set and correct: QP_PLATFORM = PLATFORM_xxx QP_TARGET = T↵ARGET_xxx where the possibles values for PLATFORM_xxx (target platform) and TARGET_xxx (executable) are defined by compiler flags.
- long **resident_memory_size** ()
- const Rational **operator+** (const Rational &lhs, const Rational &rhs)
- const Rational **operator-** (const Rational &lhs, const Rational &rhs)
- const Rational **operator*** (const Rational &lhs, const Rational &rhs)
- const Rational **operator/** (const Rational &lhs, const Rational &rhs)
- Rational **rabs** (const Rational &r)
- bool **operator==** (const Rational &lhs, const Rational &rhs)
- bool **operator!=** (const Rational &lhs, const Rational &rhs)
- bool **operator**< (const Rational &lhs, const Rational &rhs)
- bool **operator**> (const Rational &lhs, const Rational &rhs)
- bool **operator**<**=** (const Rational &lhs, const Rational &rhs)
- bool **operator**>**=** (const Rational &lhs, const Rational &rhs)
- std::ostream & **operator**<< (std::ostream &ostr, const Rational &r)
- std::istream & **operator**>> (std::istream &istr, Rational &r)
- Rational toRational (double x, int iterations=5)

  double -> Rational conversion
- double toDouble (const Rational &r)

  Rational -> double conversion.
- long trunc (const Rational &r)

  Rational -> long conversions.
- long **floor** (const Rational &r)
- long **ceil** (const Rational &r)
- Rational::Rational (long n, long d=1)

  default constructor
- const Rational & **Rational::operator+=** (const Rational &rhs)
- const Rational & **Rational::operator+=** (long rhs)
- const Rational & **Rational::operator-=** (const Rational &rhs)
- const Rational & **Rational::operator-=** (long rhs)
- const Rational & **Rational::operator*=** (const Rational &rhs)
- const Rational & **Rational::operator*=** (long rhs)
- const Rational & **Rational::operator/=** (const Rational &rhs)
- const Rational & **Rational::operator/=** (long rhs)
- const Rational & **Rational::operator++** ()
- const Rational **Rational::operator++** (int)
- const Rational & **Rational::operator--** ()
- const Rational **Rational::operator--** (int)
- void Rational::printint (std::ostream &) const

  print in format int+rat
- Rational & Rational::operator= (const Rational &rhs)

  assignment operators
- Rational & **Rational::operator=** (long rhs)

Variables

- int EVENTNB_UNKNOWN = -1

  type for MIDI event numbers
- Rational MUSTIME_UNKNOWN = Rational(-1)

  type for musical time values
- long **MUSPOINTREF_NULL** = LONG_MIN
- WeightDom CST_WEIGHT_TYPE = WeightDom::UNDEF

  weight type. value specified in grammar file or default value WeightDom::UNDEF
- long CST_MAX_GRACE = 0

  max number of grace notes specified in grammar used for abstraction of terminal transition labels see Label.hpp value specified in grammar file or default 0 (there is no abstraction of labels)
- double CST_ALPHA = 0.5

  coefficient for combining weight and distance in pernalty weight model
- double CST_SIGMA2 = 0.5

  constant for computing performance weight see PerformanceModel.hpp
- long MAX_AR = 13

  symbols for labeling RT and WTA terminal transitions max arity
- double CST_PRECISION = 0.0000001

  precision for floting point unit calculations
- long HASH_SEED = 1009

  hash function parameters Bernstein hash `http://www.eternallyconfuzzled.com/tuts/algorithms/jsw_tut_hashing.aspx` *see also Josh Bloch, Effective Java see* `http://stackoverflow.com/a/1646913/126995` *and also* `http://stackoverflow.com/questions/17016175`
- long **HASH_FACTOR** = 9176
- bool OPT_RUN_DUR = true

  optimization flag compute the duration sequences of runs. if unset, OPT_RUN_UNIT must be automatically unset value specified in ini file or default: true
- bool OPT_RUN_STRICT = false

  optimization flag compute at most one best run for a duration sequence in each record. it is the first best run added, i.e. the best with that duration sequence. OPT_RUN_DUR must be set. value specified in ini file or default: false
- bool OPT_RUN_UNIT = true

  optimization flag do not add non-terminal runs with duration sequences of the form [0. . . 0, 1]. they correspond to reducible runs of the form p(x, _,. . . , _) where x is a leaf and _ is a continuation (tie). OPT_RUN_DUR must be set. value specified in ini file or default: true
- bool OPT_NOREST = false

  option flag ignore rests in MIDI input file where a rest is the duration between a note off and the next note on msg.
- int EVENTNB_UNKNOWN

  type for MIDI event numbers
- Rational MUSTIME_UNKNOWN

  type for musical time values
- long **MUSPOINTREF_NULL**
- WeightDom CST_WEIGHT_TYPE

  weight type. value specified in grammar file or default value WeightDom::UNDEF
- long CST_MAX_GRACE

  max number of grace notes specified in grammar used for abstraction of terminal transition labels see Label.hpp value specified in grammar file or default 0 (there is no abstraction of labels)
- double CST_ALPHA

  coefficient for combining weight and distance in pernalty weight model
- double CST_SIGMA2

  constant for computing performance weight see PerformanceModel.hpp
- long MAX_AR

  symbols for labeling RT and WTA terminal transitions max arity

- double CST_PRECISION

  precision for floting point unit calculations
- long HASH_SEED

  hash function parameters Bernstein hash `http://www.eternallyconfuzzled.com/tuts/algorithms/jsw_tut_hashing.aspx` *see also Josh Bloch, Effective Java see* `http://stackoverflow.com/a/1646913/126995` *and also* `http://stackoverflow.com/questions/17016175`
- long **HASH_FACTOR**
- bool OPT_RUN_STRICT

  optimization flag compute at most one best run for a duration sequence in each record. it is the first best run added, i.e. the best with that duration sequence. OPT_RUN_DUR must be set. value specified in ini file or default: false
- bool OPT_RUN_UNIT

  optimization flag do not add non-terminal runs with duration sequences of the form [0. . . 0, 1]. they correspond to reducible runs of the form p(x, _,. . . , _) where x is a leaf and _ is a continuation (tie). OPT_RUN_DUR must be set. value specified in ini file or default: true
- bool OPT_RUN_DUR

  optimization flag compute the duration sequences of runs. if unset, OPT_RUN_UNIT must be automatically unset value specified in ini file or default: true
- bool OPT_NOREST

  option flag ignore rests in MIDI input file where a rest is the duration between a note off and the next note on msg.
- const auto console = spd::stdout_color_mt("console")

  Console logger with color const std::shared_ptr<spd::logger> console = spd::stdout_color_mt("console");.
- const int TRACE_LEVEL = 2
- const std::shared_ptr< spd::logger > console

  Console logger with color const std::shared_ptr<spd::logger> console = spd::stdout_color_mt("console");.

13.6.1 Detailed Description

The `general` module contains reusable tools and utilities, initialization of constants, and tracing functions.

13.6.2 Enumeration Type Documentation

13.6.2.1 WeightDom

```
enum WeightDom  [strong]
```

weight types

Enumerator

PENALTY	to be specified
STOCHASTIC	tropical semiring
COUNTING	Viterbi semiring. int vectors for corpus stat

13.6.3 Function Documentation

13.6.3.1 virtual_memory_size()

```
long virtual_memory_size ( )
```

Here we check that the compile flags are set and correct: QP_PLATFORM = PLATFORM_xxx QP_TARGET = TARGET_xxx where the possibles values for PLATFORM_xxx (target platform) and TARGET_xxx (executable) are defined by compiler flags.

in Xcode, the flags are defined.

13.6.4 Variable Documentation

13.6.4.1 HASH_SEED [1/2]

```
long HASH_SEED = 1009
```

hash function parameters Bernstein hash http://www.eternallyconfuzzled.com/tuts/algorithms/jsw_tut_hashing.aspx see also Josh Bloch, Effective Java see http://stackoverflow.com/a/1646913/126995 and also http://stackoverflow.com/questions/17016175

see also https://stackoverflow.com/a/1646913/6930643 constexpr int HASH_SEED = 17; constexpr int HASH_FACTOR = 31; see also https://stackoverflow.com/a/34006336/6930643

13.6.4.2 HASH_SEED [2/2]

```
long HASH_SEED
```

hash function parameters Bernstein hash http://www.eternallyconfuzzled.com/tuts/algorithms/jsw_tut_hashing.aspx see also Josh Bloch, Effective Java see http://stackoverflow.com/a/1646913/126995 and also http://stackoverflow.com/questions/17016175

see also https://stackoverflow.com/a/1646913/6930643 constexpr int HASH_SEED = 17; constexpr int HASH_FACTOR = 31; see also https://stackoverflow.com/a/34006336/6930643

13.6.4.3 TRACE_LEVEL

```
const int TRACE_LEVEL = 2
```

Todo TBR

13.7 Weight module

The weight module contains the definitions of several domains for weight values for tree automata.

Classes

- class CountingWeight

  domain : vectors of fixed dim k > 0
- class Distance

  concrete Weight domain identical to TropicalWeight with an additional constructor to compute a distance value from an Alignement, obtained as the sum of the pointwise distances.
- class FloatWeight

  concrete Weight defined as a scalar value.
- class PerfoWeight

  extention of ViterbiWeight with a model of performance.
- class SemiRing< T >

  semiring structure.
- class TropicalWeight

  concrete Weight defined as a scalar value: non-negative weights.
- class ViterbiWeight

  Viterbi semifield. concrete Weight defined as a scalar value: probability of the best derivation.
- class LetterWeight

  abstract class for concrete weight values. Every concrete weight domain must be a derived class of LetterWeight.
- class Weight

  A class of polymorphic weight domains for tree series.

Functions

- std::ostream & **operator<<** (std::ostream &o, const CountingWeight &rhs)
- std::ostream & **operator<<** (std::ostream &o, const FloatWeight &rhs)
- bool operator== (const Weight &lhs, const Weight &rhs)
- bool **operator!=** (const Weight &lhs, const Weight &rhs)
- bool operator< (const Weight &lhs, const Weight &rhs)
- bool **operator>** (const Weight &lhs, const Weight &rhs)
- bool **operator<=** (const Weight &lhs, const Weight &rhs)
- bool **operator>=** (const Weight &lhs, const Weight &rhs)
- std::ostream & operator<< (std::ostream &o, const Weight &rhs)
- CountingWeight::CountingWeight (CWType t, size_t dim)
- **CountingWeight::CountingWeight** (const CountingWeight &)
- CountingWeight & **CountingWeight::operator=** (const CountingWeight &)
- CountingWeight & CountingWeight::operator= (const LetterWeight &rhs)
- CountingWeight ∗ **CountingWeight::clone** () const
- virtual Weight CountingWeight::make (double v) const
- static Weight **CountingWeight::make_one** (size_t)
- static Weight CountingWeight::make_unit (size_t dim, size_t i)
- virtual Weight CountingWeight::get_zero () const

  return the neutral element for add (absorbing element for mult) wrapped in a Weight.
- virtual Weight CountingWeight::get_one () const

  return the neutral element for mult wrapped in a Weight.
- virtual bool CountingWeight::zero () const

- bool CountingWeight::error () const
- virtual bool CountingWeight::one () const
- virtual double CountingWeight::norm () const
- virtual void CountingWeight::scalar (double)

  add to each component.
- virtual bool CountingWeight::equal (const LetterWeight ∗rhs) const
- virtual bool CountingWeight::smaller (const LetterWeight ∗rhs) const
- virtual void CountingWeight::add (const LetterWeight ∗rhs)
- virtual void CountingWeight::mult (const LetterWeight ∗rhs)
- virtual void **CountingWeight::print** (std::ostream &) const
- virtual void **CountingWeight::rawprint** (std::ostream &) const
- Distance::Distance (const InputSegment ∗s, const AlignedInterval ∗p)

  weight which is the distance defined by alignment for input segment not unknown.
- Distance & **Distance::operator=** (const Distance &)
- virtual Distance ∗ **Distance::clone** () const
- virtual void **Distance::print** (std::ostream &) const
- FloatWeight::FloatWeight (double d=0.0)

  defaut = null weight - not unknown
- **FloatWeight::FloatWeight** (const FloatWeight &)
- FloatWeight & **FloatWeight::operator=** (const FloatWeight &)
- FloatWeight & **FloatWeight::operator=** (const LetterWeight &)
- virtual FloatWeight ∗ **FloatWeight::clone** () const
- virtual void **FloatWeight::scalar** (double)
- virtual void FloatWeight::invert ()

  multiplicative inverse.
- virtual bool FloatWeight::zero () const

  this letterweight is neutral element for add (absorbing element for mult).
- virtual bool FloatWeight::one () const

  this letterweight is neutral element for mult.
- bool **FloatWeight::equal** (const FloatWeight &rhs) const
- virtual bool FloatWeight::equal (const LetterWeight ∗rhs) const
- bool **FloatWeight::smaller** (const FloatWeight &rhs) const
- virtual bool FloatWeight::smaller (const LetterWeight ∗rhs) const
- void **FloatWeight::add** (const FloatWeight &rhs)
- virtual void FloatWeight::add (const LetterWeight ∗rhs)
- void **FloatWeight::mult** (const FloatWeight &rhs)
- virtual void FloatWeight::mult (const LetterWeight ∗rhs)
- virtual void **FloatWeight::print** (std::ostream &) const
- PerfoWeight::PerfoWeight (const InputSegment ∗s, const AlignedInterval ∗p, pre_t pre=0, pre_t post=0)

  probability of positions in the given alignement in the interval defined by the given path.
- PerfoWeight & **PerfoWeight::operator=** (const PerfoWeight &)
- PerfoWeight & PerfoWeight::operator= (const LetterWeight &rhs)
- static void **PerfoWeight::set_sigma2** (double)
- **TropicalWeight::TropicalWeight** (const TropicalWeight &)
- TropicalWeight & **TropicalWeight::operator=** (const TropicalWeight &)
- TropicalWeight & TropicalWeight::operator= (const LetterWeight &)
- TropicalWeight ∗ **TropicalWeight::clone** () const
- virtual double TropicalWeight::norm () const
- virtual void **TropicalWeight::scalar** (double)
- virtual void TropicalWeight::invert ()

  multiplicative inverse.
- virtual bool TropicalWeight::zero () const

  this letterweight is neutral element for add (absorbing element for mult).

- virtual bool TropicalWeight::one () const

  this letterweight is neutral element for mult.
- virtual bool TropicalWeight::equal (const LetterWeight ∗rhs) const
- virtual bool TropicalWeight::smaller (const LetterWeight ∗rhs) const
- virtual void TropicalWeight::add (const LetterWeight ∗rhs)

  sum is min.
- virtual void TropicalWeight::mult (const LetterWeight ∗rhs)

  product is sum.
- virtual void **TropicalWeight::print** (std::ostream &) const
- static TropicalWeight TropicalWeight::inner (size_t)

  penalty for an inner node.
- static TropicalWeight TropicalWeight::tie ()

  penalty for a tie.
- static TropicalWeight TropicalWeight::gracenote (size_t)

  penalty for given number of grace notes in a leaf.
- ViterbiWeight::ViterbiWeight (double)

  default is one
- **ViterbiWeight::ViterbiWeight** (const ViterbiWeight &)
- ViterbiWeight & **ViterbiWeight::operator=** (const ViterbiWeight &)
- ViterbiWeight & ViterbiWeight::operator= (const LetterWeight &rvalue)
- virtual LetterWeight ∗ **ViterbiWeight::clone** () const
- virtual double **ViterbiWeight::norm** () const
- virtual void **ViterbiWeight::scalar** (double)
- virtual void ViterbiWeight::invert ()

  multiplicative inverse.
- virtual bool ViterbiWeight::zero () const

  this letterweight is neutral element for add (absorbing element for mult).
- virtual bool ViterbiWeight::one () const

  this letterweight is neutral element for mult.
- bool ViterbiWeight::equal (const LetterWeight ∗rhs) const

  rhs must be a ViterbiWeight.
- bool ViterbiWeight::smaller (const LetterWeight ∗rhs) const

  rhs must be a ViterbiWeight.
- void ViterbiWeight::add (const LetterWeight ∗rhs)

  sum is min.
- void ViterbiWeight::mult (const LetterWeight ∗rhs)

  product is sum.
- void **ViterbiWeight::print** (std::ostream &) const
- virtual bool LetterWeight::equal (const LetterWeight ∗) const

  binary operators are defined only between descendant LetterWeights of same typeid.
- virtual bool LetterWeight::smaller (const LetterWeight ∗) const

  binary operators are defined only between descendant LetterWeights of same typeid.
- virtual void LetterWeight::add (const LetterWeight ∗)

  binary operators are defined only between descendant LetterWeights of same typeid.
- virtual void LetterWeight::mult (const LetterWeight ∗)

  binary operators are defined only between descendant LetterWeights of same typeid.
- virtual bool LetterWeight::zero () const

  this letterweight is neutral element for add (absorbing element for mult).
- virtual bool LetterWeight::one () const

  this letterweight is neutral element for mult.
- virtual void **LetterWeight::print** (std::ostream &o) const

- Weight::Weight (const Weight &w)

  clone the letter.
- Weight & **Weight::operator=** (const Weight &)
- Weight $*$ **Weight::clone** () const
- Weight Weight::make (double v) const
- Weight Weight::get_zero () const

  return the neutral element for add (absorbing element for mult) for the LetterWeight, if any otherwise return unknown Weight.
- Weight Weight::get_one () const

  return the neutral element for mult for the LetterWeight, if any otherwise return unknown Weight.
- bool Weight::unknown () const

  unknown weight is a Weight with NULL letter.
- bool Weight::hasType (std::string code) const
- double Weight::norm ()
- void Weight::scalar (double)

  scalar multiplication.
- void Weight::invert ()

  multiplicative inverse, for semifields
- void Weight::clear ()

  delete the letter.
- bool Weight::zero () const

  this weight is neutral element for + (absorbing element for $*$).
- bool Weight::one () const

  this weight is neutral element for $*$
- bool Weight::equal (const Weight &rhs) const

  binary operators are defined only between descendant Weights of same typeid
- bool Weight::smaller (const Weight &rhs) const
- void Weight::add (const Weight &rhs)
- void Weight::mult (const Weight &rhs)
- void **Weight::print** (std::ostream &o) const
- std::string **Weight::save_to_string** ()

Variables

- static TropicalWeight TropicalWeight::penalty [18]

  penalty by arity.

13.7.1 Detailed Description

The `weight` module contains the definitions of several domains for weight values for tree automata.

13.7.2 Function Documentation

13.7.2.1 operator==()

```
bool operator== (
            const Weight & lhs,
            const Weight & rhs )  [inline]
```

See also

equal

13.7.2.2 operator<()

```
bool operator< (
            const Weight & lhs,
            const Weight & rhs )  [inline]
```

See also

smaller

13.7.2.3 operator<<()

```
std::ostream& operator<< (
            std::ostream & o,
            const Weight & rhs )  [inline]
```

See also

print

13.7.2.4 CountingWeight()

```
CountingWeight::CountingWeight (
            CWType t,
            size_t dim )  [protected]
```

Warning

must dim > 0

13.7.2.5 operator=() [1/4]

```
CountingWeight & CountingWeight::operator= (
            const LetterWeight & rhs )
```

Parameters

rhs	must be a CountingWeight

13.7.2.6 make() [1/2]

```
Weight CountingWeight::make (
            double v ) const  [virtual]
```

Returns

ERROR should not be used

Implements LetterWeight.

13.7.2.7 make_unit()

```
Weight CountingWeight::make_unit (
            size_t dim,
            size_t i )  [static]
```

Parameters

dim	must be $>$ 0
i	must be $>$=0
i	must be $<$ dim

13.7.2.8 zero()

```
bool CountingWeight::zero ( ) const  [virtual]
```

Warning

this weight is zero (FAIL)

Reimplemented from LetterWeight.

13.7.2.9 error()

```
bool CountingWeight::error ( ) const
```

Warning

this weight is the error value

13.7.2.10 one()

```
bool CountingWeight::one ( ) const  [virtual]
```

Warning

this weight is one (null vector)

Reimplemented from LetterWeight.

13.7.2.11 norm() [1/3]

```
double CountingWeight::norm ( ) const  [virtual]
```

Warning

do not use

Implements LetterWeight.

13.7.2.12 equal() [1/4]

```
bool CountingWeight::equal (
             const LetterWeight * rhs ) const  [protected], [virtual]
```

Parameters

rhs	must be a CountingWeight

Reimplemented from LetterWeight.

13.7.2.13 smaller() [1/4]

```
bool CountingWeight::smaller (
             const LetterWeight * rhs ) const  [protected], [virtual]
```

Parameters

rhs	must be a CountingWeight

Warning

do not use

Reimplemented from LetterWeight.

13.7.2.14 add() [1/5]

```
void CountingWeight::add (
             const LetterWeight * rhs )  [protected], [virtual]
```

- FAIL is neutral
- ERROR absorbing
- VECTOR + VECTOR = ERROR

  Warning

  this and rhs must have same dimension

Reimplemented from LetterWeight.

13.7.2.15 mult() [1/5]

```
void CountingWeight::mult (
             const LetterWeight * rhs )  [protected], [virtual]
```

- VECTOR . VECTOR = VECTOR with component-wise sum
- VECTOR . FAIL = FAIL . VECTOR = FAIL
- FAIL . FAIL = FAIL
- ERROR absorbing

  Warning

  this and rhs must have same dimension

Reimplemented from LetterWeight.

13.7.2.16 invert() [1/4]

```
void FloatWeight::invert ( )  [virtual]
```

multiplicative inverse.

Warning

this weight must not be zero.

Implements LetterWeight.

13.7.2.17 equal() [2/4]

```
bool FloatWeight::equal (
             const LetterWeight * rhs ) const  [protected], [virtual]
```

Parameters

rhs	must be a FloatWeight.

Reimplemented from LetterWeight.

13.7.2.18 smaller() [2/4]

```
bool FloatWeight::smaller (
             const LetterWeight * rhs ) const  [protected], [virtual]
```

Parameters

rhs	must be a FloatWeight.

Reimplemented from LetterWeight.

13.7.2.19 add() [2/5]

```
void FloatWeight::add (
             const LetterWeight * rhs )  [protected], [virtual]
```

Parameters

rhs	must be a FloatWeight.

Reimplemented from LetterWeight.

13.7.2.20 mult() [2/5]

```
void FloatWeight::mult (
            const LetterWeight * rhs )  [protected], [virtual]
```

Parameters

rhs	must be a FloatWeight.

Reimplemented from LetterWeight.

13.7.2.21 PerfoWeight()

```
PerfoWeight::PerfoWeight (
            const InputSegment * s,
            const AlignedInterval * p,
            pre_t pre = 0,
            pre_t post = 0 )
```

probability of positions in the given alignement in the interval defined by the given path.

= product of the probabilities for the points in the alignement,

- the pre points on the left bound
- the post rightmost points in the right half of the alignment

13.7.2.22 operator=() [2/4]

```
PerfoWeight & PerfoWeight::operator= (
            const LetterWeight & rhs )
```

Parameters

rhs	must be a PerfoWeight

13.7.2.23 operator=() [3/4]

```
TropicalWeight & TropicalWeight::operator= (
            const LetterWeight & rhs )
```

Warning

rvalue must be a TropicalWeight

13.7.2.24 norm() [2/3]

```
double TropicalWeight::norm ( ) const  [virtual]
```

Warning

must not be zero (infinity)

Implements LetterWeight.

13.7.2.25 invert() [2/4]

```
void TropicalWeight::invert ( )  [virtual]
```

multiplicative inverse.

Warning

this weight must not be zero.

Implements LetterWeight.

13.7.2.26 equal() [3/4]

```
bool TropicalWeight::equal (
            const LetterWeight * rhs ) const  [protected], [virtual]
```

Parameters

rhs	must be a TropicalWeight

Reimplemented from LetterWeight.

13.7.2.27 smaller() [3/4]

```
bool TropicalWeight::smaller (
             const LetterWeight * rhs ) const  [protected], [virtual]
```

Parameters

rhs	must be a TropicalWeight

Reimplemented from LetterWeight.

13.7.2.28 add() [3/5]

```
void TropicalWeight::add (
             const LetterWeight * rhs )  [protected], [virtual]
```

sum is min.

Parameters

rhs	must be a TropicalWeight set this to the min of this and rhs

Reimplemented from LetterWeight.

13.7.2.29 mult() [3/5]

```
void TropicalWeight::mult (
             const LetterWeight * rhs )  [protected], [virtual]
```

product is sum.

Parameters

rhs	must be a TropicalWeight set this to the sum of this and rhs

Reimplemented from LetterWeight.

13.7.2.30 gracenote()

```
TropicalWeight TropicalWeight::gracenote (
             size_t n )  [static]
```

penalty for given number of grace notes in a leaf.

- 0 = 1 event, no grace note
- 1 = 1 event, 1 grace note
- 2 = 1 event, 2 grace notes
- etc

13.7.2.31 operator=() [4/4]

```
ViterbiWeight & ViterbiWeight::operator= (
             const LetterWeight & rvalue )
```

Parameters

rvalue	must be a ViterbiWeight

13.7.2.32 invert() [3/4]

```
void ViterbiWeight::invert ( )  [virtual]
```

multiplicative inverse.

Warning

this weight must not be zero.

Todo TBR

Implements LetterWeight.

13.7.2.33 add() [4/5]

```
void ViterbiWeight::add (
             const LetterWeight * rhs )  [protected], [virtual]
```

sum is min.

Parameters

rhs	must be a ViterbiWeight. set this to the min of this and rhs.

Reimplemented from LetterWeight.

13.7.2.34 mult() [4/5]

```
void ViterbiWeight::mult (
            const LetterWeight * rhs )  [protected], [virtual]
```

product is sum.

Parameters

rhs	must be a ViterbiWeight. set this to the sum of this and rhs.

Reimplemented from LetterWeight.

13.7.2.35 make() [2/2]

```
Weight Weight::make (
            double v ) const
```

See also

LetterWeight.make

13.7.2.36 hasType()

```
bool Weight::hasType (
            std::string code ) const
```

Parameters

code	is the code of the letter weight if there is one or "UNKNOWN" otherwise.

13.7.2.37 norm() [3/3]

```
double Weight::norm ( )
```

Warning

this Weight must not be unknown (letter != NULL)
not const: may need recomputations.

13.7.2.38 scalar()

```
void Weight::scalar (
             double d )
```

scalar multiplication.

Warning

this Weight must not be unknown (letter != NULL).

13.7.2.39 invert() [4/4]

```
void Weight::invert ( )
```

multiplicative inverse, for semifields

Warning

this Weight must not be zero
this Weight must not be unknown (letter != NULL)

Todo TBR : replace by div with const rhs

13.7.2.40 clear()

```
void Weight::clear ( )
```

delete the letter.

Warning

this weight becomes unknown.

13.7.2.41 equal() [4/4]

```
bool Weight::equal (
             const Weight & rhs ) const  [protected]
```

binary operators are defined only between descendant Weights of same typeid

- two unknown Weights are equal
- one unknown weight and one not unknown are not equal
- equality of two not unknown weight depends on the descendant class

13.7.2.42 smaller() [4/4]

```
bool Weight::smaller (
             const Weight & rhs ) const  [protected]
```

- unknown Weight is minimal:
- unknown Weight is smaller than any not unknown Weight
- not unknown Weight is not smaller that unknown Weight
- unknown Weight is not smaller than unknown Weight
- inequality of two not unknown weight depends on the descendant class

13.7.2.43 add() [5/5]

```
void Weight::add (
             const Weight & rhs )  [protected]
```

Warning

this and rhs must not be unknown

13.7.2.44 mult() [5/5]

```
void Weight::mult (
             const Weight & rhs )  [protected]
```

Warning

this and rhs must not be unknown

13.7.3 Variable Documentation

13.7.3.1 penalty

```
TropicalWeight TropicalWeight::penalty  [static]
```

Initial value:

```
=
{
    TropicalWeight(0.01),
    TropicalWeight(0.02),
    TropicalWeight(0.03),
    TropicalWeight(0.04),
    TropicalWeight(0.05),
    TropicalWeight(0.06),
    TropicalWeight(0.07),
    TropicalWeight(0.08),
    TropicalWeight(0.09),
    TropicalWeight(0.10),
    TropicalWeight(0.11),
    TropicalWeight(0.12),
    TropicalWeight(0.13),
    TropicalWeight(0.14),
    TropicalWeight(0.15),
    TropicalWeight(0.16),
    TropicalWeight(0.17),
    TropicalWeight(0.18)
}
```

penalty by arity.

Chapter 14

Namespace Documentation

14.1 patch Namespace Reference

trace levels:

Functions

- template<typename T >
  std::string **to_string** (const T &n)

14.1.1 Detailed Description

trace levels:

- 0: off
- 1: critical
- 2: error
- 3: warn
- 4: info
- 5: debug
- 6: trace to patch a bug in g++ see `https://stackoverflow.com/questions/12975341/to-string-is-not`

14.2 ScoreModel Namespace Reference

Classes

- class Beam
- class Duration
- class Event
- class Measure
- class Note
- class Part
- class Rest
- class Score
- class ScoreMeter
- class Sequence
- class SpanningElement
- class Tuplet
- class Voice

Typedefs

- typedef std::pair$<$ Pitch, Pitch $>$ **VoiceRange**
- typedef std::pair$<$ Note $*$, Note $*$ $>$ **Tie**

14.2.1 Detailed Description

Representation and management of beams

Author

Philippe RigauxA beam encompasses n events

Representation of duration

Author

Philippe RigauxA duration has an internal representation as a rational.

Value 1 is a whole note

Several utility methods allow to get the symbolic representation

Abstract model of Events

Author

Philippe RigauxAn Event is anything that has a duration

Measures

Author

Philippe Rigaux

Model of a part

Author

Philippe RigauxA part is a set of voices, to be played by a single instrument/performer

Model of a score

Author

Philippe RigauxThe score class: models a score content

Representation of a score meter

Author

Philippe Rigaux

A sequence = a list of events

Utility class used for sequential calculations

Author

Philippe RigauxA sequence is a list of events

Abstract class for spanning elements

Author

Philippe RigauxA spanning element provides a notation for a sequence of events.

Examples of sub-classes are: slurs, beams, tuplets

Representation of yuplets

Author

Philippe RigauxA tuplet encompasses n events, and covers a regular duration

Abstract model of voice

Author

Philippe RigauxA voice is a sequence of event, belonging to a Part

14.3 State Namespace Reference

States.

Functions

- bool **isWTA** (state_t)
- bool **isLabel** (state_t)
- bool **isMeta** (state_t)
- state_t MetaState (size_t barnb)

  Meta state corresponding to bar nb barnb.

14.3.1 Detailed Description

States.

- positive of null long: state of WTA (wta state)
- positive of null int: state of WTA or label (label symbol)
- negative long: inverse of number of bars (meta state)

Chapter 15

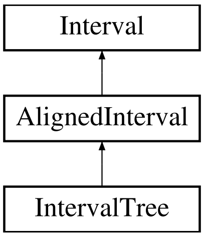
Class Documentation

15.1 AlignedInterval Class Reference

Extension of Interval with computed alignment of InputSegment points onto left- and right-bounds.

```
#include <AlignedInterval.hpp>
```

Inheritance diagram for AlignedInterval:



Public Member Functions

- AlignedInterval (const InputSegment ∗s, Rational mend=Rational(1), bool f_align=false)

  Interval covering the whole length of the given input segment with given musical time length (number of bars).
- AlignedInterval (const AlignedInterval &)

  copy.
- ∼AlignedInterval ()
- virtual AlignedInterval & **operator=** (const AlignedInterval &)
- virtual bool **operator==** (const AlignedInterval &) const
- size_t lsize () const

  number of elements of input segment in the first half of this interval.
- size_t lfirst () const

  index of the first element of input segment inside the fist half of this interval.
- size_t rsize () const

  number of elements of input segment in the second half of this interval.
- size_t rfirst () const

  index of the first element of input segment inside the second half of this interval.
- size_t size () const

  number of elements of input segment in this interval.

- size_t first () const

  index of the first element of input segment after the right bound of this interval (i.e. inside or after this interval).
- size_t next () const

  index of the first element of input segment ouside this interval (= after the right bound).
- bool inhabited () const

  this interval contains at least an element of the input segment
- size_t align (const InputSegment ∗s, size_t b)

  set the alignment parameters, starting from index b of input segment point and return the next index of point in input segment to be processed (first index at right of this interval) or the size of input segment (total # points) if end of segment is reached.
- size_t align (const InputSegment ∗s)

  same as previous but uses _seg_first instead of argument b.
- size_t rewind (const InputSegment ∗s, size_t b)

  compute only the value of the next point (the first element of input segment after the right bound of this interval) starting from index b of input segment point.
- size_t rewind (const InputSegment ∗)

  same as previous but uses _seg_first instead of arg. b.
- bool aligned () const

  this interval has been aligned.

Protected Member Functions

- AlignedInterval (const InputSegment ∗s, Rational mbeg, Rational mend, double rbeg, double rend, size_t first, bool f_align=false)

  aligned interval with musical-time bounds [mbegin, mbegin+mdur[and real-time bounds [rbegin, rbegin+rdur[for the input segment s.

Friends

- class **IntervalHeap**
- std::ostream & **operator**<< (std::ostream &, const AlignedInterval &)

Additional Inherited Members

15.1.1 Detailed Description

Extension of Interval with computed alignment of InputSegment points onto left- and right-bounds.

The result of aligment can be consulted with function l/rsize, l/rfirst. alignement is computed by function align.

The alignement is computed automatically for newly created intervals, the other created intervals (multiple-bars intervals) are not aligned.

15.1.2 Constructor & Destructor Documentation

15.1.2.1 ∼AlignedInterval()

```
AlignedInterval::∼AlignedInterval ( )  [inline]
```

Warning

do not deallocate the segment here.

15.1.3 Member Function Documentation

15.1.3.1 lsize()

```
size_t AlignedInterval::lsize ( ) const  [inline]
```

number of elements of input segment in the first half of this interval.

Warning

the interval must have been aligned.

15.1.3.2 lfirst()

```
size_t AlignedInterval::lfirst ( ) const  [inline]
```

index of the first element of input segment inside the fist half of this interval.

Returns

out_of_range (= size of segment) if l_size() == 0.

Warning

the interval must have been aligned.

15.1.3.3 rsize()

```
size_t AlignedInterval::rsize ( ) const  [inline]
```

number of elements of input segment in the second half of this interval.

Warning

the interval must have been aligned.

15.1.3.4 rfirst()

```
size_t AlignedInterval::rfirst ( ) const  [inline]
```

index of the first element of input segment inside the second half of this interval.

Returns

out_of_range (= size of segment) if r_size() == 0.

Warning

the interval must have been aligned.

15.1.3.5 size()

```
size_t AlignedInterval::size ( ) const  [inline]
```

number of elements of input segment in this interval.

Warning

the interval must have been aligned.

15.1.3.6 first()

```
size_t AlignedInterval::first ( ) const  [inline]
```

index of the first element of input segment after the right bound of this interval (i.e. inside or after this interval).

Returns

out_of_range (= size of segment) if there is none.

15.1.3.7 next()

```
size_t AlignedInterval::next ( ) const  [inline]
```

index of the first element of input segment ouside this interval (= after the right bound).

Returns

out_of_range (= size of segment) if there is none.

Warning

the interval must have been aligned.

15.1.3.8 inhabited()

```
bool AlignedInterval::inhabited ( ) const  [inline]
```

this interval contains at least an element of the input segment

Warning

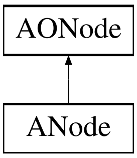
the interval must have been aligned.

The documentation for this class was generated from the following files:

- src/segment/AlignedInterval.hpp
- src/segment/AlignedInterval.cpp

15.2 ANode Class Reference

Inheritance diagram for ANode:



Public Member Functions

- **ANode** (size_t a)
- void **add** (const ONode &n)

Public Attributes

- std::vector< ONode > **children**

Additional Inherited Members

The documentation for this class was generated from the following file:

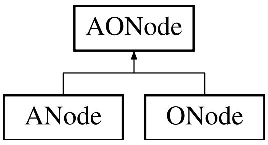
- src/schemata/SubdivisionSchema.hpp

15.3 AONode Class Reference

AND-OR alternating nested lists used by Adrien in RQ.

`#include <SubdivisionSchema.hpp>`

Inheritance diagram for AONode:



Public Member Functions

- **AONode** (size_t a)
- bool **inner** () const
- bool **leaf** () const
- size_t **value** () const

Protected Attributes

- size_t _arity

  0 for inner nodes, arity > 1 for leaf

15.3.1 Detailed Description

AND-OR alternating nested lists used by Adrien in RQ.

from RQ doc: A subdivision schema of this kind is presented in the form of a nested list.

The documentation for this class was generated from the following file:

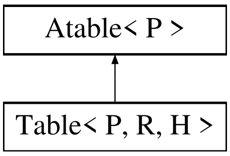
- src/schemata/SubdivisionSchema.hpp

15.4 Atable< P > Class Template Reference

abstract interface to parse table

`#include <Atable.hpp>`

Inheritance diagram for Atable< P >:



Public Member Functions

- Atable (Parser< P > ∗env, RunCompare< P > comp)
- virtual Run< P > ∗ best (const P &p)=0

  return k-best run pointed by p or NULL if there is none. k is either included in p or the default value 1.
- virtual RhythmTree ∗ bestTree (const P &p)=0

  tree corresponding to the k-best run in p.
- virtual RhythmTree ∗ bestTree (Run< P > ∗p)=0

  when the k-best run in p is already computed.
- virtual size_t add (const P &p, Run< P > ∗r, Record< P > ∗i)=0

  add possible instances of run r to the entries in table for corresp. to possible instances for p. dispatch to the four functions below according to p and r.
- virtual size_t **nb_entries** ()=0
- virtual size_t **nb_runs** ()=0

Public Attributes

- Parser< P > ∗ parent

  parsing environment.

Protected Attributes

- RunCompare< P > **_comparer**

15.4.1 Detailed Description

template<class P>
class Atable< P >

abstract interface to parse table

15.4.2 Constructor & Destructor Documentation

15.4.2.1 Atable()

```
template<class P>
Atable< P >::Atable (
            Parser< P > ∗ env,
            RunCompare< P > comp )
```

Parameters

env	environment must not be null.

15.4.3 Member Function Documentation

15.4.3.1 best()

```
template<class P>
virtual Run<P>* Atable< P >::best (
            const P & p )  [pure virtual]
```

return k-best run pointed by p or NULL if there is none. k is either included in p or the default value 1.

Parameters

p	must be complete.

Implemented in Table< P, R, H >, Table< SIPpointer, Brecord< SIPpointer >, SIPpointerHasher >, Table< Spointer, Brecord< Spo
Table< SKpointer, Krecord< SKpointer >, SKpointerHasher >, and Table< SKIPpointer, Krecord< SKIPpointer >, SKIPpointerHas

15.4.3.2 bestTree()

```
template<class P>
virtual RhythmTree* Atable< P >::bestTree (
            Run< P > * p )  [pure virtual]
```

when the k-best run in p is already computed.

Parameters

p	not used

Todo TBR param p

Warning

the run must be wta.

Implemented in Table< P, R, H >, Table< SIPpointer, Brecord< SIPpointer >, SIPpointerHasher >, Table< Spointer, Brecord< Spo
Table< SKpointer, Krecord< SKpointer >, SKpointerHasher >, and Table< SKIPpointer, Krecord< SKIPpointer >, SKIPpointerHas

15.4.3.3 add()

```
template<class P>
virtual size_t Atable< P >::add (
            const P & p,
            Run< P > * r,
            Record< P > * i )  [pure virtual]
```

add possible instances of run r to the entries in table for corresp. to possible instances for p. dispatch to the four functions below according to p and r.

Parameters

p	can be complete or partial.
r	can be complete or partial.
i	if p is complete, then i must be an iterator to the entry for p in table, otherwise (p partial), i is table.end().

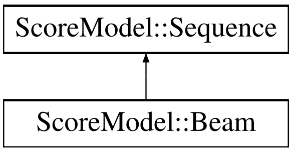
Implemented in Table< P, R, H >, Table< SIPpointer, Brecord< SIPpointer >, SIPpointerHasher >, Table< Spointer, Brecord< Spo
Table< SKpointer, Krecord< SKpointer >, SKpointerHasher >, and Table< SKIPpointer, Krecord< SKIPpointer >, SKIPpointerHas

The documentation for this class was generated from the following files:

- src/segment/InputSegment.hpp
- src/table/Atable.hpp

15.5 ScoreModel::Beam Class Reference

Inheritance diagram for ScoreModel::Beam:



Public Member Functions

- Beam (Sequence events)
- ∼Beam ()

15.5.1 Constructor & Destructor Documentation

15.5.1.1 Beam()

```
ScoreModel::Beam::Beam (
            Sequence events )
```

Main constructor.

15.5.1.2 ∼Beam()

```
ScoreModel::Beam::∼Beam ( )
```

Destructor

The documentation for this class was generated from the following files:

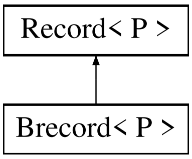
- src/scoremodel/Beam.hpp
- src/scoremodel/Beam.cpp

15.6 Brecord< P > Class Template Reference

record associated to Ptr for one-best procedures.

```
#include <Brecord.hpp>
```

Inheritance diagram for Brecord< P >:



Public Member Functions

- **Brecord** (const P &, RunCompare< P >)
- virtual void add (Run< P > ∗)

  add a run to the record.
- virtual Run< P > ∗ best (Atable< P > ∗parent, size_t k=1)

  returns the k-th best run of the record
- virtual bool **empty** () const

Protected Attributes

- Run< P > ∗ _best

  best run for the associated state.

Additional Inherited Members

15.6.1 Detailed Description

template<class P>
class Brecord< P >

record associated to Ptr for one-best procedures.

15.6.2 Member Function Documentation

15.6.2.1 best()

```
template<class P >
virtual Run<P>* Brecord< P >::best (
            Atable< P > * parent,
            size_t k = 1 )  [virtual]
```

returns the k-th best run of the record

Parameters

parent	is ignored
k	rank (as in k-best)

Implements Record< P >.

The documentation for this class was generated from the following file:

- src/table/Brecord.hpp

15.7 ComboState Class Reference

tmp state structure for construction of ComboWTA from a WTA (base schema) and an input segment casted into state_t after construction

```
#include <ComboWTA.hpp>
```

Public Member Functions

- **ComboState** (const InputSegment ∗s, IntervalHeap ∗)
- **ComboState** (state_t, IntervalTree ∗, pre_t rp=0, pre_t rr=0)
- ComboState (const ComboState &, pre_t rp=0, pre_t rr=0)
- bool **compatible** (label_t label) const
- bool **operator==** (const ComboState &s) const
- bool operator< (const ComboState &s) const

    lexicographic comparison on hash value (array[5])

Public Attributes

- state_t cs_state

    state of base schema.
- IntervalTree ∗ cs_path

    current augmented path (interval of points + alignment of input segment) share: in addComboState many ComboState constructed with the same cs_path.
- pre_t cs_pre

    guess number of points aligned to right of previous segment.
- pre_t cs_post

    guess number of points aligned to right of current segment.

Friends

- std::ostream & **operator**<< (std::ostream &, const ComboState &)

15.7.1 Detailed Description

tmp state structure for construction of ComboWTA from a WTA (base schema) and an input segment casted into state_t after construction

label (for leaves): see WTA = continuation or number of note + grace notes at left of the current path

states (q:int, p:Path, rp:int list, rr:int list) ou label (feuille) q: state of base schema p: current path (interval of points in input segment) rp: guess number of points aligned to right of previous segment rr: guess number of points aligned to right of current segment

The documentation for this class was generated from the following files:

- src/schemata/ComboWTA.hpp
- src/schemata/ComboWTA.cpp

15.8 ComboStateHasher Struct Reference

Public Member Functions

- std::size_t operator() (const ComboState &cs) const

15.8.1 Member Function Documentation

15.8.1.1 operator()()

```
std::size_t ComboStateHasher::operator() (
              const ComboState & cs ) const  [inline]
```

See also

constant.h

The documentation for this struct was generated from the following file:

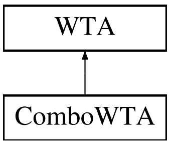
- src/schemata/ComboWTA.hpp

15.9 ComboWTA Class Reference

WTA combo: A special kind of WTA for quantization constructed from.

`#include <ComboWTA.hpp>`

Inheritance diagram for ComboWTA:



Public Member Functions

- ComboWTA (const InputSegment ∗, size_t bloc, const WTA &, pre_t pre=0)

  construction from input segment and WTA (base schema) with given max pre value and bloc number (in input segment, for alignement).
- virtual bool **hasType** (std::string code) const
- state_t initial (pre_t pre=0, pre_t post=0) const

  state representing the whole segment.

Additional Inherited Members

15.9.1 Detailed Description

WTA combo: A special kind of WTA for quantization constructed from.

- a given WTA (base schema)
- a given input segment (Alignment) the ComboWTA combines weights defined by the WTA schema (absolute measure of quality of rhythm) and a weight related to the distance of a rhythm to the given input segment.

not serializable

Warning

deprecatred

table of transitions top-down construction, given input and schema

principle:

- rp is propagated from father to leftmost child
- rr is propagated from father to rightmost child
- for every 2 states, s2 sibling and successive, s1.rr = s2.rp

given q state of schema, p path, k <= max{ n | q -> q1,...,qn | w transition of schema} mright(q, p, k) = # point d'input dans la derniere 2k partie de p

The documentation for this class was generated from the following files:

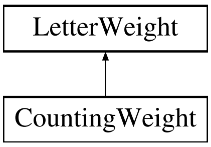
- src/schemata/ComboWTA.hpp
- src/schemata/ComboWTA.cpp

15.10 CountingWeight Class Reference

domain : vectors of fixed dim k $>$ 0

```
#include <CountingWeight.hpp>
```

Inheritance diagram for CountingWeight:



Public Member Functions

- **CountingWeight** (const CountingWeight &)
- CountingWeight & **operator=** (const CountingWeight &)
- CountingWeight & operator= (const LetterWeight &rhs)
- CountingWeight $*$ **clone** () const
- virtual Weight make (double v) const
- virtual Weight get_zero () const

  return the neutral element for add (absorbing element for mult) wrapped in a Weight.
- virtual Weight get_one () const

  return the neutral element for mult wrapped in a Weight.
- virtual double norm () const
- virtual void scalar (double)

  add to each component.
- virtual void invert ()
- virtual bool zero () const
- bool fail () const
- bool error () const
- virtual bool one () const
- virtual size_t **dim** () const
- virtual bool **hasType** (std::string code) const

Static Public Member Functions

- static Weight **make_one** (size_t)
- static Weight make_unit (size_t dim, size_t i)

Protected Types

- enum **CWType** { **VECTOR**, **FAIL**, **ERROR** }

Protected Member Functions

- CountingWeight (CWType t, size_t dim)
- virtual bool equal (const LetterWeight ∗rhs) const
- virtual bool smaller (const LetterWeight ∗rhs) const
- virtual void add (const LetterWeight ∗rhs)
- virtual void mult (const LetterWeight ∗rhs)
- virtual void **print** (std::ostream &) const
- virtual void **rawprint** (std::ostream &) const

Protected Attributes

- CWType **_type**
- size_t **_dim**
- std::vector< size_t > **_counters**

Friends

- std::ostream & **operator**<< (std::ostream &o, const CountingWeight &rhs)

15.10.1 Detailed Description

domain : vectors of fixed dim k > 0

- FAIL = stuck (0 run in state s for 1 tree)
- ERROR = ambiguity in grammar (2 runs for 1 tree)

zero = FAIL add : for all x, y vectors dim k x + y = ERROR ERROR absorbing for + one = null vector of dim k mult : for all x, y vectors dim k x . y = component-wise sum x . FAIL = FAIL . x = FAIL FAIL . FAIL = FAIL ERROR absorbing for .

15.10.2 Member Function Documentation

15.10.2.1 invert()

```
virtual void CountingWeight::invert ( )  [inline], [virtual]
```

Warning

do not use

Implements LetterWeight.

15.10.2.2 fail()

```
bool CountingWeight::fail ( ) const  [inline]
```

Warning

this weight is the error value

The documentation for this class was generated from the following files:

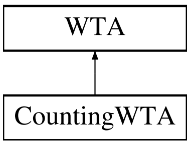
- src/weight/CountingWeight.hpp
- src/weight/CountingWeight.cpp

15.11 CountingWTA Class Reference

copy of WTA dedicated to corpus statistics.

```
#include <CountingWTA.hpp>
```

Inheritance diagram for CountingWTA:



Public Member Functions

- CountingWTA ()

  default initializer for cython
- CountingWTA (const WTA &a)

  copy base WTA reset weight values to counting weights (unit vectors)
- virtual bool **hasType** (std::string code) const
- virtual Weight eval (const RhythmTree &t) const

  special version of eval for CountingWeight with feedback in case of fail

Protected Member Functions

- Weight **evalCountingVerbose** (const RhythmTree &, state_t, Position) const
- void resetCounting (size_t dim)

  the weight of this WTA are replaced by "CountingWeight" unit vector of length dim (one unit per transition)

Protected Attributes

- OTransitionTable _tableids

  copy of transition table ordered according to the transition's ids (can be iterated).

Static Protected Attributes

- static bool(∗ _trcomp_ptr)(std::pair< state_t, Transition &>, std::pair< state_t, Transition &>) = &trcomp

  pointer to comparison functionå

Friends

- std::ostream & operator<< (std::ostream &, const CountingWTA &)

  it is important to enumerate in same order for printing and building unit weights!

Additional Inherited Members

15.11.1 Detailed Description

copy of WTA dedicated to corpus statistics.

for WTA weight estimation and WTA construction from corpus.

construction of WTA with counting weights (unit vectors) from WTA and verbose tree evaluation with feedback.

Warning

only for target SCHEMA

The documentation for this class was generated from the following files:

- src/schemata/CountingWTA.hpp
- src/schemata/CountingWTA.cpp

15.12 dagSchema Class Reference

dag whose edges are labeled by arity values two distinguished nodes:

```
#include <SubdivisionSchema.hpp>
```

Public Member Functions

- dagSchema (const ANode &)

  translation of AND-OR alternating nested lists into dag-schemas
- **dagSchema** (const ONode &)
- unsigned int **max** () const
- unsigned int **max** (const dagSchema &lhs, const dagSchema &rhs)
- void add (const ds_transition &dst)

15.12.1 Detailed Description

dag whose edges are labeled by arity values two distinguished nodes:

- a source node: 0
- a target node: _max_state

The documentation for this class was generated from the following files:

- src/schemata/SubdivisionSchema.hpp
- src/schemata/SubdivisionSchema.cpp

15.13 DepthMarking Class Reference

marking of states of a WTA with informations on the depth of their occurences initialized with a WTA, can be interrogated afterwards

```
#include <WTA.hpp>
```

Public Member Functions

- **DepthMarking** (const WTA &)
- int depth (state_t) const

  return depth mark if given state marked return -1 otherwise
- bool multiple (state_t) const

  return true if the given state can occur at multiple depths return false otherwise or if state not marked
- int mark (state_t, int)

  mark state using given depth and return new mark value can be the given depth or a greater depth with which the state had been already marked.

15.13.1 Detailed Description

marking of states of a WTA with informations on the depth of their occurences initialized with a WTA, can be interrogated afterwards

The documentation for this class was generated from the following files:

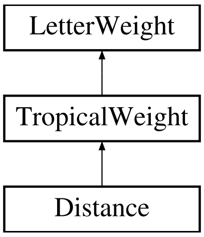
- src/schemata/WTA.hpp
- src/schemata/WTA.cpp
- src/schemata/WTA_BACKUP_31784.cpp
- src/schemata/WTA_BASE_31784.cpp
- src/schemata/WTA_LOCAL_31784.cpp
- src/schemata/WTA_REMOTE_31784.cpp

15.14 Distance Class Reference

concrete Weight domain identical to TropicalWeight with an additional constructor to compute a distance value from an Alignement, obtained as the sum of the pointwise distances.

```
#include <Distance.hpp>
```

Inheritance diagram for Distance:



Public Member Functions

- Distance (double d=0.0)

  defaut = null distance - not unknown
- Distance (const InputSegment ∗s, const AlignedInterval ∗p)

  weight which is the distance defined by alignment for input segment not unknown.
- **Distance** (const Distance &d)
- Distance & **operator=** (const Distance &)
- virtual Distance ∗ **clone** () const
- virtual Weight make (double v) const
- virtual Weight get_zero () const

  return the neutral element for add (absorbing element for mult) wrapped in a Weight.
- virtual Weight get_one () const

  return the neutral element for mult wrapped in a Weight.

Static Public Member Functions

- static Weight **make_zero** ()
- static Weight **make_one** ()

Protected Member Functions

- virtual void **print** (std::ostream &) const

Additional Inherited Members

15.14.1 Detailed Description

concrete Weight domain identical to TropicalWeight with an additional constructor to compute a distance value from an Alignement, obtained as the sum of the pointwise distances.

Warning

a Distance hasType "TropicalWeight"

ALT: implement as vector of pointwise distances

15.14.2 Member Function Documentation

15.14.2.1 make()

```
virtual Weight Distance::make (
            double v ) const  [inline], [virtual]
```

Warning

value must be positive

Todo TBR : stricly positive

Reimplemented from TropicalWeight.

The documentation for this class was generated from the following files:

- src/weight/Distance.hpp
- src/weight/Distance.cpp

15.15 ds_transition Struct Reference

dag schema

`#include <SubdivisionSchema.hpp>`

Public Member Functions

- ds_transition (unsigned int s, size_t l, unsigned int t)
- **ds_transition** (const ds_transition &dst)
- void **rename** (unsigned int s, unsigned int u)
- void shift (unsigned int n)

  increase source and target state by n
- void shift0 (unsigned int n)

  increase source and target state by n, if they are not 0

Public Attributes

- unsigned int **dst_source**
- size_t **dst_label**
- unsigned int **dst_target**

15.15.1 Detailed Description

dag schema

15.15.2 Constructor & Destructor Documentation

15.15.2.1 ds_transition()

```
ds_transition::ds_transition (
             unsigned int s,
             size_t l,
             unsigned int t )  [inline]
```

Parameters

s	source_state
l	arity_val
t	target_state

The documentation for this struct was generated from the following files:

- src/schemata/SubdivisionSchema.hpp
- src/schemata/SubdivisionSchema.cpp

15.16 ScoreModel::Duration Class Reference

Public Member Functions

- Duration (Rational ratio)
- Rational getValue () const
- void setValue (Rational value)
- int getCMN () const
- ∼Duration ()

Static Public Attributes

- static const int QUARTER_DURATION =4

15.16.1 Constructor & Destructor Documentation

15.16.1.1 Duration()

```
ScoreModel::Duration::Duration (
             Rational ratio )
```

Main constructor.

15.16.1.2 ∼Duration()

```
ScoreModel::Duration::∼Duration ( )
```

Destructor

15.16.2 Member Function Documentation

15.16.2.1 getValue()

```
Rational ScoreModel::Duration::getValue ( ) const  [inline]
```

Get the duration value as a rational: nb beats / beat unit

15.16.2.2 setValue()

```
void ScoreModel::Duration::setValue (
             Rational value )  [inline]
```

Set the duration value as a rational: nb beats / beat unit

15.16.2.3 getCMN()

```
int ScoreModel::Duration::getCMN ( ) const
```

Get the CMN code

The CMN code is a value ranging from 1 (whole note) to 256, and is always a power of 2. In the score output, intermediate durations (eg triplets) are usually obtained by applying a tuplet ratio to the CMN code.

15.16.3 Member Data Documentation

15.16.3.1 QUARTER_DURATION

```
const int ScoreModel::Duration::QUARTER_DURATION =4  [static]
```

Some constants

The documentation for this class was generated from the following files:

- src/scoremodel/Duration.hpp
- src/scoremodel/Duration.cpp

15.17 DurationList Class Reference

list of rational durations to label nodes of WTA Runs for Kbest enum.

```
#include <DurationList.hpp>
```

Public Member Functions

- DurationList ()

  empty duration list.
- **DurationList** (const DurationList &)
- DurationList (const DurationList &l, Rational q)

  copy of duration list l where all elements are multiplied by given Ratio q.
- DurationList (std::string)

  read duration list from file.
- DurationList & **operator=** (const DurationList &)
- bool **empty** () const
- bool **unit** () const
- size_t **size** () const
- Rational **cont** () const
- size_t summed () const

  for checking.
- std::list< Rational >::const_iterator **begin** () const
- std::list< Rational >::const_iterator **end** () const
- bool **complete** () const
- bool single_continuation () const

  one (non null) continuation and no event in the main list.
- bool single_event () const

  no continuation and only one event in the main list.
- bool event () const

  no continuation and some grace notes (dur=0) + one event (dur>0) in the main list.
- size_t nbgn () const

  number of grace note must be an event()
- Rational length () const

  sum of the elements of the duration list (including continuation)
- void add (Rational)

  add the event at the end of the main list.
- void addcont (Rational)

  push a continuation value.
- void normalize ()

  divide by the number of lists summed.
- DurationList & operator+= (const DurationList &rhs)

  concatenation.

Friends

- class **ValueList**
- std::ostream & **operator<<** (std::ostream &, const DurationList &)
- bool **operator==** (const DurationList &, const DurationList &)
- bool **operator!=** (const DurationList &, const DurationList &)

15.17.1 Detailed Description

list of rational durations to label nodes of WTA Runs for Kbest enum.

Duration is either positive (event w or wo continuations -ties) or null (grace note).

a duration list is made of 2 parts:

- _cont : initial duration (possibly null) tied to the previous duration list
- _main : main list of the other events (without ties) it is represented by _cont[_main]

to speed up processing, every DurationList is associated a state value, wich is one of the following:

- 0: empty list initial 0[] _cont=0, _main empty empty list assigned to a run at creation before appending of children's run lists.
- 1: single continuation 1[] _cont=1, _main empty
- 2: only-gn 0[0..0] _cont=0, _main = [0,...,0]
- 3: single event 0[1] _cont=0, _main = [1]
- 4: event 0[0..01] _cont=0, _main = [0,...,0,1]
- 5: other incomplete
- 6: other complete
- 7: empty non initial empty but other (children's) list have been appened already
- 0, 2, 5 are incomplete (sum != _summed)
- 1, 3, 4, 6, 7 are complete
- 1, 3, 4 are unit: the duration list represents one event and some grace notes

in quantization the length of list for a run correspoding to a segment is equal to the length of the segment.

The documentation for this class was generated from the following files:

- src/output/DurationList.hpp
- src/output/DurationList.cpp

15.18 DurationTree Class Reference

a tree container for duration lists. to avoid recomputation of division of duration lists.

```
#include <DurationTree.hpp>
```

Public Member Functions

- **DurationTree** (Rational len)
- **DurationTree** (const DurationList &d)
- DurationTree ∗ **sub** (size_t, size_t)

Public Attributes

- ValueList **top**

Friends

- std::ostream & **operator**<< (std::ostream &, const DurationTree &)

15.18.1 Detailed Description

a tree container for duration lists. to avoid recomputation of division of duration lists.

The documentation for this class was generated from the following files:

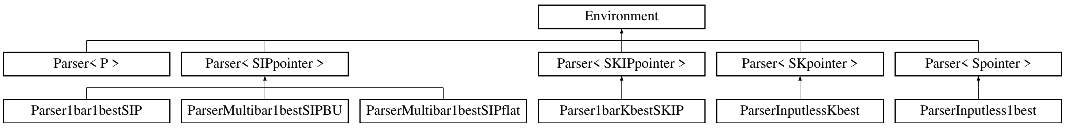
- src/output/DurationTree.hpp
- src/output/DurationTree.cpp

15.19 Environment Class Reference

wrapper abstract class embedding a standard input environment for parsing algos.

```
#include <Environment.hpp>
```

Inheritance diagram for Environment:



Public Member Functions

- Environment (InputSegment ∗s=NULL)

Public Attributes

- InputSegment ∗ segment

  input points to quantize.
- IntervalHeap ∗ iheap

  table of aligned input interval recorded.

15.19.1 Detailed Description

wrapper abstract class embedding a standard input environment for parsing algos.

it encapsulates some input data and structures for memory management.

15.19.2 Member Data Documentation

15.19.2.1 segment

```
InputSegment* Environment::segment
```

input points to quantize.

NULL when not given

15.19.2.2 iheap

```
IntervalHeap* Environment::iheap
```

table of aligned input interval recorded.

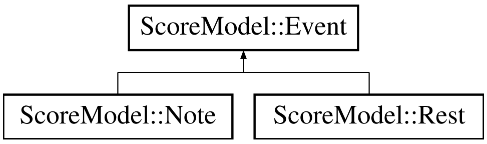
NULL when not needed (if there are no input points to process).

The documentation for this class was generated from the following files:

- src/segment/Environment.hpp
- src/segment/Environment.cpp

15.20 ScoreModel::Event Class Reference

Inheritance diagram for ScoreModel::Event:



Public Member Functions

- Event (Duration duration)
- virtual bool **isRest** () const
- virtual bool **isNote** () const
- void **setGraceNote** ()
- bool **isGraceNote** () const
- Duration getDuration () const
- void setDuration (Duration dur)
- void **setVoice** (Voice ∗voice)
- Voice ∗ **getVoicePtr** ()
- void **setMeasure** (Measure ∗measure)
- Measure ∗ **getMeasure** ()
- void **setStartBeam** (Beam ∗beam)
- Beam ∗ **getStartBeam** ()
- void **setEndBeam** (Beam ∗beam)
- Beam ∗ **getEndBeam** ()
- string **getId** () const
- void **setId** (string id)
- ∼Event ()

Static Public Attributes

- static const unsigned int **UNDEF_VELOCITY**

15.20.1 Constructor & Destructor Documentation

15.20.1.1 Event()

```
ScoreModel::Event::Event (
            Duration duration )
```

Main constructor.

15.20.1.2 ∼Event()

```
ScoreModel::Event::∼Event ( )
```

Destructor

15.20.2 Member Function Documentation

15.20.2.1 getDuration()

```
Duration ScoreModel::Event::getDuration ( ) const  [inline]
```

Get the duration of the Event

15.20.2.2 setDuration()

```
void ScoreModel::Event::setDuration (
            Duration dur )  [inline]
```

Set the duration of the Event

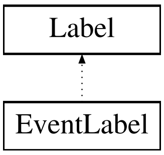
The documentation for this class was generated from the following files:

- src/scoremodel/Event.hpp
- src/scoremodel/Event.cpp

15.21 EventLabel Class Reference

```
#include <Label.hpp>
```

Inheritance diagram for EventLabel:



Public Member Functions

- **EventLabel** (unsigned int n=0)
- size_t **nbGraceNotes** () const
- void **addGraceNotes** (unsigned int)
- void **pushEvent** (Event ∗)

15.21.1 Detailed Description

Todo TBR (NOT USED)

The documentation for this class was generated from the following files:

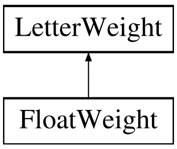
- src/output/Label.hpp
- src/output/Label.cpp

15.22 FloatWeight Class Reference

concrete Weight defined as a scalar value.

```
#include <FloatWeight.hpp>
```

Inheritance diagram for FloatWeight:



Public Member Functions

- FloatWeight (double d=0.0)

  defaut = null weight - not unknown
- **FloatWeight** (const FloatWeight &)
- FloatWeight & **operator=** (const FloatWeight &)
- FloatWeight & **operator=** (const LetterWeight &)
- virtual FloatWeight ∗ **clone** () const
- virtual Weight make (double v) const

  factory.
- virtual Weight get_zero () const

  return the neutral element for add (absorbing element for mult) wrapped in a Weight.
- virtual Weight get_one () const

  return the neutral element for mult wrapped in a Weight.
- virtual double **norm** () const
- virtual void **scalar** (double)
- virtual void invert ()

  multiplicative inverse.
- virtual bool zero () const

  this letterweight is neutral element for add (absorbing element for mult).
- virtual bool one () const

  this letterweight is neutral element for mult.
- bool **equal** (const FloatWeight &rhs) const
- bool **smaller** (const FloatWeight &rhs) const
- void **add** (const FloatWeight &rhs)
- void **mult** (const FloatWeight &rhs)
- virtual bool **hasType** (std::string code) const

Static Public Member Functions

- static Weight **make_zero** ()
- static Weight **make_one** ()

Protected Member Functions

- virtual bool equal (const LetterWeight ∗rhs) const
- virtual bool smaller (const LetterWeight ∗rhs) const
- virtual void add (const LetterWeight ∗rhs)
- virtual void mult (const LetterWeight ∗rhs)
- virtual void **print** (std::ostream &) const

Protected Attributes

- double **_val**

Friends

- std::ostream & **operator<<** (std::ostream &o, const FloatWeight &rhs)

15.22.1 Detailed Description

concrete Weight defined as a scalar value.

- domain : double
- operators
- add is +
- zero is 0.0
- mult is ∗
- one is 1.0

15.22.2 Member Function Documentation

15.22.2.1 make()

```
virtual Weight FloatWeight::make (
              double v ) const  [inline], [virtual]
```

factory.

Returns

a weight of same type as this letter, initialized with given value.

Implements LetterWeight.

The documentation for this class was generated from the following files:

- src/weight/FloatWeight.hpp
- src/weight/FloatWeight.cpp

15.23 std::hash< DurationList > Struct Template Reference

Public Member Functions

- size_t **operator()** (const DurationList &d) const

The documentation for this struct was generated from the following file:

- src/output/DurationList.hpp

15.24 std::hash< Rational > Class Template Reference

Public Member Functions

- std::size_t **operator()** (const Rational &x) const

The documentation for this class was generated from the following file:

- src/general/Rational.hpp

15.25 std::hash< ValueList > Struct Template Reference

Public Member Functions

- size_t **operator()** (const ValueList &d) const

The documentation for this struct was generated from the following file:

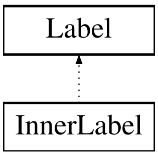
- src/output/ValueList.hpp

15.26 InnerLabel Class Reference

label for inner node. contains only arity (more info later?)

```
#include <Label.hpp>
```

Inheritance diagram for InnerLabel:



Public Member Functions

- **InnerLabel** (unsigned int)

15.26.1 Detailed Description

label for inner node. contains only arity (more info later?)

Todo TBR (NOT USED)

The documentation for this class was generated from the following files:

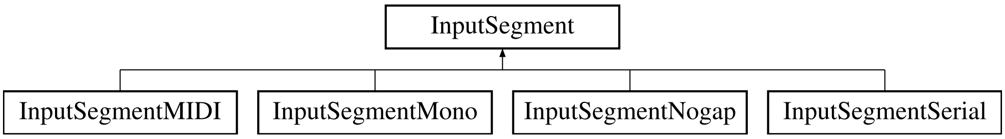
- src/output/Label.hpp
- src/output/Label.cpp

15.27 InputSegment Class Reference

intermediate representation for input performance data (sequence of timestamped events).

```
#include <InputSegment.hpp>
```

Inheritance diagram for InputSegment:



Public Member Functions

- InputSegment (double b=0, double e=0)

  constructs an empty input segment (no events)
- **InputSegment** (const InputSegment &)
- InputSegment (const InputSegment &s, double b, double e)

  copy and resize.
- double rbegin () const

  real-time start date (in seconds) of segment.
- double rend () const

  real-time end date (in seconds) of segment.
- double rduration () const

  real-time total duration (in seconds) of segment.
- Rational mduration () const

  musical total duration (in bars) of segment.
- size_t size () const

  number of non-floating points in segment.
- std::vector< MusPoint >::iterator begin ()

  iterators to the segment's contents.
- std::vector< MusPoint >::iterator **end** ()
- std::vector< MusPoint >::const_iterator **cbegin** () const
- std::vector< MusPoint >::const_iterator **cend** () const
- const MusPoint & point (long i) const

  return a ref to the point of index i.
- MusEvent ∗ event (long i) const

  return the event of the point of index i.
- double rdate (long i) const

  return the real-time date (in seconds) of the point of index i
- double rduration (long i) const

  return the real-time duration (in seconds) of the point of index i.
- double rduration (const MusPoint &p) const

  return the real-time duration (in seconds) of the given point.
- Rational & mdate (long i)

  return a reference to the musical-time date (in fraction of bar) of the point of index i.
- Rational & mduration (long i)

  return a reference to the musical-time duration (in fraction of bar) of the point of index i.

- long add_back (MusEvent ∗e, double rdate, double rdur, bool on, long link=MUSPOINTREF_NULL, Rational mdate=MUSTIME_UNKNOWN, Rational mduration=MUSTIME_UNKNOWN)

  insert new timestamped muspoint created from the parameters, at the end of the segment.
- long **add_back** (const MusPoint &)
- long add_floating (MusEvent ∗e, double rdate, double rdur, bool on, long link=MUSPOINTREF_NULL, Rational mdate=MUSTIME_UNKNOWN, Rational mduration=MUSTIME_UNKNOWN)

  create new timestamped muspoint from the parameters, and add the the heap of floating points (not in segment).
- long **add_floating** (const MusPoint &)
- void close (double e)

  set end date.
- bool quantized () const

  quantization has been applied at least once.
- template< class P >
  void quantize (Atable< P > ∗table, const P &p)

  set the musical time date and duration of events in this given input segment, according to the best run for p in given table.
- template< class P >
  size_t quantizu (Atable< P > ∗table, const P &p, size_t b=0)

  set the musical time date and duration of events in this given input segment, according to the best run for p in given table, starting from point number b in interval.
- void respell (int k=0)

  pitch spelling. unwindowed.
- void respell (Rational ws, int k=0)

  pitch spelling with a sliding window of given musical duration.
- virtual bool hasType (std::string code) const

  return wether the segment has the type of the code.
- void print (std::ostream &) const

  print size to output stream.

Protected Member Functions

- MusPoint & ncpoint (long i)

  same as point but not const.
- bool **check_index** (long i) const
- void link (long i, long j)

  the event of index i is linked to the event of index j.

Protected Attributes

- double _begin

  start date (in seconds) of segment.
- double _end

  start date (in seconds) of segment.
- double _len

  length (in seconds) of segment.
- Rational _mduration

  length (in bars) of segment.
- std::vector< MusPoint > _events

  event list.
- std::vector< MusPoint > _heap

  floating events.

Friends

- std::ostream & operator<< (std::ostream &, const InputSegment &)

  write segment content to output stream.

15.27.1 Detailed Description

intermediate representation for input performance data (sequence of timestamped events).

an input segment is made of:

- a time interval containing some timed events (muspoints), always sorted by increasing realtime date (vector of events).
- a heap of floating points.

The links in points are indexes in the input segment, where an index is a unique identifier of a point in either of the two above structures.

The realtime duration of a linked point is the difference of realtime dates (between the link and the point). the realtime duration of a point without link (with unknown link) is zero.

Todo do the same think with musical time duration.

suppr. samplestosec

suppr. member _res (resolution)

15.27.2 Member Function Documentation

15.27.2.1 mduration()

```
Rational InputSegment::mduration ( ) const [inline]
```

musical total duration (in bars) of segment.

will return MUSTIME_UNKNOWN if segment was not quantized.

15.27.2.2 hasType()

```
virtual bool InputSegment::hasType (
             std::string code ) const [inline], [virtual]
```

return wether the segment has the type of the code.

type InputSegment: plain InputSegment imported from MIDI file (or text) without filters.

15.27.3 Member Data Documentation

15.27.3.1 _mduration

```
Rational InputSegment::_mduration  [protected]
```

length (in bars) of segment.

is set at quantization

15.27.3.2 _events

```
std::vector<MusPoint> InputSegment::_events  [protected]
```

event list.

polymorphic (mono or poly)

15.27.3.3 _heap

```
std::vector<MusPoint> InputSegment::_heap  [protected]
```

floating events.

not in the list but can be linked by events on the list

The documentation for this class was generated from the following files:

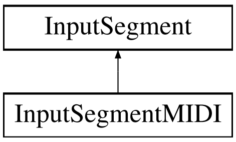
- src/segment/InputSegment.hpp
- src/segment/InputSegment.cpp

15.28 InputSegmentMIDI Class Reference

import an InputSegment from a MIDI file.

```
#include <InputSegmentMIDI.hpp>
```

Inheritance diagram for InputSegmentMIDI:



Public Member Functions

- InputSegmentMIDI (const std::string filename, int tracknb=1)

  read input segment from a MIDI file.
- InputSegmentMIDI (MidiFile &midifile, int tracknb=1)

  read input segment from a MIDI file.
- InputSegmentMIDI (const std::string filename, bool mono=true, bool norest=false, int tracknb=1)

  read input segment from a MIDI file.
- **InputSegmentMIDI** (const InputSegmentMIDI &)
- std::string **filename** () const
- size_t export_midifile (std::string, Rational)

  copy input midifile into output_midifile.
- size_t status () const

  exit status code for MIDI import
- size_t export_midifile (MidiFile &midifile, std::string midiout, Rational beatperbar)

  copy input midifile into output_midifile.
- size_t export_midifile_mono (MidiFile &midifile, std::string midiout, Rational beatperbar)

  copy input midifile into output_midifile, monophonic case.

Additional Inherited Members

15.28.1 Detailed Description

import an InputSegment from a MIDI file.

- The segment contains the NOTE-ON and NOTE-OFF events in the MIDI file, with the realtime dates.
- The musical dates and duration as set to unknown.
- Every NOTE_ON event is linked to the closest posteroir NOTE-OFF event with the same MIDI key. It is left unmatched (without warning) if there is no such matching NOTE-OFF.
- Several NOTE-ON with the same key may be linked to the same NOTE-OFF (a warning is displayed in this case).
- Unmatched NOTE-OFF are added with a warning.

15.28.2 Constructor & Destructor Documentation

15.28.2.1 InputSegmentMIDI() [1/3]

```
InputSegmentMIDI::InputSegmentMIDI (
            const std::string filename,
            int tracknb = 1 )
```

read input segment from a MIDI file.

The musical onsets and durations are all set to UNKNOWN.

Parameters

filename	name of input MIDI file
tracknb	MIDI track read

15.28.2.2 InputSegmentMIDI() [2/3]

```
InputSegmentMIDI::InputSegmentMIDI (
             MidiFile & midifile,
             int tracknb = 1 )
```

read input segment from a MIDI file.

The musical onsets and durations are all set to UNKNOWN.

Parameters

midifile	a MIDIfile object
tracknb	MIDI track read

15.28.2.3 InputSegmentMIDI() [3/3]

```
InputSegmentMIDI::InputSegmentMIDI (
             const std::string filename,
             bool mono = true,
             bool norest = false,
             int tracknb = 1 )
```

read input segment from a MIDI file.

the musical onsets and durations are all set to -1 for backward compatibility.

Parameters

mono	flag : set if we want a monophonic input segment.
norest	flag : if set, rests in MIDI file are ignored.

Todo TBR

15.28.3 Member Function Documentation

15.28.3.1 export_midifile() [1/2]

```
size_t InputSegmentMIDI::export_midifile (
            std::string ,
            Rational  )
```

copy input midifile into output_midifile.

update the onsets / offsets to the quantized values in this segment.

Warning

this segment must have been created from a midi file.
the musical date and duration must have been set in this segment.

Todo TBR mv export to segment/InputSegment∗ classes

15.28.3.2 status()

```
size_t InputSegmentMIDI::status ( ) const  [inline]
```

exit status code for MIDI import

Returns

0 if import or export worked well
error code $>$ 0 otherwise

15.28.3.3 export_midifile() [2/2]

```
size_t InputSegmentMIDI::export_midifile (
            MidiFile & midifile,
            std::string midiout,
            Rational beatperbar )
```

copy input midifile into output_midifile.

update the onsets / offsets to the quantized values in this segment.

Parameters

midifile	MIDIfile struct
midiout	name of output midifile
beatperbar	number of beats per bar (for producing output midifile)

Warning

> this segment must have been created from a midi file.
> the musical date and duration must have been set in this segment.

Todo TBR mv export to segment/InputSegment∗ classes

15.28.3.4 export_midifile_mono()

```
size_t InputSegmentMIDI::export_midifile_mono (
            MidiFile & midifile,
            std::string midiout,
            Rational beatperbar )
```

copy input midifile into output_midifile, monophonic case.

update the onsets / offsets to the quantized values in this segment.

Parameters

midifile	MIDIfile struct
midiout	name of output midifile
beatperbar	number of beats per bar (for producing output midifile)

Warning

> this segment must have been created from a midi file.
> the musical date and duration must have been set in this segment.

Todo TBR mv export to segment/InputSegment∗ classes

The documentation for this class was generated from the following files:

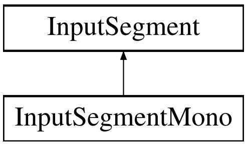
- src/input/InputSegmentMIDI.hpp
- src/input/InputSegmentMIDI.cpp

15.29 InputSegmentMono Class Reference

conversion of InputSegment to remove overlapping notes.

`#include <InputSegmentMono.hpp>`

Inheritance diagram for InputSegmentMono:



Public Member Functions

- InputSegmentMono (const InputSegment &s)

  transform the given input segment into a monophonic input segment (no two notes in the same time).

Additional Inherited Members

15.29.1 Detailed Description

conversion of InputSegment to remove overlapping notes.

if NOTEON1 is linked to NOTEOFF1 and NOTEON2 occurs between NOTEON1 and NOTEOFF1 (including N↩ OTEON1, excluding NOTEOFF1) then NOTEON1 is relinked to NOTEON2, and NOTEOFF1 is ignored if it is not linked.

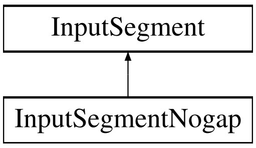
In the case where NOTEON1 = NOTEON2, we move the NOTEOFF of the shortest note (in real duration). This note becomes a grace note (duration 0).

The documentation for this class was generated from the following files:

- src/segment/InputSegmentMono.hpp
- src/segment/InputSegmentMono.cpp
- src/segment/InputSegmentMono_sumult.cpp

15.30 InputSegmentNogap Class Reference

Inheritance diagram for InputSegmentNogap:



Public Member Functions

- InputSegmentNogap (const InputSegment &s, bool norest=true)

  transform the given input segment into a new input segment without gaps.

Additional Inherited Members

The documentation for this class was generated from the following files:

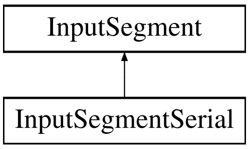
- src/segment/InputSegmentNogap.hpp
- src/segment/InputSegmentNogap.cpp

15.31 InputSegmentSerial Class Reference

serialization of an input segment in a text file.

```
#include <InputSegmentSerial.hpp>
```

Inheritance diagram for InputSegmentSerial:



Public Member Functions

- InputSegmentSerial (const std::string filename, bool mono=true)

  read input segment from a text file.
- **InputSegmentSerial** (const InputSegmentSerial &)
- std::string **filename** () const
- size_t save (const std::string filename)

  export this input segment into given file.
- size_t status () const

  return the final status for import.

Additional Inherited Members

15.31.1 Detailed Description

serialization of an input segment in a text file.

functions for import, export and comparison (evaluation).

15.31.2 Constructor & Destructor Documentation

15.31.2.1 InputSegmentSerial()

```
InputSegmentSerial::InputSegmentSerial (
            const std::string filename,
            bool mono = true )
```

read input segment from a text file.

if not present in text file, the musical onsets and durations are all set to -1.

Parameters

filename	name of input text file
mono	flag is true if we want a monophonic input segment.

15.31.3 Member Function Documentation

15.31.3.1 status()

```
size_t InputSegmentSerial::status ( ) const  [inline]
```

return the final status for import.

Returns

0 if import or export worked well
error code > 0 otherwise

The documentation for this class was generated from the following files:

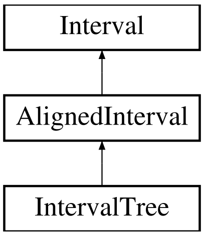
- src/input/InputSegmentSerial.hpp
- src/input/InputSegmentSerial.cpp

15.32 Interval Class Reference

an Interval in an input segment with realtime bounds (seconds) and musical bounds (fraction of bars).

```
#include <Interval.hpp>
```

Inheritance diagram for Interval:



Public Member Functions

- Interval (const InputSegment ∗s, Rational mend=Rational(1))

  top interval constructed from an input segment.
- Interval (const Interval &)

  copy.
- Interval (Interval ∗)

  used for copy of downcasted IntervalTree.
- ∼Interval ()
- virtual Interval & **operator=** (const Interval &)
- virtual bool operator== (const Interval &) const

  for using Interval as key in map.
- Rational **mduration** () const
- double **rduration** () const
- bool **insideBar** () const

Public Attributes

- Rational mbegin

  musical-time start. starting date of interval, in musical-time (number of bars) relatively (shift) to current bar start.
- Rational mend

  musical-time end.
- double rbegin

  real-time start.
- double rend

  real-time end.

Protected Member Functions

- Interval (const InputSegment ∗s, Rational mbeg, Rational mend, double rbeg, double rend)

  build an interval with musical-time bounds [mbegin, mbegin+mdur[and real-time bounds [rbegin, rbegin+rdur[for the input segment s.

Friends

- class **IntervalHeap**
- std::ostream & **operator**<< (std::ostream &, const Interval &)

15.32.1 Detailed Description

an Interval in an input segment with realtime bounds (seconds) and musical bounds (fraction of bars).

15.32.2 Constructor & Destructor Documentation

15.32.2.1 ∼Interval()

```
Interval::∼Interval ( )  [inline]
```

Warning

do not deallocate the segment here.

15.32.3 Member Data Documentation

15.32.3.1 mend

```
Rational Interval::mend
```

musical-time end.

ending date of interval, in musical-time (number of bars) relatively (shift) to current bar start.

0 for meta interval (in this case begin must be 0).

Warning

must be >= begin.

15.32.3.2 rbegin

```
double Interval::rbegin
```

real-time start.

starting date of interval, in real-time (seconds) i.e. real-time date aligned with the musical date bars + begin

15.32.3.3 rend

```
double Interval::rend
```

real-time end.

ending date of interval, in real-time (seconds) i.e. real date aligned with he musical date bars + end. must be > rbegin.

The documentation for this class was generated from the following files:

- src/segment/Interval.hpp
- src/segment/Interval.cpp

15.33 IntervalHasher Struct Reference

hash function for using interval as key in a unordered map.

```
#include <Interval.hpp>
```

Public Member Functions

- std::size_t **operator()** (const Interval &p) const

15.33.1 Detailed Description

hash function for using interval as key in a unordered map.

musical time bounds are ignored here

Todo TBR

The documentation for this struct was generated from the following file:

- src/segment/Interval.hpp

15.34 IntervalHeap Class Reference

table for storage of aligned intervals to avoid recomputation of alignments.

```
#include <IntervalHeap.hpp>
```

Public Member Functions

- bool **empty** () const
- size_t **size** () const
- IntervalTree ∗const make (const InputSegment ∗s, Rational mend, double rext=0)

  find or create (and push) a top interval of real-time duration covering the whole length of the given input segment s (root of interval tree) + the given extension.
- IntervalTree ∗const make (const InputSegment ∗s, Rational mbeg, Rational mend, double rbeg, double rend, size_t first, IntervalTree ∗p, IntervalTree ∗ps)

  get interval from heap, build it if not present.

Protected Attributes

- IntervalSet _interval_heap

  table of nodes in interval tree.
- int **_added**
- int **_found**

15.34.1 Detailed Description

table for storage of aligned intervals to avoid recomputation of alignments.

the aligned interval are indexed (in table) as intervals (i.e. indexed with bound of real-time and musical-time interval bounds).

The documentation for this class was generated from the following files:

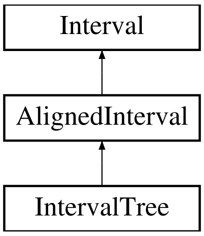
- src/segment/IntervalHeap.hpp
- src/segment/IntervalHeap.cpp

15.35 IntervalTree Class Reference

extension of Aligned Interval to define a tree of nested Alignements with sharing using hash table to store all alignment constructed.

```
#include <IntervalTree.hpp>
```

Inheritance diagram for IntervalTree:



Public Member Functions

- virtual IntervalTree ∗ parent ()
- virtual IntervalTree ∗ previous_sibling ()
- IntervalTree ∗ top (const InputSegment ∗s, IntervalHeap ∗h, Rational mend=Rational(1))

  top interval (root of interval tree) covering the whole length of the given input segment s.
- IntervalTree ∗ split (const InputSegment ∗, IntervalHeap ∗, double rdur, Rational mdur, size_t i)

  return a sub interval.
- IntervalTree ∗ split_back (const InputSegment ∗, IntervalHeap ∗, double rdur, Rational mdur, size_t i)

  return a sub interval.
- IntervalTree ∗ sub (const InputSegment ∗, IntervalHeap ∗, size_t a, size_t i)

  return a the i-1th sub-interval of the division of this interval in n equal parts. the sub-interval returned is aligned.

Protected Member Functions

- IntervalTree (const InputSegment ∗s, Rational mend=Rational(1))

  top interval (root of interval tree).
- IntervalTree (const InputSegment ∗s, Rational mbeg, Rational mend, double rbeg, double rend, size_t first, IntervalTree ∗p=NULL, IntervalTree ∗ps=NULL)

  build an interval tree with musical-time bounds [mbegin, mbegin+mdur[and real-time bounds [rbegin, rbegin+rdur[for the input segment s.

Protected Attributes

- IntervalTree ∗ **_parent**
- IntervalTree ∗ _previous_sibling

    previous sibling Interval in the Interval tree.
- std::map< size_t, std::vector< IntervalTree ∗ > > _children

    direct access to subtrees.

Friends

- class **IntervalHeap**

Additional Inherited Members

15.35.1 Detailed Description

extension of Aligned Interval to define a tree of nested Alignements with sharing using hash table to store all alignment constructed.

to construct IntervalTree use IntervalHeap.make and the members top, split, split_back and sub.

15.35.2 Member Function Documentation

15.35.2.1 parent()

```
virtual IntervalTree* IntervalTree::parent ( )  [inline], [virtual]
```

Returns

the embedding Interval in the Interval tree.
NULL if this Interval is the root of the tree.

15.35.2.2 previous_sibling()

```
virtual IntervalTree* IntervalTree::previous_sibling ( )  [inline], [virtual]
```

Returns

the previous sibling Interval in the Interval tree.
NULL if this Interval is the leftmost sibling.

15.35.3 Member Data Documentation

15.35.3.1 _previous_sibling

`IntervalTree* IntervalTree::_previous_sibling [protected]`

previous sibling Interval in the Interval tree.

NULL if this Interval is the leftmost sibling

15.35.3.2 _children

`std::map<size_t, std::vector<IntervalTree*> > IntervalTree::_children [protected]`

direct access to subtrees.

every entry in this map associate to an arity a a partition t1,...,ta of the root interval.

The documentation for this class was generated from the following files:

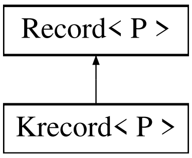
- src/segment/IntervalTree.hpp
- src/segment/IntervalTree.cpp

15.36 Krecord$<$ P $>$ Class Template Reference

record associated to Ptr for k-best procedures.

`#include <Krecord.hpp>`

Inheritance diagram for Krecord$<$ P $>$:



Public Member Functions

- **Krecord** (const P &, RunCompare$<$ P $>$)
- virtual void add (Run$<$ P $>$ $*$)

  add a run to the record.
- virtual Run$<$ P $>$ $*$ best (Atable$<$ P $>$ $*$table, size_t k=1)

  returns the k-th best run of the record. Fill the list of best runs up to (at most) k if necessary. If less than k best can be constructed (table is complete), return an null run (weight unknown), otherwise, the weight of the returned run is known.
- virtual bool **empty** () const

Protected Member Functions

- virtual void addCand (Run< P > ∗r)

  add Run r to the heap of candidates after some filtering based on optimisation flags.
- virtual void addBest (Run< P > ∗r)

  add Run r at the end of the list of best runs. record the given run r as one of the best runs of the record.
- bool **bestFilter** (const Run< P > ∗r)
- void addNext (Run< P > ∗r)

  add the candidates following Run r (lexico order for ranks) to the heap of candidates.

Protected Attributes

- std::priority_queue< Run< P > ∗, std::vector< Run< P > ∗ >, RunCompare< P > > _cand

  heap of candidate runs for the associated state. it is empty iff no more k-best can be added
- std::vector< Run< P > ∗ > _best

  ordered list of best runs for the associated state.

Additional Inherited Members

15.36.1 Detailed Description

template<class P>
class Krecord< P >

record associated to Ptr for k-best procedures.

15.36.2 Member Function Documentation

15.36.2.1 addCand()

```
template<class P >
virtual void Krecord< P >::addCand (
              Run< P > ∗ r )  [protected], [virtual]
```

add Run r to the heap of candidates after some filtering based on optimisation flags.

Parameters

r	given Run can be complete or partial (weight not fully evaluated)

15.36.2.2 addBest()

```
template<class P >
```

```
virtual void Krecord< P >::addBest (
            Run< P > * r )  [protected], [virtual]
```

add Run r at the end of the list of best runs. record the given run r as one of the best runs of the record.

Parameters

r	must be complete (weight fully evaluated).

15.36.2.3 addNext()

```
template<class P >
void Krecord< P >::addNext (
            Run< P > * r )  [protected]
```

add the candidates following Run r (lexico order for ranks) to the heap of candidates.

Parameters

r	must be complete (weight fully evaluated). the nexts Run (new candidates) will be partial

The documentation for this class was generated from the following file:

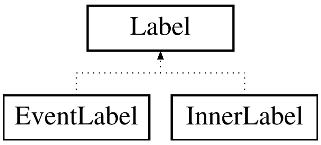
- src/table/Krecord.hpp

15.37 Label Class Reference

labels for nodes of output Rhythm Trees.

```
#include <Label.hpp>
```

Inheritance diagram for Label:



Public Member Functions

- **Label** (int a=0)
- size_t **arity** () const
- bool **isLeaf** () const
- bool **isInner** () const
- LabelKind **kind** () const

Static Public Member Functions

- static size_t **nbGraceNotes** (label_t)
- static size_t **nbEvents** (label_t)
- static bool **continuation** (label_t)
- static bool **abstract** (label_t)
- static bool **abstract** (label_t a, label_t n)
- static bool **leqabstract** (label_t a, label_t n)

Protected Attributes

- LabelKind **_type**
- size_t **_ar**

15.37.1 Detailed Description

labels for nodes of output Rhythm Trees.

Inner nodes are simply labeled by arity.

Leaves are labeled by info on:

- input (segment of unquantized points) and
- output (quantized points).

More precisely,

- the input info is about the alignement of unquantized input points on the bounds of the interval associated to the node. These are the pre and post values.
- the output info is about the number of quantized input points in this interval.
  - number 0 corresponds to a tie,
  - number 1 corresponds to a single event,
  - bigger numbers correspond to a event + grace notes.

we consider the abstract domain [0,...,MAX_GRACE] for the values of:

- i) the pre and post, and
- ii) the number of g.n. + note

for i) the meaning is

- 0: 0
- 1: 1
- ...
- MAX_GRACE: >= MAX_GRACE

for ii) the meaning is

- 0: tie (no event)
- 1: 1 note
- 2: 1 gn + 1 note
- ...
- MAX_GRACE: >= MAX_GRACE-1 gn + 1 note (appogiature) = all other cases

an abstract label is a label in abstract domain.

a concrete label is a positive integer.

an abstract label a is an abstraction of a concrete label b if

- either b <= MAX_GRACE and a = b
- or b > MAX_GRACE and a = MAX_GRACE.

  Todo TBR the class Label is not used (except for static members)

The documentation for this class was generated from the following files:

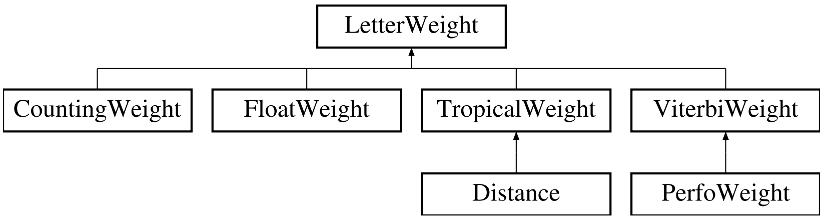
- src/output/Label.hpp
- src/output/Label.cpp

15.38 LetterWeight Class Reference

abstract class for concrete weight values. Every concrete weight domain must be a derived class of LetterWeight.

```
#include <Weight.hpp>
```

Inheritance diagram for LetterWeight:



Public Member Functions

- LetterWeight ()
- **LetterWeight** (const LetterWeight &)
- virtual ∼LetterWeight ()

  virtual destructor to ensure correct destruction of derived objects through a pointer to base Weight object.
- LetterWeight & **operator=** (const LetterWeight &)
- virtual LetterWeight ∗ **clone** () const =0
- virtual Weight make (double v) const =0

  factory.
- virtual Weight get_zero () const =0

  return the neutral element for add (absorbing element for mult) wrapped in a Weight.
- virtual Weight get_one () const =0

  return the neutral element for mult wrapped in a Weight.
- virtual double **norm** () const =0
- virtual void **scalar** (double)=0
- virtual void invert ()=0

  multiplicative inverse, for semifields.
- virtual bool zero () const

  this letterweight is neutral element for add (absorbing element for mult).
- virtual bool one () const

  this letterweight is neutral element for mult.
- virtual bool **hasType** (std::string) const =0

Protected Member Functions

- virtual bool equal (const LetterWeight ∗) const

  binary operators are defined only between descendant LetterWeights of same typeid.
- virtual bool smaller (const LetterWeight ∗) const

  binary operators are defined only between descendant LetterWeights of same typeid.
- virtual void add (const LetterWeight ∗)

  binary operators are defined only between descendant LetterWeights of same typeid.
- virtual void mult (const LetterWeight ∗)

  binary operators are defined only between descendant LetterWeights of same typeid.
- virtual void **print** (std::ostream &o) const

Friends

- class **Weight**

15.38.1 Detailed Description

abstract class for concrete weight values. Every concrete weight domain must be a derived class of LetterWeight.

15.38.2 Constructor & Destructor Documentation

15.38.2.1 LetterWeight()

```
LetterWeight::LetterWeight ( )  [inline]
```

Warning

should not happen.

15.38.3 Member Function Documentation

15.38.3.1 make()

```
virtual Weight LetterWeight::make (
             double v ) const  [pure virtual]
```

factory.

Returns

a weight of same type as this letter, initialized with given value.

Implemented in PerfoWeight, CountingWeight, ViterbiWeight, Distance, TropicalWeight, and FloatWeight.

The documentation for this class was generated from the following files:

- src/weight/Weight.hpp
- src/weight/Weight.cpp

15.39 ScoreModel::Measure Class Reference

```
#include <Measure.hpp>
```

Public Member Functions

- Measure (int id, ScoreMeter meter)
- int getId () const
- Duration getDuration () const
- ∼Measure ()

15.39.1 Detailed Description

An measure is a container of fixed duration, defined by a ScoreMeter

15.39.2 Constructor & Destructor Documentation

15.39.2.1 Measure()

```
ScoreModel::Measure::Measure (
              int id,
              ScoreMeter meter )
```

Main constructor.

15.39.2.2 ∼Measure()

```
ScoreModel::Measure::∼Measure ( )
```

Destructor

15.39.3 Member Function Documentation

15.39.3.1 getId()

```
int ScoreModel::Measure::getId ( ) const  [inline]
```

Get the measure id

15.39.3.2 getDuration()

```
Duration ScoreModel::Measure::getDuration ( ) const  [inline]
```

Get the duration of a measure

The documentation for this class was generated from the following files:

- src/scoremodel/Measure.hpp
- src/scoremodel/Measure.cpp

15.40 MEI Class Reference

```
#include <MEI.hpp>
```

Public Member Functions

- MEI ()
- void createFromScore (const ScoreModel::Score &s)
- void createScoreDef (const ScoreModel::Score &s)
- void findStartingBeam (const ScoreModel::Score &s)
- void writeInFile (const string fname)
- std::pair< string, int > chooseClef (std::pair< Pitch, Pitch > range)
- ∼MEI ()

Static Public Member Functions

- static Note ∗ **makeNote** (const ScoreModel::Note ∗noteEvent)
- static TupletSpan ∗ **makeTupletSpan** (const ScoreModel::Tuplet ∗tuplet)
- static Tie ∗ **makeTie** (const ScoreModel::Tie ∗tie)

15.40.1 Detailed Description

The main MEI class.

Takes a Rhythm tree as input, and creates a MEI score

15.40.2 Member Function Documentation

15.40.2.1 findStartingBeam()

```
void MEI::findStartingBeam (
            const ScoreModel::Score & s )
```

Find whether a beam start on an event

The documentation for this class was generated from the following files:

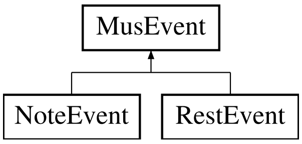
- src/output/MEI.hpp
- src/output/MEI.cpp

15.41 MusEvent Class Reference

input events

```
#include <MusEvent.hpp>
```

Inheritance diagram for MusEvent:



Public Member Functions

- **MusEvent** (int nb=EVENTNB_UNKNOWN)
- **MusEvent** (const MusEvent &)
- virtual MusEvent $*$ **clone** () const =0
- virtual bool **isRest** () const =0
- virtual bool **isNote** () const =0
- virtual void **print** (std::ostream &o) const =0

Public Attributes

- int **number**

Static Public Attributes

- static const unsigned int **UNDEF_VELOCITY** = 128

Friends

- std::ostream & **operator**$<<$ (std::ostream &o, const MusEvent &rhs)

15.41.1 Detailed Description

input events

input interface to MIDI, OpenMusic, MEI etc keep track of input event list event are not stored internaly (in WTA). we just preserve the order. and remap afterwards to input event list (with dfs). abstract class to built polymorphic event lists (in input or output).

can be downcasted to descendant class with dynamic_cast for using particular operations

The documentation for this class was generated from the following files:

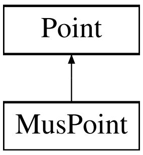
- src/segment/MusEvent.hpp
- src/segment/MusEvent.cpp

15.42 MusPoint Class Reference

Point extended with mutable musical time date and duration (expressed in fraction of bars).

```
#include <MusPoint.hpp>
```

Inheritance diagram for MusPoint:



Public Member Functions

- **MusPoint** (MusEvent ∗e, double rdate, double rdur, bool on, long link=MUSPOINTREF_NULL, Rational mdate=MUSTIME_UNKNOWN, Rational mduration=MUSTIME_UNKNOWN)
- MusPoint (const Point &p, Rational mdate=MUSTIME_UNKNOWN, Rational mduration=MUSTIME_UNKNOWN)

  copy of point.
- MusPoint (const MusPoint &)

  event (if any) is cloned.
- MusPoint & operator= (const MusPoint &)

  event (if any) is cloned.
- bool **operator==** (const Point &) const
- Rational & mdate ()
- Rational & mduration ()

Protected Member Functions

- virtual void **print** (std::ostream &o) const

Protected Attributes

- Rational _mdate

  timestamp in musical time (number of bars).
- Rational **_mduration**

Friends

- std::ostream & **operator**<< (std::ostream &o, const Point &rhs)

Additional Inherited Members

15.42.1 Detailed Description

Point extended with mutable musical time date and duration (expressed in fraction of bars).

Todo redefine musical time duration as realtime duration, with links.

replace _mduration by mduration computed from linked point's date

Warning

mduration is reset in InputSegment.quantize()

15.42.2 Member Function Documentation

15.42.2.1 mdate()

```
Rational& MusPoint::mdate ( )  [inline]
```

Warning

can be modified.

15.42.2.2 mduration()

```
Rational& MusPoint::mduration ( )  [inline]
```

Warning

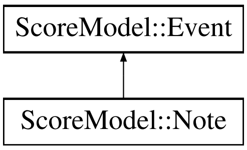
only for polyphonic events.
can be modified.

The documentation for this class was generated from the following files:

- src/segment/MusPoint.hpp
- src/segment/MusPoint.cpp

15.43 ScoreModel::Note Class Reference

Inheritance diagram for ScoreModel::Note:



Public Member Functions

- **Note** (Duration duration, Pitch p)
- virtual Event ∗ **clone** ()
- virtual bool **isRest** () const
- virtual bool **isNote** () const
- Pitch **pitch** () const
- virtual void **print** (std::ostream &o) const

Protected Attributes

- Pitch **_pitch**

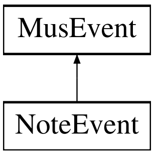
Additional Inherited Members

The documentation for this class was generated from the following files:

- src/scoremodel/Note.hpp
- src/scoremodel/Note.cpp

15.44 NoteEvent Class Reference

Inheritance diagram for NoteEvent:



Public Member Functions

- NoteEvent (unsigned int vel=MusEvent::UNDEF_VELOCITY, int nb=EVENTNB_UNKNOWN)

    unpitched note (drums).
- NoteEvent (Pitch p, unsigned int vel=MusEvent::UNDEF_VELOCITY, int nb=EVENTNB_UNKNOWN)

    pitched note.
- NoteEvent (unsigned int p, unsigned int vel=MusEvent::UNDEF_VELOCITY, int nb=EVENTNB_UNKNOWN)

    pitched note with MIDI pitch in 0..127.
- **NoteEvent** (const NoteEvent &)
- virtual MusEvent ∗ **clone** () const
- virtual bool **isRest** () const
- virtual bool **isNote** () const
- bool **unpitched** () const
- unsigned int **velocity** () const
- Pitch & pitch ()

    can be modified.
- virtual void **print** (std::ostream &o) const

Protected Attributes

- Pitch **_pitch**
- unsigned int _velocity

    MIDI velocity.

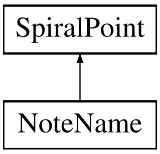
Additional Inherited Members

The documentation for this class was generated from the following files:

- src/segment/MusEvent.hpp
- src/segment/MusEvent.cpp

15.45 NoteName Struct Reference

Inheritance diagram for NoteName:



Public Member Functions

- NoteName (char n, float alt, int id)

  notename object from name, alteration and index.
- **NoteName** (const NoteName &rhs)
- NoteName & **operator=** (const NoteName &rhs)
- bool **unknown** () const

Static Public Member Functions

- static const NoteName & ofkey (int k)

  ref to a NoteName in table synonyms. ∗/
- static const NoteName & closest (unsigned int pitch, const SpiralPoint &p)

  note name (ref in table synonyms) corresponding to given midi pitch and closest to given point.

Public Attributes

- char name

  note name. 'A' to 'G' or Pitch::UNDEF_NOTE_NAME
- float alteration

  note alteration.
- int index

  position in the line of fifths relative to C

Static Public Attributes

- static const int **UNDEF_NOTE_INDEX** = 99
- static const double h = 1.0

  z distance between two successive points of the spiral (one fifth apart).
- static const double r = std::sqrt(7.5) ∗ h

  radius of the cylinder in which the spiral is embedded.
- static const NoteName **synonyms** [12][3]

Friends

- bool **operator==** (const NoteName &, const NoteName &)
- bool **operator!=** (const NoteName &, const NoteName &)
- std::ostream & **operator<<** (std::ostream &o, const NoteName &rhs)

15.45.1 Member Data Documentation

15.45.1.1 name

```
char NoteName::name
```

note name. 'A' to 'G' or Pitch::UNDEF_NOTE_NAME

See also

same has in class Pitch

15.45.1.2 alteration

```
float NoteName::alteration
```

note alteration.

in [-2.0, 2.0] where 1.0 is half tone or Pitch::UNDEF_NOTE_ALTERATION same has in class Pitch.

15.45.1.3 index

```
int NoteName::index
```

position in the line of fifths relative to C

- C has index 0 and index increases in the direction of sharps:
- G has index 1, D has index 2, F# has index 6...
- F has index -1, Bb has index -2...

values between -15 (Fbb) and 19 (B##)

TBC: it is redundant with name and alteration maybe should replace them?

The documentation for this struct was generated from the following files:

- src/segment/Spiral.hpp
- src/segment/Spiral.cpp

15.46 OMRhythmTree Class Reference

Public Member Functions

- **OMRhythmTree** (Rational lab, bool tied=false)
- **OMRhythmTree** (const RhythmTree $*$, Rational dur=Rational(1))
- bool **leaf** () const
- bool **inner** () const
- Rational **label** ()
- bool **tie** ()
- size_t **size** () const
- OMRhythmTree $*$ **child** (size_t) const
- void **add** (OMRhythmTree $*$)
- string **to_string** () const

Friends

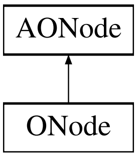
- std::ostream & **operator**$<<$ (std::ostream &, const OMRhythmTree &)

The documentation for this class was generated from the following files:

- src/output/OMRT.hpp
- src/output/OMRT.cpp

15.47 ONode Class Reference

Inheritance diagram for ONode:



Public Member Functions

- **ONode** (size_t a)
- void **add** (const ANode &n)

Public Attributes

- std::vector$<$ ANode $>$ **children**

Additional Inherited Members

The documentation for this class was generated from the following file:

- src/schemata/SubdivisionSchema.hpp

15.48 Onsets Class Reference

sequence of onsets used for merge of duration lists.

```
#include <Onsets.hpp>
```

Public Member Functions

- Onsets (const DurationList &)

  the list of onsets defined by the given duration list (IOI's) the first onset is 0.
- void **add** (Rational r)
- DurationList ioi () const

  the list of IOI associated to this list of onsets.

Friends

- const Onsets operator+ (const Onsets &lhs, const Onsets &rhs)

  ordered merge

15.48.1 Detailed Description

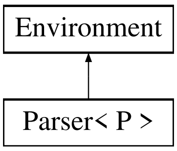
sequence of onsets used for merge of duration lists.

The documentation for this class was generated from the following files:

- src/output/Onsets.hpp
- src/output/Onsets.cpp

15.49 Parser< P > Class Template Reference

Inheritance diagram for Parser< P >:



Public Member Functions

- **Parser** (WTA *a, InputSegment *s=NULL)
- size_t **resolution** ()
- virtual size_t **addRuns** (Atable< P > *, const P &, Record< P > *)=0
- virtual void **printobest** (std::ostream &o, Atable< P > *table, const P &) const
- virtual void **printobestRun** (std::ostream &o, Atable< P > *table, Run< P > *r) const

Public Attributes

- WTA ∗ **wta**
- Environment ∗ **input**

Protected Member Functions

- virtual size_t **addWTARuns** (Atable< P > ∗, const P &, Record< P > ∗)

Protected Attributes

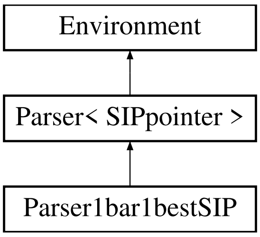
- size_t **_res**

The documentation for this class was generated from the following file:

- src/parsers/Parser.hpp

15.50 Parser1bar1bestSIP Class Reference

Inheritance diagram for Parser1bar1bestSIP:



Public Member Functions

- **Parser1bar1bestSIP** (WTA ∗a, InputSegment ∗s, bool ordering=false)
- virtual size_t **addRuns** (Atable< SIPpointer > ∗, const SIPpointer &, Record< SIPpointer > ∗)
- RhythmTree ∗ **bestTree** (pre_t pre=0, pre_t post=0) const
- Run< SIPpointer > ∗ **bestRun** (pre_t pre=0, pre_t post=0) const
- Weight **bestWeight** (pre_t pre=0, pre_t post=0) const
- void **printBest** (std::ostream &o, pre_t pre=0, pre_t post=0) const
- size_t **demo** (const string schema_file, const string input_file, pre_t pre=0, pre_t post=0)

Public Attributes

- Table< SIPpointer, Brecord< SIPpointer >, SIPpointerHasher > ∗ **table**

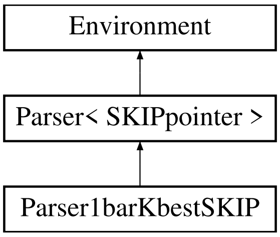
Additional Inherited Members

The documentation for this class was generated from the following files:

- src/parsers/Parser1bar1bestSIP.hpp
- src/parsers/Parser1bar1bestSIP.cpp

15.51 Parser1barKbestSKIP Class Reference

Inheritance diagram for Parser1barKbestSKIP:



Public Member Functions

- **Parser1barKbestSKIP** (WTA ∗a, InputSegment ∗s, bool ordering=false)
- virtual size_t **addRuns** (Atable< SKIPpointer > ∗, const SKIPpointer &, Record< SKIPpointer > ∗)
- RhythmTree ∗ **bestTree** (size_t k=1, pre_t pre=0, pre_t post=0) const
- Run< SKIPpointer > ∗ **bestRun** (size_t k=1, pre_t pre=0, pre_t post=0) const
- Weight **bestWeight** (size_t k=1, pre_t pre=0, pre_t post=0) const
- void **printBest** (std::ostream &o, size_t k=1, pre_t pre=0, pre_t post=0) const
- size_t **demo** (const string schema_file, const string input_file, pre_t pre=0, pre_t post=0, size_t k=1)

Public Attributes

- Table< SKIPpointer, Krecord< SKIPpointer >, SKIPpointerHasher > ∗ **table**

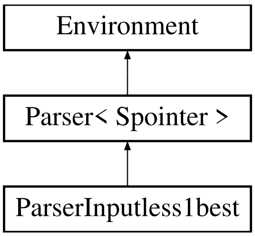
Additional Inherited Members

The documentation for this class was generated from the following files:

- src/parsers/Parser1barKbestSKIP.hpp
- src/parsers/Parser1barKbestSKIP.cpp

15.52 ParserInputless1best Class Reference

Inheritance diagram for ParserInputless1best:



Public Member Functions

- **ParserInputless1best** (WTA *a, bool ordering)
- virtual size_t **addRuns** (Atable< Spointer > *, const Spointer &, Record< Spointer > *)
- RhythmTree * **best** ()
- Weight **bestWeight** ()
- virtual void **printBest** (std::ostream &) const
- size_t **demo** (const string schema_file)

Public Attributes

- Table< Spointer, Brecord< Spointer >, SpointerHasher > * **table**

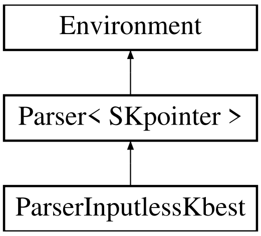
Additional Inherited Members

The documentation for this class was generated from the following files:

- src/parsers/ParserInputless1best.hpp
- src/parsers/ParserInputless1best.cpp

15.53 ParserInputlessKbest Class Reference

Inheritance diagram for ParserInputlessKbest:



Public Member Functions

- **ParserInputlessKbest** (WTA ∗a, bool ordering)
- virtual size_t **addRuns** (Atable< SKpointer > ∗, const SKpointer &, Record< SKpointer > ∗)
- RhythmTree ∗ **best** (size_t)
- Weight **bestWeight** (size_t)
- virtual void **printBest** (std::ostream &, size_t k=1) const
- size_t **demo** (const string schema_file, size_t k=1)

Public Attributes

- Table< SKpointer, Krecord< SKpointer >, SKpointerHasher > ∗ **table**

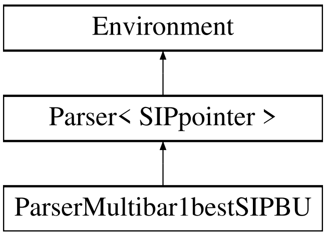
Additional Inherited Members

The documentation for this class was generated from the following files:

- src/parsers/ParserInputlessKbest.hpp
- src/parsers/ParserInputlessKbest.cpp

15.54 ParserMultibar1bestSIPBU Class Reference

Inheritance diagram for ParserMultibar1bestSIPBU:



Public Member Functions

- **ParserMultibar1bestSIPBU** (WTA ∗a, InputSegment ∗s, bool ordering=false, pre_t pre=0, pre_t post=0, double barlen=1.0, size_t nbbars=1, ScoreModel::ScoreMeter ts=ScoreModel::ScoreMeter(1, 4))
- virtual size_t **addRuns** (Atable< SIPpointer > ∗, const SIPpointer &, Record< SIPpointer > ∗)
- size_t **nbbars** () const
- Run< SIPpointer > ∗ **getBar** (size_t) const
- SIPpointer **getTarget** (size_t) const
- ScoreModel::ScoreMeter **getTimeSig** (size_t) const
- size_t **demo** (const std::string schema_file, const std::string input_file, const std::string output_file="", Rational barbeat=Rational(1))

Public Attributes

- Table< SIPpointer, Brecord< SIPpointer >, SIPpointerHasher > ∗ **table**
- const SIPpointer **endmarker_bot**
- const SIPpointer **endmarker_top**

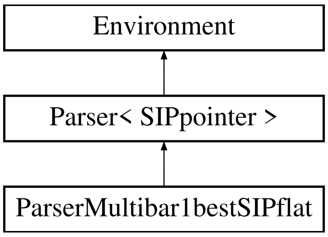
Additional Inherited Members

The documentation for this class was generated from the following files:

- src/parsers/ParserMultibar1bestSIPBU.hpp
- src/parsers/ParserMultibar1bestSIPBU.cpp

15.55 ParserMultibar1bestSIPflat Class Reference

Inheritance diagram for ParserMultibar1bestSIPflat:



Public Member Functions

- **ParserMultibar1bestSIPflat** (WTA *a, InputSegment *s, double barlen, bool ordering=false, pre_t pre=0, pre_t post=0)
- **ParserMultibar1bestSIPflat** (WTA *a, InputSegment *s, double barlen_min, double barlen_max, size_t nbsteps, bool ordering=false, pre_t pre=0, pre_t post=0)
- virtual size_t **addRuns** (Atable< SIPpointer > *, const SIPpointer &, Record< SIPpointer > *)
- size_t **nbbars** () const
- Run< SIPpointer > * **getBar** (size_t) const
- SIPpointer **getTarget** (size_t) const
- size_t **demo** (const std::string schema_file, const std::string input_file, const std::string output_file="", Rational barbeat=Rational(1))

Static Public Member Functions

- static double **barlen** (double tempo, size_t beatsperbar)

Public Attributes

- Table< SIPpointer, Brecord< SIPpointer >, SIPpointerHasher > * **table**
- const SIPpointer **endmarker_top**

Additional Inherited Members

The documentation for this class was generated from the following files:

- src/parsers/ParserMultibar1bestSIPflat.hpp
- src/parsers/ParserMultibar1bestSIPflat.cpp

15.56 ScoreModel::Part Class Reference

Public Member Functions

- Part (Score &score, std::string name)
- std::string **getName** () const
- Score & **getScore** () const
- void addVoice (Voice ∗voice)
- Voice ∗ getVoice (std::string voiceName)
- std::vector< Voice ∗ > getVoices ()
- ∼Part ()

15.56.1 Constructor & Destructor Documentation

15.56.1.1 Part()

```
ScoreModel::Part::Part (
            Score & score,
            std::string name )
```

Main constructor. Builds an empty part

15.56.1.2 ∼Part()

```
ScoreModel::Part::∼Part ( )
```

Destructor

15.56.2 Member Function Documentation

15.56.2.1 addVoice()

```
void ScoreModel::Part::addVoice (
            Voice ∗ voice )
```

Add a voice

15.56.2.2 getVoice()

```
Voice ∗ ScoreModel::Part::getVoice (
            std::string voiceName )
```

Get a voice from its name

15.56.2.3 getVoices()

```
std::vector<Voice*> ScoreModel::Part::getVoices ( )  [inline]
```

Get all voices

The documentation for this class was generated from the following files:

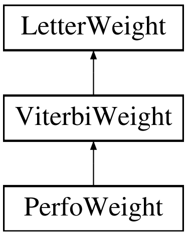
- src/scoremodel/Part.hpp
- src/scoremodel/Part.cpp

15.57 PerfoWeight Class Reference

extention of ViterbiWeight with a model of performance.

```
#include <PerformanceModel.hpp>
```

Inheritance diagram for PerfoWeight:



Public Member Functions

- **PerfoWeight** (double v)
- PerfoWeight (const InputSegment ∗s, const AlignedInterval ∗p, pre_t pre=0, pre_t post=0)

  probability of positions in the given alignement in the interval defined by the given path.
- PerfoWeight & **operator=** (const PerfoWeight &)
- PerfoWeight & operator= (const LetterWeight &rhs)
- PerfoWeight ∗ **clone** () const
- virtual Weight make (double v) const

  factory.
- virtual bool hasType (std::string code) const

Static Public Member Functions

- static void **set_sigma2** (double)

Friends

- std::ostream & **operator**<< (std::ostream &o, const PerfoWeight &rhs)

Additional Inherited Members

15.57.1 Detailed Description

extention of ViterbiWeight with a model of performance.

compute probabilities of alignement of input points to a score following a truncated Gaussian distribution with parameters mu (default 0) and sigma (default 1) and is truncated on the interval [a,b] (values in samples) and shifted.

15.57.2 Member Function Documentation

15.57.2.1 make()

```
virtual Weight PerfoWeight::make (
            double v ) const  [inline], [virtual]
```

factory.

Returns

a weight of same type as this letter, initialized with given value.

Reimplemented from ViterbiWeight.

15.57.2.2 hasType()

```
virtual bool PerfoWeight::hasType (
            std::string code ) const  [inline], [virtual]
```

Warning

type code is still "ViterbiWeight"

Reimplemented from ViterbiWeight.

The documentation for this class was generated from the following files:

- src/weight/PerformanceModel.hpp
- src/weight/PerformanceModel.cpp

15.58 Pitch Class Reference

internal representation of a pitch value.

```
#include <Pitch.hpp>
```

Public Types

- enum **PitchUnit** { **MIDI**, **MIDICENT** }

Public Member Functions

- Pitch ()

  undef pitch value.
- Pitch (char name, float alt=0.0, int oct=0)

  construct pitch from name+alteration+octave.
- Pitch (unsigned int pitch, PitchUnit u=MIDI)

  construct note from MIDI pitch
- **Pitch** (const Pitch &)
- Pitch & **operator=** (const Pitch &)
- bool **operator==** (const Pitch &) const
- bool **undef** () const
- unsigned int midicent () const

  value in MIDIcent.
- unsigned int midi () const

  value in MIDI.

Public Attributes

- char name

  note name betwen 'A' and 'G'.
- float alteration

  alteration in [-2, 2] where 1.0 is half tone.
- int octave

  octave in -10..10.

Static Public Attributes

- static const unsigned int **UNDEF_MIDICENT** = 12800
- static const char **UNDEF_NOTE_NAME** = 'X'
- static const int **UNDEF_NOTE_OCTAVE** = 128
- static const float **UNDEF_NOTE_ALTERATION** = 11

Friends

- std::ostream & **operator**<< (std::ostream &, const Pitch &)

15.58.1 Detailed Description

internal representation of a pitch value.

can be unknown value

Todo extend conversions to MIDIcent (import OM)

The documentation for this class was generated from the following files:

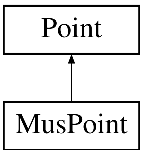
- src/segment/Pitch.hpp
- src/segment/Pitch.cpp

15.59 Point Class Reference

timestamped event.

```
#include <Point.hpp>
```

Inheritance diagram for Point:



Public Member Functions

- Point (MusEvent ∗e, double rdate, double rdur, bool on, long link=MUSPOINTREF_NULL)

  timestamped monophonic or polyphonic event.
- Point (const Point &)
- ∼Point ()
- virtual Point & operator= (const Point &)
- virtual bool **operator==** (const Point &) const
- MusEvent ∗ event () const
- double **rdate** () const
- double rduration () const

  realtime duration of polyphonic events.
- bool noteon () const

  is note on.
- bool noteoff () const

  is note off.

Public Attributes

- long linked

  link to a point in an input segment marking the end date of this point.

Protected Member Functions

- virtual void **print** (std::ostream &o) const

Protected Attributes

- MusEvent ∗ _event

  input event.
- double _rdate

  timestamp in real-time (sec).
- double _rduration

  real duration computed using the matcher's rdate.
- bool _onoff

  true if note-on, false if note-off.

Friends

- std::ostream & **operator**<< (std::ostream &o, const Point &rhs)

15.59.1 Detailed Description

timestamped event.

event extended with realtime date (in seconds) and optional [on-off] link.

every point has a realtime date (in seconds).

a point can be linked to a forward point (with a realtime date larger or equal). the link is an index in an input segment.

a linked point is also called onset or note-on. an point without link (with link = MUSPOINTREF_NULL) is called offset or note-off.

for the computation of realtime duration of points, see InputSegment.

15.59.2 Member Function Documentation

15.59.2.1 event()

```
MusEvent* Point::event ( ) const [inline]
```

Warning

can be NULL.

15.59.2.2 rduration()

```
double Point::rduration ( ) const [inline]
```

realtime duration of polyphonic events.

is computed in input segment

Todo TBR (only for backward compability)

15.59.3 Member Data Documentation

15.59.3.1 linked

```
long Point::linked
```

link to a point in an input segment marking the end date of this point.

the link is an index in an input segment structure:

- a point of segment if >= 0,
- or a floating point if < 0. if MUSPOINTREF_NULL, the duration of this point is zero.

15.59.3.2 _rduration

```
double Point::_rduration [protected]
```

real duration computed using the matcher's rdate.

Todo TBR (added for backward compatibility)

15.59.3.3 _onoff

```
bool Point::_onoff [protected]
```

true if note-on, false if note-off.

Todo TBR

The documentation for this class was generated from the following files:

- src/segment/Point.hpp
- src/segment/Point.cpp

15.60 PointedIntervalEq Struct Reference

Public Member Functions

- bool **operator()** (IntervalTree const ∗lhs, IntervalTree const ∗rhs) const

The documentation for this struct was generated from the following file:

- src/segment/IntervalHeap.hpp

15.61 PointedIntervalHash Struct Reference

Public Member Functions

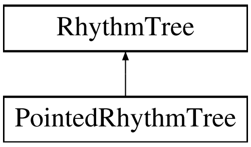
- std::size_t **operator()** (const IntervalTree ∗p) const

The documentation for this struct was generated from the following file:

- src/segment/IntervalHeap.hpp

15.62 PointedRhythmTree Class Reference

Inheritance diagram for PointedRhythmTree:



Public Member Functions

- **PointedRhythmTree** (label_t lab)
- **PointedRhythmTree** (const RhythmTree ∗, const InputSegment ∗, size_t i=0)
- size_t **next** ()

Public Attributes

- std::vector< const MusEvent ∗ > **events**

Protected Attributes

- size_t **_next**

Additional Inherited Members

The documentation for this class was generated from the following files:

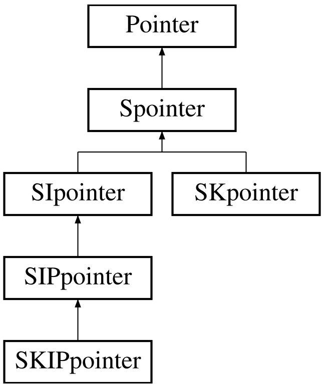
- src/output/PointedRT.hpp
- src/output/PointedRT.cpp

15.63 Pointer Class Reference

abstract class defining a signature for a class of pointer to best runs.

```
#include <Ptr.hpp>
```

Inheritance diagram for Pointer:



Public Member Functions

- virtual bool **has_S** () const
- virtual bool **has_K** () const
- virtual bool **has_I** () const
- virtual bool **has_P** () const
- virtual bool complete () const =0

  the pointer is complete i.e. all fields are set
- bool **partial** () const
- virtual size_t rank () const

  return the rank of best (k) pointed. default is 1. redefine for classes for k-best parsing.
- virtual bool divisible () const

  return wether it is worth descending (dividing) from this pointer. the result may differ according to whether this pointer has a WTA state or a Meta state. For instance:
- virtual void **incr** ()
- virtual bool compatible (const label_t, bool abstract=true) const

  return wether this pointer is compatible with the given label.
- virtual bool dummy () const =0

  return whether this pointer is a dummy pointer i.e. it was constructed with P() default false.
- virtual label_t label (const Transition &t) const =0

  return a concrete label value corresponding to this pointer when considered as a leaf position, using the label of the given transition. the given transition must be terminal.
- virtual Weight terminalWeight (const InputSegment ∗, const Transition &) const

  return the weight for a terminal Run associated to the given Transition. The transition must be terminal. This pointer must be compatible with the Transition. input segment can be NULL.
- virtual Weight innerWeight (const Transition &) const

  return the initial weight for an inner Run associated to the given Transition. the weight will have to be multiplied with all the weights of subruns. the transition must be inner. this pointer must be divisible.

15.63.1 Detailed Description

abstract class defining a signature for a class of pointer to best runs.

constructors

Every concrete subclass P (descendant) must implement the following generic constructors and class-specific operators. they are called by the templates Table and Run. Some use the encapsulator Environment.

```
P()
```

dummy ptr (unique - can not be produced by other constructors).

```
P(label_t s)
```

fake ptr containing only a label symbol to act as (singleton) body of a terminal runs. the label symbol is concrete (see Label.hpp). also used for P(state_t) -> confusion types state_t & label_t

```
P(Environment* env, const &P p, size_t a, size_t i, state_t s)
```

sub-pointer or instance as leaf s must be a wta state (e.g. initial state of wta) there are 2 cases according to a:

1. if a = 0 : construct a copy of p instanciated as a leaf with label s. i must be 0, p must be partial p must be compatible with s (s = leaf symbol in this case).
2. if a > 0 i must be in [0..a] p must be divisible p can have a meta state (meta run with a=2) or a wta state with i=0, construct the head of a run (in general a copy of p but not always) with 0 < i <= a, construct a pointer for the ith children of a run. the details and specific pre-conditions are described in every class P.

```
P(const &P p)
```

copy

```
P(const &P p, const &P p0, const &P p1)
```

instance as parent p must be partial p0 must be complete p1 must be complete Construct a copy of p instanciated as an target node of run with p0 as first child and p1 as last child.

```
P(const &P p0, const &P p1)
```

instance as next sibling p0 must be complete p1 must be partial p1 must be instanciable into a successor sibling of p0. Construct a copy of p1 instanciated as the next sibling of p0.

```
virtual P& operator= (const P& p) = 0;
virtual bool operator==(const P& p) const;
```

for using pointer as key in hash table (unordered multimap)

```
virtual bool operator<(const P& p) const;
```

for using pointer as key in search tree (multimap)

```
bool instance(const P& p)
```

return wether this ptr is an instance of p. if p is complete, then it is equality.

```
bool subsume(const P& p)
```

inverse of instance return wether p is an instance of this ptr. if this ptr is complete, then it is equality.

15.63.2 Member Function Documentation

15.63.2.1 divisible()

```
virtual bool Pointer::divisible ( ) const  [inline], [virtual]
```

return wether it is worth descending (dividing) from this pointer. the result may differ according to whether this pointer has a WTA state or a Meta state. For instance:

- for WTA state: it is not worth when this pointer corresponds to an input sub-segment not inhabited.
- for Meta state: it is not worth if this ptr corresponds to an empty segment. default true.

Reimplemented in SIpointer, and Spointer.

15.63.2.2 compatible()

```
virtual bool Pointer::compatible (
            const label_t ,
            bool abstract = true ) const  [inline], [virtual]
```

return wether this pointer is compatible with the given label.

Parameters

abstract	if flag is true (default), the label is supposed abstract (label of terminal transition). if abstract flag is false, the label is supposed concrete.

See also

Label.hpp for def. abstract/concrete labels

Returns

default return true.

Reimplemented in SIPpointer, and SIpointer.

The documentation for this class was generated from the following files:

- src/table/Ptr.hpp
- src/table/Ptr.cpp

15.64 Position Class Reference

position in a RT.

```
#include <RT.hpp>
```

Public Member Functions

- Position ()

  empty sequence = root position
- **Position** (const Position &)
- bool **empty** () const
- size_t **length** () const
- void operator+= (size_t i)

  concatenate given int to this position

Protected Member Functions

- void **print** (std::ostream &o) const

Protected Attributes

- std::vector< size_t > **_content**

Friends

- std::ostream & **operator<<** (std::ostream &o, const Position &pos)
- Position **operator+** (const Position &, const size_t &)

15.64.1 Detailed Description

position in a RT.

= sequence of integers to reach position from root.

The documentation for this class was generated from the following files:

- src/output/RT.hpp
- src/output/RT.cpp

15.65 PreState Class Reference

tmp state structure for construction of PreWTA from a WTA (base schema) casted aka state_t after construction

```
#include <PreWTA.hpp>
```

Public Member Functions

- **PreState** (state_t, pre_t pre=0, pre_t post=0)
- PreState (const PreState &)
- bool **operator==** (const PreState &s) const
- bool operator< (const PreState &s) const

  lexicographic comparison on hash value (array[5])
- state_t serialize ()

  return a state value unically associated to this PreState
- bool **compatible** (label_t label) const

Static Public Member Functions

- static bool compatible_post (state_t, const AlignedInterval ∗)

  compatible(s, al) the serialized state value s is compatible with the content of the alignment al (sub-segment of initial input corr. to an interval)

Public Attributes

- state_t ps_state
  state of base schema
- pre_t ps_pre
  guess number of points aligned to right of previous segment
- pre_t ps_post
  guess number of points aligned to right of current segment

Friends

- std::ostream & **operator**<< (std::ostream &, const PreState &)

15.65.1 Detailed Description

tmp state structure for construction of PreWTA from a WTA (base schema) casted aka state_t after construction

label (for leaves): see WTA

states (q:int, pre:int, post:int) ou label (feuille) q: state of base schema pre: guess number of points aligned to right of previous segment post: guess number of points aligned to right of current segment

The documentation for this class was generated from the following files:

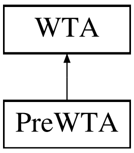
- src/schemata/PreWTA.hpp
- src/schemata/PreWTA.cpp

15.66 PreWTA Class Reference

extension of WTA where states are associated pre and post values.

```
#include <PreWTA.hpp>
```

Inheritance diagram for PreWTA:



Public Member Functions

- PreWTA (const WTA &)
  construction from WTA (base schema)
- virtual bool **hasType** (std::string code) const
- virtual state_t initial (pre_t pre=0, pre_t post=0) const
  initial(pre, port) returns state representing the whole segment, with pre points of the previous segment aligned to the left and post points of the current segment aligned to the right (i.e. to the left of the next segment)

Static Public Member Functions

- static pre_t pre (state_t)

  access to original components of new PreWTA states
- static pre_t **post** (state_t)
- static state_t **state** (state_t)

Friends

- std::ostream & **operator**<< (std::ostream &, const PreWTA &)

Additional Inherited Members

15.66.1 Detailed Description

extension of WTA where states are associated pre and post values.

- post is a number of points
- pre is the post of the previous sibling

Warning

deprecated

extension of a given schema (WTA) with pre and post information in states: during a computation, the current state is associated to an input segment,

- pre value is an abstraction of the number of points in the previous segment aligned to the left of the current segment.
- post value is an abstraction of the number of points aligned to the right of the current segment. cf. Label↩ ::abstract for the abstraction domain

top-down construction of the transition table, principle:

- pre propagate from a node to its leftmost child
- post propagte from a node to it rightmost child
- for 2 states s1, s2 at successive siblings, it holds that s1.post = s2.pre

given q state of schema, k <= max{ n | q -> q1,...,qn | w transition of schema} mright(q, k) = # point d'input dans la derniere 2k partie de p

we start with states <q0, pre0, post0> in a queue q0 = initial state of the schema pre0 arbitrary (input of problem = previous tree). ctypically 0. post0 in [0, MAX_GRACE]

for all state <q, pre, post> taken from the queue

for all final transition q -> s | w of the schema we add the transition <q, pre, post> -> <s, pre, post> | w such that the value of s abstracts the possible values of pre + lalign(path) for any path.

for all inner transition q -> (q1,...,qk) | w in schema (k > 1) for all post in [0..MAX_GRACE] we add the transitions <q, pre, post> -> (<q1, pre1, post1>,..., <qk, prek, postk>) | w such that

- postk = post
- pre1 = pre
- for all 1 ≤ i < k, posti = prei+1 we also add the states <q1, pre1, post1>,..., <qk, prek, postk> in the queue

The documentation for this class was generated from the following files:

- src/schemata/PreWTA.hpp
- src/schemata/PreWTA.cpp

15.67 QDate Class Reference

quantified onset values expressed in number of samples.

```
#include <QDate.hpp>
```

Public Member Functions

- **QDate** (size_t blocs, size_t rel)
- **QDate** (const QDate &)
- virtual QDate & **operator=** (const QDate &)
- virtual QDate ∗ **clone** () const
- size_t bloc () const

  number of bloc of length RES.
- size_t relative () const

  quantified date (samples) modulo RES (date in last bloc).
- Rational absolute (size_t res) const

  quantified date as rational value.
- void **print** (std::ostream &) const
- void print (std::ostream &, size_t) const

  fractional print using resolution value.

Protected Attributes

- size_t _quotient

  date in samples / RESOLUTION = bloc number
- size_t _modulo

  date in samples modulo RESOLUTION

Friends

- std::ostream & **operator**<< (std::ostream &, const QDate &)

15.67.1 Detailed Description

quantified onset values expressed in number of samples.

the value of RESOLUTION (total number of samples) is not stored in objects of this class. it should be the same for each date created.

The documentation for this class was generated from the following files:

- src/output/QDate.hpp
- src/output/QDate.cpp

15.68 Rational Class Reference

class of rational numbers

```
#include <Rational.hpp>
```

Public Member Functions

- Rational (long n, long d=1)

    default constructor
- Rational (const Rational &rhs)

    copy constructor
- long **numerator** (void) const
- long **denominator** (void) const
- bool **null** (void) const
- bool **integral** (void) const
- Rational & operator= (const Rational &rhs)

    assignment operators
- Rational & **operator=** (long rhs)
- Rational **operator+** (void) const
- Rational **operator-** (void) const
- Rational **invert** (void) const
- const Rational & **operator+=** (const Rational &rhs)
- const Rational & **operator-=** (const Rational &rhs)
- const Rational & **operator*=** (const Rational &rhs)
- const Rational & **operator/=** (const Rational &rhs)
- const Rational & **operator+=** (long rhs)
- const Rational & **operator-=** (long rhs)
- const Rational & **operator*=** (long rhs)
- const Rational & **operator/=** (long rhs)
- const Rational & **operator++** ()
- const Rational **operator++** (int)
- const Rational & **operator--** ()
- const Rational **operator--** (int)
- void printint (std::ostream &) const

    print in format int+rat

15.68.1 Detailed Description

class of rational numbers

The documentation for this class was generated from the following files:

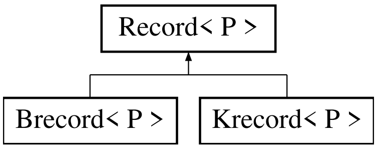
- src/general/Rational.hpp
- src/general/Rational.cpp

15.69 Record< P > Class Template Reference

abstract class describing the basic functionalities of a record.

```
#include <Record.hpp>
```

Inheritance diagram for Record< P >:



Public Member Functions

- **Record** (const P &, RunCompare< P >)
- virtual void add (Run< P > ∗)=0

  add a run to the record.
- virtual Run< P > ∗ best (Atable< P > ∗table, size_t k=1)=0

  returns the k-th best run of the record
- const P & **key** ()
- virtual bool **empty** () const =0

Public Attributes

- unsigned int state

  state - possible values: 0 : empty (record just created no run was stored) 1 : add(_key) was not called but add(p) was called for p partial and subsuming _key 2 : add(_key) was called 3 : some run has been stored but we are not in 1 or 2. should not happen.

Protected Member Functions

- bool **valid** (Run< P > ∗)

Protected Attributes

- P _key

  copy of the key associated to the record in container.
- RunCompare< P > _comp

  comparison function.
- size_t _nb_cand

  number of candidate bests constructed.
- size_t _nb_best_rejected

  number of best not added to the list because of optimization filters.

15.69.1 Detailed Description

template<class P>
class Record< P >

abstract class describing the basic functionalities of a record.

each record is associated to a Ptr it can be filled with add and can be interrogating with best, for retrieving the best runs for the associated Ptr.

when uncomplete runs are added to the record (either by the record or from outside) their weight must be computed using a table.

there are 3 similar kinds of Run∗ that should not be added in record: TBC should not be returned by best on the record ?

- NULL ptr to Run
- ptr to Run with unknown weight (i.e. weight with NULL letter) that case includes null runs.
- ptr to Run with weight == zero (acc. to test zero()). a Run not in these 3 case is called valid.

[update] the runs with weight zero (still invalid) can be added to records but an error message is displayed (for debugging).

15.69.2 Member Function Documentation

15.69.2.1 best()

```
template<class P >
virtual Run<P>* Record< P >::best (
            Atable< P > * table,
            size_t k = 1 )  [pure virtual]
```

returns the k-th best run of the record

Parameters

table	can be used to compute weights of new runs.
k	rank (as in k-best)

Implemented in Krecord< P >, and Brecord< P >.

15.69.3 Member Data Documentation

15.69.3.1 state

```
template<class P >
unsigned int Record< P >::state
```

state - possible values: 0 : empty (record just created no run was stored) 1 : add(_key) was not called but add(p) was called for p partial and subsuming _key 2 : add(_key) was called 3 : some run has been stored but we are not in 1 or 2. should not happen.

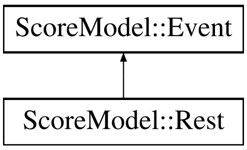
The state is not changed inside the Record class. It is changed by callers (table.add).

The documentation for this class was generated from the following files:

- src/segment/Environment.hpp
- src/table/Record.hpp

15.70 ScoreModel::Rest Class Reference

Inheritance diagram for ScoreModel::Rest:



Public Member Functions

- **Rest** (Duration duration)
- virtual Event ∗ **clone** ()
- virtual bool **isRest** () const
- virtual bool **isNote** () const
- virtual void **print** (std::ostream &o) const

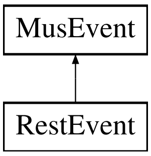
Additional Inherited Members

The documentation for this class was generated from the following files:

- src/scoremodel/Rest.hpp
- src/scoremodel/Rest.cpp

15.71 RestEvent Class Reference

Inheritance diagram for RestEvent:



Public Member Functions

- **RestEvent** (int nb=EVENTNB_UNKNOWN)
- **RestEvent** (const RestEvent &)
- virtual MusEvent ∗ **clone** () const
- virtual bool **isRest** () const
- virtual bool **isNote** () const
- virtual void **print** (std::ostream &o) const

Additional Inherited Members

The documentation for this class was generated from the following files:

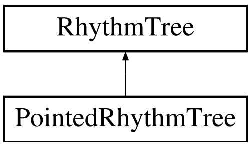
- src/segment/MusEvent.hpp
- src/segment/MusEvent.cpp

15.72 RhythmTree Class Reference

Rhythm Trees.

```
#include <RT.hpp>
```

Inheritance diagram for RhythmTree:



Public Member Functions

- RhythmTree ()

  empty inner tree (not terminal)
- RhythmTree (label_t lab)

  single leaf rhythm tree (terminal tree)
- RhythmTree (const string &)

  extract RT from string description
- bool terminal () const

  single node tree
- bool **inner** () const
- size_t arity () const

  arity of root node (0 for terminal tree)
- RhythmTree ∗ child (size_t i) const

  return the ith child of this tree
- label_t label () const

  label for terminal node
- bool continuation () const

label of terminal node is a continuation

- bool single_event () const

  label of terminal node is a single event (1 note / rest, no grace note).
- size_t nbgn () const

  number of grace notes in this terminal node.
- bool reducible () const

  this tree contains a subtree of the form.
- void add (RhythmTree ∗)

  add a subtree.
- string lily (int depth, bool tie=false) const

  LilyPond format.
- string lilydot (int depth)

  LilyPond format with dots.
- string APTED () const

  format for Tree Edit Distance Salzburg library.
- string **to_string** () const

Static Public Attributes

- static bool dot_flag = false

  global variable set if a dot is added in lilydot.

Protected Member Functions

- bool tail_redex () const

  inner and the children list is of the form.
- bool tail_reducible () const

  inner and one of the children 1..a is reducible.
- bool tie () const

  return whether this tree is a continuation (a leaf).
- bool binary () const

  return whether this tree is binary.
- bool tied () const

  return whether the leftmost innermost leaf is a tie (continuation).
- bool second_tied () const

  return whether this tree is binary and the second child is tied.
- bool dot_after () const

  return whether this tree is binary and the left son is a dot (continuation after the dotted note).
- bool dot_before () const

  return whether this tree is binary and the right son is a dot (continuation before the dotted note).
- string lilydot (int depth, bool tie, bool dot, bool ignore_first, bool ignore_second)

  LilyPond format with dots.

Protected Attributes

- long _label

  for leaves, it is the positive integer stored in the leaf; for inner tree, it is a negative integer.
- std::vector< RhythmTree ∗ > **_children**

Friends

- std::ostream & **operator**<< (std::ostream &, const RhythmTree &)

15.72.1 Detailed Description

Rhythm Trees.

for the value of leaf labels

See also

WTA.hpp
Label.hpp

15.72.2 Member Data Documentation

15.72.2.1 _label

```
long RhythmTree::_label  [protected]
```

for leaves, it is the positive integer stored in the leaf; for inner tree, it is a negative integer.

long int : for comparison with state_t = size_t = unsigned long (=label of terminal Run)

The documentation for this class was generated from the following files:

- src/output/RT.hpp
- src/output/APTED.cpp
- src/output/LilyPond.cpp
- src/output/RT.cpp

15.73 Run< P > Class Template Reference

a run is a compact representation of parse trees as a tuple of pointers to subruns.

```
#include <Rune.hpp>
```

Public Member Functions

- Run ()

  construct a null run (special)
- Run (Weight w)

  Run with empty body and given weight. the run is marked as meta. the first PartialorUpdate child is 0.
- Run (Environment ∗, const P &, const Transition &)

  WTA Run with head the given pointer. the given pointer must be a wta ptr. the body is build according to the transition t.
- Run (const Run< P > &)

  copy.
- Run (const Run< P > &r, size_t i)

  copy r and increase rank of pointer number i.
- Run (const Run< P > &r, const P &p)

  copy/update constructor
- Run< P > & **operator=** (const Run< P > &)
- bool **operator==** (const Run< P > &s) const
- bool null () const

  this run is null - constructed with Run().
- bool terminal () const

  this run is terminal (leaf).
- bool inner () const

  this run is inner.
- bool meta () const

  meta run: inner, binary and second child is a meta state.
- bool allcomplete () const

  all ptr in children list are complete.
- bool complete () const

  all ptr in children list are complete and the weight of this run has been evaluated.
- bool **partial** () const
- label_t label () const

  this run must be terminal
- size_t arity () const

  return the number of children of this Run: = 0 in the case of terminal run

  1 for inner run

  = 2 for meta run
- bool **filter** ()
- const P & operator[ ] (size_t i) const

  return the ith subrun of this run.
- const P & first () const

  first children.
- const P & last () const

  last children.
- const P & firstPartialorUpdate () const

  index of first children which is either either
- void insert (const P &)

  append the given ptr at the end of body.
- void update (const Weight &w, const DurationList &dl=DurationList())

  update the weight and duration lists of this run with given weight and duration list.

Public Attributes

- Weight weight

    current weight. totally evaluated when evaluated() = true.

- DurationList duration

    list of relative durations.

Friends

- std::ostream & **operator**<< (std::ostream &o, const Run< P > &r)

15.73.1 Detailed Description

template<class P>
class Run< P >

a run is a compact representation of parse trees as a tuple of pointers to subruns.

a run stores

- a list of children represented by pointers (template type) compatible with a transition
- a weight (to evaluate)
- a temporary weight value (initialy the weight of the parent transition)
- a list of relative durations.

a run can be of 3 kinds:

- null run:
    - unknown current weight,
    - unknown tmp weight,
    - no children,
    - empty duration list.
- terminal (leaf) run
    - created from terminal (length 1) parent transition:
    - current weight unknown or current weight known (evaluated),
    - tmp weight known,
    - 1 child : fake pointer containing as state the transition label (and rank 0 if the pointer class has a rank)
    - duration list with single continuation or single event preceeded graces notes
- inner run
    - created from inner (length > 1) parent transition:
    - current weight unknown or current weight known (evaluated)
    - tmpt weight known
    - nb children = length parent transition
    - duration list == empty (unknown) or not (evaluated).

Todo suppr. null runs

15.73.2 Constructor & Destructor Documentation

15.73.2.1 Run() [1/4]

```
template<class P>
Run< P >::Run (
            Weight w )
```

Run with empty body and given weight. the run is marked as meta. the first PartialorUpdate child is 0.

Parameters

w	must not be unknown weight .

Warning

the body must be completed with insert().

15.73.2.2 Run() [2/4]

```
template<class P>
Run< P >::Run (
            Environment * ,
            const P & ,
            const Transition &  )
```

WTA Run with head the given pointer. the given pointer must be a wta ptr. the body is build according to the transition t.

- terminal run if t is terminal,
  - the given ptr must be compatible with the transition's label.
  - singleton children list with fake ptr containing only label.
  - the run is complete.
  - the weight of run is set to a combination of transition's weight and a distance returned by terminalWeight.
- inner run if t is inner (using states in the body of t)
  - children list contains pointers of type P to the 1-best runs for the given transition for transition (s1,...,sn), the 1-best is (<s1,1>,...,<sn,1>).
  - the ptrs in body are registered.
  - the run is partial.
  - the weight of run is set to innerWeight and must be mult. by weights of subruns.
- null run when it is not possible to construct one of the children.

15.73.2.3 Run() [3/4]

```
template<class P>
Run< P >::Run (
            const Run< P > & r,
            size_t i )
```

copy r and increase rank of pointer number i.

Parameters

r	must be inner.
i	(child) must have rank, index i must be between 0 and arity of r - 1. the run is reset (partial):

Warning

> the weight and duration list of the run must be recomputed (the weight is reset to the weight of creator transition).

15.73.2.4 Run() [4/4]

```
template<class P>
Run< P >::Run (
            const Run< P > & r,
            const P & p )
```

copy/update constructor

Parameters

r	must be partial.
p	must be complete. copy r and replace first partialorUpdate child by p,

15.73.3 Member Function Documentation

15.73.3.1 operator[]()

```
template<class P>
const P& Run< P >::operator[] (
            size_t i ) const  [inline]
```

return the ith subrun of this run.

Parameters

i	index of subrun

Warning

> the number of children must be at least i+1

15.73.3.2 first()

```
template<class P>
const P& Run< P >::first ( ) const
```

first children.

Warning

> this run must be inner with arity $>$ 0

15.73.3.3 last()

```
template<class P>
const P& Run< P >::last ( ) const
```

last children.

Warning

> this run must be inner with arity $>$ 0

15.73.3.4 firstPartialorUpdate()

```
template<class P>
const P& Run< P >::firstPartialorUpdate ( ) const
```

index of first children which is either either

- partial, or
- whose weight did not contribute to run's weight or arity() if there is no such children.

the index of first children is 0, the index of last children is arity - 1.

Warning

> complete() must not hold (otherwise there is no such children).
> this run must be inner with arity $>$ 0.

15.73.3.5 insert()

```
template<class P>
void Run< P >::insert (
             const P &  )
```

append the given ptr at the end of body.

Warning

the run must be marked as meta.

15.73.3.6 update()

```
template<class P>
void Run< P >::update (
             const Weight & w,
             const DurationList & dl = DurationList() )
```

update the weight and duration lists of this run with given weight and duration list.

Parameters

w	the given weight, must not be zero it must be the weight of best run for the first partialorupdate children (this cannot be checked!).
dl	must be the duration list of best run for the first partialorupdate children (this cannot be checked!).

Warning

this run must not be complete.
the first partialorupdate children must exist and be complete.

the index to first partialorupdate children is incremented.

The documentation for this class was generated from the following files:

- src/segment/InputSegment.hpp
- src/table/Rune.hpp

15.74 ScoreModel::Score Class Reference

Public Member Functions

- Score ()
- Score (std::string name, ScoreMeter meter)
- ScoreMeter getMeter () const
- std::string **getName** () const

- Voice ∗ getVoice (std::string partName, std::string voiceName)
- std::vector< Part ∗ > getParts () const
- void addPart (Part ∗p)
- Measure ∗ addMeasure ()
- std::vector< Measure ∗ > getMeasures () const
- ∼Score ()

15.74.1 Constructor & Destructor Documentation

15.74.1.1 Score() [1/2]

```
ScoreModel::Score::Score ( )
```

Main constructor. Builds an empty score in 4/4

15.74.1.2 Score() [2/2]

```
ScoreModel::Score::Score (
            std::string name,
            ScoreMeter meter )
```

Monody score constructor.

takes the single part/name voice, and the meter

15.74.1.3 ∼Score()

```
ScoreModel::Score::∼Score ( )
```

Destructor

15.74.2 Member Function Documentation

15.74.2.1 getMeter()

```
ScoreMeter ScoreModel::Score::getMeter ( ) const
```

Getter/setter for meter

15.74.2.2 getVoice()

```
Voice * ScoreModel::Score::getVoice (
            std::string partName,
            std::string voiceName )
```

Get a voice from the part and voice name

15.74.2.3 getParts()

```
std::vector< Part * > ScoreModel::Score::getParts ( ) const
```

Get all parts

15.74.2.4 addPart()

```
void ScoreModel::Score::addPart (
            Part * p )
```

Add a new part

15.74.2.5 addMeasure()

```
Measure * ScoreModel::Score::addMeasure ( )
```

Add a new measure

15.74.2.6 getMeasures()

```
std::vector< Measure * > ScoreModel::Score::getMeasures ( ) const
```

Iterator to scan measures

The documentation for this class was generated from the following files:

- src/scoremodel/Score.hpp
- src/scoremodel/Score.cpp

15.75 ScoreModel::ScoreMeter Class Reference

```
#include <ScoreMeter.hpp>
```

Public Member Functions

- ScoreMeter (int meter_count, int meter_unit)
- int **getCount** () const
- int **getUnit** () const
- Duration **getMeasureDuration** () const
- ∼ScoreMeter ()

15.75.1 Detailed Description

The score class: models a score content

15.75.2 Constructor & Destructor Documentation

15.75.2.1 ScoreMeter()

```
ScoreModel::ScoreMeter::ScoreMeter (
             int meter_count,
             int meter_unit )  [inline]
```

Main constructor.

15.75.2.2 ∼ScoreMeter()

```
ScoreModel::ScoreMeter::∼ScoreMeter ( )  [inline]
```

Destructor

The documentation for this class was generated from the following files:

- src/scoremodel/ScoreMeter.hpp
- src/scoremodel/ScoreMeter.cpp

15.76 SemiRing< T > Class Template Reference

semiring structure.

```
#include <SemiRing.hpp>
```

Friends

- bool **operator==** (const T &lhs, const T &rhs)
- bool **operator!=** (const T &lhs, const T &rhs)
- bool **operator<** (const T &lhs, const T &rhs)
- bool **operator>** (const T &lhs, const T &rhs)
- bool **operator<=** (const T &lhs, const T &rhs)
- bool **operator>=** (const T &lhs, const T &rhs)
- std::ostream & **operator<<** (std::ostream &o, const T &rhs)

15.76.1 Detailed Description

template<typename T>
class SemiRing< T >

semiring structure.

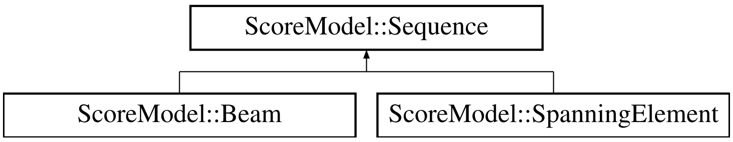
- add is associative, commutative
- zero is neutral element for plus
- mult is associative
- one is neutral element for mult
- zero is absorbing element for mult
- mult distributes over plus

The documentation for this class was generated from the following file:

- src/weight/SemiRing.hpp

15.77 ScoreModel::Sequence Class Reference

Inheritance diagram for ScoreModel::Sequence:



Public Member Functions

- Sequence ()
- void addEvent (Event ∗event)
- std::vector< Event ∗ > getEvents () const
- void concatenate (Sequence seq)
- int nbEvents () const
- Event ∗ getFirstEvent () const
- Event ∗ getLastEvent () const
- ∼Sequence ()

15.77.1 Constructor & Destructor Documentation

15.77.1.1 Sequence()

```
ScoreModel::Sequence::Sequence ( )
```

Main constructor.

15.77.1.2 ∼Sequence()

```
ScoreModel::Sequence::∼Sequence ( )
```

Destructor

15.77.2 Member Function Documentation

15.77.2.1 addEvent()

```
void ScoreModel::Sequence::addEvent (
             Event * event )
```

Add an event

15.77.2.2 getEvents()

```
std::vector< Event * > ScoreModel::Sequence::getEvents ( ) const
```

Get events

15.77.2.3 concatenate()

```
void ScoreModel::Sequence::concatenate (
             Sequence seq )
```

Concatenate a sub-sequence

15.77.2.4 nbEvents()

```
int ScoreModel::Sequence::nbEvents ( ) const
```

Nb events

15.77.2.5 getFirstEvent()

```
Event * ScoreModel::Sequence::getFirstEvent ( ) const
```

First event

15.77.2.6 getLastEvent()

```
Event * ScoreModel::Sequence::getLastEvent ( ) const
```

Last event

The documentation for this class was generated from the following files:

- src/scoremodel/Sequence.hpp
- src/scoremodel/Sequence.cpp

15.78 SerialLabel Class Reference

static functions for serializable int encoding of input and output leaf symbols containing the following info:

```
#include <SerialLabel.hpp>
```

Static Public Member Functions

- static label_t serialize (pre_t pre, pre_t post, size_t nb)

  return the leaf label encoding the given
- static pre_t pre (label_t)

  return the pre value of the given leaf label
- static pre_t post (label_t)

  return the post value of the given leaf label
- static size_t nbGraceNotes (label_t)

  return the number of grace node encoded in given leaf label
- static bool continuation (label_t)

  the given leaf label is a continuation (no event, no grace note)
- static size_t nbEvents (label_t)

  number of note + grace notes encoded in given leaf label

15.78.1 Detailed Description

static functions for serializable int encoding of input and output leaf symbols containing the following info:

- [input info]
  - pre value: number of events from previous segment aligned to left of current input segment
  - post value: number of events aligned to right of current input segment
- [output info]
  - number of grace notes in output
  - number of events in output (notes + grace notes)

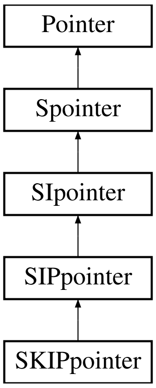
the encoding is
```
pre * (MAX_GRACE+1)^2 + post * (MAX_GRACE+1) + number_events
```

The documentation for this class was generated from the following files:

- src/output/SerialLabel.hpp
- src/output/SerialLabel.cpp

15.79 SIpointer Class Reference

Inheritance diagram for SIpointer:



Public Member Functions

- SIpointer ()

    dummy ptr
- SIpointer (label_t)

    fake ptr for terminal run, contains only a label symbol it is considered as complete see description in Ptr.hpp
- SIpointer (Environment ∗env, state_t s, Rational mdur=Rational(1), double rext=0)

    class specific top ptr (covering the whole input segment + given extension in realtime, of given musical duration.
- SIpointer (Environment ∗, const SIpointer &p, double rdur, Rational mdur, bool position, size_t i, state_t s)

    split ptr p in 2 parts.
- SIpointer (Environment ∗, const SIpointer &p, size_t a, size_t i, state_t s)

    sub-pointer or instance as leaf.
- SIpointer (const SIpointer &)

    copy.
- SIpointer (const SIpointer &p, const SIpointer &p0, const SIpointer &p1)

    instance as parent.
- SIpointer (const SIpointer &p0, const SIpointer &p1)

    instance as next sibling.
- virtual SIpointer & operator= (const SIpointer &)
- virtual bool operator== (const SIpointer &) const

    for use as key in a unordered_multimap.
- virtual bool **operator!=** (const SIpointer &) const
- virtual bool operator< (const SIpointer &) const

    for use as key in a multimap.
- virtual bool instance (const SIpointer &p) const
- virtual bool subsume (const SIpointer &p) const
- virtual bool **has_I** () const
- virtual bool **has_P** () const
- IntervalTree ∗ **interval** () const
- virtual bool complete () const
- virtual label_t label (const Transition &t) const
- virtual bool divisible () const
- virtual bool compatible (const label_t, bool abstract=true) const
- virtual bool dummy () const
- virtual Weight terminalWeight (const InputSegment ∗, const Transition &) const

Protected Member Functions

- bool **equal_node** (const SIpointer &) const

Protected Attributes

- IntervalTree ∗ **_node**

Friends

- std::ostream & **operator**<< (std::ostream &o, const SIpointer &p)

The documentation for this class was generated from the following files:

- src/table/PtrSI.hpp
- src/table/PtrSI.cpp

15.80 SIpointerHasher Struct Reference

hash function for using as key in a table. rank is ignoreds : same as SpointerHasher

```
#include <PtrSI.hpp>
```

Public Member Functions

- std::size_t **operator()** (const SIpointer &p) const

15.80.1 Detailed Description

hash function for using as key in a table. rank is ignoreds : same as SpointerHasher

The documentation for this struct was generated from the following file:

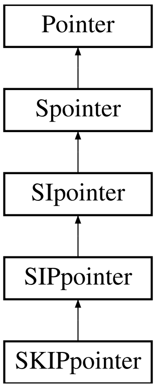
- src/table/PtrSI.hpp

15.81 SIPpointer Class Reference

key in a parse table. pointer to a (best) run for 1-best parsing for WTA and input segment.

```
#include <PtrSIP.hpp>
```

Inheritance diagram for SIPpointer:



Public Member Functions

- SIPpointer (pre_t pre=PP_UNKNOWN, pre_t post=PP_UNKNOWN)

  dummy ptr.
- SIPpointer (label_t)

  fake ptr for terminal run, contains only a label symbol. it is considered as complete
- SIPpointer (Environment ∗env, state_t s, pre_t pre=0, pre_t post=0, Rational mdur=Rational(1), double rext=0)

  class specific top ptr (covering the whole input segment
- SIPpointer (Environment ∗, const SIPpointer &p, double rdur, Rational mdur, bool position, size_t i, state_t s)

  split ptr p in 2 parts.
- SIPpointer (Environment ∗, const SIPpointer &p, size_t a, size_t i, state_t s)

  sub-pointer or instance as leaf.
- SIPpointer (const SIPpointer &)

  copy.
- SIPpointer (const SIPpointer &p, const SIPpointer &p0, const SIPpointer &p1)

  instance as parent.
- SIPpointer (const SIPpointer &p0, const SIPpointer &p1)

  instance as next sibling.
- virtual SIPpointer & operator= (const SIPpointer &)
- virtual bool operator== (const SIPpointer &) const

  for use as key in a unordered_multimap.
- virtual bool **operator!=** (const SIPpointer &) const
- virtual bool operator< (const SIPpointer &) const

  for use as key in a multimap.
- virtual bool instance (const SIPpointer &p) const
- virtual bool subsume (const SIPpointer &p) const
- virtual bool **has_I** () const
- virtual bool **has_P** () const
- IntervalTree ∗ **interval** () const
- pre_t **pre** () const
- pre_t **post** () const
- virtual bool complete () const
- label_t label (const Transition &t) const
- virtual bool compatible (const label_t, bool abstract=true) const
- virtual bool dummy () const
- virtual Weight terminalWeight (const InputSegment ∗s, const Transition &t) const

Protected Attributes

- pre_t _pre

  pre and post contain concrete labels (number of events)
- pre_t _post

  pre and post contain concrete labels (number of events)

Friends

- std::ostream & **operator<<** (std::ostream &o, const SIPpointer &p)

Additional Inherited Members

15.81.1 Detailed Description

key in a parse table. pointer to a (best) run for 1-best parsing for WTA and input segment.

a SIPpointer contains

- a state symbol: either a WTA state or a meta state or a leaf label
- an aligned interval in the input segment
- pre and post values, known or not (partial ptr)

15.81.2 Member Data Documentation

15.81.2.1 _pre

```
pre_t SIPpointer::_pre  [protected]
```

pre and post contain concrete labels (number of events)

See also

Label.hpp

15.81.2.2 _post

```
pre_t SIPpointer::_post  [protected]
```

pre and post contain concrete labels (number of events)

See also

Label.hpp

The documentation for this class was generated from the following files:

- src/table/PtrSIP.hpp
- src/table/PtrSIP.cpp

15.82 SIPpointerHasher Struct Reference

hash function for using as key in a table rank is ignoreds : same as SpointerHasher

```
#include <PtrSIP.hpp>
```

Public Member Functions

- std::size_t **operator()** (const SIPpointer &p) const

15.82.1 Detailed Description

hash function for using as key in a table rank is ignoreds : same as SpointerHasher

See also

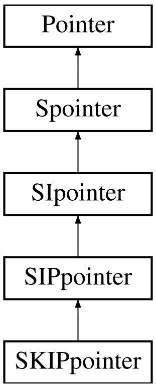
constant.h

The documentation for this struct was generated from the following file:

- src/table/PtrSIP.hpp

15.83 SKIPpointer Class Reference

Inheritance diagram for SKIPpointer:



Public Member Functions

- SKIPpointer ()

  dummy ptr.
- SKIPpointer (label_t, size_t k=1)

  specific fake ptr for terminal run, contains only a label symbol. it is considered as complete
- SKIPpointer (Environment ∗env, pre_t pre=0, pre_t post=0, bool bar=false, size_t k=1)
- SKIPpointer (Environment ∗env, state_t s, pre_t pre=0, pre_t post=0, Rational mdur=Rational(1), size_t k=1)

  class specific top ptr (covering the whole input segment.
- SKIPpointer (Environment ∗env, const SKIPpointer &p, size_t a, size_t i, state_t s)

  sub-pointer or instance as leaf.
- SKIPpointer (const SKIPpointer &)

  copy.
- SKIPpointer (const SKIPpointer &p0, const SKIPpointer &p1)

  next sibling.
- SKIPpointer (const SKIPpointer &p, const SKIPpointer &p0, const SKIPpointer &p1)

  instance as parent.
- virtual SKIPpointer & operator= (const SKIPpointer &)
- virtual bool operator== (const SKIPpointer &) const
- virtual bool instance (const SKIPpointer &p) const
- virtual bool subsume (const SKIPpointer &p) const
- virtual bool **has_K** () const
- virtual size_t rank () const

  return the rank of best (k) pointed. default is 1. redefine for classes for k-best parsing.
- virtual void **incr** ()

Protected Attributes

- size_t _rank

  k as in k-best.

Friends

- std::ostream & **operator**<< (std::ostream &o, const SKIPpointer &p)

Additional Inherited Members

The documentation for this class was generated from the following files:

- src/table/PtrSKIP.hpp
- src/table/PtrSKIP.cpp

15.84 SKIPpointerHasher Struct Reference

hash function for using as key in a table.

```
#include <PtrSKIP.hpp>
```

Public Member Functions

- std::size_t **operator()** (const Spointer &p) const

15.84.1 Detailed Description

hash function for using as key in a table.

Warning

> rank is ignored : same as SpointerHasher

The documentation for this struct was generated from the following file:

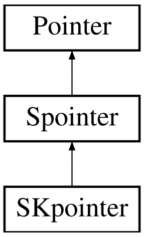
- src/table/PtrSKIP.hpp

15.85 SKpointer Class Reference

pointer to a (best) run. for k-best parsing with standard WTA a SKpointer contains

```
#include <PtrSK.hpp>
```

Inheritance diagram for SKpointer:



Public Member Functions

- SKpointer ()

  specific
- SKpointer (label_t, size_t k=1)

  specific
- SKpointer (WTA ∗a, Environment ∗env, pre_t pre=0, pre_t post=0, Rational mlen=Rational(1), size_t k=1)

  top ptr.
- SKpointer (Environment ∗, const SKpointer &p, size_t a, size_t i, state_t s)

  sub-pointer or instance as leaf.
- SKpointer (const SKpointer &)

  copy.
- SKpointer (const SKpointer &p0, const SKpointer &p1)

  next sibling.
- SKpointer (const SKpointer &p, const SKpointer &p0, const SKpointer &p1)

  instance as parent.
- virtual SKpointer & operator= (const SKpointer &)
- virtual bool operator== (const SKpointer &) const
- virtual bool instance (const SKpointer &p) const
- virtual bool subsume (const SKpointer &p) const
- virtual bool **has_K** () const
- virtual size_t rank () const

  return the rank of best (k) pointed. default is 1. redefine for classes for k-best parsing.
- virtual void **incr** ()

Protected Attributes

- size_t _rank

  k as in k-best

Friends

- std::ostream & **operator**<< (std::ostream &o, const SKpointer &p)

15.85.1 Detailed Description

pointer to a (best) run. for k-best parsing with standard WTA a SKpointer contains

- a state symbol (see Spointer)
- a rank:
  - 0 if the state symbol is a leaf label,
  - $>$ 0 otherwise in the case of Viterbi algo (1-best), the rank is defaulted to 1

all SKpointer's are complete.

The documentation for this class was generated from the following files:

- src/table/PtrSK.hpp
- src/table/PtrSK.cpp

15.86 SKpointerHasher Struct Reference

hash function for using as key in a table rank is ignoreds : same as SpointerHasher

```
#include <PtrSK.hpp>
```

Public Member Functions

- std::size_t **operator()** (const Spointer &p) const

15.86.1 Detailed Description

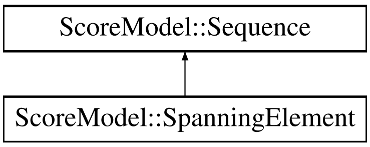
hash function for using as key in a table rank is ignoreds : same as SpointerHasher

The documentation for this struct was generated from the following file:

- src/table/PtrSK.hpp

15.87 ScoreModel::SpanningElement Class Reference

Inheritance diagram for ScoreModel::SpanningElement:



Public Member Functions

- SpanningElement ()

15.87.1 Constructor & Destructor Documentation

15.87.1.1 SpanningElement()

```
ScoreModel::SpanningElement::SpanningElement ( )
```

Main constructor.

The documentation for this class was generated from the following files:

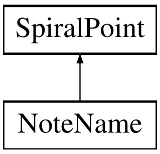
- src/scoremodel/SpanningElement.hpp
- src/scoremodel/SpanningElement.cpp

15.88 SpiralPoint Struct Reference

Elaine Chew's spiral of fifths.

```
#include <Spiral.hpp>
```

Inheritance diagram for SpiralPoint:



Public Member Functions

- **SpiralPoint** (double, double, double)
- **SpiralPoint** (const SpiralPoint &rhs)
- SpiralPoint & **operator=** (const SpiralPoint &)
- bool isnormal () const
- void **operator+=** (const SpiralPoint &rhs)
- void **operator-=** (const SpiralPoint &rhs)
- void **operator*=** (double a)
- double distance (const SpiralPoint &rhs) const

Public Attributes

- double **x**
- double **y**
- double **z**

Friends

- bool **operator==** (const SpiralPoint &, const SpiralPoint &)
- bool **operator!=** (const SpiralPoint &, const SpiralPoint &)
- std::ostream & **operator**<< (std::ostream &o, const SpiralPoint &rhs)

15.88.1 Detailed Description

Elaine Chew's spiral of fifths.

for pitch spelling.

The documentation for this struct was generated from the following files:

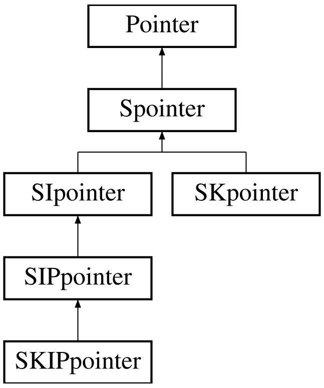
- src/segment/Spiral.hpp
- src/segment/Spiral.cpp

15.89 Spointer Class Reference

key in a parse table.

`#include <PtrS.hpp>`

Inheritance diagram for Spointer:



Public Member Functions

- Spointer ()

  specific
- Spointer (label_t)

  specific
- Spointer (WTA ∗a, Environment ∗env, pre_t pre=0, pre_t post=0, Rational mlen=Rational(1), size_t k=1)

  top ptr (head of the main Run).
- Spointer (Environment ∗env, const Spointer &p, size_t a, size_t i, state_t s)

  sub-pointer or instance as leaf.
- Spointer (const Spointer &)

  copy.
- Spointer (const Spointer &p0, const Spointer &p1)

  next sibling.
- Spointer (const Spointer &p, const Spointer &p0, const Spointer &p1)

  instance as parent.
- virtual Spointer & operator= (const Spointer &)
- virtual bool operator== (const Spointer &) const

  for use as key in a unorered_multimap.
- virtual bool operator< (const Spointer &) const

  for use as key in a multimap.
- virtual bool instance (const Spointer &p) const
- virtual bool subsume (const Spointer &p) const
- virtual bool **has_S** () const
- state_t **state** () const
- virtual bool complete () const

  the pointer is complete i.e. all fields are set
- virtual label_t label (const Transition &t) const

  return a concrete label value corresponding to this pointer when considered as a leaf position, using the label of the given transition. the given transition must be terminal.
- virtual bool dummy () const

  return whether this pointer is a dummy pointer i.e. it was constructed with P() default false.
- virtual bool divisible () const

Protected Attributes

- state_t **_state**

Friends

- std::ostream & **operator<<** (std::ostream &o, const Spointer &p)

15.89.1 Detailed Description

key in a parse table.

pointer to a (best) run for 1-best parsing for standard WTA.

a Spointer contains

- a state symbol: either
  - a WTA state or
  - a leaf label or
  - a meta state

that defines two kind of pointers

- state pointer : points to a state of a WTA
- bar pointer : points to a solution for the n first bars in an input segment = a sequence of n solutions pointing to the initial state of WTA.

all Spointers are complete

The documentation for this class was generated from the following files:

- src/table/PtrS.hpp
- src/table/PtrS.cpp

15.90 SpointerHasher Struct Reference

Public Member Functions

- std::size_t **operator()** (const Spointer &p) const

The documentation for this struct was generated from the following file:

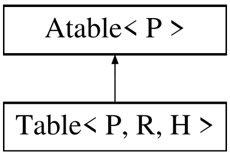
- src/table/PtrS.hpp

15.91 Table< P, R, H > Class Template Reference

parse table.

```
#include <Table.hpp>
```

Inheritance diagram for Table< P, R, H >:



Public Member Functions

- Table (Parser< P > *env, RunCompare< P > comp)

    concrete table.
- virtual Run< P > * best (const P &p)

    return k-best run pointed by p or NULL if there is none. k is either included in p or the default value 1.
- virtual RhythmTree * bestTree (const P &p)

    tree corresponding to the k-best run in p.
- virtual RhythmTree * bestTree (Run< P > *r)

    when the k-best run in p is already computed.
- Record< P > * add (const P &p)

    if p complete, create a new record in table for it and process it (add runs), if p partial, process it (register instances to table) with addPartial.
- virtual size_t add (const P &p, Run< P > *r, Record< P > *i)
- virtual size_t **nb_entries** ()
- virtual size_t **nb_runs** ()
- void **dump_table** () const
- void **dump_instances** () const

Additional Inherited Members

15.91.1 Detailed Description

template<class P, class R, class H>
class Table< P, R, H >

parse table.

table defines two main undorered mappings:

map table:
```
map key -> value
```

where

- key of type P = Pointer
- value of type R = Record

(stores some Run

*)

- H = Hasher for P
- equal+to is op. == defined in P

table of instances:
```
multimap: key -> keys
```

where

- key of type P = Pointer (partial)
- keys of type P = complete Pointer instances of key
- H = Hasher for P
- equal+to is op. == defined in P

15.91.2 Constructor & Destructor Documentation

15.91.2.1 Table()

```
template<class P, class R, class H>
Table< P, R, H >::Table (
             Parser< P > * env,
             RunCompare< P > comp )
```

concrete table.

See also

Atable for arguments

Parameters

env	the parsing environment must not be NULL

15.91.3 Member Function Documentation

15.91.3.1 best()

```
template<class P, class R, class H>
virtual Run<P>* Table< P, R, H >::best (
             const P & p ) [virtual]
```

return k-best run pointed by p or NULL if there is none. k is either included in p or the default value 1.

Parameters

p	must be complete.

Implements Atable< P >.

15.91.3.2 add() [1/2]

```
template<class P, class R, class H>
Record<P>* Table< P, R, H >::add (
             const P & p )
```

if p complete, create a new record in table for it and process it (add runs), if p partial, process it (register instances to table) with addPartial.

Parameters

p	can be partial or complete.

Warning

p must have yet no associated record in table when complete.
p must not have been added before if partial (no registered instances).

Returns

a pointer to the newly created record if p complete.
a NULL pointer in this case if p partial.

15.91.3.3 add() [2/2]

```
template<class P, class R, class H>
virtual size_t Table< P, R, H >::add (
            const P & p,
            Run< P > * r,
            Record< P > * i ) [virtual]
```

Parameters

p	can be complete or partial.
r	can be complete or partial.
i	if p is complete, then i must be an pointer to the entry for p in table, otherwise (p partial), i is NULL.

add possible instances of run r to the entries in table for corresp. to possible instances for p. dispatch to the four functions below according to p and r.

Returns

0 if the run or one instance of the run (at least) was added to the table.
> 0 otherwise.

Implements Atable< P >.

The documentation for this class was generated from the following file:

- src/table/Table.hpp

15.92 Transition Class Reference

a Transition is defined by a sequence of antecedent states (body) the weight must be not null (null weight means a missing transition).

```
#include <Transition.hpp>
```

Public Member Functions

- Transition ()

  transition with unknown weight and empty body.
- Transition (const Weight &)

  Transition(w) creates a transition with weight a copy of w and empty body.
- Transition (LetterWeight ∗)

  Transition(lw) creates a transition with weight a wrapper of the letter lw (must be non null)
- Transition (std::vector< state_t >, const Weight &)

  Transition(v, w) creates a transition with weight a copy of w and body a copy of the vector v.
- Transition (std::vector< state_t >, LetterWeight ∗)

  Transition(v, lw) creates a transition with weight a wrapper of the letter lw (must be non null) and body a copy of the vector v.
- Transition (state_t, const Weight &)

  Transition(s, w) creates a transition with weight a copy of w and body (of size 1) the singleton (s) (terminal symbol).
- Transition (state_t, LetterWeight ∗)

  Transition(s, lw) creates a transition with weight a wrapper of the letter lw (must be non null) and body (of size 1) the singleton (s) (terminal symbol).
- bool **inner** () const
- bool **terminal** () const
- size_t **id** () const
- label_t label () const
- Weight **weight** () const
- void **setWeight** (const Weight &w)
- void scalar (double)

  modify weight of transition.
- void **invert** ()
- state_t at (size_t i) const

  at(i) returns the ith state in the body.
- void push (state_t)

  add given state at the end of the body of this transition.
- size_t size () const

  size of body.
- size_t **arity** () const
- Transition_const_iterator begin () const

  iterator pointing to the first state in the body of the transition.
- Transition_const_iterator end () const

  iterator pointing to the end of the body of the transition.
- bool member (state_t) const

  whether the given state belongs to the body of this transition.
- bool allin (const std::set< state_t > &) const

  every state of the body is in the given set.
- bool nonein (const std::set< state_t > &) const

  no state of the body is in the given set.

Friends

- std::ostream & operator<< (std::ostream &, const Transition &)

  write content of body and weight to output stream.

15.92.1 Detailed Description

a Transition is defined by a sequence of antecedent states (body) the weight must be not null (null weight means a missing transition).

a transition can be of two kinds:

- inner transition: the body has length > 1 the arity is the length of the body
- terminal (leaf) transition: the body has length 1 and contains a leaf label the arity is zero

leaf label (terminal transitions): number of note + grace notes at (left of) current node 0 = continuation 1 = 1 note | rest (au + 1 note) 2 = 1 grace notes + 1 note 3 = 2 grace notes + 1 note etc

See also

Label for the functions managing these labels

The documentation for this class was generated from the following files:

- src/schemata/Transition.hpp
- src/schemata/Transition.cpp

15.93 TransitionList Class Reference

Public Member Functions

- bool empty () const

  zero transition
- size_t size () const

  number of transitions.
- size_t fullsize () const

  total size of transition table (sum of transition sizes. = number of occurences of states)
- void **add** (const Transition &)
- void **remove** (TransitionList_iterator)
- void remove (state_t)

  remove all transitions of length > 1 in the list containing the given state do not remove length 1 transitions to terminal symbols
- void **clear** ()
- TransitionList_const_iterator **begin** () const
- TransitionList_const_iterator **end** () const
- TransitionList_iterator nc_begin ()

  non constant iterator.
- TransitionList_iterator nc_end ()

  non constant iterator.

Friends

- class **WTA**

The documentation for this class was generated from the following files:

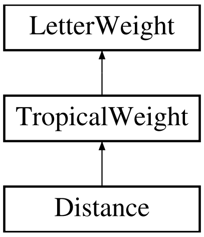
- src/schemata/WTA.hpp
- src/schemata/WTA.cpp
- src/schemata/WTA_BACKUP_31784.cpp
- src/schemata/WTA_BASE_31784.cpp
- src/schemata/WTA_LOCAL_31784.cpp
- src/schemata/WTA_REMOTE_31784.cpp

15.94 TropicalWeight Class Reference

concrete Weight defined as a scalar value: non-negative weights.

```
#include <TropicalWeight.hpp>
```

Inheritance diagram for TropicalWeight:



Public Member Functions

- **TropicalWeight** (const TropicalWeight &)
- TropicalWeight & **operator=** (const TropicalWeight &)
- TropicalWeight & operator= (const LetterWeight &)
- TropicalWeight ∗ **clone** () const
- virtual Weight make (double v) const
- virtual Weight get_zero () const

  return the neutral element for add (absorbing element for mult) wrapped in a Weight.
- virtual Weight get_one () const

  return the neutral element for mult wrapped in a Weight.
- virtual double norm () const
- virtual void **scalar** (double)
- virtual void invert ()

  multiplicative inverse.
- virtual bool zero () const

  this letterweight is neutral element for add (absorbing element for mult).
- virtual bool one () const

  this letterweight is neutral element for mult.
- virtual bool **hasType** (std::string code) const

Static Public Member Functions

- static Weight **make_zero** ()
- static Weight **make_one** ()
- static TropicalWeight inner (size_t)

  penalty for an inner node.
- static TropicalWeight tie ()

  penalty for a tie.
- static TropicalWeight gracenote (size_t)

  penalty for given number of grace notes in a leaf.

Static Public Attributes

- static TropicalWeight penalty [18]

  penalty by arity.

Protected Member Functions

- TropicalWeight ()

  default is one
- **TropicalWeight** (double v)
- virtual bool equal (const LetterWeight ∗rhs) const
- virtual bool smaller (const LetterWeight ∗rhs) const
- virtual void add (const LetterWeight ∗rhs)

  sum is min.
- virtual void mult (const LetterWeight ∗rhs)

  product is sum.
- virtual void **print** (std::ostream &) const

Protected Attributes

- double **_val**

Friends

- std::ostream & **operator**<< (std::ostream &o, const TropicalWeight &rhs)

15.94.1 Detailed Description

concrete Weight defined as a scalar value: non-negative weights.

- domain : positive or null double + infinity
- operators of tropical algebra:
- add is min
- zero is infinity
- mult is +
- one is 0

15.94.2 Member Function Documentation

15.94.2.1 make()

```
virtual Weight TropicalWeight::make (
            double v ) const  [inline], [virtual]
```

Warning

value must be positive

Todo TBR : stricly positive

Implements LetterWeight.

Reimplemented in Distance.

The documentation for this class was generated from the following files:

- src/weight/TropicalWeight.hpp
- src/weight/TropicalWeight.cpp

15.95 ScoreModel::Tuplet Class Reference

Public Member Functions

- Tuplet (Duration duration, Sequence events, int arity)
- ∼Tuplet ()
- Duration getDuration () const
- int nbEvents () const
- Duration getBaseDuration () const
- int getArity () const
- int getNumBase () const
- Event ∗ getFirstEvent () const
- Event ∗ getLastEvent () const
- std::vector< Event ∗ > getEvents () const

15.95.1 Constructor & Destructor Documentation

15.95.1.1 Tuplet()

```
ScoreModel::Tuplet::Tuplet (
             Duration duration,
             Sequence events,
             int arity )
```

Main constructor.

15.95.1.2 ∼Tuplet()

```
ScoreModel::Tuplet::∼Tuplet ( )
```

Destructor

15.95.2 Member Function Documentation

15.95.2.1 getDuration()

```
Duration ScoreModel::Tuplet::getDuration ( ) const
```

Duration of the tuplet

15.95.2.2 nbEvents()

```
int ScoreModel::Tuplet::nbEvents ( ) const
```

Nb events

15.95.2.3 getBaseDuration()

```
Duration ScoreModel::Tuplet::getBaseDuration ( ) const
```

Base duration = duration of regular tuplet events, before applying the ratoi

Example: a triplet of eigthth, the base duration is the eighth

15.95.2.4 getArity()

```
int ScoreModel::Tuplet::getArity ( ) const
```

Arity = the number of duration-equal timespans the tuplet is decomposed in

15.95.2.5 getNumBase()

```
int ScoreModel::Tuplet::getNumBase ( ) const
```

Numbase = the regular number of base duration in the tuplet

Example: a triplet of eigthth correspond to 2 (two) regular eigthth

15.95.2.6 getFirstEvent()

```
Event * ScoreModel::Tuplet::getFirstEvent ( ) const
```

Get the first event of the tuplet sequence

First event

15.95.2.7 getLastEvent()

```
Event * ScoreModel::Tuplet::getLastEvent ( ) const
```

Last event

15.95.2.8 getEvents()

```
std::vector< Event * > ScoreModel::Tuplet::getEvents ( ) const
```

Get events

The documentation for this class was generated from the following files:

- src/scoremodel/Tuplet.hpp
- src/scoremodel/Tuplet.cpp

15.96 ValueList Class Reference

list of rational durations as components of value states.

```
#include <ValueList.hpp>
```

Public Member Functions

- **ValueList** (Rational)
- **ValueList** (const DurationList &)
- **ValueList** (const ValueList &)
- **ValueList** (std::string)
- ValueList & **operator=** (const ValueList &)
- bool **empty** () const
- size_t **size** () const
- Rational **length** () const
- Rational **cont** () const
- std::list< Rational >::const_iterator **begin** () const
- std::list< Rational >::const_iterator **end** () const
- bool **complete** () const
- bool **single_continuation** () const
- bool **single_event** () const
- bool **event** () const
- size_t **nbgn** () const
- void **add** (Rational)
- void **addcont** (Rational)
- Rational **front** () const
- Rational **pop** ()
- Rational **popcont** ()
- void **popcont** (Rational)

Friends

- class **DurationTree**
- std::ostream & **operator**<< (std::ostream &, const ValueList &)
- bool **operator==** (const ValueList &, const ValueList &)
- bool **operator!=** (const ValueList &, const ValueList &)

15.96.1 Detailed Description

list of rational durations as components of value states.

Each duration is either positive (event w or wo continuations -ties) or null (grace note).

a value list is made of 2 parts:

- _cont : initial duration (possibly null) tied to the previous duration list
- _main : main list of the other events (without ties) it is represented by _cont[_main]

The documentation for this class was generated from the following files:

- src/output/ValueList.hpp
- src/output/ValueList.cpp

15.97 ValueState Class Reference

Public Member Functions

- **ValueState** (state_t, DurationTree ∗)
- state_t **state** () const
- ValueList **value** () const
- DurationTree ∗ **tree** () const
- bool **operator==** (const ValueState &s) const
- bool **compatible** (label_t label) const

Friends

- std::ostream & **operator**<< (std::ostream &, const ValueState &)

The documentation for this class was generated from the following files:

- src/schemata/ValueWTA.hpp
- src/schemata/ValueWTA.cpp

15.98 ValueStateHasher Struct Reference

Public Member Functions

- std::size_t **operator()** (const ValueState &vs) const

The documentation for this struct was generated from the following file:

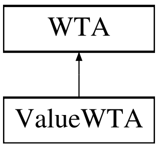
- src/schemata/ValueWTA.hpp

15.99 ValueWTA Class Reference

Value WTA is a special kind of WTA associated to an initial WTA (schema) and a rhythmic value (DurationList).

```
#include <ValueWTA.hpp>
```

Inheritance diagram for ValueWTA:



Public Member Functions

- ValueWTA (const DurationList &, const WTA &)

  construction from given initial list and WTA (base schema)
- virtual bool **hasType** (std::string code) const

Additional Inherited Members

15.99.1 Detailed Description

Value WTA is a special kind of WTA associated to an initial WTA (schema) and a rhythmic value (DurationList).

It characterizes the trees of the schema language (with weight defined by schema) having the given rhythmic value.

table of transitions construction top-down, given an initial schema (WTA) and a DurationList

The documentation for this class was generated from the following files:

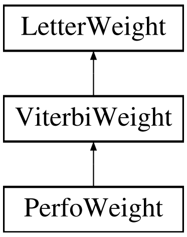
- src/schemata/ValueWTA.hpp
- src/schemata/ValueWTA.cpp

15.100 ViterbiWeight Class Reference

Viterbi semifield. concrete Weight defined as a scalar value: probability of the best derivation.

```
#include <ViterbiWeight.hpp>
```

Inheritance diagram for ViterbiWeight:



Public Member Functions

- **ViterbiWeight** (const ViterbiWeight &)
- ViterbiWeight & **operator=** (const ViterbiWeight &)
- ViterbiWeight & operator= (const LetterWeight &rvalue)
- virtual LetterWeight $*$ **clone** () const
- virtual Weight make (double v) const

  factory.
- virtual Weight get_zero () const

  return the neutral element for add (absorbing element for mult) wrapped in a Weight.
- virtual Weight get_one () const

  return the neutral element for mult wrapped in a Weight.
- virtual double **norm** () const
- virtual void **scalar** (double)
- virtual void invert ()

  multiplicative inverse.
- virtual bool zero () const

  this letterweight is neutral element for add (absorbing element for mult).
- virtual bool one () const

  this letterweight is neutral element for mult.
- virtual bool **hasType** (std::string code) const

Static Public Member Functions

- static Weight **make_zero** ()
- static Weight **make_one** ()

Protected Member Functions

- ViterbiWeight (double)

  default is one
- bool equal (const LetterWeight ∗rhs) const

  rhs must be a ViterbiWeight.
- bool smaller (const LetterWeight ∗rhs) const

  rhs must be a ViterbiWeight.
- void add (const LetterWeight ∗rhs)

  sum is min.
- void mult (const LetterWeight ∗rhs)

  product is sum.
- void **print** (std::ostream &) const

Protected Attributes

- double **_val**

Friends

- std::ostream & **operator**<< (std::ostream &o, const ViterbiWeight &rhs)

15.100.1 Detailed Description

Viterbi semifield. concrete Weight defined as a scalar value: probability of the best derivation.

- domain : positive or null rational numbers in [0, 1]
- operators:
- add is max
- zero is 0
- mult is ∗
- one is 1

15.100.2 Member Function Documentation

15.100.2.1 make()

```
virtual Weight ViterbiWeight::make (
            double v ) const  [inline], [virtual]
```

factory.

Returns

a weight of same type as this letter, initialized with given value.

Implements LetterWeight.

Reimplemented in PerfoWeight.

The documentation for this class was generated from the following files:

- src/weight/ViterbiWeight.hpp
- src/weight/ViterbiWeight.cpp

15.101 ScoreModel::Voice Class Reference

Public Member Functions

- Voice (Part ∗part, std::string name)
- std::string **getName** () const
- Part ∗ **getPart** () const
- Score & **getScore** () const
- void addEvent (Event ∗event)
- void addTie (Note ∗e1, Note ∗e2)
- void addTuplet (Tuplet ∗tuplet)
- void addBeam (Beam ∗beam)
- Sequence addFromRhythmTree (Measure ∗measure, const PointedRhythmTree ∗pointedRT, Duration rtDuration, int level=0)
- Voice trimMeasure (Measure ∗m)
- VoiceRange getRange () const
- std::vector< Event ∗ > getEvents () const
- std::vector< Tie ∗ > getTies () const
- std::vector< Tuplet ∗ > getTuplets () const
- std::vector< Beam ∗ > getBeams () const
- ∼Voice ()

15.101.1 Constructor & Destructor Documentation

15.101.1.1 Voice()

```
ScoreModel::Voice::Voice (
            Part * part,
            std::string name )
```

Main constructor.

15.101.1.2 ∼Voice()

```
ScoreModel::Voice::∼Voice ( )
```

Destructor

15.101.2 Member Function Documentation

15.101.2.1 addEvent()

```
void ScoreModel::Voice::addEvent (
            Event * event )
```

Add an event

15.101.2.2 addTie()

```
void ScoreModel::Voice::addTie (
            Note * e1,
            Note * e2 )
```

Add a tie between two notes

15.101.2.3 addTuplet()

```
void ScoreModel::Voice::addTuplet (
            Tuplet * tuplet )
```

Add a tuplet

15.101.2.4 addBeam()

```
void ScoreModel::Voice::addBeam (
            Beam * beam )
```

Add a beam

15.101.2.5 addFromRhythmTree()

```
Sequence ScoreModel::Voice::addFromRhythmTree (
            Measure * measure,
            const PointedRhythmTree * pointedRT,
            Duration rtDuration,
            int level = 0 )
```

Add a new measure or part of a measure from a rhythm tree

The method return a sub-voice containing the added elements

15.101.2.6 trimMeasure()

```
Voice ScoreModel::Voice::trimMeasure (
            Measure * m )
```

Extract the part that belongs to a measure

15.101.2.7 getRange()

```
VoiceRange ScoreModel::Voice::getRange ( ) const
```

get the range of a voice a a pair of pitches

15.101.2.8 getEvents()

```
std::vector< Event * > ScoreModel::Voice::getEvents ( ) const
```

Get events

15.101.2.9 getTies()

```
std::vector< Tie * > ScoreModel::Voice::getTies ( ) const
```

Get ties

15.101.2.10 getTuplets()

```
std::vector< Tuplet * > ScoreModel::Voice::getTuplets ( ) const
```

Get tuplets

15.101.2.11 getBeams()

```
std::vector< Beam * > ScoreModel::Voice::getBeams ( ) const
```

Get beams

The documentation for this class was generated from the following files:

- src/scoremodel/Voice.hpp
- src/scoremodel/Voice.cpp

15.102 Weight Class Reference

A class of polymorphic weight domains for tree series.

```
#include <Weight.hpp>
```

Public Member Functions

- Weight (LetterWeight *w=NULL)

  wrapper and unknown weight constructor (empty envelope, default).
- Weight (const Weight &w)

  clone the letter.
- Weight & **operator=** (const Weight &)
- LetterWeight * operator-> () const
- Weight * **clone** () const
- bool unknown () const

  unknown weight is a Weight with NULL letter.
- Weight make (double v) const
- bool hasType (std::string code) const
- double norm ()
- void scalar (double)

  scalar multiplication.
- bool zero () const

  *this weight is neutral element for + (absorbing element for *).*
- Weight get_zero () const

  return the neutral element for add (absorbing element for mult) for the LetterWeight, if any otherwise return unknown Weight.
- void operator+= (const Weight &rhs)
- bool one () const

  *this weight is neutral element for **
- Weight get_one () const

  return the neutral element for mult for the LetterWeight, if any otherwise return unknown Weight.
- void operator*= (const Weight &rhs)
- void invert ()

  multiplicative inverse, for semifields
- void clear ()

  delete the letter.
- std::string **save_to_string** ()

Protected Member Functions

- bool equal (const Weight &rhs) const

    binary operators are defined only between descendant Weights of same typeid
- bool smaller (const Weight &rhs) const
- void add (const Weight &rhs)
- void mult (const Weight &rhs)
- void **print** (std::ostream &o) const

Protected Attributes

- LetterWeight ∗ _letter

    letter always points to an object of one of the derived ∗Weight classes never to an object of the Weight base class.

Friends

- bool operator== (const Weight &, const Weight &)
- bool operator< (const Weight &, const Weight &)
- std::ostream & operator<< (std::ostream &o, const Weight &rhs)

15.102.1 Detailed Description

A class of polymorphic weight domains for tree series.

Every concrete weight domain must be a derived class of Weight.

the type Weight is the union of an unknown weight value and different weight domain.

it is implemented as an envelope, containing either

- a null letter. in this case, we have an unknown weight value.
- a non-null letter, pointing to an object of a derived weight class (concrete weight). In this case, the envelope is a wrapper for the object of the derived class, corresponding to an actual (known) weight value. see Envelope Letter Idiom (wikibooks)

Client code only uses the Weight class (not the derived classes), except for allocation of new concrete weights values by Weight(new DerivedWeight(...))

15.102.2 Member Function Documentation

15.102.2.1 operator->()

```
LetterWeight∗ Weight::operator-> ( ) const  [inline]
```

Warning

must not be unknown

15.102.2.2 operator+=()

```
void Weight::operator+= (
             const Weight & rhs )  [inline]
```

See also

add

15.102.2.3 operator∗=()

```
void Weight::operator*= (
             const Weight & rhs )  [inline]
```

See also

mult

15.102.3 Friends And Related Function Documentation

15.102.3.1 operator==

```
bool operator== (
             const Weight & lhs,
             const Weight & rhs )  [friend]
```

See also

equal

15.102.3.2 operator<

```
bool operator< (
             const Weight & lhs,
             const Weight & rhs )  [friend]
```

See also

smaller

15.102.3.3 operator<<

```
std::ostream& operator<< (
            std::ostream & o,
            const Weight & rhs ) [friend]
```

See also

print

The documentation for this class was generated from the following files:

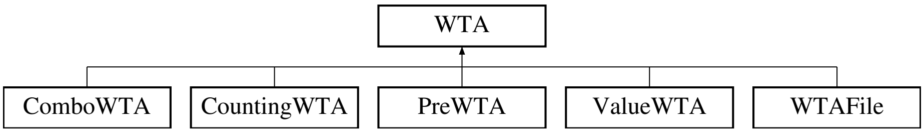
- src/weight/Weight.hpp
- src/weight/Weight.cpp

15.103 WTA Class Reference

class of schemas = weighted tree automata = weighted CFG.

```
#include <WTA.hpp>
```

Inheritance diagram for WTA:



Public Member Functions

- WTA ()

  nullary constructor for cython
- WTA (Weight seed, pre_t pre=0, pre_t post=0)

  empty automaton
- virtual bool **hasType** (std::string code) const
- size_t size () const

  number of states
- bool **empty** () const
- bool isRegistered (state_t) const

  the state is present in the automaton
- bool isInitial (state_t) const

  the state is an initial state
- virtual state_t initial (pre_t pre=0, pre_t post=0) const

  initial(pre, post) pre and post are use for quantification and ignored in this version (useless for schemas)
- size_t **resolution** () const
- TransitionList & add (state_t, bool initial=false)

  add(s, i) register state s if s was already registered, return a reference to its transition list. otherwise, create state s with an empty transition list and returns a reference to it. moreover s is set as initial if i = true.

- TransitionList & add (state_t, const Transition &, bool initial=false)

    add(s, t) add a transition with head s and with body/weight described in t if s was not registered, it is registered the transition t is added to the transition list of s and a reference to this transition list is returned moreover s is set as initial if i = true.
- void remove (state_t)

    remove the entry for given state s in the table of the table i.e. all transitions targeted to s, and all the transitions with s in their body. if s was in the initial set, it is also removed from this set. s must be registered.
- TransitionList_const_iterator begin (state_t) const

    begin(s) returns an iterator pointing to the first transition with head state s. s must be registered. not for modifying transition list of s. use add(...) methods for this.
- TransitionList_const_iterator end (state_t) const

    begin(s) returns an iterator pointing to the past-the-end transition with head state s. s must be registered. not for modifying transition list of s. use add(...) methods for this.
- size_t oftarget (state_t) const

    oftarget(s) return the number of transitions of head state s. s must be registered.
- bool isClean () const

    the WTA has no empty states
- std::set< state_t > emptyStates () const

    returns the set of all non-inhabited (zero weight) states in wta
- void clean ()

    remove states not inhabited and transitions containing these states
- void normalize (unsigned int flag=0)

    for all state q, for all transition tr to q (in the transition list TL(q) of q). recompute weights to get a probabilistic WTA.
- void CountingtoStochastic ()

    cast weights in all transitions.
- void PenaltytoCounting ()

    cast weights in all transitions.
- void StochastictoPenalty ()

    cast weights in all transitions.
- void CountingtoPenalty ()

    cast weights in all transitions.
- void abstract (bool flag=false)

    abstract the leaf label values in domain [0..MAX_GRACE] every value > MAX_GRACE is casted to MAX_GRACE the weights are summed accordingly
- size_t countStates () const

    number of states
- size_t countTransitions () const

    number of transition
- size_t countAll () const

    number of symbols (state occurences)
- bool hasWeightType (std::string code) const

    return wether the weights in transition have the type of the code (code of the letter weight if there is one or "UNKN↩ OWN" otherwise).
- virtual Weight weight_zero () const

    return the 0 value in the weight domain in this WTA
- virtual Weight weight_one () const

    return the 1 value in the weight domain in this WTA
- virtual Weight eval (const RhythmTree &t) const

    evaluate the weight of the tree t for WTA in initial state
- virtual Weight **eval** (const RhythmTree &t, state_t s) const
- pre_t **max_pre** () const
- pre_t **max_post** () const
- void print (std::ostream &) const

    print sizes to output stream

Public Attributes

- std::set< state_t > initials

  set of initial states

Protected Member Functions

- std::set< state_t > step (const std::set< state_t > &)

  step(s) returns the set of states reachable in one transition step by this WTA from the given state set s. all the states in the set s must be registered.
- std::set< state_t > allStates () const

  returns the set of all states occuring in wta (in head or body)

Protected Attributes

- std::map< state_t, TransitionList > _table

  transition table
- state_t **_initial**
- size_t _cpt_tr

  number of transitions
- size_t _cpt_size

  full size (number of occurences of states)
- pre_t _max_pre

  used only in descendant classes
- pre_t **_max_post**
- Weight _seed

  arbitrary (polymorphic) weight value. for generation of weights in same domain with get_zero, get_one

Friends

- class **TransitionList**
- std::ostream & operator<< (std::ostream &, const WTA &)

  write table content to output stream

15.103.1 Detailed Description

class of schemas = weighted tree automata = weighted CFG.

state (and non-terminals): int

transition table = map state -> (transition = state list), weight state: head state state list: see Transition.hpp body states if length > 1 label if length = 1

in other terms transition rules have one of the forms

> s -> (s1,...,sk) w where k > 1, s, s1, ..., sk are states and w weight s -> (s1) w where s1 is an leaf label = int encoding

leaf label (terminals): number of note + grace notes at (left of) current node

> 0 = continuation 1 = 1 note | rest (au + 1 note) 2 = 1 grace notes + 1 note 3 = 2 grace notes + 1 note >etc

See also

> Label for the functions managing these labels transition Table module:
>
> > head state -> vector of (state vector, Weight module)

weights are concrete weight values embedded in a Weight envelop we consider 3 kinds of weights for WTA serialized in file:

- counting model: weight = # of subresettrees in corpus parsed by rule implemented as FloatWeight
- penalty model: weight = penalities to sum implemented as TropicalWeight e.g. inverse of counting model (normalized?)
- probabilistic model, fulfilling stochastic condition (sum of weight of transition from a state = 1) implemented as ViterbiWeight e.g. (# of subtrees in corpus parsed by rule) / (# of subtrees matching lhs state)

15.103.2 Member Function Documentation

15.103.2.1 normalize()

```
void WTA::normalize (
            unsigned int flag = 0 )
```

for all state q, for all transition tr to q (in the transition list TL(q) of q). recompute weights to get a probabilistic WTA.

with arg = 0, we assume the current WTA is a penalty model. the probability is then obtained by dividing (Weight.scalar) the inverse of the norm (Weight.norm) of the weight of the tr by the sum of inverses of norms of transitions in TL(q).

with arg = 1, we assume the current WTA is a counting model. the probability is then obtained by dividing (Weight.scalar) the norm (Weight.norm) of the weight of the tr by the sum of norms of transitions in TL(q).

15.103.3 Member Data Documentation

15.103.3.1 initials

```
std::set<state_t> WTA::initials
```

set of initial states

Todo SUPPR

The documentation for this class was generated from the following files:

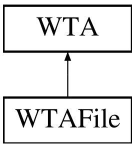
- src/schemata/WTA.hpp
- src/schemata/WTA.cpp
- src/schemata/WTA_BACKUP_31784.cpp
- src/schemata/WTA_BASE_31784.cpp
- src/schemata/WTA_LOCAL_31784.cpp
- src/schemata/WTA_REMOTE_31784.cpp

15.104 WTAFile Class Reference

wrapper for constructing WTA with various flags for weight type.

```
#include <WTAFile.hpp>
```

Inheritance diagram for WTAFile:



Public Member Functions

- WTAFile ()

  default constructor for cython.
- WTAFile (const std::string filename, WeightDom wt=WeightDom::UNDEF, pre_t pre=0, pre_t post=0)

  read weight type and schema from file.
- WTAFile (const std::string filename, bool count_flag=false, bool penalty_flag=true, bool stochastic_flag=false)

  read schema from file
- ∼WTAFile ()

  same as WTA destructor.
- size_t save (string filename)

  save to file.

Static Public Member Functions

- static bool readTimesignature (const std::string filename, ScoreModel::ScoreMeter &ts)

  read time signature from schema file

Additional Inherited Members

15.104.1 Detailed Description

wrapper for constructing WTA with various flags for weight type.

15.104.2 Constructor & Destructor Documentation

15.104.2.1 WTAFile() [1/2]

```
WTAFile::WTAFile (
             const std::string filename,
             WeightDom wt = WeightDom::UNDEF,
             pre_t pre = 0,
             pre_t post = 0 )
```

read weight type and schema from file.

Parameters

filename	input text file specifying the schema
wt	weight type (forced)

if another weight type is given as argument (forced weight type)

- use it as weight type for reading schema if no weight type found in file
- use it to cast schema (force type). the WTA can be empty in case of error.

15.104.2.2 WTAFile() [2/2]

```
WTAFile::WTAFile (
             const std::string filename,
             bool count_flag = false,
             bool penalty_flag = true,
             bool stochastic_flag = false )
```

read schema from file

Parameters

filename	input text file specifying the schema
count_flag	flag to determine the type of weights
penalty_flag	flag to determine the type of weights
stochastic_flag	flag to determine the type of weights

casts weights according to compile options if needed.

Todo TBR

15.104.3 Member Function Documentation

15.104.3.1 readTimesignature()

```
bool WTAFile::readTimesignature (
             const std::string filename,
             ScoreModel::ScoreMeter & ts )  [static]
```

read time signature from schema file

Returns

0 if a time signature was found in filename in that case, the time signature is stored in the given ts, otherwise, ts is left unchanged.

Warning

must be in the form "[timesig int int]" in the file, with int $>$ 0.

The documentation for this class was generated from the following files:

- src/input/WTAFile.hpp
- src/input/Schema.cpp
- src/input/WTAFile.cpp

Index