Using answer set programming to detect and compose music in the structure of twelve bar blues

Flavio Everardo[1,2,3], Gabriel Rodrigo Gil[1], Oscar B. Pérez Alcalá[1], and Gabriela Silva Ballesteros[1]

[1] Tecnológico de Monterrey, Puebla Campus, Mexico.
[2] University of Potsdam, Germany
[3] 750 Records
flavio.everardo@{tec.mx;cs.uni-potsdam.de}

Abstract. The inclusion of Artificial Intelligence (AI) in the music domain has attracted the attention of different research directions towards intelligent solutions for different users, being students or professionals. On the other hand, Answer Set Programming (ASP), has over a decade of music production-related contributions going in directions from composition up to mixing. To continue to empower the user with intelligent solutions under ASP, we focus this paper to combine ASP with the Twelve Bar Blues musical structure. This structure is well known and a very common musical structure used in many popular genres. This work intends to support students during their music formation. To do this, we present the first of a two-stages ASP-based system towards helping the student to train her listening, and at the same time, be aware and understand musical structures. This initial stage receives a (partial) composition, perform an analysis, and returns feedback to the user in the form of error detection and notifications of chords or partial structures out of the Twelve Bar Blues.

Keywords: Answer Set Programming, ASP, Twelve Bar Blues, Musical Structures, Intelligent Music Production

1 Introduction

The inclusion of Artificial Intelligence (AI) in the music domain has attracted the attention of different research directions to contribute to any stage of music production [19,20], where intelligent solutions are required to help the user to outperform at any level, being under their studies or professionally. Regardless of the level where the user is currently situated (may it be a composer, sound designer, mixing, or mastering engineer to say a few), there is a fundamental aspect that all share at a professional level, and it is a proper listening and musical training. Talking concretely about students, they should work to develop the necessary skills under their formation. This may sound simpler in principle, however, it takes practice and years to achieve.

Continuing with intelligent music solutions, Answer Set Programming [1] or ASP for short, has over a decade of music production-related contributions, either for composition or even to mixing. ASP and music met back in 2008 when *Anton*, the first system for music composition [9] abled to compose melodic, harmonic and rhythmic music in the style of Palestrina. This system also helps to diagnose errors in human compositions and serve as a computer-aided composition tool. Since then, composing in ASP derived in other works. In 2011, an *Anton*-based system named *Armin* [10] composed electronic music, particularly a primer for the trance music genre. This tool can compose basic beats progressions as a starting point for electronic music producers. In a similar direction, [11] proposed melodic variations to any given input composition. From 2015, other systems started to compose with different chords progression. In [12], the authors presented *chasp*, an automatic music composition system that creates simple accompanying pieces of different genres focusing on chord progressions and harmony rules. [13] shows a computational system that invents novel cadences and chord progressions based on blending. A year later, [14] presented a song generation system that combines music and lyrics generation into a single generative phase. The advantage of this approach in comparison to previous work on automated song composition is in its seamless combination of language and music. *Haspie* [15] is a musical harmonization and composition assistant with features such as preferred harmonization and score completion from 2016. Currently, the only work outside the composition and arrangement is focus on multitrack mixing [16] from 2017, proving a concept for multitrack stereo mixing based on best practices from the literature. This work aims to demonstrate the use of this declarative approach to achieve a well-balanced mix by compiling a knowledge base with rules and constraints about what professional music producers and audio engineers suggest creating a good mix.

We have described all the published works where music and ASP converge. Still, at the moment of this writing, we are aware of a complete work, but not yet published. [4] This work takes musical pieces converted to MIDI [21] files and extracts different patterns. These patterns form an essential block to compose music ideally close to the input songs. In other words, these common features of similar music are exploited to create new music, making this system genre-independent, and both the analysis and the composer are entirely encoded in ASP.

As we show above, it's been over a decade and still, the combination of ASP and music is in its first steps. To continue to empower the user with intelligent solutions under ASP, we focus this paper to combine ASP with the Twelve Bar Blues musical structure [17].

The Twelve Bar Blues is a well-known and very common musical structure used in many genres for decades including pop, rock, and jazz music to say a few [17]. We refer to this structure independently of the genre of the blues, however as the name says, they are related. Musically, this structure is taught during the formation of students and is key to understanding more complex compositions. This structure, or sometimes referred to as pattern, consists exactly of 12 bars repeated over time with some hard rules. Some of these rules involve a specific progression build from chord structures and duration. In its essence, the progression relies predominantly based on the I, IV, and V chords of a key [17,18]. making the compositions to sound "recognizable" or "predictable".

[4] This work is an undergraduate thesis by Leo Pinetzki, supervised by the first author of this paper.

To help the student to train her listening at the same time be aware and understand musical structures, this work presents the design and an initial implementation of a two-stages ASP-based system. This initial stage receives a (partial) composition, perform an analysis, and returns feedback to the user in the form of error detection and notifications of chords or partial structures out of the Twelve Bar Blues.

The remainder of the paper is structured as follows. Section 2 introduces the fundamentals of the Twelve Bar Blues structure and the formalities of ASP. Section 3 presents our system proposal and some ASP encodings of particular subtasks as part of the analysis, and lastly, Section 4 directs future work and concludes the paper.

2 Background

In this section, we formally provide a brief account of what the Twelve Bar Blues structure is, including progressions, chords, and structure. Then, we introduce in a high-level the foundations of Answer Set Programming, including actual syntax from the *state-of-the-art* and *award-winner* ASP system *clingo* [3].

2.1 The Twelve Bar Blues Structure

The Twelve Bar Blues is a progression of of chords consisting of four bars of the tonic (I), two bars of the subdominant (IV), two bars of the tonic (I), a bar of the dominant seventh (V7), a bar of the subdominant (IV), and two final bars of the tonic (I). Following this structure it is understandable for a musician to follow the blues on a given key, for instance "a blues in F" [17].

This structure follows a "call-and-response" succession in which a lead voice or instrument states a phrase that is answered by other voices or players. An example of the most common Twelve Bar song pattern, the leader sings two repeated lines, each of which is answered by an instrumental phrase, then a rhyming third line that is answered by a final instrumental passage [17]. This pattern is the most common blues form, but it does not fully define the style. Other common characteristics of this structure are that almost invariably, the songs will be in 4/4 time, the chords structure consist of three basic chords, built on the first, fourth and fifth notes of the scale. In the scale of C, the chords will be C, F, and G7 [18]. These aspects also impact the lyrics.

For the pass of the years, this progression has developed certain variations to make music more interesting. Examples of these involve a subdominant in the second bar, and the 10th and the 12th bar could be dominant, to mention a few.

Figure 1 shows an example of a score representing this progression. We can see that it follows the rules mentioned above, starting with four bars of the tonic. In this case, this blues is in C, and the rest is calculated accordingly.

2.2 Answer Set Programming

Answer Set Programming (ASP; [1]), is many things in one. On the one hand, it is a knowledge-based Artificial Intelligence (AI) oriented to solve high-combinatorial optimization problems. On the other hand (and more properly), ASP is a rule-based

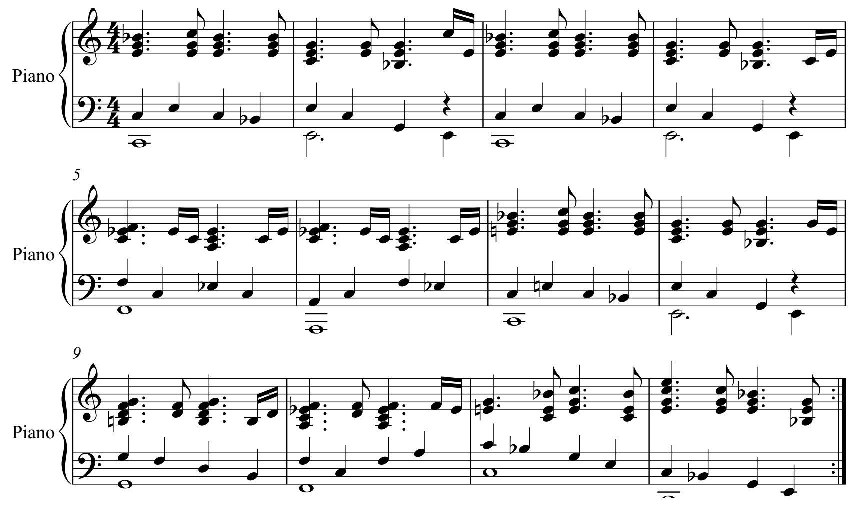

Fig. 1: Example of a score representing a Twelve Bar Blues structure

formalism for modeling and solving knowledge-intense combinatorial (optimization) problems as part of the area of knowledge representation and reasoning (KRR) [4]. [5]

Besides the success in both academic and industry in areas like planning, scheduling, configuration design, biology, logistics, and even music [2], what makes ASP attractive is its combination of a declarative modeling language with highly effective solving engines. This allows you to concentrate on specifying—rather than programming the algorithm for solving—a problem at hand. In resume, just define the problem, the solution path is found autonomously [7].

Formally, an ASP program consists of rules of the form:

a_1`;...;`a_m `:-` a_{m+1}`,...,`a_n`, not` a_{n+1}`,..., not` a_o`.`

where each a_i is an atom of form `p(`t_1`,...,`t_k`)` and all t_i are terms, composed of function symbols and variables. If an expression contains no variables, it is said to be ground and `not` denotes (default) negation. A rule is called a fact if $m = o = 1$, normal if $m = 1$, and an integrity constraint if $m = 0$. An *answer set* (or an *interpretation*) is a set of atoms that satisfies the ASP program in question.

An ASP solver takes information as input and starts *propagating* what it knows towards deriving new information. A very basic example program is defined below in Listing 1.1. It consists of four normal rules and one fact, and we read them from left to right. [6] [7]

For instance, we can read that `b` becomes true if `a` is proven to be true. Similarly, `c` becomes true if we have evidence that both `a` and `b` are true and no proof that `f` is

[5] For a non-technical reading about ASP, we refer to [8]

[6] For more information about ASP syntax, particularly for *clingo*, we refer to [4,5,6]

[7] For more details of *clasp*'s trophies and tracks, see `http://potassco.sourceforge.net/trophy.html.`

true. In other words, `f` must be false. [8] The following two rules (lines 3 and 4) yield no information because to prove `e` we need `d` to be true, and the same happens when trying to prove `d`. Lastly, the line 5 has a choice rule in the head. This rule is an example of the benefit of the non-determinism capabilities of ASP, by choosing between subsets of atoms, where any subset of its head atoms can be included in a stable model if the body holds (in this case `c`). For this example, this unrestricted (unbounded) choice rule selects between the subsets { } or {`g`} as part of the answer set.

If `a` is given as a fact (or assigned to true as shown in line 6) we have two answer sets, being {`a,b,c`} and {`a,b,c,g`}. Another example could be if both `a` and `f` are given as facts, the only answer set is {`a,b,f`}. The atom `c` cannot be inferred because of the existence of `f`, thus, the rule in line 5 will be automatically discarded.

```
1  b :- a.
2  c :- b, a, not f.
3  e :- d.
4  d :- e.
5  {g} :- c.
6  a.
```

Listing 1.1: A basic ASP encoding.

For an easy grasp to *clingo*, its syntax and simple examples, please go to: `http://flavioeverardo.com/clingo/` or `http://potassco.org/`

3 Towards an ASP System with Twelve Bar Blues Knowledge

Now that we have the Twelve Bar Blues and ASP foundations, our approach is represented in Figure 2.

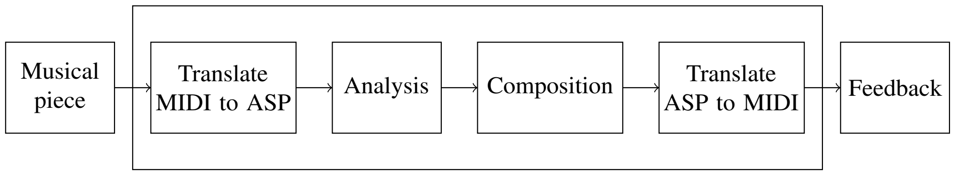

Fig. 2: System architecture interpreting a given score for ASP processing

For this first stage of our system, we cover only the first two blocks from the proposed architecture, and part of the feedback output. The composition block as well as the translation back to MIDI are left for future work. Being more specific, our approach starts with the input of a given (incomplete) composition in MIDI format [21] and translates it to ASP facts. Then, we proceed to analyze the piece by means of ASP

[8] ASP uses logical operators such as AND ($\wedge$), interpreted by a comma, IF ($\leftarrow$), interpreted by a colon and hyphen, and negation (as failure) written as *not* or tilde ($\sim$). In a formal representation, we would rewrite `b :- a.` as $b \leftarrow a$, and `c :- b, a,not f.` as $c \leftarrow b \wedge a \wedge not\, f$

encodings. Currently, we have implemented three ASP encodings being the detection of the structure, the tonality detection for calculating the composition mode, and chords detection. All the encodings have additional rules that contribute to user feedback. That is, some rules catch errors or suggestions to return to the user.

The translation to ASP facts (also known as instance in the ASP vocabulary) consist in capturing the continuity of MIDI events over time. [9] MIDI stands for Musical Instrument Digital Interface and it is a technical standard that describes a communications protocol, digital interface, and electrical connectors to communicate a wide variety of electronic musical instruments, computers, and music production audio devices. MIDI files are very common in the music production process when handling music. It has been a standard file to use it in music production software such as Digital Audio Workstations (DAW) or score writer. [10] The MIDI files represent all the events and timings from a composition, and each event is consecutively stored within the file. For instance, Listing 1.2 shows an excerpt of a MIDI file, particularly for the *note_on* and *note_off* events. [11]

```
note_on channel=0 note=48 velocity=81 time=0
note_on channel=0 note=36 velocity=81 time=0
note_off channel=0 note=48 velocity=0 time=960
note_on channel=0 note=52 velocity=78 time=0
note_off channel=0 note=64 velocity=0 time=589
note_off channel=0 note=67 velocity=0 time=0
```

Listing 1.2: Example of the events inside a MIDI file

An example of the output of this process, is shown in Listing 1.3, where these facts represent the blues shown in Figure 1.

```
1 note(1,48,81,(1,4),(0,1)). note(1,64,81,(3,8),(0,1)).
2 note(1,67,81,(3,8),(0,1)). note(1,70,81,(3,8),(0,1)).
3 note(1,36,81,(1,1),(0,1)).
4 note(2,52,78,(1,4),(1,4)).
5 note(3,64,78,(1,8),(1,8)). note(3,67,78,(1,8),(1,8)).
6 note(3,72,78,(1,8),(1,8)).
```

Listing 1.3: Extract of the ASP facts capturing the first three events from the first bar shown in Figure 1

We can read this instance from the atom `note` of arity 5, where the first parameter is number of a *note_on* event. The second parameter is the MIDI note number, followed by the velocity. The last two correspond to the duration of the event, and the distance from the last played note, respectively. For instance, from lines 1-3 we see the first chord, being its lowest note the C_2. Line 4 is the second event, which lasts one quarter, and it is represented by its fraction of 1/4.

Then, as mentioned in Section 2.1, there are several tasks or checks that the system needs to perform to detect a valid composition in the Twelve Bar Blues structure, and as

[9] For this process, we use ASPI, an open translator from MIDI to ASP and back developed by Leo Pinetzki. Available at `https://github.com/pinetzki/ASPI`

[10] Examples of DAW and music notation software are Ableton Live and MuseScore, respectively

[11] The *note_on* event is triggered as soon as a key is pressed, and the *note_off* event sends the signal to stop the sound as soon as the key is not pressed anymore.

mentioned above (for the scope of this work), the first one is the calculation of the structure as shown in Listing 1.4. To calculate the structure we prefer to perform calculations over integer numbers instead of reals or fractions, so we convert each distance from the atom `note` by applying the rule:
`duration(E,F):- figure(F,DST), note(E,N,V,DR,DST),event(E).`
This rule means that for each event `E`, we take the distance and map it to an integer number. For example, the atom `figure(64,(1,1)).`, tells that a whole note equals to 64, and `figure(16,(1,4)).`, represents a quarter note.[12]

Now, we proceed to the bars detection encoding (Listing 1.4) which consists of four lines.

```
1  selectedMeasure(1,E,E,F) :- duration(E,F), event(E), E=0.
2  selectedMeasure(M,S,E,R) :- duration(E,F1),
       selectedMeasure(M,S,E-1,F2), event(E), E>0, R=(F1+F2),
       R<=TL, timeLimit(TL).
3  measure(M,S,F,F-S)  :- selectedMeasure(M,S,F,R), timeLimit
       (TL), R\TL==0, F!=0.
4  selectedMeasure(M+1,E,E,F1) :- duration(E,F1),
       selectedMeasure(M,S,E-1,_), event(E), E=F+1, measure(M,
       S,F,F-S).
```

Listing 1.4: Encoding to detect the structure of the composition

The system first analyses the piece and perform a set of tasks being the first one, the calculation of the structure. As described in Section 2.1, the music style in question is built from 12 bars and repetitions, so the first step is to have this musical structure identified. The first line handles the first event of the input. In other words, the first event is directly assigned to the first bar. Then, line 2 captures the rest of the events from the first bar. The constraint in this rule is that the durations of the events in the bar must not exceed the limit where in a 4/4 case, is 64. Benefiting from the recursion power from the ASP system *clingo*, lines 3 and 4 captures the rest of the input from the deductions from line 2. The atom `measure/4` captures the essential information after this analysis. For example, the only answer set for this instance has the atoms `measure(1,1,6,5)` `measure(2,7,13,6)` `measure(3,14,19,5)` capturing that the first measure or bar comprises the events from 1 to 6. The events 7 to 13 belongs to the second bar and the third one has the events 14 to 19. The last argument is the number of events per bar. [13]

```
selectedMeasure(1,1,1,16) selectedMeasure(1,1,2,24)
selectedMeasure(1,1,3,32) selectedMeasure(1,1,4,48)
selectedMeasure(1,1,5,56) selectedMeasure(1,1,6,64)

selectedMeasure(2,7,7,16) selectedMeasure(2,7,8,24)
selectedMeasure(2,7,9,32) selectedMeasure(2,7,10,48)
selectedMeasure(2,7,11,56) selectedMeasure(2,7,12,60)
selectedMeasure(2,7,13,64)

selectedMeasure(3,14,14,16) selectedMeasure(3,14,15,24)
```

[12] This fraction-integer convention was proposed in [10]
[13] The encoding 1.4 is deterministic. This is why there is only one answer set per instance.

```
selectedMeasure(3,14,16,32) selectedMeasure(3,14,17,48)
selectedMeasure(3,14,18,56) selectedMeasure(3,14,19,64)
```

Listing 1.5: Answer Set capturing the first three bars from the input composition shown in Figure 1

Listing 1.5 shows an extract of the answer set where the first three bars from the input composition are detected. We can see that the first bar is represented in lines 1-3, followed by the second and third in lines 5-8, and 10-12, respectively. The structure of the atom `selectedMeasure/4` shows the bar number as the first argument. The second argument is the first MIDI event of that particular bar, followed by the consecutive enumeration of events. The last argument keeps track of the note duration.

Our second encoding is shown in Listing 1.6. To illustrate the purpose of our encoding, we show only the detection of three chord types, however, we have rules to identify up to 13 chords including, major, minor, augmented, half-diminished, dominant-seventh, maj7, diminished, and dominant-seventh 9th. This encoding is straightforward where each rule captures a chord type. For example, the first rule states that a chord in a singular event `E` is major if it has a note `N` and its respective pitches `N+4, N+7`, and there is no evidence of `N+10` in the same event. This default negation is necessary to avoid the calculation of the major chord as a subset from a dominant-seventh. With this encoding, we can identify that the chords shown in the first and third events from Listing 1.3, are dominant-seventh, and major respectively.

```
chordType(E,major) :- note(E,1,N), note(E,2,N+4), note(E
    ,3,N+7), not note(E,4,N+10).
chordType(E,minor) :- note(E,1,N), note(E,2,N+3), note(E
    ,3,N+7), not note(E,4,N+10).
chordType(E,augmented) :- note(E,1,N), note(E,2,N+4), note
    (E,3,N+8).
```

Listing 1.6: Extract of the chords detection encoding

Another key feature from the analysis is the identification of the composition mode or tonality. Once we capture the structure and chords, we can easily identify all the notes used in the composition. For example, consider the notes $C\#, A, B, G\#, E, F\#$ and $D\#$ represented by the atoms `note(cs). note(a). note(b). note(gs). note(e). note(fs).` and `note(ds).`, respectively. Now, we can apply the encoding in Listing 1.7.

This encoding starts by counting the number of notes in the input. For our example, we have seven. Then, we have our knowledge base for all the major and minor scales as shown in the atoms `scale/3` from lines 5-31. To calculate which scale the given notes belong to, we need to perform some steps. The first step is a choice rule that computes subsets of notes that matches the number of input notes (line 34). Then, in line 37, we do not allow guesses without the input notes. Lines 30 and 43, avoid crosses or overlaps between fundamentals and modes from the knowledge base, and lastly, we get the used scale in line 46. Running the example with the seven notes listed above, there are two answer sets $\{$`scale(e,major)`$\}$ and $\{$`scale(cs,minor)`$\}$. This tell us that the given composition is in $E\ major$ or in its relative minor key $C\#\ minor$.

One of the main advantages of this encoding is that it also works on incomplete data. For example, if we remove $D\#$ or `note(ds).` from the input notes, we get four answer sets {`scale(e,major)`}, {`scale(cs,minor)`}, {`scale(a,major)`} and {`scale(fs,minor)`}. In other words, it proposes all the different composition modes on which a subset of notes belong.

```
count(C) :- C = { note(N) }.

%% Major scales
%                W, W, H, W, W, W, H
scale(c ,major,(c ;d ;e ;f ;g ;a ;b)).
scale(cs,major,(cs;ds;f ;fs;gs;as;c)).
scale(d ,major,(d ;e ;fs;g ;a ;b ;cs)).
scale(ds,major,(ds;f ;g ;gs;as;c ;d)).
scale(e ,major,(e ;fs;gs;a ;b ;cs;ds)).
scale(f ,major,(f ;g ;a ;as;c ;d ;e)).
scale(fs,major,(fs;gs;as;b ;cs;ds;f)).
scale(g ,major,(g ;a ;b ;c ;d ;e ;fs)).
scale(gs,major,(gs;as;c ;cs;ds;f ;g)).
scale(a ,major,(a ;b ;cs;d ;e ;fs;gs)).
scale(as,major,(as;c ;d ;ds;f ;g ;a)).
scale(b ,major,(b ;cs;ds;e ;fs;gs;as)).

%% Minor scales
%                W, H, W, W, H, W, W
scale(a ,minor,(a ;b ;c ;d ;e ;f ;g)).
scale(as,minor,(as;c ;cs;ds;f ;fs;gs)).
scale(b ,minor,(b ;cs;d ;e ;fs;g ;a)).
scale(c ,minor,(c ;d ;ds;f ;g ;gs;as)).
scale(cs,minor,(cs;ds;e ;fs;gs;a ;b)).
scale(d ,minor,(d ;e ;f ;g ;a ;as;c)).
scale(ds,minor,(ds;f ;fs;gs;as;b ;cs)).
scale(e ,minor,(e ;fs;g ;a ;b ;c ;d)).
scale(f ,minor,(f ;g ;gs;as;c ;cs;ds)).
scale(fs,minor,(fs;gs;a ;b ;cs;d ;e)).
scale(g ,minor,(g ;a ;as;c ;d ;ds;f)).
scale(gs,minor,(gs;as;b ;cs;ds;e ;fs)).

%% Calculate all subsets of size C for each scale
C { guess(F,M,N) : scale(F,M,N) } C :- count(C).

%% Discard guesses not having the given notes
:- not guess(_,_,N), note(N).

%% Not allowed to have two guesses with different
    fundamentals and different notes
:- guess(F1,_,N1), guess(F2,_,N2), F1 != F2, N1 != N2.

%% Discard the subsets pointing to opposite modes
:- scale(F,M1), scale(F,M2), M1!=M2.

%% Get the scale
```

```
scale(F,M) :- guess(F,M,_).
```

Listing 1.7: Mode detection encoding

Our last encoding is related to the previous ones, as it characterizes rules to give feedback to the user (Listing 1.8). The encoding catches three types of errors, such as structure, chord, and scale errors. The structure error determines if the given composition does not have 12 bars. The chord error states that a chord is not recognizable for some event E, and the scale error is derived when there is not a scale that comprises all the input notes.

```
#const err_str = "Structure error. There are no 12 bars.".
error(err_str) :- not measure(12,_,_,_).

#const err_chord = "Not a valid chord at this event".
error(err_chord,E) :- not chordType(E,_), event(E).

#const err_scl = "Not a valid scale".
error(err_scl) :- not guess(F,M,_), scale(F,M).
```

Listing 1.8: Examples of feedback messages to the user

4 Discussion

In this work we present an initial phase to build an ASP system capable of reasoning about basic features of the Twelve Bar Blues structure. This system has the intention to support the students to train their audition and composition skills at the same time they comprehend musical structures by benefiting from feedback. Currently, we have ASP encodings to detect the structure of a given score in MIDI format, detect chords for progression, and the scaled used in the composition. We have several challenges before we complete this work. For the musical efforts, we aim to detect variations of the mode from the blues structure, as mentioned in Section 2.1.

In the case of ASP, one of the main aspects to consider is proper performance evaluation over different size compositions to test our encodings. Another upcoming work is to build the composer that can suggest a score completion or alternative compositions associated with the input. To do so, we are studying the different systems of music composition with ASP. Lastly, as the work progresses, we need to enrich our ASP-system with more feedback options.

References

1. Lifschitz, V.: Answer set planning. In International Conference on Logic Programming and Nonmonotonic Reasoning. pp. 373–374. Springer, Berlin, Heidelberg (1999).
2. Erdem, E., Gelfond, M., and Leone, N.: Applications of ASP. In AI Magazine. 37(3) (2016).
3. Gebser M., Kaminski R., Kaufmann B., and Schaub T.: Clingo = ASP + control: Preliminary report. In Technical Communications of the Thirtieth International Conference on Logic Programming (ICLP'14), (2014). Available at http://arxiv.org/abs/1405.3694.
4. Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T.: Answer set solving in practice. Synthesis lectures on artificial intelligence and machine learning, 6(3), 1-238 (2012).

5. Gebser, M., Kaminski, R., Kaufmann, B., Lindauer, M., Ostrowski, M., Romero, J., Schaub, T., Thiele, S., and Wanko, P.: Potassco User Guide. 2 edn. (2019), `http://potassco.org`
6. Answer Set Programming at Wikipedia: `https://en.wikipedia.org/wiki/Answer_set_programming`
7. Schaub, T. Interview at University of Potsdam: The Universal Problem Solver – "AI Made in Potsdam" on its Triumphal Procession around the World Short URL: `shorturl.at/kvLTX`
8. Schaub, T., and Woltran, S.: Answer set programming unleashed!. KI-Künstliche Intelligenz, 32(2-3), 105-108, (2018).
9. Boenn, G., Brain, M., De Vos, M., and Ffitch, J.: Automatic Composition of Melodic and Harmonic Music by Answer Set Programming. In *International Conference on Logic Programming*, ICLP08. Lecture Notes in Computer Science, vol. 4386. Springer Berlin / Heidelberg, 160–174 (2008).
10. Everardo, F., and Aguilera, A.: Armin: Automatic trance music composition using answer set programming. In *Fundamenta Informaticae*, vol. 113, no. 1, pp. 79–96, (2011).
11. Everardo, F.: A logical approach for melodic variations. In Latin American New Methods of Reasoning, pp. 141–150 (2011).
12. Opolka, S., Obermeier, P., and Schaub, T.: Automatic genre-dependent composition using answer set programming. Proceedings of the 21st International Symposium on Electronic Art. Vancouver, Canada (2015).
13. Eppe, M., Confalonieri, R., Maclean, E., Kaliakatsos, M., Cambouropoulos, E., Schorlemmer, M., Codescu, M., and Kühnberger, K. U.: Computational invention of cadences and chord progressions by conceptual chord-blending. In Twenty-Fourth International Joint Conference on Artificial Intelligence (2015).
14. Toivanen., J. M.: Methods and models in linguistic and musical computational creativity (2016).
15. Cabalar, P., and Martín, R.: Haspie-A Musical Harmonisation Tool Based on ASP. In EPIA Conference on Artificial Intelligence (pp. 637-642). Springer, Cham (2017).
16. Everardo, F.: Towards an Automated Multitrack Mixing Tool using Answer Set Programming. In Lokki T., Pätynen J., and Välimäki V. (eds.) Proceedings of the 14th Sound and Music Computing Conference. pp. 422–428 (2017).
17. Wald, E.: The blues: a very short introduction. Oxford University Press (2010).
18. Weissman, D.: Blues: the basics. Routledge (2004).
19. De Man, B., Stables, R., & Reiss, J. D.: Intelligent Music Production. Routledge (2019).
20. Everardo, F.: Over a Decade of Producing Music with Answer Set Programming. A Survey. Submitted to the Workshop on Trends and Applications of Answer Set Programming (TAASP 2020).
21. Moog, R. A.: Midi: Musical instrument digital interface. Journal of the Audio Engineering Society, 34(5), 394-404, (1986).