# Part 2: Technical dive

- Generative grammars
- Markov chains
- Cellular automata
- Genetic algorithms
- Transformers
- MusicGen

# Part 2: Technical dive

- Generative grammars
- Markov chains
- Cellular automata
- Genetic algorithms
- Transformers
- MusicGen

# 9. Generative grammars

*Generative Music AI*

THE SOUND OF AI

upf. Universitat Pompeu Fabra Barcelona

MTG Music Technology Group

# Overview

- Intuition
- Formalisation
- Probabilistic grammars
- Music generative grammars
- Guidelines
- L-System

# Intuition

Generative grammars are a set of *rules* and *symbols* that systematically describe how strings of a language (or musical elements in the case of music) can be generated.

# Intuition

- Describe a *formal language*

# Intuition

- Describe a *formal language*
- Either *parse* or *generate* sentences for that language

# Intuition

- Describe a *formal language*
- Either *parse* or *generate* sentences for that language
- Rule-based system

# Intuition

- Describe a *formal language*
- Either *parse* or *generate* sentences for that language
- Rule-based system
- Generate ton of strings from a few rules

# Intuition

- Describe a *formal language*
- Either *parse* or *generate* sentences for that language
- Rule-based system
- Generate ton of strings from a few rules
- Come from linguistics

# Formalizing generative grammars

- $G = (N, T, P, S)$

# Formalizing generative grammars

- $G = (N, T, P, S)$
- $N$ is a set of non-terminal symbols (variables)

# Formalizing generative grammars

- $G = (N, T, P, S)$
- $N$ is a set of non-terminal symbols (variables)
- $T$ is a set of terminal symbols (alphabet)

# Formalizing generative grammars

- $G = (N, T, P, S)$
- $N$ is a set of non-terminal symbols (variables)
- $T$ is a set of terminal symbols (alphabet)
- $P$ is a set of production rules

# Formalizing generative grammars

- $G = (N, T, P, S)$
- $N$ is a set of non-terminal symbols (variables)
- $T$ is a set of terminal symbols (alphabet)
- $P$ is a set of production rules
- $S$ is a starting symbol

# Language example

- $T$ = {I, like, apples}

# Language example

- $T$ = {I, like, apples}
- $N$ = {S, PN, V, N}

# Language example

- $T$ = {I, like, apples}
- $N$ = {S, PN, V, N}
- $S$ = S

# Language example

- *T* = {I, like, apples}
- *N* = {S, PN, V, N}
- *S* = S
- *P* = {
  S -> PN V N
  PN -> I
  V -> like
  N -> apples
  }

# Example expansion

1. Start with S

# Example expansion

1. Start with S
2. Apply *S -> PN V N*

# Example expansion

1. Start with S
2. Apply *S* -> *PN V N*
3. Apply *PN -> I*; I V N

# Example expansion

1. Start with S
2. Apply *S -> PN V N*
3. Apply *PN -> I*; I V N
4. Apply *V -> like*; I like N

# Example expansion

1. Start with S
2. Apply *S -> PN V N*
3. Apply *PN -> I*; I V N
4. Apply *V -> like*; I like N
5. Apply *N -> apples*; I like apples

# Example expansion

1. Start with S
2. Apply *S -> PN V N*
3. Apply *PN -> I*; I V N
4. Apply *V -> like*; I like N
5. Apply *N -> apples*; I like apples

Apply production rules until you have all terminal symbols in the generated string

# Language example

- *T* = {I, like, apples}
- *N* = {S, PN, V, N}
- *S* = S
- *P* = {
  S -> PN V N
  PN -> I
  V -> like
  N -> apples
  }

deterministic

# Probabilistic grammar

- *T* = {I, you, like, love, apples}
- *N* = {S, PN, V, N}
- *S* = S
- *P* = {

  S -> PN V N

  PN -> (0.5) I | (0.5) you

  V -> (0.7) like | (0.3) love

  N -> apples

  }

# Probabilistic grammar expansion

1. Start with S

# Probabilistic grammar expansion

1. Start with S
2. Apply *S -> PN V N*

# Probabilistic grammar expansion

1. Start with S
2. Apply *S* -> *PN V N*
3. Apply *PN -> I* | *you* (choose *you*); you V N

# Probabilistic grammar expansion

1. Start with S
2. Apply *S -> PN V N*
3. Apply *PN -> I | you* (choose *you*); you V N
4. Apply *V -> like | love* (choose *love*); you love N

# Probabilistic grammar expansion

1. Start with S
2. Apply *S -> PN V N*
3. Apply *PN -> I | you* (choose *you*); you V N
4. Apply *V -> like | love* (choose *love*); you love N
5. Apply *N -> apples*; you love apples

# Music example

*T* = {C, D, E, F, G, A, B, Whole, Half, Quarter}

*N* = {Melody, Phrase, Pitch, Duration}

*S* = Melody

*P* = {

Melody -> Phrase Phrase

Phrase → Pitch Duration | Pitch Pitch Duration

Pitch -> C | D | E | F | G | A | B

Duration -> Whole | Half | Quarter

}

# Music example

1. Start with *Melody*

# Music example

1. Start with *Melody*
2. Apply *Melody -> Phrase Phrase*

# Music example

1. Start with *Melody*
2. Apply *Melody -> Phrase Phrase*
3. Apply *Phrase -> Pitch Duration | Pitch Pitch Duration*; Pitch Duration Phrase

# Music example

1. Start with *Melody*
2. Apply *Melody -> Phrase Phrase*
3. Apply *Phrase -> Pitch Duration | Pitch Pitch Duration*; Pitch Duration Phrase

# Music example

1. Start with *Melody*
2. Apply *Melody -> Phrase Phrase*
3. Apply *Phrase -> Pitch Duration | Pitch Pitch Duration*; Pitch Duration Phrase
4. Apply *Pitch -> C | D | E | F | G | A | B*; C Duration Phrase

# Music example

1. Start with *Melody*
2. Apply *Melody -> Phrase Phrase*
3. Apply *Phrase -> Pitch Duration* | *Pitch Pitch Duration*; Pitch Duration Phrase
4. Apply *Pitch -> C* | *D* | *E* | *F* | *G* | *A* | *B*; C Duration Phrase

# Music example

1. Start with *Melody*
2. Apply *Melody -> Phrase Phrase*
3. Apply *Phrase -> Pitch Duration* | *Pitch Pitch Duration*; Pitch Duration Phrase
4. Apply *Pitch -> C* | *D* | *E* | *F* | *G* | *A* | *B*; C Duration Phrase
5. Apply *Duration -> Whole* | *Half* | *Quarter*; C Quarter Phrase

# Music example

1. Start with *Melody*
2. Apply *Melody -> Phrase Phrase*
3. Apply *Phrase -> Pitch Duration* | *Pitch Pitch Duration*; Pitch Duration Phrase
4. Apply *Pitch -> C* | *D* | *E* | *F* | *G* | *A* | *B*; C Duration Phrase
5. Apply *Duration -> Whole* | *Half* | *Quarter*; C Quarter Phrase

# Music example

1. Start with *Melody*
2. Apply *Melody -> Phrase Phrase*
3. Apply *Phrase -> Pitch Duration | Pitch Pitch Duration*; Pitch Duration Phrase
4. Apply *Pitch -> C | D | E | F | G | A | B*; C Duration Phrase
5. Apply *Duration -> Whole | Half | Quarter*; C Quarter Phrase
6. Apply *Phrase -> Pitch Duration | Pitch Pitch Duration*; C Quarter Pitch Pitch Duration

# Music example

1. Start with *Melody*
2. Apply *Melody -> Phrase Phrase*
3. Apply *Phrase -> Pitch Duration | Pitch Pitch Duration*; Pitch Duration Phrase
4. Apply *Pitch -> C | D | E | F | G | A | B*; C Duration Phrase
5. Apply *Duration -> Whole | Half | Quarter*; C Quarter Phrase
6. Apply *Phrase -> Pitch Duration | Pitch Pitch Duration*; C Quarter Pitch Pitch Duration

# Music example

1. Start with *Melody*
2. Apply *Melody -> Phrase Phrase*
3. Apply *Phrase -> Pitch Duration* | *Pitch Pitch Duration*; Pitch Duration Phrase
4. Apply *Pitch -> C* | *D* | *E* | *F* | *G* | *A* | *B*; C Duration Phrase
5. Apply *Duration -> Whole* | *Half* | *Quarter*; C Quarter Phrase
6. Apply *Phrase -> Pitch Duration* | *Pitch Pitch Duration*; C Quarter Pitch Pitch Duration
7. Apply *Pitch -> C* | *D* | *E* | *F* | *G* | *A* | *B*; C Quarter E Pitch Duration

# Music example

1. Start with *Melody*
2. Apply *Melody -> Phrase Phrase*
3. Apply *Phrase -> Pitch Duration* | *Pitch Pitch Duration*; Pitch Duration Phrase
4. Apply *Pitch -> C* | *D* | *E* | *F* | *G* | *A* | *B*; C Duration Phrase
5. Apply *Duration -> Whole* | *Half* | *Quarter*; C Quarter Phrase
6. Apply *Phrase -> Pitch Duration* | *Pitch Pitch Duration*; C Quarter Pitch Pitch Duration
7. Apply *Pitch -> C* | *D* | *E* | *F* | *G* | *A* | *B*; C Quarter E Pitch Duration

# Music example

1. Start with *Melody*
2. Apply *Melody -> Phrase Phrase*
3. Apply *Phrase -> Pitch Duration | Pitch Pitch Duration*; Pitch Duration Phrase
4. Apply *Pitch -> C | D | E | F | G | A | B*; C Duration Phrase
5. Apply *Duration -> Whole | Half | Quarter*; C Quarter Phrase
6. Apply *Phrase -> Pitch Duration | Pitch Pitch Duration*; C Quarter Pitch Pitch Duration
7. Apply *Pitch -> C | D | E | F | G | A | B*; C Quarter E Pitch Duration
8. Apply *Pitch -> C | D | E | F | G | A | B*; C Quarter E D Duration

# Music example

1. Start with *Melody*
2. Apply *Melody -> Phrase Phrase*
3. Apply *Phrase -> Pitch Duration* | *Pitch Pitch Duration*; Pitch Duration Phrase
4. Apply *Pitch -> C* | *D* | *E* | *F* | *G* | *A* | *B*; C Duration Phrase
5. Apply *Duration -> Whole* | *Half* | *Quarter*; C Quarter Phrase
6. Apply *Phrase -> Pitch Duration* | *Pitch Pitch Duration*; C Quarter Pitch Pitch Duration
7. Apply *Pitch -> C* | *D* | *E* | *F* | *G* | *A* | *B*; C Quarter E Pitch Duration
8. Apply *Pitch -> C* | *D* | *E* | *F* | *G* | *A* | *B*; C Quarter E D Duration

# Music example

1. Start with *Melody*
2. Apply *Melody -> Phrase Phrase*
3. Apply *Phrase -> Pitch Duration | Pitch Pitch Duration*; Pitch Duration Phrase
4. Apply *Pitch -> C | D | E | F | G | A | B*; C Duration Phrase
5. Apply *Duration -> Whole | Half | Quarter*; C Quarter Phrase
6. Apply *Phrase -> Pitch Duration | Pitch Pitch Duration*; C Quarter Pitch Pitch Duration
7. Apply *Pitch -> C | D | E | F | G | A | B*; C Quarter E Pitch Duration
8. Apply *Pitch -> C | D | E | F | G | A | B*; C Quarter E D Duration
9. Apply *Duration -> Whole | Half | Quarter*; C Quarter E D Whole

# Music example

1. Start with *Melody*
2. Apply *Melody -> Phrase Phrase*
3. Apply *Phrase -> Pitch Duration | Pitch Pitch Duration*; Pitch Duration Phrase
4. Apply *Pitch -> C | D | E | F | G | A | B*; C Duration Phrase
5. Apply *Duration -> Whole | Half | Quarter*; C Quarter Phrase
6. Apply *Phrase -> Pitch Duration | Pitch Pitch Duration*; C Quarter Pitch Pitch Duration
7. Apply *Pitch -> C | D | E | F | G | A | B*; C Quarter E Pitch Duration
8. Apply *Pitch -> C | D | E | F | G | A | B*; C Quarter E D Duration
9. Apply *Duration -> Whole | Half | Quarter*; C Quarter E D Whole

# How do you determine rules?

- Extract manually (music theory)
- Learn from dataset

# Generative tasks

- Melody generation
- Chord progressions
- Music structure
- Full track generation
- …

# Designing a generative grammar

- Finding the correct music representation is key

# Designing a generative grammar

- Finding the correct music representation is key
- What musical dimensions do I want to capture?

# Designing a generative grammar

- Finding the correct music representation is key
- What musical dimensions do I want to capture?
- What do symbols represent?

# Lindenmayer system

- Particular type of generative grammar

# Lindenmayer system

- Particular type of generative grammar
- Apply all production rules at once for each iteration

# Lindenmayer system

- Particular type of generative grammar
- Apply all production rules at once for each iteration
- Generate fractals

# Lindenmayer system

- Particular type of generative grammar
- Apply all production rules at once for each iteration
- Generate fractals
- Describe growth patterns (bacteria, plants)

# L-System: Modelling growth of algae

- *A* (alphabet) = {A, B}
- *S* (axiom) = A
- *P* = {

  A -> AB

  B -> A

  }

# L-System: Modelling growth of algae

n = 0 : A

# L-System: Modelling growth of algae

n = 0 : A

n = 1 : AB

A -> AB

# L-System: Modelling growth of algae

n = 0 : A

n = 1 : AB

n = 2 : ABA

A -> AB
B -> A

# L-System: Modelling growth of algae

n = 0 : A

n = 1 : AB

n = 2 : ABA

n = 3 : ABAAB

A -> AB
B -> A

# L-System: Modelling growth of algae

n = 0 : A

n = 1 : AB

n = 2 : ABA

n = 3 : ABAAB

n = 4 : ABAABABA

# L-System: Modelling growth of algae

n = 0 : A

n = 1 : AB

n = 2 : ABA

n = 3 : ABAAB

n = 4 : ABAABABA

n = 5 : ABAABABAABAAB

# L-System: Modelling growth of algae

n = 0 : A

n = 1 : AB

n = 2 : ABA

n = 3 : ABAAB

n = 4 : ABAABABA

n = 5 : ABAABABAABAAB

n = 6 : ABAABABAABAABABAABABA

# L-System: Modelling growth of algae

n = 0 : A

n = 1 : AB

n = 2 : ABA

n = 3 : ABAAB

n = 4 : ABAABABA

n = 5 : ABAABABAABAAB

n = 6 : ABAABABAABAABABAABABA

n = 7 : ABAABABAABAABABAABABAABAABABAABAAB

# Key takeaways

- Generative grammars use symbols and rules to generate strings

# Key takeaways

- Generative grammars use symbols and rules to generate strings
- Few rules generate a ton of strings

# Key takeaways

- Generative grammars use symbols and rules to generate strings
- Few rules generate a ton of strings
- Used for many generative music tasks

# Key takeaways

- Generative grammars use symbols and rules to generate strings
- Few rules generate a ton of strings
- Used for many generative music tasks
- Probabilistic grammars add randomness to generation

# Key takeaways

- Generative grammars use symbols and rules to generate strings
- Few rules generate a ton of strings
- Used for many generative music tasks
- Probabilistic grammars add randomness to generation
- Rules can be manually encoded or learnt

# Key takeaways

- Generative grammars use symbols and rules to generate strings
- Few rules generate a ton of strings
- Used for many generative music tasks
- Probabilistic grammars add randomness to generation
- Rules can be manually encoded or learnt
- Right mapping between symbols and musical entities is key

# Key takeaways

- Generative grammars use symbols and rules to generate strings
- Few rules generate a ton of strings
- Used for many generative music tasks
- Probabilistic grammars add randomness to generation
- Rules can be manually encoded or learnt
- Right mapping between symbols and musical entities is key
- L-Systems are generative grammars with parallel re-writing rules

# What next?

Implementation of L-System for chord generation