VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HUDEBNÍ NOTACE PRO KLAVÍR JAKO FORMÁLNÍ JAZYK

MUSIC NOTATION FOR PIANO AS A FORMAL LANGUAGE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB KLOUB**
AUTHOR

VEDOUCÍ PRÁCE **prof. RNDr. ALEXANDR MEDUNA, CSc.**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

161809

Ústav: Ústav informačních systémů (UIFS)
Student: **Kloub Jakub**
Program: Informační technologie
Název: **Hudební notace pro klavír jako formální jazyk**
Kategorie: Teoretická informatika
Akademický rok: 2024/25

Zadání:

1. Dle instrukcí vedoucího se seznamte s hudební notací pro klavír, formálními jazyky a různými formálními modely, např. modely gramatik a automatů.
2. Zaveďte nové verze těchto modelů vhodné pro definici této hudební notace.
3. Studujte vlastnosti těchto modelů a jejich jazyků dle instrukcí vedoucího.
4. Aplikujte modely z bodu 2 v hudbě, např. pro generování vybraných hudebních pasáží.
5. Implementujte aplikace navržené v bodě 4. Zhodnoťte takto vzniklou implementaci na bázi velmi pečlivé konzultace s vedoucím o způsobu tohoto vyhodnocení.
6. Zhodnoťte dosažené výsledky. Diskutujte další vývoj projektu.

Literatura:

- Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, Volume 1-3, Springer, 1997, ISBN 3-540-60649-1.
- Schulze, W. A Formal Language Theory Approach To Music Generation. Matieland 7602, South Africa, 2009. University of Stellenbosch. Vedouci práce Merwe, A. van der.
- Krakowski, S. Rhythmically-Controlled Automata Applied to Musical Improvisation. Rio de Janeiro, 2009. Phd thesis. IMPA. Music.
- Zeyu, Y. Formal semantics for music notation control flow. Carnegie Mellon University, Pittsburg, 2013.
- Jurish, B. Music as a formal language. Universität Potsdam, 2004, Phd thesis.
- Roads, C., Wieneke, P. Grammars as Representations for Music. *Computer Music Journal* 3, no. 1 (1979): 48–55. DOI: 10.2307/3679756.
- Zuidema, W. et al. Formal models of Structure Building in Music, Language and Animal Songs. DOI: 10.48550/arXiv.1901.05180.

Při obhajobě semestrální části projektu je požadováno:
Body 1, 2, 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Meduna Alexandr, prof. RNDr., CSc.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2024
Termín pro odevzdání: 14.5.2025
Datum schválení: 22.10.2024

Abstrakt

Tato bakalářská práce se zabývá generováním hudební notace pomocí L-systémů. Cílem práce je analýza struktury hudební notace společně s návrhem formálního modelu, který by umožňoval její generování pomocí L-systémů. V rámci práce je navržen stochastický kontextově závislý L-systém, jehož výstup je překládán na základě uživatelských parametrů do hudebních not. Teoretický model je následně implementován v programovacím jazyce Rust, přičemž výsledná aplikace zahrnuje nástroje pro definici gramatiky, převod do obrázkového formátu nástrojem LilyPond a přehrání s pomocí FluidSynth. Součástí řešení je i grafické rozhraní, které umožňuje interaktivní úpravu vstupních parametrů a podrobností L-systému a zobrazuje výsledné noty. Aplikace nachází využití v oblasti algoritmické kompozice a slouží jako výchozí bod pro experimenty s formálními modely v hudbě.

Abstract

This bachelor's thesis focuses on the generation of musical notation using L-systems. The aim of the thesis is to analyze the structure of music notation and to design a formal model that enables its generation through L-systems. A stochastic context-sensitive L-system is proposed, whose output is translated into musical notation based on user-defined parameters. The theoretical model is implemented in the Rust programming language. The resulting application includes tools for grammar definition, conversion to graphical notation using LilyPond, and audio playback via FluidSynth. The solution also provides a graphical appliaction that allows for interactive modification interpret parameters and L-system details, as well as visualization of the resulting musical notation. The application is intended for use in algorithmic composition and serves as a starting point for further experimentation with formal models in music.

Klíčová slova

Formmální jayzk, Hudební notace, Klavír, L-Systém, Generace, GUI

Keywords

Formal language, Music notation, Piano, L-System, Generation, GUI

Citace

KLOUB, Jakub. *Hudební notace pro klavír jako formální jazyk*. Brno, 2025. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. RNDr. Alexandr Meduna, CSc.

Hudební notace pro klavír jako formální jazyk

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. RNDr. Alexandra Meduny CSc. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jakub Kloub
13. května 2025

Poděkování

Rád bych poděkoval vedoucímu své bakalářské práce, prof. RNDr. Alexandru Medunovi, CSc., za jeho odborné vedení, cenné rady a trpělivost během zpracování této práce.

Dále bych chtěl poděkovat své rodině za jejich stálou podporu a povzbuzení po celou dobu studia. Mé díky patří také mým přátelům, kteří mi poskytli druhý úhel pohledu, vedli se mnou podnětné diskuze a významně přispěli k rozvoji mých myšlenek i celé práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Hudba a její notace je rozsáhlá problematika, kterou se lidé zabývají již mnoho let. Je to jedním z důvodů, proč se vyskytuje mnoho způsobů, jak hudbu zapisovat. Dnes lze již s relativní jistotou říct, že hudební notace (nejenom) pro klavír je mezinárodně ustanovená. Existují sice různé varianty značení, mezi které patří například notace pro značení repeticí, ale ve výsledku lze usoudit, že zde se jedná o méně časté případy.

Jednotná struktura otevírá dveře k problematice formální reprezentace not. Když lze hudební notaci formálně reprezentovat, tak by mělo být možné ji i generovat. Vytváření hudby je umělecký proces, což znamená, že její generace může být náročná. Existují k tomuto účelu různé modely a způsoby, kde každý z nich se snaží zachytit jiné aspekty hudební skladby.

Formální reprezentace hudební notace je užitečná nejen pro hudební teoretiky, ale i pro vývojáře softwaru. Zde se jedná například o vytváření nástrojů pro skladatele, výuku hudby a algoritmické kompozice. Generování skladeb může zpřístupnit vytváření hudby i širšímu publiku, nebo pro vytvoření částí děl, které autor z různých důvodů nedokončil. Dále může mít význam v oboru umělé inteligence [4].

Oblast generativní hudby a její formální modelování má dlouhou historii. Dokonce již Mozart, Haydn a C.P.E Bach měli zájem o generativní hudbu. Mozart vytvořil skladbu *Musikalisches Würfelspiel*, která zahrnovala několik částí, které se za sebe skládaly podle hodu kostkou [14]. První počítačově generovanou kompozici vytvořili *Lerajen Hiller* a *Leonard Isaacson* v roce 1955-56 na univerzitě *Illinois* [7]. Dnes jsou již k dispozici pokročilejší technologie, jako jsou neuronové sítě a algoritmy strojového učení, které analyzují i generují komplexní hudební struktury.

Výběr tohoto tématu podpořil osobní zájem o hudbu a její propojení s informatikou. Studovaná oblast je zajímavá zejména svým poměrně jednoduchým přístupem k docela komplexnímu procesu, jako je vytváření hudby. Nabyté znalosti o formálních modelech a jejich interpretaci pro vytváření hudební notace lze jistě využít i v budoucnu. Navíc je motivující možnost vytvářet nástroje, které umožní interaktivní úpravu pravidel, zobrazení a přehrání generovaných not.

Cílem této práce je navrhnout a implementovat vhodnou metodu pro generování hudebních klavírních not. Tento přístup by měl umožnit vytvářet hudební noty na základě formálně definovaných pravidel, která lze snadno upravovat a rozšiřovat. Výsledkem by měl být nástroj, který přispěje k pochopení konkrétní možnosti algoritmické kompozice.

Nejdříve se kapitola 2 zabývá základními pojmy, které jsou potřebné pro celkové pochopení této práce. Záměr je zde na strukturu hudební notace (kapitola 2.1), L-Systémy (kapitola 2.6), způsoby mapování L-Systémů na notaci (kapitola 2.7) a různými možnostmi

uložení hudby (kapitola 2.9). Dále v kapitole 3 navrhnu vhodný L-Systém, společně s jeho interpretací v sekci 3.4 pro vytvoření hudebních not. Nakonec v kapitole 4 popisuji způsob implementace navržené gramatiky a grafického prostředí společně se zhodnocením dosažených výsledků.

Kapitola 2

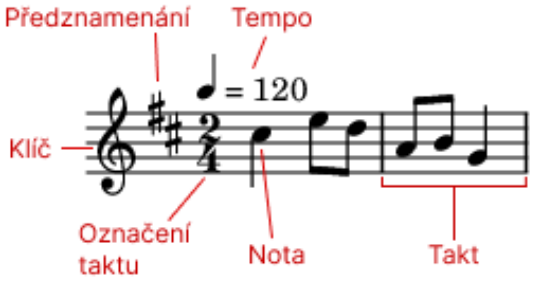
Základní koncepty a definice

Tato kapitola se zabývá pojmy a koncepty, které jsou využívány ve zbytku práce. Shrnuje a vysvětluje základní strukturu hudební notace a různé definice. Dále se krátce věnuje možnostem interpretace L-Systémů a nakonec i způsoby uložení hudby.

2.1 Hudební notace

Hudební notace je systém grafických symbolů, které slouží k zaznamenání hudby a umožňují její následnou interpretaci. Jejím cílem je poskytnout hudebníkům pokyny k tomu, jak mají zahrát dané hudební dílo. Tyto pokyny obsahují informace o výškách tónů, jejich délce, repetice atp.

Mezi základní prvky hudební notace patří *tón*, *notová osnova*, *předznamenání*, *takt*, *tempo* a *notový klíč*. Tyto prvky nám stačí pro vytvoření základních not. Mezi další užitečné prvky patří značení repeticí a variant. Těmto prvkům se však tato práce nevěnuje.



Obrázek 2.1: Základní prvky hudební notace

Tato sekce čerpá hlavně z [12] a [15].

Tón a nota

Tón je hudební zvuk, který má následující vlastnosti: **výšku**, **sílu**, **délku** a **barvu**. Grafickou reprezentací tónu je *nota*, kterou lze vidět na obrázku 2.1.

Výška tónu je určena pozicí noty (linka) v notové osnově. Dále může být ovlivněna předznamenáním, nebo křížkem/béčkem (posuvky) před notou.

Délka noty je hlavně určena vzhledem noty. Rozlišujeme noty celé, půlové, čtvrťové, osminové, šestnáctinové a dále dle potřeby. Na obrázku 2.1 lze vidět noty čtvrťové a osminové. Pokud se vedle noty vyskytuje tečka, tak se délka noty prodlužuje o polovinu.

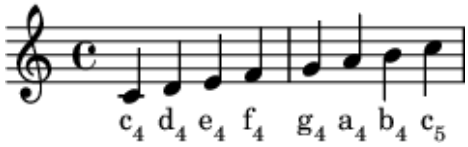
Síla tónu se značí pomocí značek dynamiky, mezi které patří například *Akcent*. Protože se zabýváme notací pro klavír, tak předpokládáme, že tón má barvu zvuku klavíru.

Notová osnova a pomocné linky

Notová osnova slouží k umístění not. Skládá se z pěti linek a čtyř mezer. Pokud je to potřeba, tak se nad nebo pod osnovu přidávají pomocné linky, kterých může být nejvíce 5. Přidává se jich vždy jenom tolik, kolik je potřeba pro daný tón.

Poloha notové hlavičky v osnově může být buď na lince, nebo mezi linkami (noty přímo nad/pod osnovou nevyžadují pomocné linky). Výška tónu je určena polohou hlavičky a notovým klíčem.

Každá nota má jiný název, který je mimo jiné dán notovou abecedou a klíčem. Notová abeceda obsahuje 7 základních not: C, D, E, F, G, A, H/B, které mají mezi sebou v tomto pořadí vzdálenost celého tónu s výjimkou not E-F a B-C, které mají vzdálenost půltónu.



Obrázek 2.2: Stupnice C dur v houslovém klíči. Na obrázku lze vidět jména not v závislosti na jejich pozici v osnově. Malé čísílko vedle noty značí oktávu.

Notový klíč

Notový klíč je značka na začátku osnovy (může se vyskytovat i vícekrát, ale na začátku musí být vždy), která přiřazuje pozicím v osnově dané výšky tónů. Výška tónu může být ještě ovlivněna předznamenáním.

Noty pro klavír typicky obsahují klíč *houslový* (F klíč) a *basový* (G klíč).

Křížky, béčka a předznamenání

Vzdálenost mezi sousedními notami v osnově může být celý tón, nebo půltón. Pokud se před notou nachází křížek, tak se výška tónu posune o půltón nahoru. Pokud se zde nachází béčko, tak se výška naopak posune o půltón dolů. Existují i další posuvky, jako je například dvojitý křížek, ale ty nejsou v kontextu této práce využity.

Křížky a béčka lze také zapsat na začátek osnovy za notový klíč. Potom to znamená, že pro všechny noty ve všech oktávách na dané pozici platí daná posuvka implicitně.

Stupnice a Tónina

Noty se řadí do takzvaných stupnic a tónin. Každá stupnice obsahuje jiný počet křížků a béček. To znamená, že každá stupnice má jiné předznamenání.

Stupnice je posloupnost tónů, které jsou uspořádány podle nějakého pravidla, které udává počet tónů v rozsahu oktávy a jaké jsou mezi nimi intervaly.

Existuje mnoho druhů stupnic. V kontextu této práce se zajímáme hlavně o *durové stupnice*. Jedná se o stupnici, která se skládá z řady 8 tónů, kde mezi 3.—4. a 7.—8. stupněm je půltón. Mezi ostatními je celý tón. Z tohoto pravidla se také odvíjí předznamenání pro každou stupnici. Pokud se začne na notě G, tak aby mezi 7. a 8. stupněm byl půltón, musí se přidat křížek před notu F.

Tónina je podobná stupnici, akorát zde jsou noty v neuspořádaném pořadí. Značí tedy pouze skupinu not.

Akord

Akord je seskupení nejméně tří not, které se v notách píšou "nad sebe". To značí, že všechny noty začínají ve stejnou dobu.

Takt a označení taktu

Noty v díle se seskupují do skupin zvaných *takt*. Každý takt je vyznačen taktovými čarami a trvá stejný počet dob. Na začátku osnovy se typicky nachází označení taktu, které udává, která nota má jednu dobu (spodní číslo) a kolik dob je v jednom taktu (horní číslo). Obrázek 2.1 obsahuje příklad takového označení.

2.2 Formální koncepty

Tato sekce je věnována vysvětlení některých formálních konceptů a modelů, které jsou dále využívány v této práci. Stanovuje vědomostní bázi, na které jsou postaveny další kapitoly.

2.3 Abeceda, řetězec a konkatenace

Základním konceptem v teorii formálních jazyků je *abeceda* a *řetězec*. Tyto dva pojmy znamenají skoro to stejné co v psaném jazyce, tedy abeceda představuje symboly reprezentující slova jazyka.

Definice 2.3.1 *Abeceda* je konečná, neprázdná množina elementů, které nazýváme *symboly*.

Příklad 2.3.1 Abeceda symbolů používaných v desítkové soustavě pro zápis čísel:

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

Definice 2.3.2 *Řetězec* Nechť Σ je abeceda. Potom:

1. ϵ je řetězec nad abecedou Σ
2. pokud je x řetězec nad Σ a $a \in \Sigma$, potom xa je řetězec nad abecedou Σ

Řetězce lze seskupovat dohromady, čímž lze vytvořit nové řetězce.

Definice 2.3.3 *Konkatenace (zřetězení) řetězců*
Nechť x a y jsou dva řetězce nad abecedou Σ. *Konkatenace* x a y je řetězec xy

Dále je zapotřebí definovat *prefix* a *sufix* řetězce, které jsou dále využity při uspořádání kontextových pravidel v sekci 2.6. Tyto pojmy jsou významově velice podobné předponě a příponě slova.

Definice 2.3.4 *Pefix řetězce*

Nechť x a y jsou dva řetězce nad abecedou Σ; x je prefixem y, pokud existuje řetězec z nad abecedou Σ, přičemž platí $xz = y$. Pokud $x \notin \{\varepsilon, y\}$, pak x je *vlastním prefixem* řetězce y.

Definice 2.3.5 *Sufix řetězce*

Nechť x a y jsou dva řetězce nad abecedou Σ; x je sufixem y, pokud existuje řetězec z nad abecedou Σ, přičemž platí $zx = y$. Pokud $x \notin \{\varepsilon, y\}$, pak x je *vlastním sufixem* řetězce y.

2.4 Markovovy řetězce

Jedním z přímých a jednoduchých způsobů, jak generovat sekvenci not, jsou *Markovovy řetězce*. Jedná se o formální model, který má tu hlavní vlastnost, že jeho následující stav je závislý pouze na stávajícím stavu.

Definice 2.4.1 *Markovův řetězec ve spojitém čase (DTMC)* je uspořádaná trojice $(S, s_0, \boldsymbol{P})$, kde

- S je konečná množina stavů
- $s_0 \in S$ je počáteční stav
- $\boldsymbol{P} : S \times S \to \langle 0, 1\rangle$ je pravděpodobnostní přechodová matice, kde platí: $\forall s \in S : \sum_{s' \in S} \boldsymbol{P}(s, s') = 1$.

[16]

Pokud je stav modelovaný notou, tak další stavy tvoří noty, které mohou stav následovat [9].

2.5 Binární relace

Jedním z užitečných typů množiny jsou *binární relace*. Představují možnost, jak reprezentovat existenci zvoleného vztahu mezi dvěma prvky. Pokud takový vztah existuje, tak se v této množině nachází uspořádaná dvojice s těmito prvky. Mezi typické příklady patří například relace „je menší“, také často značená symbolem $<$. [6]

Definice 2.5.1 *Binární relace* mezi množinami A_1, A_2, je libovolná množina R, přičemž $R \subseteq A_1 \times A_2$.

- Říkáme, že prvek $a_1 \in A_1$ je v relaci s prvem $a_2 \in A_2$, zapsáno jako:

$$(a_1, a_2) \in R, \text{ nebo také } a_1 R a_2$$

- Binární relace na množině A, je tedy libovolná množina uspořádaných dvojic, přičemž $R \subseteq A^2$

Vlastnosti

Binární relace může mít několik vlastností, které mohou mít vliv na její možnosti použití. Tyto vlastnosti se odvíjí od obsahu relace.

Definice 2.5.2 *Vlastnosti relace*

Nechť R je relace na množině A. Říkáme, že relace R je:

- **Reflexivní** na množině A, pokud $\forall a \in A : aRa$
- **Symetrická** na množině A, pokud $\forall a, b \in A : aRb \Rightarrow bRa$
- **Tranzitivní** na množině A, pokud $\forall a, b, c \in A : aRb \wedge bRc \Rightarrow aRc$
- **Antisymetrická** na množině A, pokud $\forall a, b \in A : aRb \wedge bRa \Rightarrow a = b$

Uzávěr

Pokud je potřeba, aby relace měla určité vlastnosti, můžeme vykonat tzv. *uzávěr*. Jedná se o způsob, jakým doplnit do relace všechny dvojice, které zde chybí pro splnění dané vlastnosti.

Definice 2.5.3 *Uzávěr binární relace*

Nechť R je binární relace na množině A.

- **Reflexivní uzávěr** R je relace $R \cup \{(x, x) \mid x \in A\}$
- **Symetrický uzávěr** R je relace $\overset{\leftrightarrow}{R} = \{(x, y) \mid (x, y) \in R \text{ nebo } (y, x) \in R\}$
- **Tranzitivní uzávěr** R je relace $R^+ = \bigcup_{i=1}^{\infty} T^i(R)$, kde T je funkce, která pro každou binární relaci S vrací relaci:

$$T(S) = S \cup \{(x, z) \mid \exists y : (x, y), (y, z) \in S\}$$

$T^i = \underbrace{T \circ ... \circ T}_{i}$, je i-krát iterovaná aplikace funkce T.

Typy relací

Binární relace se dále rozdělují na různé typy, dle jejich použití. Mezi nejčastější patří relace *ekvivalence*, která značí vztah mezi prvky, které jsou si rovny. Dále existují relace *uspořádání*, které díky jejich asymetrii značí uspořádání mezi dvěma prvky.

Definice 2.5.4 *Relace ekvivalence*

Nechť R je relace na množině A. Říkáme, že R je relace **ekvivalence** na A, pokud R je reflexivní, symetrická a tranzitivní na A.

Příklad 2.5.1 Relace $=$ na množině reálných čísel $\mathbb{R}$.

Definice 2.5.5 *Uspořádání*

Relace $R \subseteq A \times A$ je **částečné uspořádání** právě když R je reflexivní, antisymetrická a tranzitivní.

Příklad 2.5.2 Relace $\leq$ na množině reálných čísel $\mathbb{R}$: $1 \leq 2 \leq 3$, a zároveň $1 \leq 1$

Definice 2.5.6 *Uspořádaná množina* je dvojice $(M, \preceq)$, kde M je množina a $\preceq$ je (částečné) uspořádání na M.

Částečně uspořádané množiny mají mnoho využití a jsou klíčem pro definici pravého derivačního kroku v nadcházejících sekcích (viz 2.6.11). V těchto množinách lze najít prvky, které jsou minimální a maximální. Jedná se o prvky, které v celé množině existují pouze na jedné straně uspořádání.

Definice 2.5.7 *Minimální prvek*

Nechť $(M, \preceq)$ je uspořádaná množina. Prvek $x \in M$ je **minimalní** právě když:

$$\forall y \in M : y \preceq x \Rightarrow x \preceq y$$

Definice 2.5.8 *Maximální prvek*

Nechť $(M, \preceq)$ je uspořádaná množina. Prvek $x \in M$ je **maximální** právě když:

$$\forall y \in M : x \preceq y \Rightarrow y \preceq x$$

Další důležitý pojem je *pokrytí*, který značí relaci mezi dvěma prvky uspořádané množiny, které již nejsou v netriviálním tranzitivním vztahu. Příkladem by zde mohla být uspořádaná množina rodinných členů s relací „je rodič“, kde přímí potomci jsou pokryti svými rodiči.

Definice 2.5.9 *Pokrytí prvku*

Nechť $(M, \preceq)$ je uspořádaná množina. Říkáme, že prvek $x \in M$ **pokrývá** prvek $y \in M$ právě když:

$$x \neq y \wedge x \preceq y \wedge (\nexists z \in M : x \neq z \neq y \wedge x \preceq z \preceq y)$$

Hasseův diagram

Pro grafické znázornění uspořádaných množin slouží tzv. *Hasseův diagram*. Tento diagram zobrazuje všechny prvky v jednotlivých vrstvách, které jsou pokrývány předešlými vrstvami. Jedná se o užitečný přístup ke znázornění těchto množin, který je v této práci použit pro prezentaci uspořádaných kontextových pravidel.

Definice 2.5.10 *Hasseův diagram* konečné uspořádané množiny $(M, \preceq)$ je (jednoznačné) grafické znázornění, které vznikné následujícím postupem:

- Do první „vrstvy“ zakreslíme body odpovídající minimálním prvkům $(M, \preceq)$
- Pokud máme „vrstvu“ i, tak do „vrstvy“ $i+1$ (která je „nad“ vrstvou i), zakreslíme všechny nezakreslené prvky, které pokrývají pouze prvky „vrstev“ $\leq i$. Pokud prvek x „vrstvy“ $i+1$ pokrývá prvek y „vrstvy“ $\leq i$, spojíme x a y neorientovanou hranou.

2.6 L-systém

L-systémy jsou typem gramatiky, která produkuje řetězce z počátečního axiomu a sady pravidel. Poprvé byly představeny Lindenmayerem v roce 1968. [10]

Na rozdíl od Chomské gramatiky, která aplikuje pravidla sekvenčně jedna po druhé, jsou pravidla aplikována paralelně na všechny znaky ve slově. [10] [14].

L-Systémy byly původně navržené pro modelování topologie rostlin, ale později se začaly používat i pro více komplexní modely růstu. S použitím tzv. "želví grafiky"lze vizuálně znázornit různé druhy L-Systémů (sekce 2.8).

0L-Systém

Snad nejjednodušší druh L-systému je tzv. *0L-Systém*.

Definice 2.6.1 *0L-systém* je trojice $G = (V, \omega, P)$, kde

- V je abeceda systému
- $\omega \in V^+$ je neprázdný řetězec zvaný axiom
- $P : V \times V^*$ je množina všech pravidel tvaru $\alpha \to w$, kde $\alpha \in V$ je znak abecedy a $w \in V^*$ je řetězec, které nám říká, že všechny výskyty znaku α se mají nahradit řetězcem w

Pojem, který byl pro tuto práci definován, je *identitní uzávěr*. Jedná se o speciální typ uzávěru, který přidá do identitní pravidla $x \to x$ do množiny pravidel, pokud pro x neexistuje žádné pravidlo. Tento uzávěr umožňuje jednodušeji definovat derivační kroky a také znázorňuje ten fakt, že pokud pro znak v řetězci nelze zvolit pravidlo, tak se nijak nezmění.

Definice 2.6.2 *Identitní uzávěr 0L pravidel*

Nechť P je množina pravidel 0L systému. Identitní uzávěr je množina

$$P^I = P \cup \{x \to x \mid \nexists y : x \to y \in P\}$$

L-Systémy přepisují řetězce použitím pravidel na všechny znaky paralelně. Jedno takové přepsání se nazývá *Derivační krok*.

Definice 2.6.3 *Derivační krok 0L-Systému*

Nechť $L = (V, \omega, P)$ je 0L-Systém. Nechť $x_1, x_2, ..., x_n \in V, y_1, y_2, ..., y_n \in V^*$.
Nechť $p_i = x_i \to y_i \in P^I$, pro všechna $i = 1, 2, ..., n$
Říkáme, že $x_1x_2...x_n$ derivuje $y_1y_2...y_n$ za použití pravidel $p_1, p_2, ..., p_n$, zapsáno jako:

$$x_1x_2, ...x_n \Rightarrow y_1y_2...y_n[p_1, p_2, ..., p_n],$$

nebo zjednodušeně:

$$x_1x_2, ...x_n \Rightarrow y_1y_2...y_n$$

Vykonání více derivačních kroků se často nazývá tzv. *sekvence kroků*, pro kterou existuje zkrácený zápis, díky kterému lze zapsat i komplikovanou derivační cestu jednou relací. Toto je zejména užitečné, pokud je potřeba například zapsat, že z axiomu lze získat výsledný řetězec.

Definice 2.6.4 *Sekvence derivačních kroků 0L-Systému*

1. Nechť $u \in V^*$. L provede *nula* derivačních kroků z u do u; zapisujeme:

$$u \Rightarrow^0 u[\epsilon], \text{ nebo zjednodušeně } u \Rightarrow^0 u$$

2. Nechť $u_0, u_1, ..., u_n \in V^*, n \geq 1$ a $u_{i-1} \Rightarrow u_i[p_i], p_i \in P^I$, pro všechna $i = 1, ..., n$, což znamená:

$$u_0 \Rightarrow u_1[p_1] \Rightarrow u_2[p_2] \Rightarrow ... \Rightarrow u_n[p_n]$$

Pak, L provede n *derivačních kroků* z u_0 do u_n; zapisujeme:

$$u_0 \Rightarrow^n u_n[p_1...p_n], \text{ nebo zjednodušeně } u_0 \Rightarrow^n u_n$$

D0L-systém

Pokud pro každý znak abecedy $\alpha \in V$ existuje nejvýše jedno pravidlo tvaru $\alpha \to w$, tak se systém značí jako *Deterministický 0L-Systém*. Pokud pro znak α neexistuje žádné pravidlo, tak se implicitně předpokládá $\alpha \to \alpha$. [9]

Definice 2.6.5 *D0L-systém* je *0L-Systém*, pro který platí: pokud je $\alpha \to w \in P$, tak množina $P \setminus \{\alpha \to w\}$ neobsahuje žádné pravidlo se znakem α na levé straně.

S0L-systém

Další možné rozšíření L-Systémů je přidání pravděpodobností k jednotlivým pravidlům. V takovém případě se jedná o tzv. *Stochastických L-Systémech* (S0L-Systém).

Definice 2.6.6 *Stochastický L-Systém* je čtveřice $G = (V, \omega, P, p)$, kde (V, ω, P) je *0L-Systém* a $p : P \to \langle 0, 1\rangle$ je funkce, pro kterou platí:

$$\forall \alpha \in V : \sum_{\alpha \to w \in P} p(\alpha \to w) = 1$$

Věta 2.6.1 Pro každý Markovův řetězec existuje ekvivalentní S0L-systém. [9]

Stochastické L-systémy bývají velmi užitečné pro vyjádření neurčitosti v systému. Při generaci not tím lze přinést variaci pro různé části skladby.

Kontextově závislé L-Systémy

Další možnou variací L-systému jsou tzv. *Kontextově závislé L-Systémy* [1]. Tyto systémy jsou podobné 0L-systémům s tím rozdílem, že pravidla, která se aplikují na znak, jsou závislá na okolních znacích.

V 2L kontextových systémech jsou pravidla kontextová, ale mají omezenou délku kontextových informací. Levá strana pravidla zde může mít maximálně délku tří znaků (tedy až 2 „sousedy“).

Příklad 2.6.1 Pravidlo v kontextově závislém 2L-Systému:

$$x\alpha y \to w$$

Definice 2.6.7 *Kontextově závislý 2L-Systém (C2L-Systém)* je trojice (V, ω, P), kde

- V je abeceda systému
- $\omega \in V^+$ je počáteční řetězec zvaný axiom
- $P : V^* \times V^*$ je množina kontextových pravidel tvaru $\alpha \to w$, kde $\alpha, w \in V^*$ a zároveň platí $|a| \in < 1, 3 >$.

Derivační krok C2L-Systému je podobně definovaný jako u 0L-Systému (viz. 2.6.3) s tím rozdílem, že se zde nenahrazuje pouze jeden znak řetězce, ale až tři v závislosti na levé straně použitého pravidla.

Definice 2.6.8 *Identitní uzávěr C2L pravidel*

Nechť P je množina pravidel C2L-systému. Identitní uzávěr je množina

$$P^I = P \cup \{x \to x \mid x \in V \wedge \nexists y, z : yx \to z \in P\}$$

Obecný derivační krok kontextového L-systému se nijak zásadně neliší od jeho bezkontextové varianty. Jediným rozdílem je zde použití pravidel pouze v případě, že nahrazená část slova je stejná jako celá levá strana pravidla.

[1] z anglického názvu *Context-Sensitive L-System*

Definice 2.6.9 *Derivační krok C2L-systému*

Nechť $L = (V, \omega, P)$ je 0L-Systém a $x_1, x_2, ..., x_n \in V^*, y_1, y_2, ..., y_n \in V^*$.
Nechť $p_i = x_i \to y_i \in P^I$, pro všechna $i = 1, 2, ..., n$
Říkáme, že $x_1x_2...x_n$ derivuje $y_1y_2...y_n$ za použití pravidel $p_1, p_2, ..., p_n$, zapsáno jako:

$$x_1x_2, ...x_n \Rightarrow y_1y_2...y_n[p_1, p_2, ..., p_n],$$

nebo zjednodušeně:

$$x_1x_2, ...x_n \Rightarrow y_1y_2...y_n$$

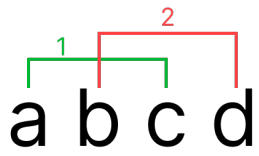
V kontextových L-systémech může existovat více derivačních kroků pro jeden řetězec. následující příklad ukazuje tuto situaci.

Příklad 2.6.2

Nechť existuje C2L-systém $L = (\{a, b, c, d\}, abcd, \{abc \to 1, bcd \to 2\})$. První derivační krok lze vykonat dvěma způsoby:

1. $abcd \Rightarrow 1d\ [abc \to 1]$
2. $abcd \Rightarrow a2\ [bcd \to 2]$

Důvodem zde je, že pravidla sdílejí kontext a není jasné, které se má použít. Obrázek 2.3 ilustruje tuto situaci na axiómu z příkladu 2.6.2.



Obrázek 2.3: Sdílený kontext pravidel 1

Pokud by se použilo pravidlo $bcd \to 2$, tak by se jednalo o tzv. *pravou derivaci*. Sdílejí-li dvě a více pravidel kontext, vždy se zvolí to pravější. Pokud je jedno z pravidel příponou jiného pravidla, je preferováno delší z nich. Toto se dělá z předpokladu, že delší pravidlo, kterému vyhovuje kontext, je více důležité, než pravidlo kratší, které může vyhovovat více kontextům.

Nejprve je tedy definována relace, která uspořádá dvě pravidla, která sdílejí kontext. Poté lze říci, že pokrývající pravidlo je preferováno nad pravidlem pokrytým. Z toho tedy vyplývá, že tato relace musí splňovat vlastnosti relace uspořádání.

Definice 2.6.10 *Relace sdíleného kontextu*

Nechť $L = (V, \omega, P)$ je C2L-Systém. Nechť $p_1 = a_1 \to b_1 \in P$ a $p_2 = a_2 \to b_2 \in P$.
Nechť $X = \{x \mid x \text{ je postfix } a_1\}$ a $Y = \{y \mid y \text{ je prefix } a_2\}$. Říkáme, že p_1 sdílí kontext s p_2, pokud platí jedno z následujících:

1. $X \cap Y \neq \emptyset$
2. a_1 je postfix a_2,

zapsáno jako:

$$p_1 \preceq p_2$$

Příklad 2.6.3 Pravidla $abc \to 1 \preceq bcd \to 2$, ale $bcd \to 2 \not\preceq abc \to 1$.

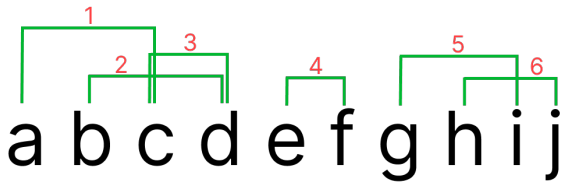
Tato relace pomáhá s výběrem pravidla v případě sdílených kontextů při derivačním kroku. Bohužel toto uspořádání stále není dostačující pro deterministický výběr pravidla, protože stále může existovat více pravidel pro stejný kontext. Tento problém řeší například stochastická pravidla navržena v sekci 2.6.

Příklad 2.6.4

Nechť $L = (\{a, b, c, d, e, f, g, h, i, j\}, abcdefghij, P)$ je C2L-systém, kde P:

1. $abc \to 1$
2. $bcd \to 2$
3. $cd \to 3$
4. $ef \to 4$
5. $ghi \to 5$
6. $hij \to 6$

Obrázek 2.4 ukazuje možné aplikace pravidel na axiom. Pokud by se vykonala první pravá derivace, výsledkem by byl řetězec: $abcdefghij \Rightarrow_R a24g6$

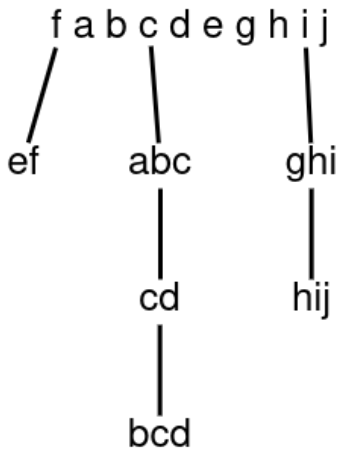


Obrázek 2.4: Sdílený kontext pravidel 2

Relace sdíleného kontextu na množině pravidel P vypadá následovně:

$$\preceq = \{(abc, bcd), (abc, cd), (bcd, cd), (ghi, hij)\}$$

Hasseův diagram (viz. 2.5.10) uspořádané množiny $U = (P^I, \preceq^+)$ vypadá takto:



Obrázek 2.5: Hasseův diagram uspořádané množiny pravidel. Na obrázku jsou zakresleny pouze levé strany pravidel, které nahrazují tečky Hasseova diagramu.

Hasseův diagram ilustruje maximální prvky uspořádané množiny U reprezentující pravidla, která je zapotřebí prioritně aplikovat v případě sdílených kontextů. Pravidla s druhou nejvyšší prioritou jsou právě ty, které je pokrývají atp. až k minimálním prvkům.

Jedním ze způsobů, jak docílit pravé derivace, je vykonání sekvence derivačních kroků, kde pro každý krok jsou omezená pravidla, která lze použít. První krok by používal pravidla, jenž odpovídají maximálním prvkům množiny U, v druhém pravidla, která je pokrývají atp. dokud se nepoužijí pravidla odpovídající minimálním prvkům U.

Definice 2.6.11 *Pravý derivační krok C2L-Systému*

Nechť $L = (V, \omega, P)$ je C2L-Systém. Nechť $U = (P^I, \preceq^+)$ je uspořádaná množina, kde $\preceq$ je relace sdíleného kontextu. *Pravý derivační krok* $x_0 \Rightarrow_R x_n; x_0, x_n \in V^+$ je roven sekvenci derivačních kroků

$$x_0 \Rightarrow x_1[P_1^I] \Rightarrow x_2[P_2^I] \Rightarrow ... \Rightarrow x_n[P_n^I],$$

kde $P_1 \subseteq P$ je množina všech maximálních prvků množiny U, množina $P_2 \subseteq P$ je množina všech prvků pokrývajících prvky z množiny P_1, ..., $P_n \subseteq P$ je množina všech minimálních prvků U.

Příklad 2.6.5 Pravá derivace pro axiom z příkladu 2.6.4:

$$\begin{aligned} abcdefghij &\Rightarrow a24g6[ef \rightarrow 4, bcd \rightarrow 2, hij \rightarrow 6, a \rightarrow a, ...] \\ &\Rightarrow a24g6[cd \rightarrow 3, ghi \rightarrow 5, a \rightarrow a, ...] \\ &\Rightarrow a24g6[abc \rightarrow 1, a \rightarrow a, ...] \\ &\Rightarrow a24g6[a \rightarrow a, ...] \end{aligned}$$

je tedy ekvivalentní:

$$abcdefghij \Rightarrow_R a24g6$$

Stochastické kontextově závislé L-Systémy

Tato práce využívá tzv. stochastických 2L-systémů (SC2L-systémů), které se liší přidáním funkce, která přiřazuje každému pravidlu jeho pravděpodobnost použití. Tento systém tedy nemusí být deterministický, aby bylo možné jej implementovat v praxi.

Příklad 2.6.6 Pravidlo ve stochastickém C2L-Systému:

$$x\alpha y \rightarrow_{1/4} w$$

Definice 2.6.12 *Stochastický C2L-Systém (SC2L-Systém)* je čtveřice (V, ω, P, p), kde (V, ω, P) je C2L-Systém a $p : P \rightarrow < 0, 1 >$ je funkce, pro kterou platí:

$$\forall a \in V : \sum_{xa \rightarrow w \in P} p(xa \rightarrow w) = 1$$

2.7 Mapování L-Systémů do hudební notace

Pro využití L-Systémů ke generování hudební notace, je zapotřebí nejdříve ustanovit abecedu systému. Každý symbol z abecedy se musí nějakým způsobem interpretovat. Zde existuje mnoho způsobů, ale většinou se řídí pravidlem, že velikost abecedy je relativně malá.

Protože L-Systémy mohou mít významné grafické reprezentace, tak se některé techniky inspirují grafickou reprezentací pro mapování na noty. Například [10] ukazuje způsob mapování Hilbertovy křivky na jednotlivé výšky tónů. Jiné přístupy zase využívají vzhled rostlin (viz. [14]), kde větvení může znamenat změnu tónu, času atp.

Výška tónu

Častým způsobem pro mapování výšky tónu bývá zvolení nějaké referenční výšky, která se následně zvyšuje, snižuje anebo nahrazuje pomocí různých symbolů v gramatice. [13] [14]

Také se často bere v potaz tonalita. Výsledná nota tedy patří do nějaké tóniny a změny výšky tónu posouvají notu v tónině nahoru nebo dolů.

Délka tónu

Přístup k interpretaci délky tónu je úzce spjatý s konkrétní gramatikou a jejími pravidly. Jedna z možných implementací je uchovávat si vnitřní stav aktuální délky jedné noty. Délka doby je následně měněna různými znaky z abecedy: půlení délky, ukládání stavu na zásobník atp.

Akordy

Jedna z možností, jak generovat akordy, je s využitím tzv. *akordních prostorů* [2] [5]. Tento přístup spočívá v tom, že existují tři notové osy, aktuální vektor směru a vektor pozice. Jednotlivé znaky jsou poté interpretovány jako změna směrového vektoru, jehož aplikace na pozici ovlivní výsledný akord.

pozn.: tento přístup není v této práci využit, ale jedná se o vhodné rozšíření do budoucna.

2.8 Model Želva

Při interpretaci L-Systémů se často mluví o tzv. „želvě“. V kontextu této práce je želva automat, jehož stav je šestice $S = (\boldsymbol{P}, \alpha, h, t)$, kde

- $\boldsymbol{P}$ je uspořádaná dvojice (x, y), která reprezentuje pozici ve 2D prostoru
- α je aktuální úhel natočení, který reprezentuje směr budoucího pohybu automatu.
- h je tloušťka čáry, která reprezentuje délku noty
- t je modelový čas

Želva přijímá několik příkazů, které řídí její akce a mění její stav [8]. Mezi základní příkazy patří:

[2] z anglického názvu *Chord spaces*

- *Dopředu* Želva se pohne vpřed a zanechává za sebou čáru
- *Vlevo* Želva se otočí o daný počet stupňů vlevo
- *Vpravo* Želva se otočí o daný počet stupňů vpravo
- *Dozadu* Želva se pohne vzad a zanechává za sebou čáru

[1]

2.9 Způsoby uložení

Dnes již existuje několik způsobů, jak ukládat hudbu. Liší se od sebe hlavně v záměru jejich použití. *Musical XML* a *LilyPond* se zaměřují na ukládání notace a dovolují programům ukládat díla v textové podobě. Na druhou stranu formát *MIDI* se zaměřuje na uložení hudební interpretace díla.

MusicXML

Protože strukturu hudebních not si lze představit jako stromovou strukturu, tak je přirozené jí reprezentovat jazykem, který má také stromovou strukturu. *MusicXML* [3] je XML formát, který je schopný ukládat většinu hudebních not. Jedná se o jeden z nejpopulárnějších způsobů ukládání not, který využívá řada editorů a jiných hudebních programů. [11]

MusicXML je ve své podstatě formát XML, jehož struktura je definována pomocí schématu. Tento umožňuje ukládat velkou většinu hudebních symbolů, značek a popisků. Dnes se jedná o de facto standard pro sdílení hudby mezi programy.

Lilypond

Lilypond [4] je program, který umožňuje generování hudebních not z jazyka, který je podobný TeX. Narozdíl od MusicXML, který je spíše výstupem nějakého programu, je zde přístup k vytvoření hudebních not opačný. Noty se píšou rovnou ve formátu LilyPond a následně se z něj vygenerují noty.

Lilypond je nejenom schopný generovat hudební notaci, ale i MIDI soubor. Tato schopnost je ideální pro využití v tomto projektu, který ho předá programu `FluiSynth` pro vytvoření zvukové reprezentace hudebních not.

[3] https://www.musicxml.com/
[4] https://lilypond.org/

```
\version "2.25.20"

\relative c' {
  \clef "treble"
  \time 2/4
  \key d \major

  \tempo 4=120

  cis'4 e8 d8

  a8 b8 g4
}
```

Výpis 2.1: Kód pro generování hudební notace z obrázku 2.1

MIDI

Musical Instrument Digital Interface (MIDI) je standard, který definuje binární přenos informací z digitálního hudebního nástroje, které obsahují instrukce pro přehrání daného zvuku. Mezi tyto informace patří například zmáčknutá klávesa, podrobnosti o úderu a podobně.

MIDI soubor tedy neobsahuje samotné audio, ale pouze instrukce k tomu, jak ho vytvořit.

Kapitola 3

Návrh L-Systému

Tato kapitola se zabývá návrhem L-systému, který bude v kapitole 4 implementován. V první sekci vymezuji požadavky na generovanou hudební notaci. Dále se shrnují pravidla, která se typicky využívají při skládání hudby. Poté je definován navržený L-systém, kde interpretace výsledného řetězce v sekci 3.4 vysvětlí jeho vztah k hudební notaci. Předposlední sekce 3.5 se věnuje možnostem rozšíření a úpravám navrženého L-systému. V poslední sekci je shrnut postup pro vygenerování hudební notace pomocí daných formálních modelů.

3.1 Omezení notace

Protože hudební notace je velice obsáhlá vzhledem k symbolům a pravidlům, je potřeba jí omezit, aby bylo možné reprezentovat její podmnožinu v implementovaném programu.

Zavedená omezení:

- Pouze je den hlas [1]
- Bez akord
- Pevně stanovený taktový předpis
- Noty mají pouze výšku a délku. Dynamika a podobné prvky se neberou v potaz.

Tyto omezení úspěšně omezují hudební notaci natolik, aby bylo možné jí generovat i s pomocí L-systémů.

3.2 Hudební pravidla

Pro definici pravidel jsou dodržovány některé zaběhlé hudební transformace, mezi které patří [4]:

- *Pravidlo opakování* - opakování noty
- *Pravidlo předání* - doplnění mezery mezi dvěma noty
- *Pravidlo souseda* - rozšíření noty přidání nové noty, jejíž tón je blízko

[1] Plánuji v budoucnu implementovat

- *Pravidlo úniku* - doplnění mezery notou, která je přidána v opačném směru od sousedící noty

[4] tato pravidla popisuje do větší hloubky a následně je používá pro generování not pomocí pravděpodobnostních gramatik.

3.3 Stochastický Kontextově-Citlivý L-Systém

Z důvodu, aby bylo jednoduché definovat pravidla ze sekce 3.2, je použit kontextově závislý stochastický L-systém, u kterého má každé předešlé pravidlo určitou pravděpodobnost použití. Například pravidlo souseda lze zapsat jako: : FF $\rightarrow_{\frac{1}{2}}$ F+F-F, kde F je zaznamenání noty a $+/-$ mění výšku tónu.

Abeceda

Abeceda navrženého L-systému vypadá následovně:

$$V = \{F, +, -, [,], d\}$$

Pravidla

Seznam všech pravidel vypadá následovně:

1. *Identita*: F $\rightarrow_{\frac{1}{2}}$ F
2. *Opakování*: F $\rightarrow_{\frac{1}{15}}$ FF
3. *Duplikace1*: F $\rightarrow_{\frac{1}{15}}$ F+F
4. *Duplikace2*: F $\rightarrow_{\frac{1}{15}}$ F-F
5. *Soused1*: FF $\rightarrow_{\frac{1}{40}}$ [Fd+F-F]
6. *Soused2*: FF $\rightarrow_{\frac{1}{40}}$ [Fd-F+F]
7. *Soused3*: FF $\rightarrow_{\frac{1}{40}}$ [dF+F]F
8. *Soused4*: FF $\rightarrow_{\frac{1}{40}}$ [dF-F]F
9. *Předání1*: F+F $\rightarrow_{\frac{1}{40}}$ [Fd+F+F]
10. *Předání2*: F-F $\rightarrow_{\frac{1}{40}}$ [Fd-F-F]
11. *Předání3*: F+F $\rightarrow_{\frac{1}{40}}$ [dF+F]++F
12. *Předání4*: F-F $\rightarrow_{\frac{1}{40}}$ [dF-F]- -F
13. *Únik1*: F+F $\rightarrow_{\frac{1}{40}}$ [Fd++F]- -F
14. *Únik2*: F-F $\rightarrow_{\frac{1}{40}}$ [Fd–F]++F
15. *Únik3*: F+F $\rightarrow_{\frac{1}{40}}$ [Fd+++F- -F]
16. *Únik4*: F+F $\rightarrow_{\frac{1}{40}}$ [Fd- - - -F++F]

První 4 pravidla slouží k rozšiřování délky díla a zbylá pravidla slouží k vytvoření melodie rozdělením tónů s využitím pravidel ze sekce 3.2. Definice pravidel je inspirována [14].

Axiom

Počáteční axiom:

$$\omega = F + + + + F - - F + + F - - - - F$$

3.4 Interpretace symbolů abecedy

Vysvětlení symbolů abecedy:

- *Symbol* '*F*': Zapsání tónu s danonu výškou do not. Pokud se nachází více symbolů F za sebou, tak se délka tónu prodlužuje.
- *Symbol* '+': Posun výšky aktuální noty o tón směrem nahoru vzhledem ke stupnici.
- *Symbol* '−': Posun výšky aktuální noty o tón směrem dolů vzhledem ke stupnici.
- *Symbol* '[': Uložení stavu na zásobník. Stav zahrnuje aktuální výšku a délku.
- *Symbol* ']': Vybrání stavu ze zásobníku.
- *Symbol* '*d*': Půlení aktuální délky noty.

Stav

Pro úspěšnou interpretaci je zapotřebí v průběhu ukládat a spravovat stav, který obsahuje následující hodnoty:

- Aktuální notu pro reprezentaci výšky tónu
- Aktuální délku noty

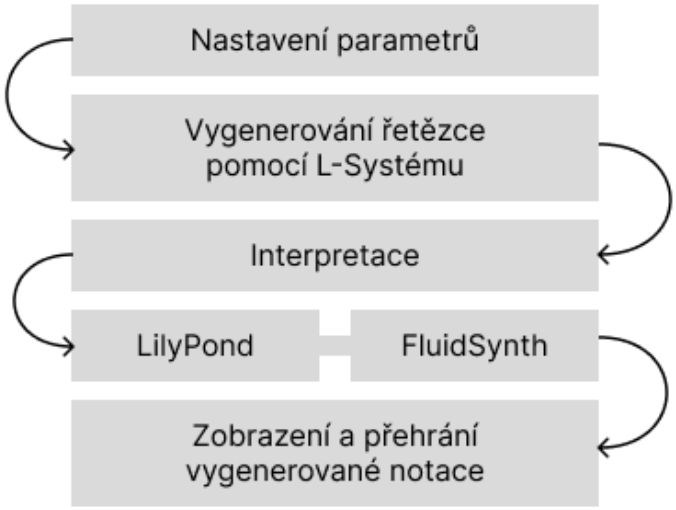
3.5 Možná rozšíření

Navržený model je v plánu do budoucna rozšířit. Tato rozšíření by přispěla ke vytváření složitějších kompozic. Konkrétně se jedná o následující:

- Náhodný výběr počátečního axiomu z množiny počátečních axiomů.
- Generování akordů aplikováním principů demonstrovaných v [5].
- Více hlasů. Noty pro klavír typicky obsahují 2 hlasy: jeden pro každou ruku.

3.6 Proces generování not

Pro využití navrženého L-Systému ke generaci not je zapotřebí zvolit počáteční parametry, mezi které patří tónina, počáteční nota a délka noty. Z těchto parametrů je možné vytvořit počáteční stav systému.



Obrázek 3.1: Shrnutí procesu pro vygenerování not

Dalším krokem je zvolení počtu iterací a aplikace pravidel na počáteční axiom. Výsledkem této operace bude řetězec, který lze následně interpretovat. Po interpretaci vznikne vnitřní reprezentace not, kterou je možné uložit ve zvoleném formátu `MusicXML` nebo `LilyPond` (viz sekce 2.9).

Uložení notace v `LilyPond` formátu umožňuje generaci grafické a MIDI reprezentace not, což nám dovoluje jí zobrazit uživateli ve výsledném programu. Výstupní soubor MIDI lze dále pomocí programu `FluidSynth` a virtuálního klavíru (ve formátu Soundfont) převést do hudebního souboru, který je poté přehrán uživateli.

Kapitola 4

Implementace

Tato kapitola se zabývá implementací programu, který pomocí vygenerovaného řetězce L-systému vytvoří hudební noty a zobrazí je uživateli v grafickém rozhraní. Aplikace také umožňuje uživateli přizpůsobit počáteční parametry překladu tak, aby vyhovovaly jeho použití. Program je napsaný v jazyce `Rust` a využívá nástrojů `Lilypond` a `FluidSynth`.

Implementace je rozdělena na několik logických částí. Nejprve je vysvětlen modul, který zabaluje logiku generování řetězců pomocí L-Systému. Dále je popsána interní reprezentace not a LilyPond formátu. Poté se sekce věnují interpretaci generovaného řetězce. Ukončující sekce vysvětlují implementaci grafického rozhraní, které umožňuje práci s daným řetězcem a jeho výsledné přeložení do hudební notace.

4.1 Požadavky

Cílem této implementace je vytvořit program, který bude schopný generovat hudební notaci na základě uživatelem specifikovaných parametrů. Program také bude schopný generované noty přehrát.

Tento program je schopen zachovat stav z jeho předchozích spuštění a umožňuje uživateli exportovat výsledné noty do archivu `TAR`.

4.2 L-systém

Tato sekce se zabývá implementací kontextově závislého stochastického L-systému, popsaného v sekci 2.6. Implementace zahrnuje definici datových struktur pro uložení pravidel, stavu systému a zajištění kontextové závislosti pravidel při přepisu řetězců.

Abeceda a axiom

Abeceda systému je v této implementaci reprezentována všemi znaky, které se vyskytují v definovaných pravidlech.

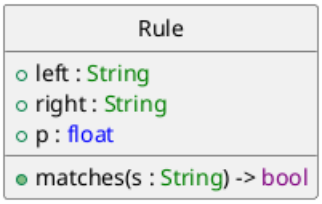
Axiom definuje uživatel, přičemž musí platit, že každý znak v axiomu se vyskytuje alespoň v jednom z pravidel.

Reprezentace pravidla

Jedním z prvních problémů je reprezentace pravidla. Z definice 2.6.12 vyplývá:

- Levá strana pravidla může být reprezentována 3. znaky z abecedy.
- Pravá strana může být reprezentována řetězcem znaků

Dále je zapotřebí, aby byla pravidla stochastická. Po vzoru definice 2.6.6 lze každému pravidlu přiřadit číslo, které bude určovat jeho pravděpodobnost použití v daném kontextu.



Obrázek 4.1: Třída `Rule`

Obrázek 4.1 ukazuje třídu *Rule*, která reprezentuje pravidlo. Pro jeho validitu musí program zajistit, aby $len(left) <= 3$ a zároveň pravděpodobnost $p \in < 0, 1 >$.

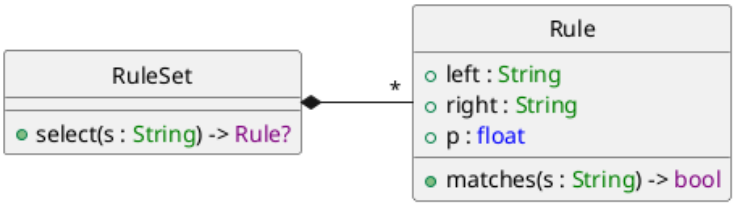
Dále je definována metoda *matches()*, která vrací pravdivou hodnotu *TRUE*, pokud nejpravější část řetězce vyhovuje levé straně pravidla a hodnotu *FALSE* jinak. Algoritmus 4.2.1 ukazuje možnou implementaci.

Kód 4.2.1: *Pseudo-kód funkce Rule::matches()*

```
1 function matches(rule, S) -> bool:
2     let end = last min(S.length, rule.left.length) characters of string
      S
3     return S == rule.left
```

Soubor pravidel

Pro uložení skupiny pravidel je definována třída *RuleSet*, která spravuje instance pravidel a dovoluje výběr jediného pravidla k danému řetězci.



Obrázek 4.2: Třída `RuleSet`

Třída obsahuje metodu *select()*, která slouží pro vybrání jednoho náhodného pravidla ze sady pravidel, které lze použít pro daný řetězec.

Pro výběr pravidla následuje definici 2.6.11. Tedy při překrývajících kontextech pravidel je vždy vybráno právě to nejpravější.

Příklad 4.2.1 Pravidla $P = \{abc \rightarrow 1, bcd \rightarrow 2\}$ a axiom $\omega = abcd$. Pro řetězec axiomu by se tedy vybralo pravidlo $bcd \rightarrow 2$.

Jedním z problémů zde je, že pravidla jsou stochastická. Důsledkem toho je, že výběr pravidla nemusí být vždy jednoznačný. Také definice 2.6.6 stanovuje, že součet pravděpodobností pravidel, které mají stejný poslední znak levé strany pravidla, musí být roven 1. Zbytek kontextu levých pravidel nemá vliv na jejich pravděpodobnost. Z tohoto důvodu je pravidlo vybíráno následujícím způsobem:

1. Nalezení všech pravidel, které splňují kontext zprava
2. Sečtení všech jejich pravděpodobností
3. Vygenerování náhodného čísla od 0 do součtu
4. Postupné akumulování součtu pravděpodobností pravidel a vybrání prvního z pravidel, jehož přičtením by byl akumulovaný součet vyšší, než náhodně vybrané číslo.

Pseudokód výběru tedy vypadá následovně:

Kód 4.2.2: *Pseudokód vybrání pravidla pro daný kontext*

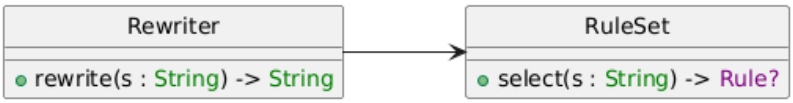
```
1  function select(self, ctx) -> Rule:
2      let matching_rules = self.rules.where(rule -> matches(rule, ctx));
3      let total_probability = matching_rules.map(rule -> r.prob()).sum();
4      let rand = random(0, 1) * total_probability;
5      let acc = 0;
6
7      for rule in matching_rules:
8          acc += rule.prob();
9          if acc > rand:
10             return rule;
11
12     return matchin_rules.last();
```

Tento způsob výběru pravidla zanechá poměry pravděpodobností i v případě, že se nesčítají do 1 (sčítají se do čísla < 1).

Přepis řetězce

V tuto chvíli lze již vykonat pravý derivační krok. V této implementaci je nazýván jako jeden *přepis řetězce*.

Za tímto účelem je implementována třída *Rewriter* (viz. obrázek 4.3), jejíž jediný účel je, přepis jednoho řetězce na druhý s využitím sady pravidel.



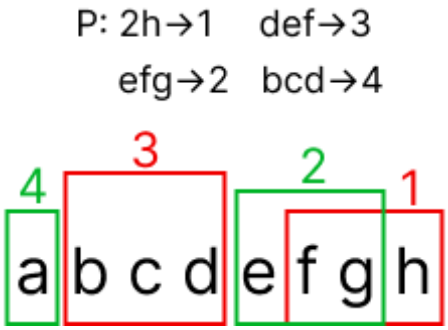
Obrázek 4.3: Třída `Rewriter`

Způsob přepisu vychází z definice 2.6.11. Začne se vpravo a postupně se přepisuje řetězec směrem vlevo. V průběhu je zapotřebí dát pozor na to, aby se nepřepsala část řetězce, která již byla použita v kontextu jednoho z použitých pravidel v předešlých krocích. (viz. definice 2.6.9).

Délky levých stran pravidel jsou variabilní, ale jejich maximální délka je 3 (viz. definice 2.6.7). Z tohoto důvodu je řetězec procházen zprava a vždy se zkouší najít pravidlo pro kontext daného charakteru a jeho dvou levých sousedů (kód 4.2.2). Po vybrání pravidla přepis posouvá o délku použitého pravidla vlevo, aby další kontroly nepřekrývaly kontext. Pokud není žádné pravidlo pro daný kontext nalezeno, předpokládá se použití pravidla identity (definice 2.6.9); charakter je zanechán tak a přepis je posunut o jeden znak vlevo.

Příklad 4.2.2

Nechť existuje systém $L = (V, abcdefgh, \{2h \rightarrow_{p_1} 1, efg \rightarrow_{p_2} 2, def \rightarrow_{p_3} 4, bcd \rightarrow_{p_4} 4\})$. Obrázek 4.5 ukazuje jednotlivé kontexty, pro které byla vybírána pravidla. Podle indexů lze vidět, v jakém pořadí se kontexty testovaly.



Obrázek 4.4: Přepis řetězce

Pseudokód funkce *Rewrite* vypadá následovně:

Kód 4.2.3: *Pseudokód funkce Rewrite*

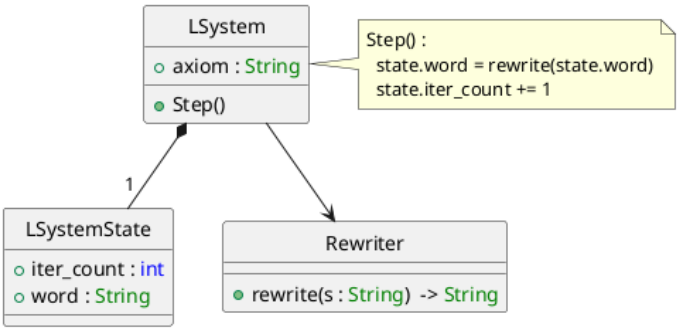
```
function rewrite(self, str) -> String:
    let window = last_3_characters_of(str);
    let rewritten_str = empty_string();

    while window.is_valid():
        if let rule = self.ruleset.select(window):
            rewritten_str.prepend(rule.left);
            move_window_left_by(length(rule.left));
        else:
            rewritten_str.prepend(last_character_of(window));
            move_window_left_by(1);

    return result;
```

Správa a modifikace stavu

Pro zjednodušení práce s třídou *Rewriter* byla vytvořena třída *LSystem*, která zapouzdřuje stav systému – například aktuální řetězec, počet iterací a podobně – a poskytuje funkce pro jeho úpravu, včetně iterace systému prostřednictvím třídy *Rewriter*.

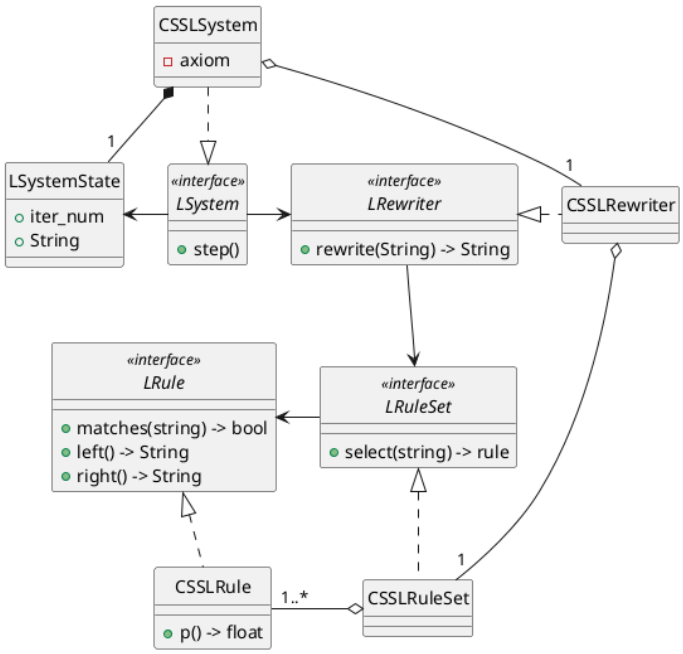


Obrázek 4.5: Třída `LSystem`

Třída poskytuje metodu *Step()*, která slouží k vykonání jednoho derivačního kroku (přepisu stavového řetězce).

Výsledný diagram tříd modulu LSystem

Výsledný diagram ukazuje obrázek 4.6, který mimo jiné obsahuje i rozhraní. Program je napsaný tak, aby bylo možné jej v budoucnu rozšířit i o jiné typy pravidel a přepisování.



Obrázek 4.6: Třída `LSystem`

V případě budoucí implementace například *D0L-systému* (viz definice 2.6.5) by postačilo vytvořit konkrétní implementace příslušných rozhraní a upravit způsob instanciace objektů. Ostatní části systému, které na tento modul navazují, by bylo možné ponechat beze změny, čímž by se minimalizoval zásah do existujícího kódu.

Jedním z hlavních záměrů tohoto návrhu je vytvořit otevřený rámec, jenž v budoucnu umožní experimentální začlenění různých variant L-systémů a jejich srovnání při generování hudební notace.

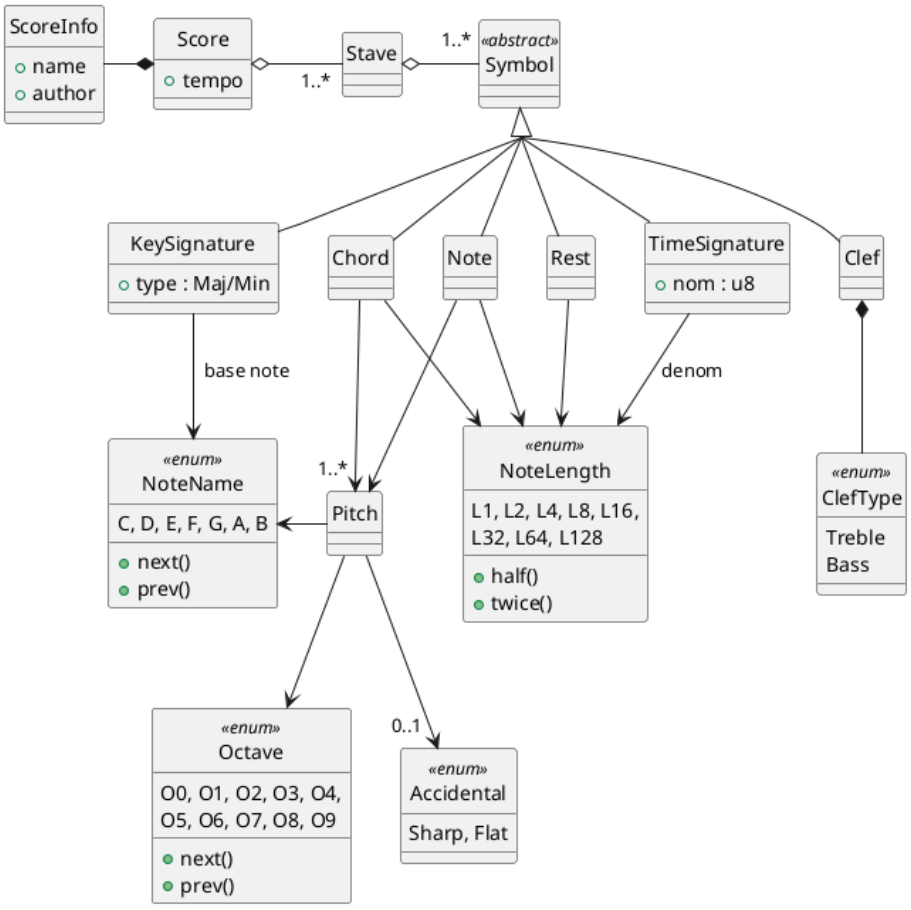
4.3 Interní reprezentace notace

Tato sekce se věnuje návrhu a popisu interní reprezentace datového modelu hudební notace. Model je koncipován tak, aby byl nezávislý na cílovém výstupním formátu (například LilyPond), do něhož bude v budoucnu převáděn. Zároveň je navržen s ohledem na snadnou „generovatelnost“ během interpretace řetězce získaného z L-systému.

Reprezentované prvky notového zápisu

Tato práce se věnuje omezenému počtu prvků hudební notace. Záměrně je abstrahována od některých složitějších prvků, které lze najít ve standardním notovém zápisu. Hudební notace jako taková obsahuje mnoho vyjadřovacích prostředků, jako jsou například artikulační znaménka (např. staccato), opakovací značky (repetice, dal segno apod.) či rytmická rozšíření (nota s tečkou). Tyto prostředky nejsou popsány navrženou gramatikou (viz sekce 3.3), což je důvodem k jejich nezahrnutí do interní reprezentace.

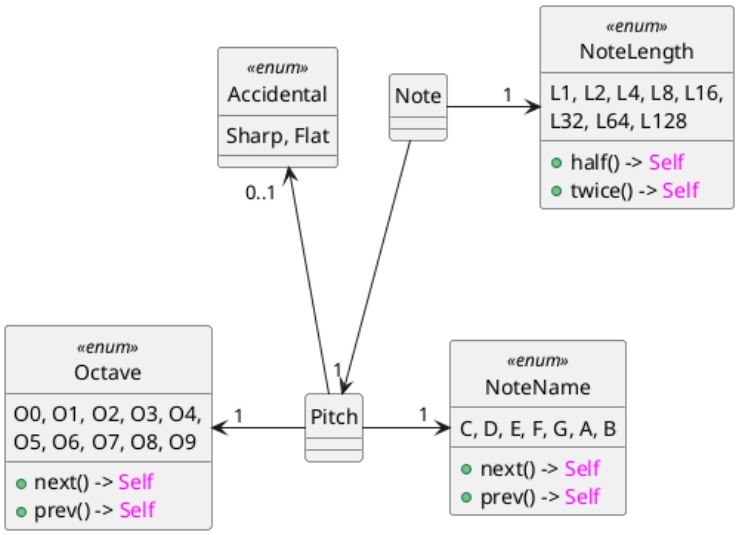
Návrh tedy obsahuje následující prvky hudební notace: více notových osnov, notu a akord se stejnou délkou not, křížky a béčka, a nakonec houslový a basový klíč s předznamenáním. Celková struktura reprezentace těchto prvků je zahrnuta v diagramu tříd, který lze vidět na obrázku 4.7



Obrázek 4.7: Diagram tříd modulu `notation`

Nota

Nota je grafickým vyjádřením tónu (viz kapitola 2.1). V běžném notovém zápisu však nota reprezentuje primárně výšku a délku tónu. Barva tónu je určena použitým hudebním nástrojem a dynamika tónu je vyjádřena pomocí dynamických znamének (např. akcenty, crescendo, decrescendo a další).



Obrázek 4.8: Třída `Note` a jiné

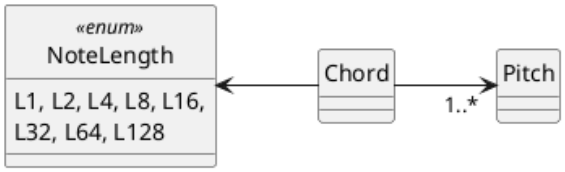
Reprezentace výšky tónu je navržena tak, aby umožňovala snadný převod do notového zápisu. Z tohoto důvodu je definováno sedm základních tónů: C, D, E, F, G, A a B/H. Pro vyjádření půltónů je zavedena enumerace `Accidental`, která reprezentuje zvýšení (křížek) nebo snížení (béčko) tónu. Kombinací základních tónů a těchto úprav lze reprezentovat všech dvanáct tónů v rámci jedné oktávy. Pro eliminaci problémů s omezeným počtem oktáv jsou i jednotlivé oktávy reprezentovány pomocí enumerace.

Délku tónu lze vyjádřit pomocí zlomku. Nejdelší hodnotu představuje celá nota, přičemž ostatní délky vznikají jejím postupným dělením. Existuje více způsobů dělení, avšak v rámci této práce je uvažováno výhradně půlení. Tento přístup byl zvolen z důvodu, že navržená gramatika obsahuje pouze symbol *d*, který reprezentuje půlení délky noty. Není proto nutné zavádět reprezentaci třetin, pětin a dalších dělení. Délky not jsou opět reprezentovány pomocí enumerace, která zahrnuje osm hodnot: celá nota, $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$, $\frac{1}{32}$, $\frac{1}{64}$ a $\frac{1}{128}$.

Implementovaná reprezentace noty má určitá omezení. Například není možné reprezentovat notu s tečkou, dynamická znaménka, artikulační znaménka a další hudební prvky. Přesto je tato zjednodušená reprezentace plně postačující pro účely práce a navrženou gramatiku. Pro budoucí rozšíření o tyto aspekty by bylo potřeba tuto třídu upravit.

Akord

Akord se skládá z více not. Dává tedy smysl jej reprezentovat polem not, avšak z důvodu jednoduchosti implementace a použití akordů je v této práci reprezentován skupinou výšek a jednou délkou noty. Tato variace má za důsledek, že můžeme modelovat pouze akordy, kde mají všechny noty stejnou délku.



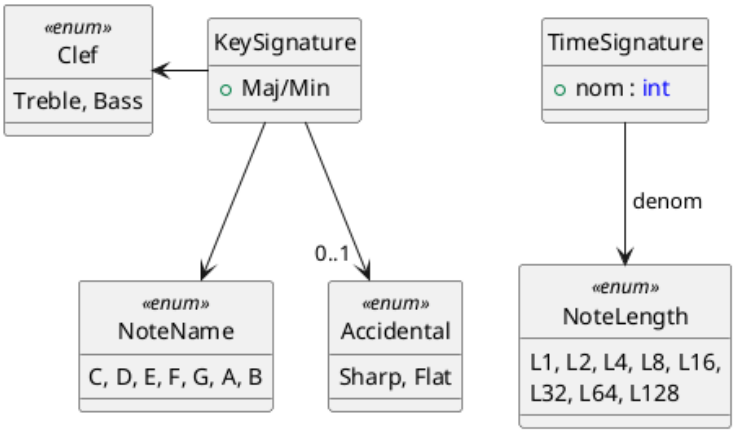
Obrázek 4.9: Třída `Chord`

Alternativní přístup k modelování akordů by spočíval v uchovávání základní noty a typu akordu, například durového nebo molového kvintakordu a jeho obratů. Tento přístup se však ukázal jako příliš složitý kvůli velkému počtu typů akordů. Jednodušší variantou je tedy přímo ukládat jednotlivé noty, které lze v budoucnu generovat na základě typu akordu.

Klíč, předznamenání a tempo

Každá sekce not začíná předznamenáním, které určuje zejména tóninu, v níž je daná část skladby zapsána. Součástí úvodu sekce je rovněž deklarace tóniny a časového označení taktu. Všechny tyto prvky zásadně ovlivňují sémantický význam následného notového zápisu.

V této interní reprezentaci není předznamenání ukládáno jako samostatný celek. Každý jeho prvek je reprezentován vlastním typem symbolu (stejně jako jednotlivé noty). Pro správnou interpretaci tedy postačuje definovat odpovídající datové vztahy mezi těmito prvky (viz obrázek 4.11).



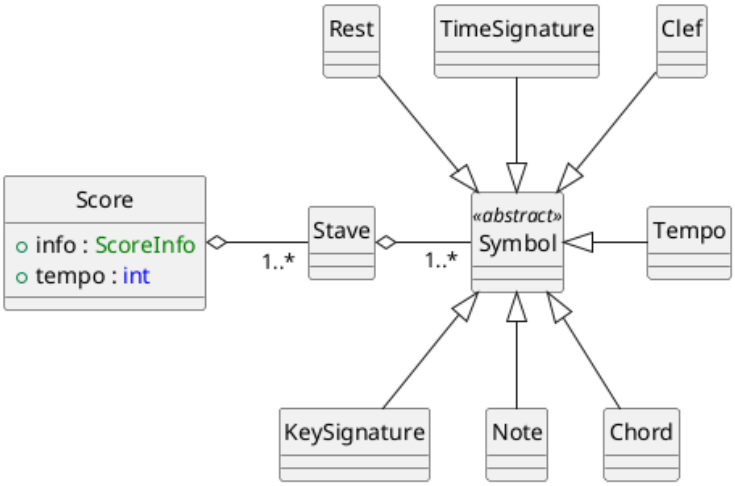
Obrázek 4.10: Třídy předznamenání

Časové označení (`TimeSignature`) je v interní reprezentaci modelováno pomocí dvou atributů: čitatele (`nom`) číselného typu a jmenovatele (`denom`), který je reprezentován enumerací `NoteLength`. Tento přístup nám umožňuje jednoduše popsat časová označení ve formátu zlomku.

Symbolická reprezentace členů předznamenání nám umožňuje vkládat tyto hudební prvky na libovolné místo ve skladbě. Zároveň poskytuje flexibilitu pro budoucí rozšíření modelu o další prvky, například o definici tempa nebo legata, bez nutnosti zásadních úprav stávající struktury.

Notová osnova

Další velmi důležitou částí tohoto modelu je notová osnova. Pro klavír nám typicky stačí pouze jedna, nebo dvě (pro každou ruku). Někdy se ale osnov, z různých důvodů, vyskytuje více. Model tuto skutečnost realizuje pomocí ukládání pole notových osnov, kde každá osnova obsahuje hudební prvky.



Obrázek 4.11: Třídy reprezentující osnovu

Tento model předpokládá, že musí vždy existovat alespoň jedna notová osnova, která bude obsahovat alespoň jeden symbol. Tímto způsobem se snažíme vyjádřit, že osnova by měla obsahovat alespoň klíč. Třída `Symbol` je zde přítomná pouze za účelem seskupení různých typů hudebních prvků. Neposkytuje tedy žádnou významnou společnou funkcionalitu mezi jednotlivými prvky.

4.4 Lilypond a převod

Lilypond je jedním z populárních nástrojů pro psaní hudebních not. Funguje jako překladač vlastního textového formátu do různých výstupních formátů, například PNG nebo PDF. V rámci této práce se zaměřuji především na formáty PNG a MIDI.

Nejprve je tedy zapotřebí převést interní reprezentaci hudebních not do formátu Lilypond, který vzápětí bude poskytnut jako jeho vstup. Pro tento účel jsem definoval interní objektový model, který reprezentuje strukturu formátu Lilypond. Tato infrastruktura je podobná interní reprezentaci, ale lépe reprezentuje některé objekty. Mezi tyto rozdíly patří například uložení oktáv.

Poté je nutné převést interní reprezentaci hudebních not do formátu Lilypond, který následně slouží jako vstup pro překlad. Za tímto účelem jsem navrhl interní objektový model, jenž odpovídá struktuře formátu Lilypond. Jeho struktura se do určité míry podobá původní interní reprezentaci, ale některé prvky vyjadřuje vhodněji. Příkladem může být odlišný způsob uchovávání informace o oktávách. Pro každý prvek tohoto modelu je definováno jak jej získat z původní interní reprezentace.

Nakonec je potřeba převést nově vytvořený objektový model do jeho textové reprezentace. K tomu je využit trait[1] *Display*, který v jazyce Rust slouží k definici textového výstupu objektů. Každá třída (struct) v modelu tento trait implementuje, což umožňuje snadný převod modelu do textové podoby pomocí makra `format!`[2].

Tento způsob převodu je sice datově relativně náročný, avšak nabízí značnou míru flexibility. Pokud by bylo v budoucnu potřeba převádět interní reprezentaci i do jiných formátů, například do MusicXML (viz kapitola 2.9), stačilo by vytvořit nový objektový model odpovídající danému formátu a definovat příslušný převod.

Skóre a notová osnova

Základem formátu Lilypond jsou informace o notovém zápisu, které mohou zahrnovat například verzi použitého programu, autora, název skladby, podtitul a další metadata. V této práci se zaměřuji pouze na základní prvky potřebné pro vytvoření výstupu.

Kód 4.4.1: *Struktura not v lilypond*

```
\version "2.25.25"

\language "deutsh"

\score {
    ... staffs or notes ...
    \layout{}
    \midi{}
}
```

Typická struktura začíná direktivou určující verzi programu (např. `\version "2.25.25"`), po níž následuje volba jazyka (`\language "deutsch"`), která ovlivňuje způsob zápisu not — například nota H se v německé notaci zapisuje jako `h` namísto `b`. Hlavní blok `\score` poté obsahuje samotný notový zápis, případně notové osnovy, a také sekce `\layout{}` pro vizuální výstup a `\midi{}` pro generování MIDI souboru.

Tato struktura odpovídá textové reprezentaci generované ze třídy *Lilypond* a tvoří výstupní podobu notového zápisu v této práci.

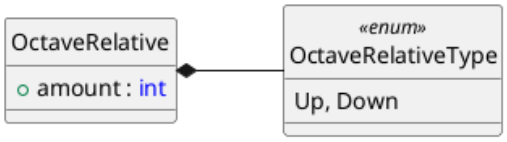
Relativní oktáva

Jedním z rozdílů oproti obecné interní reprezentaci je způsob zápisu oktáv. Lilypond nabízí několik možností, jak oktávu noty vyjádřit. V této práci používám tzv. absolutní mód, ve kterém je nejprve určena základní oktáva a následně se noty zapisují s relativním posunem směrem nahoru nebo dolů.

Tento přístup se výrazně liší od interního modelu, kde jsou oktávy reprezentovány jako prvky enumerace s devíti hodnotami. Převod mezi těmito dvěma způsoby je však poměrně přímočarý – základní oktáva v Lilypondu odpovídá třetí oktávě v interním modelu. Stačí tedy k dané notě přičíst nebo odečíst příslušný posun, což lze realizovat jednoduchou aritmetikou.

[1] `Trait` - Vlastnost = je prvek v Rustu, který nám umožňuje deklarovat množinu funkcí, které musí definovat každá struktura, jenž tento trait implementuje. Funguje podobně jako *rozhraní* v jiných jazycích.

[2] Makro v jazyce Rust, které slouží k formátování řetězců. Jeho obdobou v jazyce C je funkce `sprintf`.



Obrázek 4.12: Třídy reprezentující relativní oktávu

Ve formátu Lilypond se oktáva zapisuje pomocí znaku ’ pro posunutí o oktávu nahoru, nebo , pro posunutí o oktávu dolů. V našem modelu ukládáme kolikrát se jeden z těchto znaků musí opakovat, abychom dostali požadovanou oktávu.

Nota a notové osnovy

Základní zápis noty ve formátu Lilypond se skládá ze tří částí: názvu noty, posunutí oktávy a délky noty. Posun o půltón zde není nutné explicitně vyjadřovat, protože je již obsažen v názvu noty (například `cis` pro C#).

Kód 4.4.2: *Stupnice A-dur v lilypond formátu*

```
1 a4 h4 cis’4 d’4 e’4 fis’4 gis’4 a’4
```

Délka noty je vyjádřena pomocí čísla, které odpovídá jmenovateli zlomku celé noty. Například číslo 4 značí čtvrťovou notu ($\frac{1}{4}$), číslo 8 osminovou notu ($\frac{1}{8}$), a celá nota se zapisuje bez čísla.

Noty se následně ukládají v notových osnovách, které mají vlastní blok `\\Staff{}` a nacházejí se přímo v bloku `\\sore{}`. Z tohoto důvodu jsou noty uloženy, společně s jinými symboly, v polích, které reprezentují jednotlivé notové osnovy.

Předznamenání

Definice předznamenání nemá ve formátu LilyPond vlastní blok. Každý z prvků, které jej tvoří, má svou vlastní samostatnou direktivu. Konkrétně pracuji s následujícími prvky: klíč, tempo, časové označení a tónina. Všechny tyto prvky lze vložit na libovolné místo v rámci notové osnovy.

Každý prvek má svůj specifický způsob zápisu. Klíč se určuje pomocí sekvence `\clef treble` nebo `\clef bass`. Tónina se zapisuje pomocí direktivy `\key <jméno noty> \major` nebo `\key <jméno noty> \minor`. Tempo se definuje ve tvaru `\tempo <délka noty> = <rychlost>`, například `\tempo 4 = 120` a časové označení se zapisuje pomocí `\time <čitatel>/<jmenovatel>`, například `\time 4/4`.

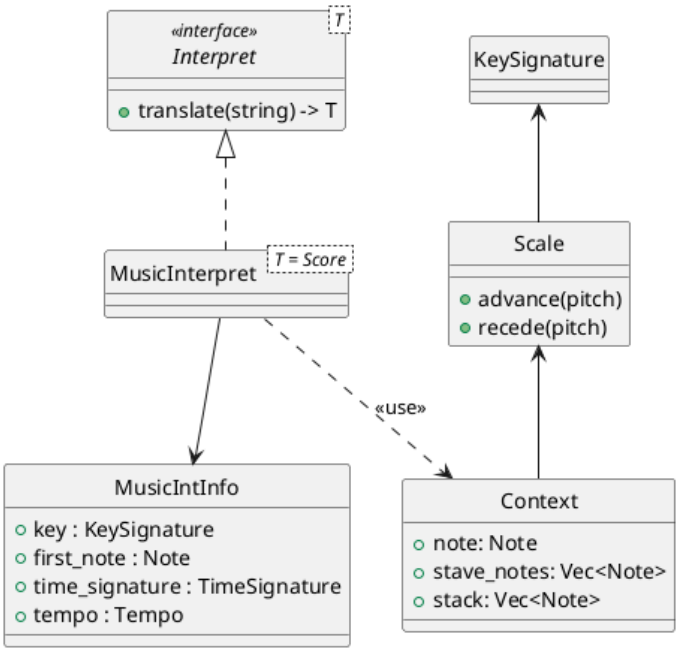
Kód 4.4.3: *Příklad předznamenání*

```
1 \Staff {
2     \clef treble \key d \major \tempo 4 = 120 \time 4/4 d4 e4 fis4 g4
3 }
4 \Staff {
5     \clef bass \key d \major \time 4/4 d,4 e,4 fis,4 g,4
6 }
```

4.5 Interpretace

Výstupem L-systému je řetězec, který slouží jako vstup pro interpret, který jej převádí do interní reprezentace. Tento interpret přijímá řetězec a jako výstup vrací instanci třídy `Score`, která reprezentuje výsledné hudební skóre.

Lze říci, že interpret implementuje jistou variantu tzv. „želvy“ popsané v sekci 2.8. Udržuje si interní stav (kontext) a postupně, zleva doprava, vykonává příkazy definované jednotlivými znaky vstupního řetězce. Výsledkem zpracování každého znaku je buď úprava aktuální noty, nebo vytvoření nové noty.



Obrázek 4.13: Diagram tříd modulu `Interpret`

Pro vytvoření kompletního skóre a jeho správnou interpretaci je nutné dodat interpretu také doplňující informace. Tyto informace mohou zahrnovat různé prvky, jako je například autor, název skladby a další metadata. V rámci této implementace je však vyžadováno pouze předznamenání, které není popsáno navrženou gramatikou L-systému. Musí být tedy poskytnuto zvlášť.

Kontext

Interpret si uchovává kontextové informace v instanci třídy `Context`. Tato data jsou průběžně modifikována během zpracování jednotlivých příkazů.

Jeden kontext odpovídá vždy jedné notové osnově a obsahuje mimo jiné informaci o aktuálně upravované notě (první nota je interpretu předána explicitně), množinu dosud vytvořených not a zásobník, který slouží návratu do předchozích stavů (znaky `[` a `]`).

Stupnice (Scale)

Pro zjednodušení interpretace je implementována třída `Scale`, jejímž účelem je reprezentace stupnice pro libovolnou výchozí notu. Třída poskytuje dvě hlavní funkce: `advance(pitch)`

a `recede(pitch)`, které přijímají výšku tónu a posunou ji nahoru, resp. dolů v rámci dané stupnice (odpovídající znakům `+` a `-` v abecedě L-systému).

Tento přístup umožňuje snadno zvolit typ použité stupnice. V rámci této implementace jsou podporovány durové, mollové a „jazz-like“ stupnice, ale teoreticky lze rozšířit funkcionalitu i o další typy.

Implementace je poměrně přímočará — pro každou stupnici je definována posloupnost intervalů mezi po sobě jdoucími tóny. Například u durové stupnice je známo, že mezi třetím a čtvrtým a mezi sedmým a osmým tónem je interval půltón, zatímco ostatní sousední tóny jsou od sebe vzdáleny o celý tón.

Pro získání následující noty je tedy nejdříve získáno pořadí dané noty ve stupnici, díky čemuž lze určit, jestli následující nota je vzdálena o celý tón, nebo jenom o půltón. Z toho důvodu definuje třída `Pitch` také metodu `value_halftone()`, která poskytuje právě tuto hodnotu. Posledním krokem je zvýšení (resp. snížení) oktávy, pokud následující (resp. předchozí) nota ve stupnici je v jiné oktávě.

Nevýhodou tohoto přístupu je, že jednu a tutéž notu lze vyjádřit více způsoby v závislosti na použité tónině a okolním kontextu. Například nota `C` v tónině D dur může být zapsána buď jako `His`, nebo jako `C` s případnou odrážkou. Bez znalosti okolního prostředí není možné jednoznačně určit, která z těchto variant je vhodnější.

Třída `Scale` navíc nebere v úvahu, zda by měly být preferovány křížky, nebo béčka. Tento nedostatek řeší tzv. „sanitizační fáze“, která je provedena po dokončení interpretace. V tomto kroku dochází k ekvivalentním úpravám výsledných not za účelem zvýšení čitelnosti zápisu (podrobněji viz sekce 4.6).

Výsledný postup

Pro úspěšnou interpretaci definuje třída `MusicInterpret` metodu `translate`, která nejprve vytvoří kontext. K tomu je zapotřebí znát počáteční notu a další doplňující informace, které jsou získány z předaného objektu `MusicIntInfo`. Po sestavení kontextu se do každé notové osnovy vloží symboly tvořící předznamenání. Následně se iteruje přes každý znak překládaného řetězce a provádějí se odpovídající akce nad kontextem:

- `+` – Zvýší výšku aktuálního tónu pomocí třídy `Scale`.
- `-` – Sníží výšku aktuálního tónu pomocí třídy `Scale`.
- `F` – Zapíše aktuální notu do pole tvořícího notovou osnovu.
- `[` – Uloží aktuální notu na vrchol zásobníku.
- `]` – Nahradí aktuální notu notou ze zásobníku.
- `d` – Zmenší délku aktuální noty na polovinu.

Nakonec se ze všech získaných osnov a metadat ze třídy `MusicIntInfo` vytvoří objekt `Score`, který je výsledkem interpretace.

4.6 Sanitizace výstupu

Výsledné skóre, které získáme interpretací, může být hůře čitelné kvůli častému zaměňování posuvek. Tento problém vzniká především kvůli bezkontextovému chování třídy `Scale`,

která pouze doplňuje správnou výšku tónů, ale již neřeší, jaká alterace (křížek, béčko, odrážka) by měla být použita.

Pro částečné vyřešení tohoto problému je implementována tzv. „sanitizační fáze“, která se spouští nad výsledným skórem a pomocí ekvivalentních úprav se snaží zlepšit čitelnost not. Relativně jednoduchým způsobem, jak toho dosáhnout, je nahrazení not s béčkem (resp. křížkem) za stejnou notu, která buď nemá žádnou posuvku (případně má odrážku), nebo má posuvku odpovídající použité tónině.

Základní tóniny lze rozdělit na dvě skupiny: tóniny s křížky a tóniny s béčky. Pro zjednodušení implementace předpokládáme, že ve většině případů preferujeme v křížkových tóninách křížky a v béčkových tóninách béčka. Výjimku z tohoto pravidla tvoří půltóny — například místo noty `C` nepreferujeme zápis `His` a podobně.

I když tato jednoduchá pravidla nejsou vždy optimální, poskytují relativně dobré výsledky pro náš účel. Výsledná funkcionalita je implementována ve třídě `Sanitizer`, která rozděluje základní tóniny do následujících skupin:

- **Preferovaná béčka** - C dur, G dur, D dur, A dur, E dur, H dur, Eis dur, His dur, Fes dur, Cis dur
- **Preferované křížky** - F dur, Hes dur, Es dur, As dur, Des dur, Ges dur, Ces dur, Ais dur, Dis dur, Gis dur, Fis dur

Mimo obecnou interní reprezentaci je upravována i LilyPond interní reprezentace. Konkrétně se jedná o doplnění konců osnov na vhodná místa. Tento krok je zapotřebí, protože LilyPond není vždy schopen tyto konce správně doplnit, což může vést k přetečení řádků mimo viditelnou část not. Pro vyřešení tohoto problému jsou využity dva uživatelem specifikované parametry: maximální počet not na jeden řádek a maximální počet taktů na jeden řádek. Pokud je jediná z těchto hodnot v jednom řádku překročena, sanitizér zde vloží symbol konce řádku.

Kód 4.6.1: *Pseudokód přidání konce řádků*

```
function add_breaks(notes[], max_notes, max_bars, total_bar_duration):
    let current_notes = 0;
    let current_bars = 0;
    let current_bar_duration = 0;

    for each note in notes:
        current_notes += 1;
        current_bar_duration += note.duration;

        if current_bar_duration > total_bar_duration:
            current_bars += 1;
            current_bar_len -= total_bar_duration;

        if current_notes >= max_notes OR current_bars >= max_bars:
            add_break_after(note);
            current_notes = 0;
            current_bars = 0;
```

4.7 Grafické rozhraní

Pro vizualizaci generované notace bylo vytvořeno grafické uživatelské rozhraní, které umožňuje interaktivní úpravu L-systému a interpretačních parametrů. Uživatel tak může aktivně definovat a upravovat pravidla L-systému a okamžitě sledovat jejich vliv na výslednou notaci. Mimo samotné zobrazení not, aplikace také umožňuje jejich přehrání pomocí syntetizéru `Fluidsynth`[3]. Uživatel si tak může poslechnout, jak by daná notace zněla například na klavíru či jiném zvoleném nástroji.

Knihovna egui

`egui: an easy-to-use GUI in pure Rust`[4] je knihovna pro jazyk Rust určená k tvorbě grafických uživatelských rozhraní. Nabízí širokou škálu předpřipravených widgetů[5], které usnadňují vývoj grafických aplikací. Knihovna je navržena s ohledem na snadnou integraci do stávajících Rust projektů bez nutnosti velké modifikace existující infrastruktury. [3]

Knihovna využívá přímého renderování, jehož výhodou je jeho jednoduchost použití. Jedná se o způsob vykreslování grafického rozhraní, při kterém se uživatelské rozhraní kompletně znovu vykresluje v každém snímku (frame) na základě aktuálního stavu aplikace. Toto má velký důsledek na strukturu kódu popisujícího vzhled aplikace. Nepíšeme zde pouze, co se má zobrazit a jak, ale můžeme rovnou kontrolovat stav widgetů a interakce s nimi. To nám dovoluje ihned reagovat změnou stavu aplikace. [3]

Na druhé straně spektra je tzv. *reatined mode* vykreslování. V tomto režimu se struktura uživatelského rozhraní ukládá jako strom objektů, kde každý z nich si uchovává vnitřní stav, ke kterému explicitně přistupují.

Velkou výhodou přímého renderování je tedy jeho jednoduchost - uživatelské rozhraní vždy odráží aktuální stav aplikace. Oproti retained vykreslování jsou zde prakticky eliminovány problémy s nekonzistencí stavů. Jednou z nevýhod je jeho neefektivnost. Sestavování celého rozhraní při každém snímku může být výpočetně mnohem náročnější než mít sestavený strom a obnovování pouze potřebných částí UI.

Knihovna EFrame

Knihovna `egui` sama o sobě slouží pouze k definici widgetů, jejich rozložení a interakcí s uživatelem. Nezabývá se však samotným vykreslováním, díky čemuž ji lze snadno integrovat do projektů, které již mají vlastní vykreslovací řetězec (například herní engine). [3]

Zmíněnou funkcionalitu doplňuje například knihovna `eframe`, která poskytuje kompletní framework pro provoz `egui` aplikací. Umožňuje snadné vykreslování `egui` komponent jak v nativním prostředí (např. na desktopu), tak ve webovém prohlížeči prostřednictvím WebAssembly. Zároveň zajišťuje správu aplikačních oken, událostí a další běžnou funkcionalitu potřebnou k vytvoření plnohodnotné GUI aplikace. [2]

Objektová struktura

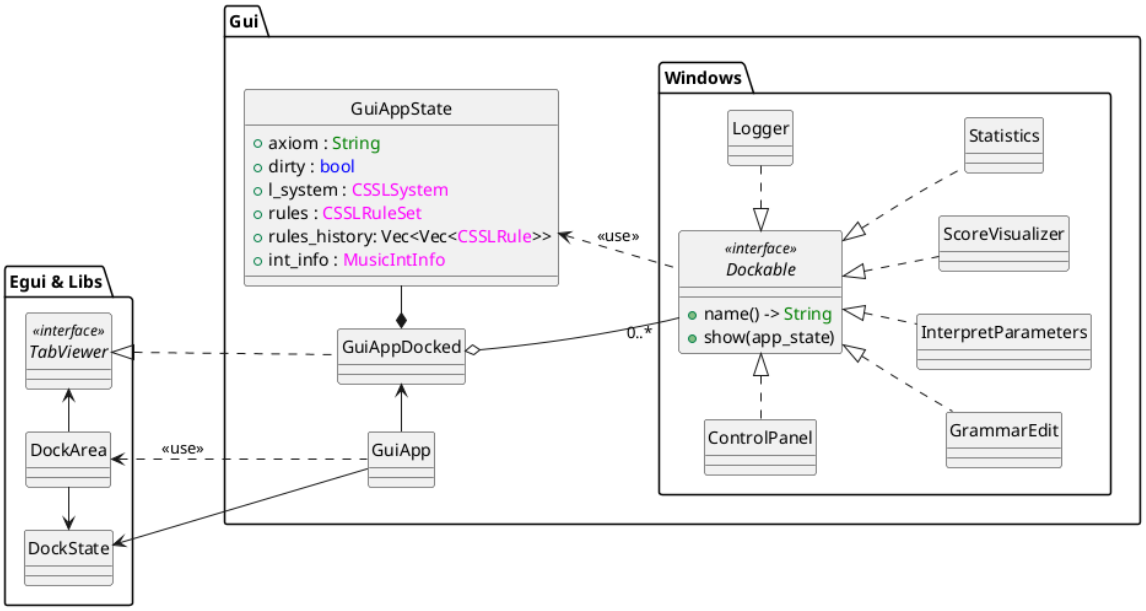
Navržené uživatelské rozhraní je poměrně přímočaré. Aplikace umožňuje zobrazení detailních informací o L-systému, včetně použitých kontextově závislých pravidel a axiomu. Zá-

[3] Fluidsynth je program určený k přehrávání MIDI souborů. Mimo jiné podporuje také export do formátu WAVE, což umožňuje uložit syntetizovaný zvuk jako běžný audio soubor.

[4] `https://github.com/emilk/egui`

[5] `Widget` je prvek grafického rozhraní, který umožňuje interakci s uživatelem – například textové pole, tlačítko nebo posuvník.

roveň je nutné poskytnout možnost nastavení parametrů interpreteru a zobrazit výslednou generovanou notaci. Uvedené informace je možné také upravovat, což uživateli umožňuje interaktivně testovat vliv různých pravidel na chování L-systému.



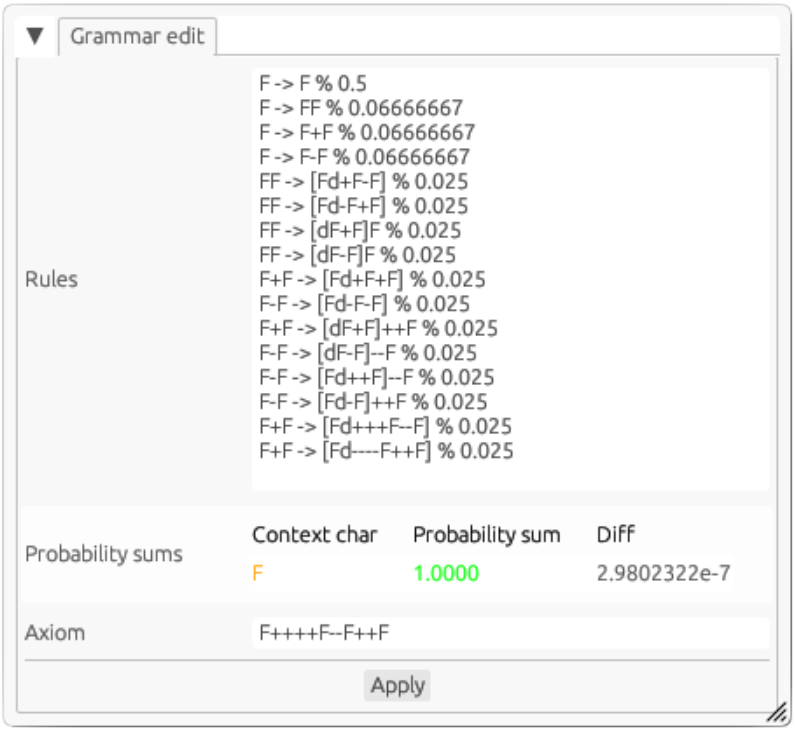
Obrázek 4.14: Diagram tříd modulu `GUI`

Potřebná funkcionalita je rozdělena do jednotlivých oken, které je možné dokovat[6] mezi sebou. Tyto okna pracují nad společnými daty, které jsou uložené v instanci třídy `GuiAppState` a jsou spravovány objektem `GuiAppDocked`. Obrázek 4.14 ukazuje tuto závislost v diagramu tříd.

Editace pravidel

Pravidla kontextově závislého stochastického L-systému jsou v aplikaci zobrazena v textovém poli, kde je možné je jednotlivě přidávat, upravovat nebo mazat. Každé pravidlo je reprezentováno jako řetězec tvaru `<left> -> <right> % <probability>`, kde `left` je levá strana pravidla, která může a nemusí být kontextově závislá, `right` je pravá strana a `probability` je pravděpodobnost použití pravidla při vhodném kontextu.

[6] Okna se chovají jako karty. Je možné například rozpůlit jedno okno napůl a vložit zde jinou kartu a podobně.



Obrázek 4.15: Okno s editací pravidel

Tento přístup je sice velice jednoduchý na implementaci, ale při úpravě pravidel je dosti snadné udělat chybu v jejich formátu, či součtu pravděpodobností. Z toho důvodu aplikace poskytuje kontrolu pravidel před jejich použitím při generaci. Při kontrole se poukazuje na pravidla se špatným formátem a kontroluje se, že součet pravděpodobností všech pravidel se stejným kontextovým znakem (nejpravější znak levé strany pravidla) je roven 1.

Editace parametrů interpretru

Pro správnou interpretaci výsledného slova L-systému je nutné nastavit počáteční parametry. Uživatelské rozhraní k tomuto účelu nabízí možnost úpravy předznamenání, které zahrnuje prvky jako klíč, tóninu, metrum [7] a tempo. Pro každý z těchto prvků byly navrženy vlastní ovládací prvky (widgety), které umožňují intuitivní volbu noty, výšky, posuvek, oktávy a délky noty.

Mezi parametry interpreteru patří také počáteční nota, tedy první nota, která je při překladu uložena do interního stavu interpreteru. Dalšími důležitými parametry jsou maximální počet not na jeden řádek a maximální počet taktů na řádek. Tyto hodnoty jsou následně využity během sanitizační fáze převodu interní reprezentace do formátu LilyPond, kde slouží k vkládání vhodných konců řádků pro lepší čitelnost výsledného notového zápisu a vyhnutí se přetečení osnovy mimo stranu.

Ovládací panel

Ovládací panel slouží k řízení překladu do notového zápisu ze vstupního řetězce L-systému. Uživatel zde může spustit přepis aktuálního řetězce (krok L-systému), přičemž výsledný

[7] Metrum - časování taktu

překlad se projeví v právě zobrazené notaci. Panel kromě možnosti vykonání jednoho kroku vpřed nabízí také krok zpět. Tato funkce je zvláště užitečná vzhledem ke stochastické povaze pravidel, která mohou generovat různé výstupy — uživatel tak může vyzkoušet více variant jednoho kroku generace.

Součástí ovládacího panelu je také tlačítko pro obnovení notace, které vyvolá nový překlad aktuálního řetězce. Tato funkce je klíčová například tehdy, pokud uživatel upraví počáteční parametry interpreteru a chce, aby se tyto změny promítly i do interpretace již existujícího řetězce.

Překlad, zobrazení a přehrání not

Ve sdíleném stavu aplikace je mimo jiné uchováván aktuální stav L-systému a jeho aktuálního řetězce. Aplikace sleduje jeho změny a v případě potřeby automaticky generuje odpovídající interní reprezentaci (třída `Score`). Po jejím vytvoření je tento model dále převeden do podoby hudebních not (obrázky) a odpovídající zvukové reprezentace. Program následně umožňuje přehrávání notového zápisu a procházení jednotlivých vygenerovaných stránek not.

Tato funkcionalita je soustředěna do komponenty `ScoreVisualizer`, která uživateli poskytuje možnost v reálném čase sledovat výsledek jednotlivých kroků L-systému pomocí zobrazení generovaných notových stránek a přehrávače generované hudby.

Proces převodu následuje schéma znázorněné na obrázku 3.1. Komponenta nejprve detekuje změnu aktuální iterace ve stavu objektu `CSSLSystem` a následně pomocí objektu `MusicInterpret` přeloží aktuální řetězec, čímž vznikne interní reprezentace hudební notace.

Pro zajištění čitelnosti not je tato reprezentace dále zpracována pomocí objektu `ScoreSanitizer` (viz kapitola 4.6), který upraví zápis bez zásahu do sémantiky not. Následně je tato struktura převedena do interního formátu `Lilypond`, který zachovává význam not, ale přizpůsobuje jej požadavkům nástroje `LilyPond`. V této fázi však model stále nemusí být vhodný k vizuálnímu vykreslení, protože může docházet k přetékání osnovy mimo obrázek. Tento problém je způsoben absencí explicitních konců řádků.

Z tohoto důvodu je nutné provést ještě další úpravy pomocí objektu `LilySanitizer`, který přidá nezbytné prvky specifické pro formát `LilyPond`, a připraví tak reprezentaci pro finální zobrazení.

Výsledný zápis je uložen do souboru na disku a následně předán jako vstup programu `LilyPond`, jehož spuštěním se vygenerují notové obrázky a odpovídající MIDI soubor. Obrázky jsou načteny zpět do aplikace a slouží jako vizuální reprezentace (noty) výsledného řetězce L-systému.

Pro převod MIDI do zvukového formátu je využit nástroj `Fluidsynth`, který umožňuje přehrávat nebo exportovat MIDI soubory pomocí virtuálního piana do formátu WAVE. Výstupní WAVE soubor je načten do paměti a následně přehráván objektem `AudioController`, který slouží jako adaptér[8] nad knihovnou `Rodio`[9]. Pro ovládání tohoto kontroléru byl vytvořen vlastní widget `AudioPlayer`, který uživateli poskytuje jednoduché rozhraní pro přehrávání výsledné hudební stopy generované L-systémem.

[8] Třída, která slouží k úpravě rozhraní jiné třídy tak, aby odpovídalo očekávanému formátu.

[9] Rust knihovna poskytující nástroje pro přehrávání různých audio formátů.

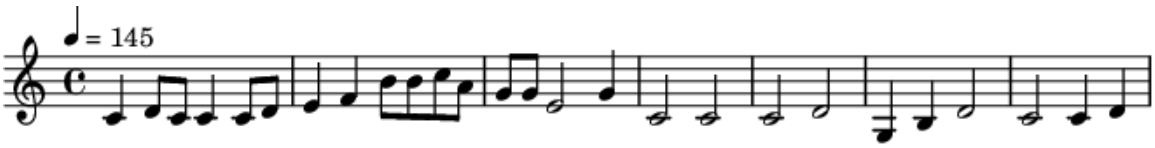
4.8 Dosažené výsledky

Implementovaná aplikace je zcela funkční a dovoluje interaktivně generovat noty. Tato sekce obsahuje ukázky generovaných not a jejich vazbu na počáteční parametry překladače.

Při použití navržených pravidel ze sekce 3.3 a následujících parametrů interpretu:

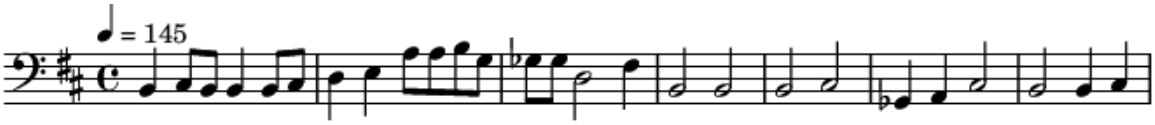
- Notový klíč — houslový
- Tónina — C dur
- První nota — C1 v houslovém klíči o délce půl doby
- Zápis taktu — 4/4
- Tempo — 145
- Maximální počet not na řádek — 45
- Maximální počet taktů na řádek — 7

jsou výsledkem noty, které jsou vidět na obrázku 4.21. První řádek těchto not zobrazuje obrázek 4.16. Vygenerovaná notace má dohromady jeden a čtvrt strany a délka přehrání tvoří cca 3 minuty.



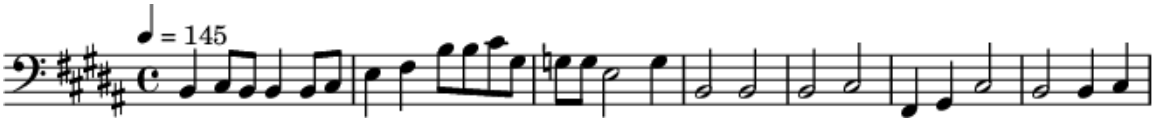
Obrázek 4.16: První řádek vygenerovaných not

Díky navrženým pravidlům dodržují noty relativně dobrou strukturu, která se rozbíjí pouze v některých kontextech.



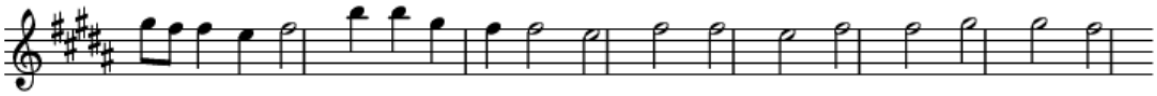
Obrázek 4.17: První řádek vygenerovaných not v H moll

Obrázek 4.17 ukazuje interpretaci stejného řetězce jako obrázek 4.16, s jinými parametry interpretu. Konkrétně se zde jedná o změnu tóniny na H-moll přirozenou, změnu houslového klíče na basový a změna počáteční noty na H. Zde je vidět, že sanitizační fáze nefunguje úplně nejlépe pro mollové tóniny. Například místo noty *fis* je zde zvolena nota *ges*.



Obrázek 4.18: První řádek vygenerovaných not v jazzové tónině

Obrázek 4.18 opět ukazuje interpretaci stejného řetězce, ale s tím rozdílem, že zde je naopak použita „jazz-like“ tónina.



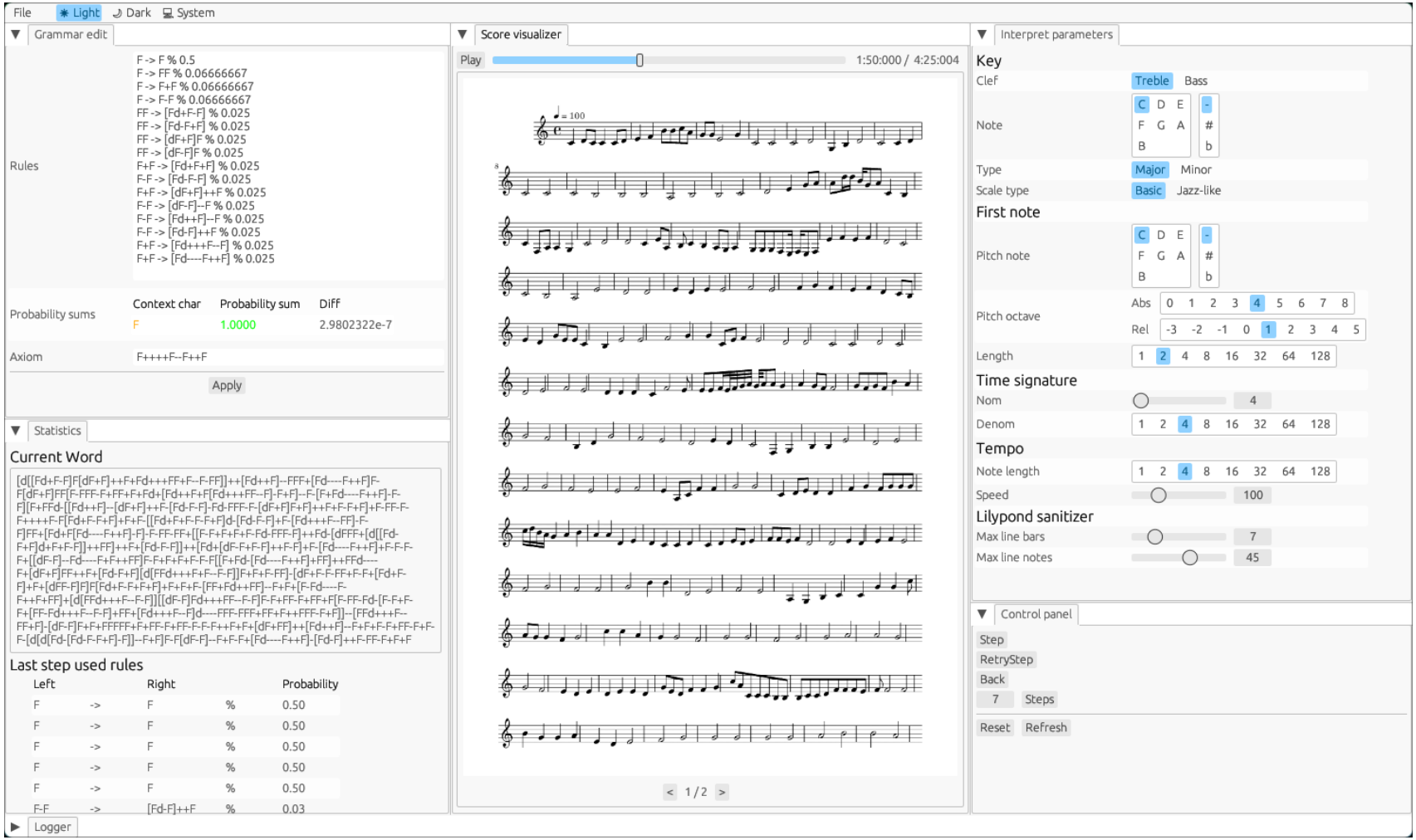
Obrázek 4.19: Řádek obsahující chybu v taktu — doby jednotlivých not se nesčítají do jednoho délky taktu (4/4).

Na obrázku 4.19 je znázorněna praktická ukázka chybně vygenerovaného taktu. Přestože je v předznamenání uveden takt 4/4, první takt obsahuje pět čtvrťových dob (půlová nota je započítávána jako dvě čtvrťové). Je patrné, že pokud by byla tato nota rozdělena na dvě svázané čtvrťové noty, mohla by být první z nich umístěna na závěr prvního taktu a druhá na začátek následujícího. Takové řešení by mohlo částečně eliminovat vzniklý problém, nicméně by neodstranilo jeho příčinu.

Za hlavní příčinu této chyby lze považovat pravidla 1-4 (viz sekce 3.3), která při přepisování nevyužívají půlení délky noty. Tím dochází k duplikaci i těch not, jejichž délka se liší od celé doby. Potlačení této chyby by bylo možné eliminací daných pravidel, čímž by však došlo k výraznému omezení generativní schopnosti systému, protože právě tato pravidla rozšiřují délku not.

Jedním z přístupů, který by mohl být dále experimentálně ověřen, by mohlo být rozdělení systému na dva samostatné systémy. První L-systém by obsahoval pouze pravidla 1–4, zatímco druhý by zahrnoval zbylá pravidla. Derivační kroky by pak byly prováděny nejprve s prvním systémem a následně s druhým. Tento přístup by mohl napomoci lepší kontrole nad taktovým členěním a přispět k dodržení správných délek taktů.

Obrázek 4.20: Okno statistik a ovládacího panelu



Obrázek 4.21: Výsledné grafické rozhraní

Testování

Jednotlivé moduly, s výjimkou grafického rozhraní, byly doplněny o automatizované jednotkové testy relevantních částí. Tyto testy slouží k ověření správnosti implementovaných konceptů a zároveň poskytují jistotu, že i při budoucích úpravách zůstane zachována požadovaná funkčnost systému.

Testování bylo zaměřeno především na moduly `Lilypond` a `LSystem`, v jehož rámci byly ověřovány třídy `Rule`, `RuleSet` a `Rewriter`. Na základě úspěšného testování těchto komponent lze usuzovat, že pravý derivační krok byl implementován v souladu s teoretickými požadavky definovanými v předchozích kapitolách. Přehled konkrétních testovaných případů je uveden v tabulce 4.1.

Testovaná metoda	Ověření rovnosti		
Rule::from()	rule.left()	rule.right()	rule.p()
"a->b%1/2"	"a"	"b"	0.5
"abc->def%1/4"	"abc"	"def"	0.25
"abc->def%0.125"	"abc"	"def"	0.125
Rule::from("ab->cd%1").matches()	Return		
"ab"	True		
"12345ab"	True		
"b"	False		
"ab1234b"	False		

Tabulka 4.1: Testování třídy `Rule`

Pro otestování třídy `RuleSet`, byla vždy vytvářena jedna z těchto dvou instancí:

- `BasicSet` = $\{\mathrm{a} \rightarrow_{0.5} \mathrm{b}, \mathrm{a} \rightarrow_{0.5} \mathrm{c}, \mathrm{b} \rightarrow_{1} \mathrm{d}\}$
- `BasicSet2` = $\{\mathrm{a} \rightarrow_{1} 1, \mathrm{b} \rightarrow_{1} 2, \mathrm{d} \rightarrow_{1} 3\}$
- `ContextSet` = $\{\mathrm{F} \rightarrow_{0.25} \mathrm{AA}, +\mathrm{F} \rightarrow_{0.25} \mathrm{BB}, \mathrm{F}+\mathrm{F} \rightarrow_{0.25} \mathrm{CC}, \mathrm{F}\text{-}\mathrm{F} \rightarrow_{0.25} \mathrm{DD}\}$,

kde jednotlivé testované případy ukazuje tabulka 4.2.

Testy třídy `Rewriter` pracují s následujícími pravidly:

- `Simple` = $\{a \rightarrow_{1} 1, b \rightarrow_{1} 2, c \rightarrow_{1} 3\}$
- `Complex` = $\{\mathrm{def} \rightarrow_{1} 11, \mathrm{bcd} \rightarrow_{1} 22, \mathrm{bc} \rightarrow_{1} 33, \mathrm{ab} \rightarrow_{1} 44, \mathrm{a} \rightarrow_{1} 55\}$
- `Example` = $\{2\mathrm{h} \rightarrow_{1} 1, \mathrm{efg} \rightarrow_{1} 2, \mathrm{def} \rightarrow_{1} 3, \mathrm{bcd} \rightarrow_{1} 4\}$,

kde testované varianty jsou vidět v tabulce 4.3.

Dále byly testovány stupnice, konkrétně třída `Scale` a její metody `Scale::advance()` a `Scale::recede()`. Ověření bylo provedeno pro všechny tři typy stupnic: durovou, mollovou a tzv. „jazz-like“ stupnici. U každého typu byla zvolena výchozí nota, na jejímž základě bylo následně vygenerováno sedm dalších po sobě jdoucích tónů. Tyto tóny byly následně porovnány s očekávanými hodnotami a bylo ověřeno, že odpovídají definici dané stupnice pro zvolenou bázovou notu.

Dále byly vytvořeny testy konverze interní reprezentace formátu LilyPond do řetězcové podoby. Tato funkcionalita je zásadní pro možnost využití interního modelu s programem

RuleSet varianta	RuleSet::select()	Opakování testu	Ověření rovnosti (bez P)
BasicSet	"a"	5	a → b nebo a → c
BasicSet	"b"	1	b → d
BasicSet	"ab"	1	b → d
BasicSet	"ba"	5	a → b nebo a → c
BasicSet	"abc"	1	nic
BasicSet2	"bac"	1	nic
BasicSet2	"aslkdbca"	1	a → 1
BasicSet2	"ca"	1	a → 1
BasicSet2	"a"	1	a → 1
BasicSet2	"bad"	1	d → 3
ContextSet	"FFF"	10	F → AA
ContextSet	"F-F+F"	10	F → AA nebo +F → BB nebo F+F → CC
ContextSet	"F-F"	10	F → AA nebo F-F → DD
ContextSet	"–F"	10	F → AA
ContextSet	"++F"	10	F → AA nebo +F → DD

Tabulka 4.2: Testování třídy `RuleSet`

RuleSet varianta	Rewriter::rewrite()	Ověření rovnosti
Simple	"abcdef"	"12c3ef"
Complex	"abcdef"	"553311"
Example	"abcdefgh"	"a42h"

Tabulka 4.3: Testování třídy `Rewriter`

`LilyPond`. Testován byl převod jednotlivých symbolů, notových osnov a výsledného skóre. Ve všech případech byla volána metoda `ToString()`, jejíž výstup byl následně porovnán s očekávanou řetězcovou reprezentací příslušného prvku.

Funkčnost grafického rozhraní nebyla testována automatizovaně, avšak byla průběžně ověřována ručně v průběhu vývoje. Testování se zaměřilo na správné předávání uživatelských vstupů, jako je úprava pravidel a nastavení parametrů překladače, do odpovídajících modulů aplikace. Dále bylo ověřováno správné zobrazení vygenerovaných not, jehož integrita byla ručně porovnávána s odpovídajícím řetězcem vytvořeným L-systémem. Přehrávání hudby a jeho soulad s generovanou notací byl ověřován poslechem. V tomto případě se předpokládala korektnost nástrojů `LilyPond` a `FluidSynth`, přičemž testování se soustředilo na správné předání dat mezi těmito nástroji. Dále byla testována i robustnost aplikace – bylo ověřeno, že se program v případě chybových stavů neukončí neočekávaně, ale zobrazí uživateli srozumitelné chybové hlášení.

Kapitola 5

Závěr

Tato práce byla zaměřena na zkoumání hudební notace z pohledu formálního jazyka. Cílem bylo prozkoumat možnosti formálního popisu hudební notace, analyzovat její gramatiku a zvolit vhodný přístup pro generování notace pomocí formálních prostředků. Výsledkem práce je implementace programu, v němž jsou navržené formální modely využívány k interaktivnímu generování a přehrávání různých notových zápisů pro klavír.

V teoretické části byla nejprve představena hudební notace, její struktura a základní prvky, které se v notovém zápisu běžně vyskytují. Důraz byl kladen zejména na význam tónu a jeho reprezentaci notou, řazení not do osnovy a roli a vliv předznamenání. Tato část poskytla základní stavební prvky potřebné pro následný návrh gramatického modelu.

Následně byly popsány formální koncepty, přičemž pozornost byla nejprve věnována základům teorie množin. Byly definovány klíčové pojmy jako množina a relace, které tvoří teoretický základ pro další části práce. Poté byly představeny konkrétní formální prostředky využitelné při generování hudební notace, především různé varianty L-systémů. Jako první model byl uveden základní Lindenmayerův systém (0L-systém), který byl dále rozšiřován o kontextová a stochastická pravidla. Důraz byl kladen na přesné definice a popis derivačních kroků, které jsou zásadní pro korektní a deterministickou implementaci těchto systémů.

Jako vhodný formální model pro generování not byl zvolen stochastický kontextově závislý L-systém, který v sobě kombinuje výhody obou přístupů. V další části práce byly popsány možnosti mapování výstupů L-systémů na jednotlivé prvky hudební notace, přičemž byly shrnuty běžné přístupy a diskutovány jejich výhody a nevýhody.

Navazující část se věnovala tzv. modelu želvy, který představuje jeden z možných způsobů grafické interpretace L-systémů a představuje základní myšlenku implementovaného interpretu. Na závěr byly popsány běžné formáty pro uložení hudební notace v digitální podobě, konkrétně formáty `MusicXML` a `LilyPond`, jež lze také považovat za formální jazyky sloužící k popisu struktury hudebního zápisu a jejichž použití umožňuje generování výsledných not.

Praktická část práce se v první řadě zaměřila na návrh použitého formálního modelu. V této kapitole byly detailně popsány jednotlivé komponenty navrženého kontextového stochastického L-systému, jeho možná rozšíření do budoucna a celkový proces generování not s použitím nástrojů LilyPond a FluidSynth.

Navržený model byl následně využit při implementaci, v jejímž rámci byla podrobně rozebrána struktura aplikace a funkce jejích jednotlivých vrstev. Ačkoli byla výsledná aplikace realizována v programovacím jazyce Rust, popis implementace byl koncipován tak, aby nebyl závislý na konkrétní technologii, ale aby reflektoval obecnou architekturu a chování systému. V první části této kapitoly byla popsána implementace L-systému, přičemž byl

vysvětlen způsob reprezentace a uchovávání pravidel, jejich výběr na základě kontextu a přepis vstupního řetězce pomocí pravé derivace. Součástí této sekce byly rovněž uvedeny pseudokódy použitých relevantních algoritmů, které odrážejí definované pojmy teoretické části.

V další části byla popsána reprezentace dvou modelů hudební notace – obecného interního modelu a specifického modelu odpovídajícího formátu LilyPond. Byly zdůvodněny výhody obou přístupů a vysvětlen proces převodu z obecného modelu do konkrétního výstupního formátu. Interní model byl následně využit v rámci implementace interpreteru, kde byl popsán způsob udržování interního stavu (kontextu) a překlad výstupu L-systému do této reprezentace. Dále byla popsána implementace různých stupnic použitých při překladu a na závěr byly představeny postupy pro sanitizaci výstupu s cílem zlepšit čitelnost a hudební srozumitelnost vygenerovaných not.

Závěrečnou částí implementace byl popis vytvořeného grafického rozhraní, které slouží jako nadstavba nad již existujícími moduly. V této části byl popsán způsob reprezentace jednotlivých částí aplikace, jako je například úprava gramatiky a axiomu L-systému nebo volba počátečních parametrů překladače. Hlavní pozornost byla věnována propojení všech doposud implementovaných tříd a jejich využití pro generování výsledného notového zápisu.

Dále byl popsán proces překladu interní reprezentace do formátu zpracovatelného nástrojem `LilyPond`, stejně jako vytvoření zvukové reprezentace not využitím `FluidSynth`. Na závěr byly shrnuty dosažené výsledky a provedeno jejich zhodnocení včetně návrhů na možné budoucí rozšíření a vylepšení.

Pro jednotlivé moduly implementace, s výjimkou grafického rozhraní, byly vytvořeny jednotkové a v některých případech také modulární testy. Tyto testy slouží nejen jako podpora vývoje založeného na testování, ale také jako nástroj pro ověřování správné funkčnosti programu při budoucích změnách a rozšířeních.

Aplikace má velký potenciál na to být dále rozvíjena. Mezi možná rozšíření patří například přidání více notových osnov využitím více instancí L-systému s potencionálně různými definicemi. V takovém případě by vznikly dva samostatné řetězce, které by po překladu byly vloženy do již implementovaného pole osnov v interní reprezentaci. Tento postup by si však vyžádal dodatečné mechanismy zajišťující harmonii mezi jednotlivými osnovami, především z hlediska synchronizace a nekonfliktnosti not.

Identifikovaný nedostatek souvisí s nesprávným počtem dob v některých taktech výsledného notového zápisu. Součet délek not v jednotlivých taktech nemusí vždy odpovídat zadanému taktovému označení. Tato chyba je důsledkem generativních pravidel (pravidla, která nepůlí dobu noty) L-systému. Jedním z možných řešení je rozdělení pravidel do dvou skupin – generativní a strukturální – a zavedení dvoufázového procesu generování. V první fázi by byla použita pravidla určující základní melodickou strukturu, zatímco ve druhé fázi by byly aplikovány pravidla definující hudební detail taktů.

Alternativní přístup k tomuto problému by spočíval v zavedení nového symbolu pro vázání not. Výstupní sanitizér by detekoval takty s nesprávnou délkou a konfliktní noty by následně rozdělil na dvě vázané noty tak, aby byla taktová délka zachována.

Další možné rozšíření se týká generování akordů. Za tímto účelem by mohl být navržen nový symbol abecedy, který by byl s určitou pravděpodobností vkládán místo symbolu `F` a reprezentoval by přidání noty do akordu namísto vytvoření nové noty v čase. V takovém případě by bylo nutné zajistit integritu akordů, například dodržením rozsahu či intervalu mezi jednotlivými tóny. Ještě ambiciózněji vylepšením by bylo využití tzv. akordových prostorů s využitím dané tóniny, což představuje zajímavou oblast s potenciálem pro další výzkum.

Vyvinutá aplikace může najít uplatnění jako počáteční bod pro testování různých typů pravidel při algoritmické generaci hudebních not pomocí L-systémů. Poskytuje okamžitou odezvu ve formě not a jejich hudební reprezentace, což umožňuje uživateli rychleji zjistit vliv jednotlivých pravidel na výslednou notaci a dovoluje mu prohloubit si znalosti v této oblasti informatiky.

Literatura

[1] Abelson, H. a diSessa, A. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics* online. MIT Press Direct, 1981. Dostupné z: https://direct.mit.edu/books/oa-monograph/4663/Turtle-GeometryThe-Computer-as-a-Medium-for. [cit. 2025-01-10].

[2] emilk. *Eframe: the egui framework*. 2025. Dostupné z: https://github.com/emilk/egui/tree/master/crates/eframe. Accessed: 2025-05-05.

[3] emilk. *Egui: an easy-to-use GUI in pure Rust*. 2025. Dostupné z: https://github.com/username/repository. Accessed: 2025-05-05.

[4] Gilbert, E. a Conklin, D. A Probabilistic Context-Free Grammar for Melodic Reduction *. In: *International Workshop on Artificial Intelligence and Music, 20th International Joint Conference on Artificial Intelligence*. 2007. Dostupné z: https://api.semanticscholar.org/CorpusID:18020161.

[5] Gogins, M. Score Generation in Voice-Leading and Chord Spaces. In: *International Conference on Mathematics and Computing*. 2006. Dostupné z: https://api.semanticscholar.org/CorpusID:30297526.

[6] Hliněná, D. *Relace*. 2025. Prezentace z předmětu IDM.

[7] iv john a. maurer. *A Brief History of Algorithmic Composition* online. 1999. Dostupné z: https://ccrma.stanford.edu/~blackrse/algorithm.html. [cit. 2025-01-01].

[8] Manousakis, S. *Musical L-Systems* online. 2006. 133 s. Diplomová práce. The Royal Conservatory, The Hague. Dostupné z: https://quod.lib.umich.edu/i/icmc/bbp2372.2006.122/1. [cit. 2025-01-10].

[9] McCormack, J. Grammar-based music composition. *Complexity International*, duben 1996, sv. 3. ISSN 1320-0682.

[10] Prusinkiewicz, P. Score Generation with L-Systems. In: *International Conference on Mathematics and Computing*. 1986. Dostupné z: https://api.semanticscholar.org/CorpusID:16101545.

[11] Schulze, W. *A Formal Language Theory Approach To Music Generation* online. Private Bag X1, Matieland 7602, South Africa., 2009. 126 s. Diplomová práce. Department of Mathematics, Applied Mathematics and Computer Science, University of Stellenbosch. Vedoucí práce Merwe, P. A. van der. Dostupné z: https://www.researchgate.net/publication/45635438_A_formal_language_theory_approach_to_music_generation. [cit. 2025-01-10].

[12] Semily, Z. *ZÁKLADNÍ HUDEBNÍ TEORIE* online. ZUŠ Semily, 2016. Dostupné z: https://www.zussemily.cz/images/ke_stazeni/ZAKLADNI_POJMY.pdf. [cit. 2025-01-10].

[13] Tfirn, M. *THE MUSICAL MAPPINGS OF L-SYSTEMS* online. 2012. 140 s. Diplomová práce. Faculty of Wesleyan University. Vedoucí práce Matthusen, D. P. Dostupné z: https://www.academia.edu/24235602/THE_MUSICAL_MAPPINGS_OF_L_SYSTEMS. [cit. 2025-01-10].

[14] Worth, P. a Stepney, S. Growing Music: Musical Interpretations of L-Systems. In: Rothlauf, F.; Branke, J.; Cagnoni, S.; Corne, D. W.; Drechsler, R. et al., ed. *Applications of Evolutionary Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, s. 545–550. ISBN 978-3-540-32003-6.

[15] Zifčáková, M. Z. *ZÁKLADNÍ HUDEBNÍ POJMY* online. ZŠ Želatovská, únor 2013. Dostupné z: https://www.atlaso.cz/data/uploads/2021/03/ZAKLADNI-HUDEBNI-POJMY.pdf. [cit. 2025-01-10].

[16] Češka, M. a Andriushchenko, R. *Introduction to Markov Models and their Analysis*. Fakulta informačních technologií VUT v Brně, 2024. [cit. 2025-01-10]. Prezentace z předmětu IAM.