

The `mung2musicxml` module

NPGR070 Project Report

Bc. Vojtěch Dvořák

May 2026

Abstract

We present `mung2musicxml`, a converter from the Music Notation Graph (MuNG) format to MusicXML, addressing a long-standing gap in the optical music recognition (OMR) ecosystem. MuNG is a graph-based representation of Common Western Music Notation that captures the relational structure of notation objects, yet has seen limited adoption partly due to the absence of export to widely used digital music formats. Our converter bridges this gap through a pipeline of preprocessing steps — multi-stem resolution, pitch inference, duration and onset inference, and voice assignment — followed by export to MusicXML via ScoreGraph, a typed internal music representation designed to be constructed incrementally from potentially incomplete visual annotations. We further introduce a grammar system for validating MuNG annotations, extending prior work. The system is publicly available as a submodule of the `mung` Python library.

1 Introduction

The Music Notation Graph (MuNG) format, first proposed and implemented with the MUSCIMA++ dataset [9], was an elegant formalisation of the writing system of Common Western Music Notation (CWMN). It has long been known that in the object detection-based pipelines for optical music recognition, a “notation assembly” step is required after detecting the individual notation objects, because CWMN encodes musical semantics through combinations of individual notation objects rather than the isolated glyphs [2, 21]. Rather than relying on constituency grammars that group graphical primitives into complex “symbols” such as chords or beamed groups [4, 22, 23], MuNG models this “featural” nature of CWMN just by adding graph edges between symbols that need to be read in relation to each other. Most importantly, individual noteheads are connected by syntax-type edges to their applicable stems, beams, flags, accidentals, staff lines, and further

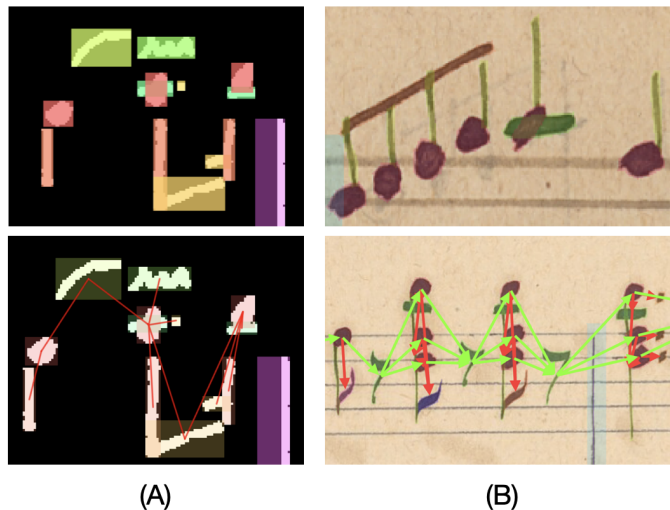


Figure 1: Examples of the MuNG representation: on the MUSCIMA++ dataset (A), and on newer data provided by the OmniOMR project (B). Top panel: symbol annotations only; in MUSCIMA++, the binary underlying images mean that only foreground pixels participate in the masks and hence bounding boxes are shown for clarity, while in the OmniOMR case the pixel-accurate symbol masks are shown. On the bottom, the edges are shown: syntactic (red), and precedence (green).

symbols that govern their interpretation in terms of pitch and duration; key signatures and clefs are connected to staves and staff lines as well. Precedence-type edges then connect symbols that encode duration (overwhelmingly noteheads and rests), so that onsets can be derived recursively. Examples of MuNG are shown in Figure 1.¹

MuNG was the first representation that allowed for a fully machine learning-driven pipeline for OMR for notations of arbitrary complexity [10], with the final step of inferring musical semantics from the graph implemented deterministically to output MIDI [10, 13].

However, beyond these initial works, MuNG has not seen much adoption. All the potential uses of MuNG — i.e. data synthesis [17], scribal identification, musical palaeography — have stayed mostly in the motivations and “future work” sections of corresponding OMR papers. A major reason for this is the lack of technical maturity on part of MuNG. While the proposed graph representation is elegant, and while prior work serves as solid proof of concept, the standard itself was developed only for the MUSCIMA++ dataset, which — while a substantial resource for OMR development —

¹Note: this report assumes the reader is at least somewhat familiar with the MuNG format. If not, we recommend the video explaining MuNG linked from the `mung` package README: <https://github.com/OmniOMR/mung>

does not quite represent the true scope of CWMN symbols and their relationships. And yet more importantly, only limited MIDI export, not MusicXML,² was implemented.

We report that the MuNG ecosystem is no longer in this unsatisfactory state. Our work implements a converter from the MuNG format to MusicXML, the most widely adopted standard format for digitally representing music notation. Specifically, we implemented the `mung2musicxml` module of the `mung` library. The conversion process has two main parts: preprocessing, where musical semantics such as pitches, durations, onsets of individual notes, or their assignment to voices and measures must be inferred, and the ScoreGraph representation that can be constructed from the preprocessed MuNG but mirrors the structure MusicXML defines and therefore can then be directly exported. To check for input quality, a new grammar for MuNG validation was also created.³ We thus provide a bridge between MuNG and the rest of the OMR ecosystem, and the ecosystem of software for manipulating music notation more broadly.

2 Related work

Beyond the recognition process itself [10, 20], graph-based structures have been used in OMR evaluation. The `musicdiff` system [6] operates on trees; the previously proposed tree edit distance metric [11] has been recently superseded by the more promising `musicdiff`-based OMR-NED [15].

A method that predicts a graph in a single step instead of two has been attempted [8]. Graph neural networks have also been tried for OMR directly [1], though only to a simplified synthetic dataset. A GNN-based approach has also been used for predicting voice and chord assignment, an essential step in estimating scores from MIDI [7].

Jiří Mayer developed SMASHCIMA [19], a tool for synthesizing handwritten music notation from MusicXML, where concurrent export of MuNGs for these synthesized pages is planned. Additionally, initial proof-of-concept systems have demonstrated that image-to-graph pipelines may offer a promising direction for OMR [5, 14].

Within the OmniOMR project [12], 100 pages of handwritten musical scores were annotated both in the MuNG format and in MusicXML[24]. The MuNG annotations were created using a recently published annotation tool MuNG Studio [18].

²<https://www.w3.org/2021/06/musicxml40/>

³Throughout this process, multiple other quality-of-life features were added to the rest of the `mung` library, though these are strictly speaking not a part of the `mung2musicxml` module.

3 Converting MuNG into MusicXML

Turning MuNG into MusicXML happens in two stages: **preprocessing** (multi-stem notehead resolving, pitch, duration, onset and voice inference), and **conversion** (loading the enriched MuNG into ScoreGraph, our music graph representation, and exporting that into MusicXML).

Additionally, we implemented a grammar architecture that is used during MuNG annotation to catch potential mistakes in relations between objects. This was essential to ensure that input data to the converter is valid.

3.1 Preprocessing

Before pitch, duration, and voice inference can take place, the raw MuNG graph requires several preparatory steps. Multi-stemmed noteheads — which arise when two voices share a pitch — must be split into separate noteheads with modified links. Pitch inference then resolves staff positions into concrete pitches. Duration and onset inference computes the length and start time of each durable, processing the score measure by measure to avoid cross-boundary conflicts. Finally, voice inference assigns each durable to a voice, respecting shared beams, chords, and tuplets.

Pitch and duration inference engines are based on approaches implemented in [9].

3.1.1 Multi-stem resolution

Before any other inference steps are run, noteheads with more than one stem must be resolved into two separate noteheads. Multi-stemmed noteheads arise when two voices share the same pitch simultaneously. The resolution duplicates the notehead and modifies links in the surrounding graph accordingly. The complexity comes from the fact that the two notes may have different durations and belong to different musical constructions (beams, tremolos, slurs, etc.). An example is shown in Figure 2.

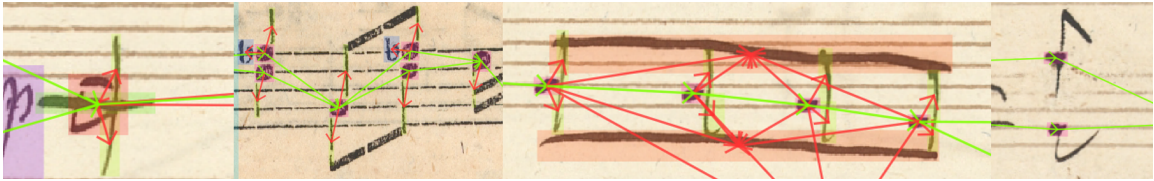


Figure 2: Examples of double-stemmed noteheads. The rightmost example resolves into two noteheads with differing durations. Images and annotations are taken from [24].

Since the original notehead carries links to other MuNG objects, its linked objects are separated into two categories: *shared* objects, to which both noteheads have a link, and *divided* objects, which

are assigned to exactly one of the two noteheads.

By default, *shared* objects include staff lines, staff spaces, staves, ledger lines, and accidentals; *divided* objects include flags, beams, single tremolos, and tremolo beams. Each *divided* object is assigned to the notehead whose stem points toward it — for instance, a beam above the noteheads is assigned to the notehead with an upward stem.

3.1.2 Pitch inference

For each staff, pitch modifiers (clefs and key signatures), noteheads, and measure separators are collected and sorted left to right. These symbols are then processed sequentially: the engine infers pitches for noteheads and updates the current pitch state based on clefs, key signatures, accidentals, and measure separators⁴. Notehead is assigned pitch based on the pitch state and its linkage to staff positions (staff space or staff line) or ledger lines.

To handle incomplete notation at system boundaries, we introduced a carry-over mechanism: staff inherits the corresponding modifiers from staff in the previous system and adjusts its pitch state accordingly before processing any of its noteheads. This helps to mitigate missing clef or key signature at the start of a system. We introduce the concept of instruments — previously, staves were processed in isolation, with no mechanism to propagate state across consecutive systems. An example of such a score is shown in Figure 3.



Figure 3: Example of a score where clef and key signature are missing in the second system.

3.1.3 Duration and onset inference

Duration of a durable is computed based on its type (whole, half, quarter, etc.) and modifiers linked to it — flags, beams, augmentation dots, tuplets and tremolo beams.

⁴Accidentals affect only subsequent noteheads within the same measure, until the next measure separator is met.

Every durable in a system measure should end before any durable from the succeeding system measure starts. To entirely rule out the possibility of violation of this rule, we separate the score into individual system measure and process them in sequence individually. The MuNG precedence graph is a DAG, we topologically sort it, every durable then asks its precedence parents for the maximal end onset (onset + duration). The algorithm is described further in Algorithm 1.

Algorithm 1 Onset inference

Require:

MuNG precedence graph of a single system measure $G = (V, E)$,
preceding system measure end onset *preceding_end_onset*

```

1: sorted_vertices  $\leftarrow$  TOPOLOGICALSORT( $G$ )
2: for each  $v$  in sorted_vertices do
3:   parents  $\leftarrow$  GETPRECEDENCEPARENTS( $v$ )
4:   parents  $\leftarrow$  parents  $\cap V(G)$  ▷ Take only those that are in the system measure
5:   if parents =  $\emptyset$  then
6:      $v.onset \leftarrow preceding\_end\_onset$ 
7:   else
8:      $max\_end \leftarrow 0$ 
9:     for each  $p \in parents$  do
10:       $end\_onset \leftarrow p.onset + p.duration$ 
11:       $max\_end \leftarrow \max(max\_end, end\_onset)$ 
12:     end for
13:      $v.onset \leftarrow max\_end$ 
14:   end if
15: end for
16: return  $\{v.onset \mid v \in V\}$ 

```

3.1.4 Voice inference

A voice is a sequence of musical events that proceeds linearly in time. Music notation can record multiple voices concurrently; an example of two overlapping voices is in Figure 4. The concept of voices is not exclusive to MusicXML, other formats such as **music21**⁵ and ****kern**⁶ use it. Voices are

⁵<https://music21.org/music21docs/>

⁶<https://www.humdrum.org/guide/ch02/>



Figure 4: Example of two overlapping voices on the same staff, first one is blue, second one is green.

an essential part of the internal logic of music, but they are not explicitly a part of MuNG, because they are an abstraction over sets of notes; therefore, they must be inferred during the conversion process.

Based on observation of how MuseScore Studio 4⁷ (MSS4) handles voices, we define a list of conventions:

1. Durables that share beam, chord, or tuple must be assigned to the same voice.
2. Durables on the same staff, that overlap, and do not belong to the same chord, must be assigned to different voices.
3. In grand staff, voices are notated using ids 1-4 for the top staff and 5-8 for the bottom staff. Single staff instruments have voice ids 1-4. (This is directly derived from MSS4 output, MusicXML documentation does not imply this in any way.)
4. Voice ids are sorted and assigned to durables based on the total time played (inside the measure) and the average pitch of notes assigned. (Voice with the lowest id takes up the most of the duration of a measure.)
5. The lower the voice id, the more important it is.

When inferring voices, we work at the level of measures. If working with grandstaff, we first we need to separate the durables into top and bottom staff (*Convention 3*). This task would be trivial, if it wasn't for cross-staff notation — triplets and beams that are linked to noteheads in both staves. Even chords can span across multiple staves. Example of a cross-staff notation is in Figure 5. We solve this by grouping durables by shared cross-staff objects and then assigning the whole group to a staff which is linked to the earliest (lowest onset) durable in that group, chords are assigned to topmost staff.

⁷The most widely used free software for score editing, <https://musescore.org/>.

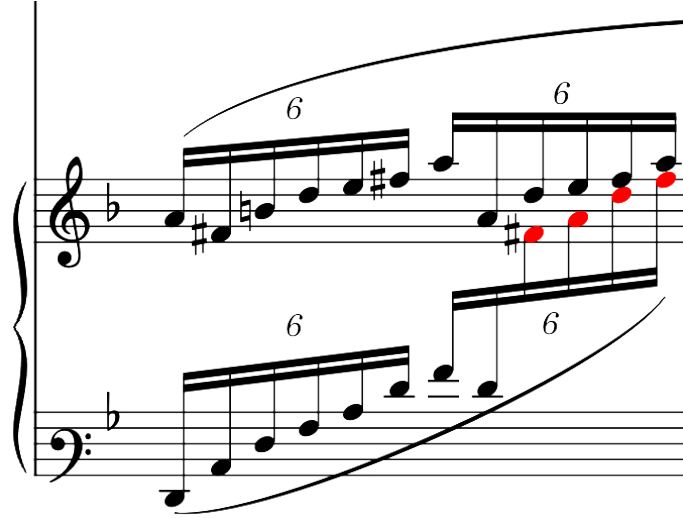


Figure 5: Example of a cross staff notation. Red noteheads belong to different staff than the rest of the noteheads that share beams with them.

Working at measure level now, the voice graph is constructed. For every source, edge to the closest preceding durable is added, if it exists, if multiple durables exist with the same end onset, we add multiple edges. This ensures correct propagation of voices throughout the graph. Overview of a voice graph construction is in Figure 6.

Next, we find the widest part of the voice graph, we imagine the durables as events in a schedule and find the place where the most events overlap. Durable from the widest place are sorted from top to bottom and assigned voices $1 - n$, then they are put into priority queue with priority computed as tuple $(\text{voice_id}, \text{y_coordinate})$. Durables are popped from the queue, then their children and parents are assigned available voice ids iteratively, if they don't have them yet. The algorithm is implemented in `mun2musicxml/preprocessing/voices`.

3.2 ScoreGraph and MusicXML export

After all the necessary preprocessing is done, MuNG is loaded into our internal music representation called *ScoreGraph* — the central component of this work. *ScoreGraph* is a graph of interconnected typed classes. It can be loaded from a (preprocessed) MuNG file and then directly exported to MusicXML, because the class implementation follows the official MusicXML documentation.⁸

The problem *ScoreGraph* solves is that it must be constructed gradually bottom-up from possibly

⁸Loader, exported and *ScoreGraph* implementation are independent of each other and derived from abstract classes so that loaders/exporters from/to other formats can be implemented.

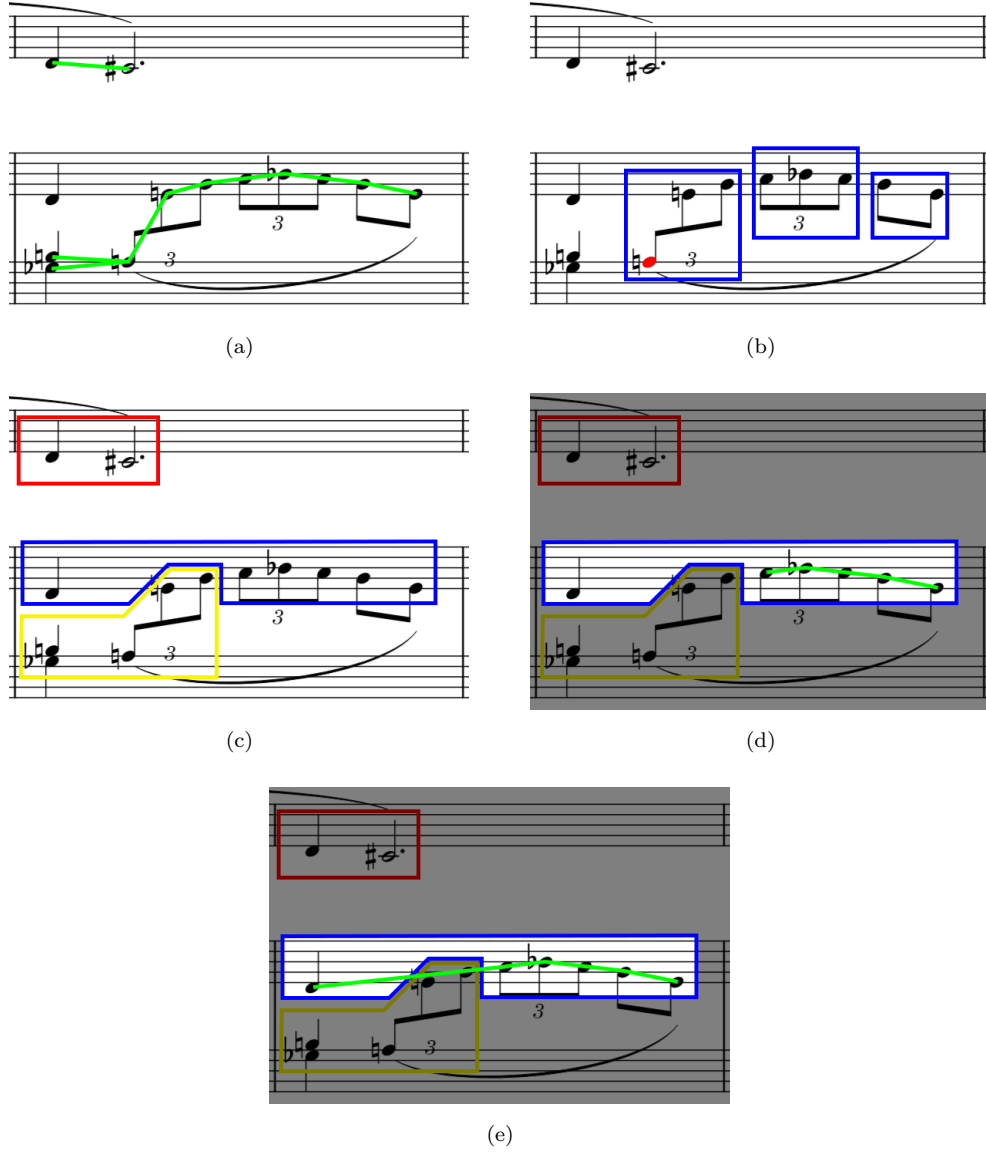


Figure 6: Example of a graph construction to infer voices. MuNG precedence graph in **(a)** contains two instruments, single-staff at the top and grandstaff at the bottom, edges are oriented from left to right. In **(b)**, possibly cross-staff objects are found and their durables are grouped together, the highlighted red notehead causes the its tuplet to be assigned to the bottom staff. **(c)** shows the final separation of durables into measures. One of these measures is shown in **(d)**. Now working with individual measure, source noteheads are linked to their closest preceding noteheads, as shown in **(e)**.

incomplete or malformed notation. This is something that existing data models for notated music (such as music21) do not support: they assume an encoding of the music notation already exists that contains the logical elements such as systems, measures, or voice assignments, and can be parsed top-down. MuNG only captures the visual elements of notation on the page, not this interpretative logical layer, which must first be constructed. This also precludes directly re-using the internal representation used in the Smashcima synthesis system [19], which operates with a fully formed MusicXML file as input.

3.2.1 ScoreGraph

Unlike MuNG, which uses a single generic node type distinguished only by an attribute (e.g., `Node("notehead")`), ScoreGraph defines a dedicated class for each musical object. This allows for further specialization: a relation such as `Node("beam") -> Node("notehead")` becomes the more expressive `Beam() -> Notehead()`. Score hierarchy is implemented via dedicated score, part, part measure and score measure classes. A simplified example of the graph structure is visualized in Figure 7.

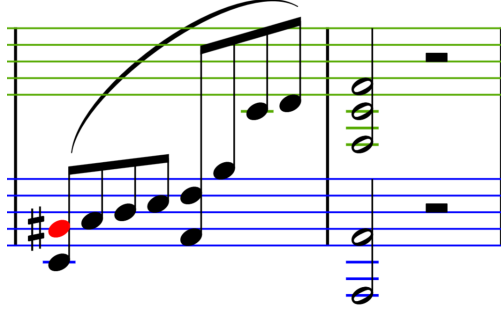
The complexity of ScoreGraph does not lie in any algorithmic issues; rather, it lies in the breadth of music notation itself, and the incompatibility between the bottom-up approach (necessitated by the direction in which OMR processes an image, and therefore adopted by MuNG), and the top-down approach used to design MusicXML (which follows from “rules of music notation” and a specific object model that, while useful for music notation editors, is not necessarily always followed in real-world musical scores). MuNG defines 151 distinct symbol classes, which all appear in the OmniOMR data.

We separate classes defined by MuNG into four types: implemented (i.e. noteheads, beams), implemented as part of other ScoreGraph object (i.e. flags, stems), partially implemented (grace noteheads), not supported (i.e. repeats). In the UFAL.OmniOMR dataset [24], there are 100 pages with full MuNG annotations, with a total of 105k objects. Out of these, 50k are implemented, 52k are part of another object, 400 are partially implemented and only 2k are not supported. The full list of MuNG classes that are supported in MusicXML export can be found in exporter’s documentation⁹

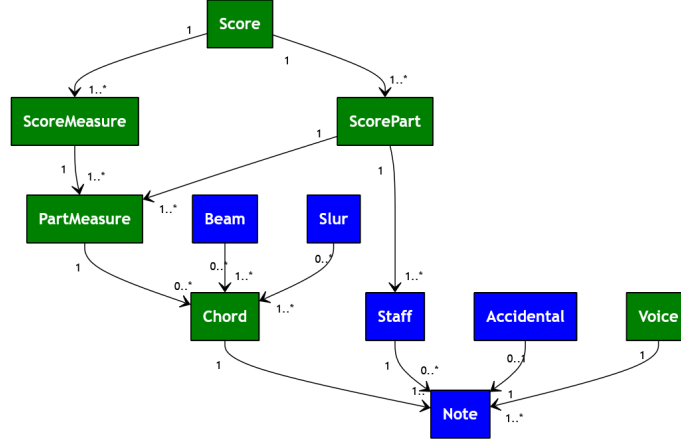
3.2.2 Loading MuNG into ScoreGraph

First, the MuNG staves are separated into individual instruments (parts), which then must be linked together across multiple systems. Example of instruments is in Figure 8. The algorithm can handle

⁹[https://github.com/OmniOMR/mung/blob/main/mung2musicxml/score_graph/docs/supported-mung-classes.](https://github.com/OmniOMR/mung/blob/main/mung2musicxml/score_graph/docs/supported-mung-classes.md)



(a)



(b)

Figure 7: Example of a possible score **(a)** consisting of two instruments: a single-staff instrument shown in red, and a grand staff represented by the green and blue staves. The note highlighted in red belongs both to a chord and to the blue staff. Connections of a notehead highlighted in red to other ScoreGraph classes are illustrated in graph **(b)**. ScoreGraph objects directly constructed from MuNG objects are highlighted in blue and the abstract classes introduced in our ScoreGraph implementation are highlighted in green. A Chord is defined as the union of all notes sharing the same stem. A ScorePart is a unique object representing a single instrument and contains all of its staves and measures (PartMeasures). A ScoreMeasure groups all PartMeasures across instruments that occur simultaneously in time. The Score object forms the root of the ScoreGraph hierarchy. Voice is a sequence of musical events that proceeds linearly in time, each ScorePart defines its own Voice objects.



Figure 8: Example of a score separation into instruments. There two systems, both made out of two instruments, single-staff in red and grandstaff in blue. Instrument is a list of staves created by collecting its staves throughout the systems from top to bottom. [24]

situation where staff of one of the instruments is missing in some of the systems, i.e. vocal with a piano, where the vocal is not present in the first system.

Then the staff (or grandstaff) is processed by individual measures. Symbols inside a measure are converted into their own ScoreGraph equivalents and are linked to the instrument’s Staff and Voice objects.

3.2.3 Exporting ScoreGraph to MusicXML

Exporting MusicXML from a ScoreGraph representation is then done recursively part-wise and then measure-wise within each part. The ScoreGraph objects are designed to correspond to MusicXML objects and their hierarchy as much as possible. However, the ScoreGraph is a DAG. Certain elements must therefore be visited more than once: typically, `<beam>` and other elements that span over multiple noteheads must be created for each participating note. If an error occurs during MusicXML export within a measure, an empty measure is output (with an invisible measure rest).

3.3 Known limitations

We encountered three types of limitations: those related to the exporter itself, limitations of MusicXML, and in the process we also found limitations of MuseScore for inspecting the conversion results and using it for MusicXML canonization.

MusicXML is unable to handle one rare phenomenon that we in fact encountered in the notation: slurs and ties cannot be connected to symbols for repeating a measure. The list of notation phenomena that are captured in MuNG 2.0 but are not currently exportable into MusicXML are mainly related to jumps (segno, coda) and complex repeats. The full list of limitations is part of

the documentation.¹⁰.

4 Converter evaluation

The converter was evaluated against ground-truth MusicXML transcriptions using our in-house UFAL.OmniOMR dataset [24], which contains 100 musical scores annotated in both MuNG and MusicXML. Following the taxonomy of [3], we partition the dataset into four notation complexity categories: monophonic, homophonic, polyphonic, and pianoform.

In Table 1, we report results under three evaluation scenarios using the OMR Normalized Edit Distance (OMR-NED) metric [6, 16], which operates on musical objects rather than raw XML text, as implemented in `musicdiff`¹¹:

- **All** – All objects in both scores are compared, including those not supported by our converter (`musicdiff` flag: `allobjects`).
- **Implemented** – Only objects fully supported by our converter are compared (`musicdiff` flags: `notesandrests`, `beams`, `tremolos`, `ties`, `slurs`, `articulations`).
- **Bare** – Only notes and rests are compared, using staff position rather than diatonic pitch to avoid cascading errors from clef or key signature mismatches (`musicdiff` flags: `notesandrests`, `notestaffposition`).

Notation Complexity	Bare	Implemented	All	Count
Monophonic	0.12	0.17	0.21	26
Homophonic	0.14	0.22	0.26	28
Polyphonic	0.16	0.28	0.32	13
Pianoform	0.07	0.08	0.13	33
All	0.11	0.16	0.20	100

Table 1: OMR-NED scores per notation complexity category and evaluation scenario (lower is better), computed as a micro average. Results are rounded to two decimal places.

Visual inspection confirms that the converter produces output closely matching the input scores. However, three systematic issues inflate the reported error rates. First, approximately 10% of the ground-truth MusicXML transcriptions contain incorrectly assigned instruments, which can drive

¹⁰https://github.com/OmniOMR/mung/blob/dev/mung2musicxml/score_graph/docs/

¹¹<https://github.com/gregchapman-dev/musicdiff>

OMR-NED values as high as 0.85 and disproportionately affect aggregate results. Second, several scores lack explicit clef annotations, and the corresponding MusicXML transcriptions use clefs that differ from the project-agreed default (C clef). This discrepancy accounts for the pronounced gap between the **Implemented** and **Bare** scenarios. Third, annotators tend to make assumptions and fill in gaps or correct other errors in the score; naturally, our converter cannot do these operations. Authors of the dataset were notified of these errors.

5 MuNG grammar

A necessary precondition for the converter was to write a validation module for the MuNG specification, so that malformed inputs would be caught before passed to the converter and preferably during the annotation process.¹² The grammar checks links between individual objects. Example of a warning raised by the grammar inside MuNG Studio [18] is shown in Figure 9.

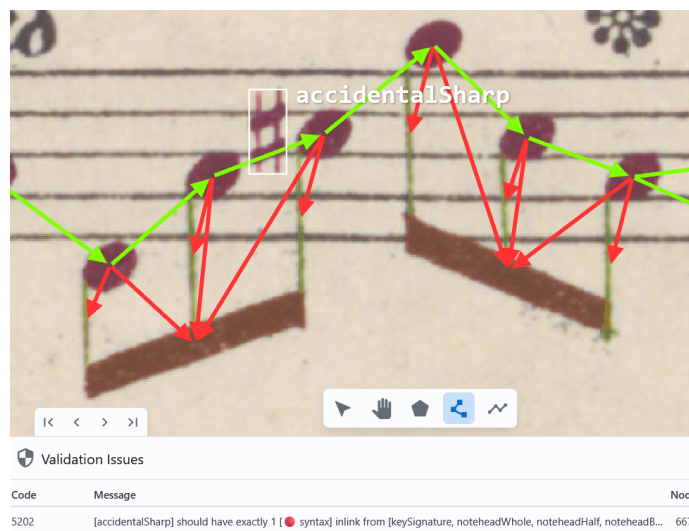


Figure 9: Example usage of the new grammar inside MuNG Studio [18]. The highlighted accidental should be connected to exactly one key signature or notehead. Validation issues are shown to the annotator on request on the bottom of the UI.

The old grammar developed as part of MUSCIMA++ [9] was able to define allowed links as (`notehead*` | `augmentationDot`) — notehead can have an outgoing edge to augmentation dot; and cardinalities as (`(` | `beam1,`) — beam has to have at least one incoming link from **any** other object.

¹²Work on this started already as the project proposal was being written and continued in M1-2

This is not specific enough to eliminate non-trivial annotation errors, which is why we introduce a new grammar system. The main difference is the introduction of *tokens* that allow the user to be more specific. Tokenized rules contain token on at least one side. Token is a keyword that groups together multiple symbols. This is an upgrade over the previous implementation, where defining a rule i.e. “Articulation has to be linked to a notehead” was not possible. We were able allow the existence of edges (`noteheadFull | articulation`), (`noteheadHalf | articulation`), ..., but introducing any cardinalities would lead to a much stricter rule that does not hold in general. In the end, we would have to settle for a rule (`notehead* | articulation*`), without any cardinality specification, which allows the existence of articulations without any connection to a notehead.

We introduce two of these tokens with examples. `ANYOF` results in the grammar checking the aggregated cardinality of all the symbols enclosed by the token. We can define the above mentioned rule for articulations as:

```
ANYOF(notehead*) | articulation*{1,}
```

- Noteheads can link to any number of articulations.
- Articulation has to have link from at least one notehead.

The second token is `EXACTLYONE`, allowing the grammar to specify exclusive relations, i.e. “Accidental should always be a child of some notehead or (exclusive) a key signature”:

```
EXACTLYONE(notehead* keySignature) | accidental*{1}
```

The grammar is loaded from a list of rules and can be used directly on a `MuNG`, the validator returns a list of found errors. More examples are in the grammar’s documentation¹³.

6 Module overview

The `mung2musicxml` converter is implemented as a submodule of the `mung` Python library whose working directory is available on GitHub, <https://github.com/OmniOMR/mung/mung2musicxml>. It is structured into the following subdirectories

- `demo` – Assets that can be used for demo conversion immediately after cloning.
- `grammar_new` – Working directory of the new implemented grammar.
- `inference` – Pitch, duration and onset inference wrappers based on the original implementation from [9].

¹³https://github.com/OmniOMR/mung/tree/main/mung2musicxml/grammar_new

- **preprocessing** – Multi-stem notehead resolution and voice inference.
- **score_graph** – Implementation of a internal score representation with typed classes for each musical object.

7 Publications and future use

This work is a key part of a submission *The Music Notation Graph (MuNG) ecosystem 2.0: ready for production* by the Prague Music Computing Group¹⁴ and our OmniOMR colleagues from MZK to WoRMs,¹⁵ a workshop at the ICDAR 2026 conference.

Parts of this work were used as infrastructure elements for the paper *A dataset for the recognition of historical and handwritten music scores in western notation*: snapping of noteheads to staves and staff positions, and grammar-based graph validation. These functions were essential for efficient quality control of the resulting dataset. The paper was co-authored by Jan Hajič jr., Jiří Mayer, colleagues from Moravská Zemská Knihovna (MZK), and the OMR research group of prof. Alicia Fornés at the Universitat Autònoma de Barcelona; it was submitted to the Nature Scientific Data journal and is available online at arXiv.¹⁶

Furthermore, this converter will serve as a basis for my future OMR research and diploma thesis. It will also be essential for experimental work within the OmniOMR project: without it, a head-to-head comparison between the MuNG-based object detection pipeline and end-to-end model paradigms of OMR would not be possible.

8 Changes compared to the Proposal

In the original proposal, we planned to distribute the converter as part of the `mung` package via `pip`, with contributions merged into the parent OMR-Research repository¹⁷. During the project, however, the team determined that both the `mung` and `mung2midi` modules require extensive refactoring outside the scope of this project before this is feasible. In the meantime, all contributions are available in our fork and will be merged into the main repository once the refactoring is complete.

¹⁴<https://ufal.mff.cuni.cz/pmcg>

¹⁵Workshop on reading music systems, <https://icdar2026.org/index.php/workshops/>

¹⁶<https://arxiv.org/abs/2605.18436>

¹⁷<https://github.com/OMR-Research/mung>

9 Contribution of other team members

- `mung2musicxml` is part of the `mung` library originally developed by Jan Hajič jr. and Alexander Pacha.
- Annotation instructions for MuNG that are essential for understanding the MuNG format and for ScoreGraph loader implementation were written by Jiří Mayer.¹⁸
- MuNG grammar, that was defined and implemented as part of this project, is used in MuNG Studio¹⁹ developed by Jiří Mayer na Čeněk Řehoř, to catch potential mistakes in annotations. They did not participate in grammar implementation; they integrated it into MuNG studio. Jiří Mayer wrote the grammar rules²⁰ which are then parsed by our grammar implementation.
- The UFAL.OmniOMR dataset (part of MusiCorpus 1.0), on which the converter and grammar were tested, was created by our colleagues from MZK — Filip Bím, Martina Dvořáková, Markéta Herzanová Vlková, Petr Žabička.
- Introduction for this report was co-written with Jan Hajič jr.

¹⁸<https://github.com/OmniOMR/mung/blob/main/docs/annotation-instructions/annotation-instructions.md>

¹⁹<https://github.com/OmniOMR/mung-studio>

²⁰https://github.com/OmniOMR/mung-studio/blob/main/pyodide/mstudio/mstudio/validation/grammar_

References

- [1] Arnau Baró, Pau Riba, and Alicia Fornés. “Musigraph: Optical Music Recognition Through Object Detection and Graph Neural Network”. In: *Frontiers in Handwriting Recognition*. Springer International Publishing, 2022, pp. 171–184. ISBN: 9783031216480. DOI: 10.1007/978-3-031-21648-0_12. URL: http://dx.doi.org/10.1007/978-3-031-21648-0_12.
- [2] Jorge Calvo-Zaragoza, Jan Hajič Jr., and Alexander Pacha. “Understanding Optical Music Recognition”. In: *ACM Comput. Surv.* 53.4 (July 2021), pp. 1–35. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3397499. (Visited on 10/26/2022).
- [3] Jorge Calvo-Zaragoza, Jan Hajič jr., and Alexander Pacha. “Understanding Optical Music Recognition”. In: *ACM Computing Surveys* 53.4 (2020), p. 77. DOI: 10.1145/3397499.
- [4] Bertrand Coüasnon. “DMOS, a generic document recognition method: application to table structure analysis in a general and in a specific way”. en. In: *International Journal of Document Analysis and Recognition (IJDAR)* 8.2 (June 2006), pp. 111–122. ISSN: 1433-2825. DOI: 10.1007/s10032-005-0148-5. URL: <https://doi.org/10.1007/s10032-005-0148-5> (visited on 09/17/2024).
- [5] Vojtěch Dvořák. “Fast Optical Music Recognition Using the YOLO Platform”. Bachelor’s thesis. Praha: Charles University, Faculty of Mathematics and Physics, 2025. URL: <http://hdl.handle.net/20.500.11956/200867>.
- [6] Francesco Foscarin, Florent Jacquemard, and Raphael Fournier-S’niehotta. “A diff procedure for music score files”. In: *6th International Conference on Digital Libraries for Musicology*. 2019, pp. 58–64.
- [7] Francesco Foscarin, Emmanouil Karystinaios, Eita Nakamura, and Gerhard Widmer. “Cluster and Separate: A GNN Approach to Voice and Staff Prediction for Score Engraving”. In: *Proceedings of the 25th International Society for Music Information Retrieval Conference, ISMIR 2024, San Francisco, California, USA and Online, November 10-14, 2024*. Ed. by Blair Kaneshiro, Gautham J. Mysore, Oriol Nieto, Chris Donahue, Cheng-Zhi Anna Huang, Jin Ha Lee, Brian McFee, and Matthew C. McCallum. 2024, pp. 503–510. DOI: 10.5281/ZENODO.14877385. URL: <https://doi.org/10.5281/zenodo.14877385>.
- [8] Carlos Garrido-Munoz, Antonio Rios-Vila, and Jorge Calvo-Zaragoza. “A holistic approach for image-to-graph: application to optical music recognition”. en. In: *International Journal on Document Analysis and Recognition (IJDAR)* (Sept. 2022). ISSN: 1433-2825. DOI: 10.1007/

- s10032-022-00417-4. URL: <https://doi.org/10.1007/s10032-022-00417-4> (visited on 10/25/2022).
- [9] Jan Hajič and Pavel Pecina. “The MUSCIMA++ Dataset for Handwritten Optical Music Recognition”. In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 01. Nov. 2017, pp. 39–46. DOI: 10.1109/ICDAR.2017.16.
 - [10] Jan Hajič jr., Matthias Dorfer, Gerhard Widmer, and Pavel Pecina. “Towards Full-Pipeline Handwritten OMR with Musical Symbol Detection by U-Nets”. In: *19th International Society for Music Information Retrieval Conference (ISMIR)*. Paris, France, 2018, pp. 225–232. URL: http://ismir2018.ircam.fr/doc/pdfs/175_Paper.pdf.
 - [11] Jan Hajič jr., Jiří Novotný, Pavel Pecina, and Jaroslav Pokorný. “Further Steps towards a Standard Testbed for Optical Music Recognition”. In: *17th International Society for Music Information Retrieval Conference (ISMIR)*. New York, USA, 2016, pp. 157–163. URL: <https://wp.nyu.edu/ismir2016/event/proceedings/>.
 - [12] Jan Hajič jr., Petr Žabička, Jan Rychtář, Jiří Mayer, Martina Dvořáková, Filip Jebavý, Markéta Vlková, and Pavel Pecina. “The OmniOMR Project”. In: *5th International Workshop on Reading Music Systems (WoRMS)*. Milan, Italy, 2023, pp. 12–14. DOI: 10.48550/arXiv.2311.04091.
 - [13] Jan Hajic jr., Marta Kolárová, Alexander Pacha, and Jorge Calvo-Zaragoza. “How current optical music recognition systems are becoming useful for digital libraries”. In: *Proceedings of the 5th International Conference on Digital Libraries for Musicology, DLfM 2018, Paris, France, September 28, 2018*. Ed. by Kevin R. Page. ACM, 2018, pp. 57–61. DOI: 10.1145/3273024.3273034. URL: <https://doi.org/10.1145/3273024.3273034>.
 - [14] Yipeng Liu, Ruimin Wu, Yifan Wu, Lijie Luo, and Wei Xu. “A Stave-Aware Optical Music Recognition on Monophonic Scores for Camera-Based Scenarios”. In: *Applied Sciences* 13.16 (2023). ISSN: 2076-3417. DOI: 10.3390/app13169360. URL: <https://www.mdpi.com/2076-3417/13/16/9360>.
 - [15] Juan C. Martinez-Sevilla, Joan Cerveto-Serrano, Noelia Luna, Greg Chapman, Craig Sapp, David Rizo, and Jorge Calvo-Zaragoza. *Sheet Music Benchmark: Standardized Optical Music Recognition Evaluation*. June 2025. DOI: 10.48550/arXiv.2506.10488. arXiv: 2506.10488 [cs]. (Visited on 01/19/2026).
 - [16] Juan C. Martinez-Sevilla, Joan Cerveto-Serrano, Noelia Luna-Barahona, Greg Chapman, Craig Sapp, David Rizo, and Jorge Calvo-Zaragoza. “Sheet Music Benchmark: Standardized Optical Music Recognition Evaluation”. In: *Proceedings of the 26th International Society for*

- Music Information Retrieval Conference* (Daejeon, South Korea and Online). ISMIR, Sept. 2025, pp. 618–625. DOI: 10.5281/zenodo.17811446. URL: <https://doi.org/10.5281/zenodo.17811446>.
- [17] Jirí Mayer and Pavel Pecina. “Synthesizing Training Data for Handwritten Music Recognition”. In: *16th International Conference on Document Analysis and Recognition, ICDAR 2021, Lausanne, Switzerland, September 5-10, 2021, Proceedings, Part III*. Ed. by Josep Lladós, Daniel Lopresti, and Seiichi Uchida. Lecture Notes in Computer Science. Springer, 2021, pp. 626–641. DOI: 10.1007/978-3-030-86334-0_41. URL: https://doi.org/10.1007/978-3-030-86334-0_41.
- [18] Jirí Mayer, Filip Jebavý, Markéta Herzánová Vlková, Martina Dvořáková, Pavel Pecina, and Jan Hajič. “MuNG Studio: Annotation Tool for Music Notation Graph”. In: *Proceedings of the 12th International Conference on Digital Libraries for Musicology. DLfM ’25*. New York, NY, USA: Association for Computing Machinery, Sept. 2025, pp. 114–118. ISBN: 979-8-4007-2083-3. DOI: 10.1145/3748336.3748379. (Visited on 09/29/2025).
- [19] Jirí Mayer, Pavel Pecina, and Jan Hajič. “Smashcima: Full-Page Handwritten Music Document Synthesizer”. In: *Proceedings of the 12th International Conference on Digital Libraries for Musicology. DLfM ’25*. New York, NY, USA: Association for Computing Machinery, Sept. 2025, pp. 119–123. ISBN: 979-8-4007-2083-3. DOI: 10.1145/3748336.3748380. (Visited on 09/29/2025).
- [20] Alexander Pacha, Jorge Calvo-Zaragoza, and Jan Hajic jr. “Learning Notation Graph Construction for Full-Pipeline Optical Music Recognition”. In: *Proceedings of the 20th International Society for Music Information Retrieval Conference, ISMIR 2019, Delft, The Netherlands, November 4-8, 2019*. Ed. by Arthur Flexer, Geoffroy Peeters, Julián Urbano, and Anja Volk. 2019, pp. 75–82. URL: <http://archives.ismir.net/ismir2019/paper/000006.pdf>.
- [21] Ana Rebelo, Ichiro Fujinaga, Filipe Paszkiewicz, Andre R. S. Marcal, Carlos Guedes, and Jaime S. Cardoso. “Optical Music Recognition: state-of-the-art and open issues”. In: *International Journal of Multimedia Information Retrieval* 1.3 (Mar. 2012), pp. 173–190. ISSN: 2192-662X. DOI: 10.1007/s13735-012-0004-6. URL: <http://dx.doi.org/10.1007/s13735-012-0004-6>.
- [22] Philippe Rigaux, Bertrand Couasnon, Christophe Guillotel-Nothmann, Fabien Guilloux, and Aurélie Lemaitre. “The CollabScore dataset -towards robust and generalized OMR evaluation”. tex.hal.id: hal-05515751 tex.hal.version: v1. Jan. 2026. DOI: 10.1145/nnnnnnnn.nnnnnnnn. URL: <https://hal.science/hal-05515751>.

- [23] Pau Torras, Sanket Biswas, and Alicia Fornés. “A Unified Representation Framework for the Evaluation of Optical Music Recognition Systems”. In: *IJDAR* 27.Special issue on “advanced topics in document analysis and recognition” (July 2024), pp. 379–393. ISSN: 1433-2825. DOI: 10.1007/s10032-024-00485-8. (Visited on 07/24/2024).
- [24] Pau Torras, Jiří Mayer, Carles Badal, Martina Dvořáková, Markéta Herzanová Vlková, Gerard Asbert, Vojtěch Dvořák, Samuel Šomorjai, Jan Hajič, and Alicia Fornés. *A Dataset for the Recognition of Historical and Handwritten Music Scores in Western Notation*. 2026. DOI: 10.48550/ARXIV.2605.18436. URL: <https://arxiv.org/abs/2605.18436>.