# Music as a Formal Language

## *Finite-State Automata and Pd*

Bryan Jurish

moocow@ling.uni-potsdam.de

Universität Potsdam, Institut für Linguistik,

Potsdam, Germany

# Overview

- The Big Idea: Music as a Formal Language
- A Brief Tour of Formal Language Theory
- Finite-State Machines
- The `pd-gfsm` Pd Externals
- Future Directions
- Concluding Remarks

# The Big Idea

## Music as a formal language

- Syntactic characterization
- Generic musical structure

## Formal languages for music

- Expressive power
- Practical requirements
- Applications

## Open Questions

- Cognitive plausibility ?
- Musical language $\sim$ spoken language ?

# Formal Language Theory

## Alphabets and Strings

- Finite (terminal) Alphabet $\Sigma = \{a_1, \ldots, a_n\}$
- Free Monoid $\langle \Sigma^*, \circ \rangle$ (finite strings over $\Sigma$)

## Formal Languages

- Formal Language $L \subseteq \Sigma^*$

## Grammars and Automata

- Grammar $G = \langle V, \Sigma, P, S \rangle$ *generates* language $L = \mathcal{L}(G)$ by *production rules* $\alpha \to \beta \in P$
- Automaton $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ *recognizes* language $L = \mathcal{L}(M)$ by *transition rules* $\in \delta$

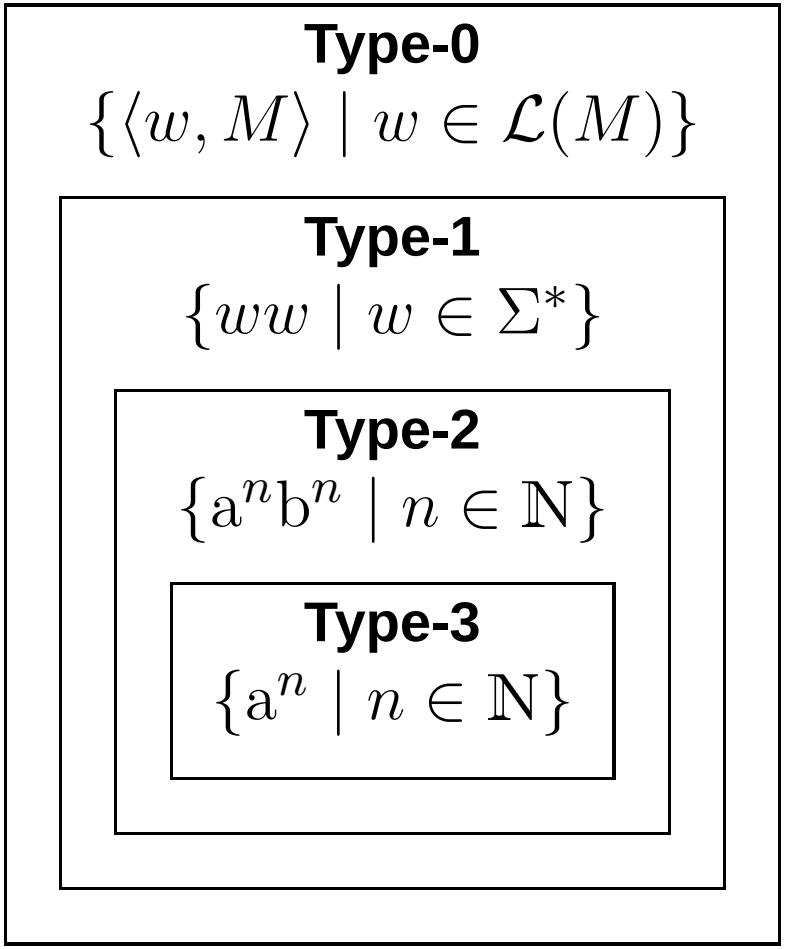
# Chomsky Inclusion Hierarchy

*Turing Machines*

*Linear Bounded Turing Machines*

*Pushdown Automata*

*Finite-State Automata*




*Rule Grammars*

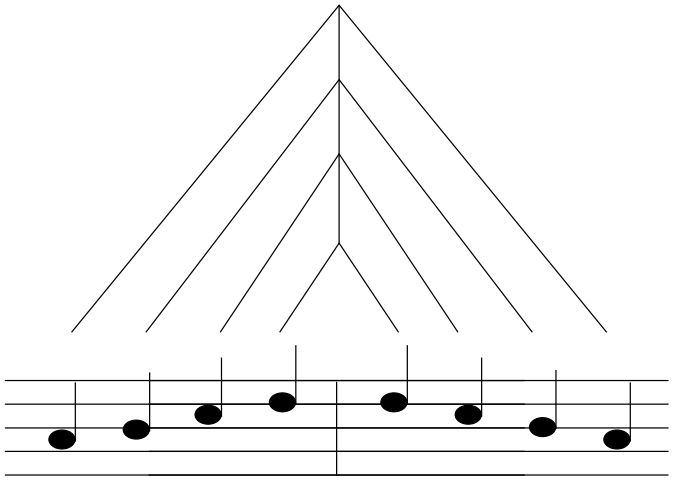
*Context-Sensitive Grammars*

*Context-Free Grammars*

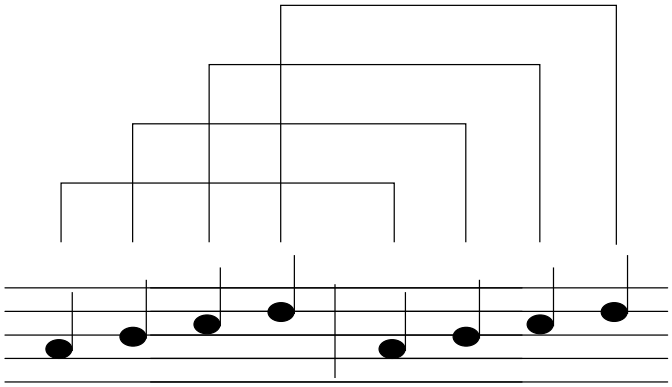
*Regular Grammars*

# Musical Structure

## Tone Dependencies



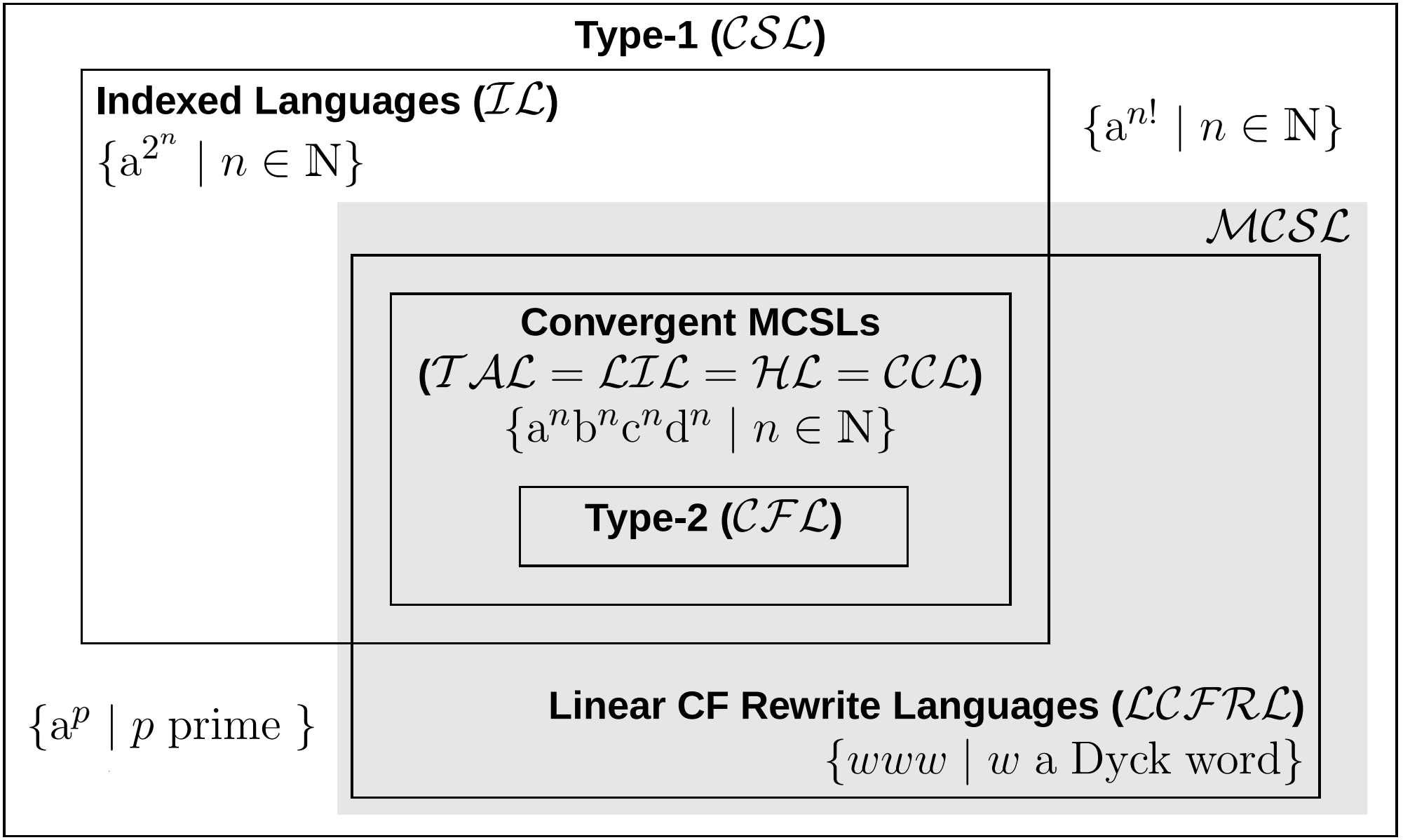
Context-Free



Non-Context-Free

$\Rightarrow$ Musical Languages $\mathcal{ML} \supsetneq \mathcal{CFL}$

# Context-Sensitive Languages



# Practical Considerations

## Improvisation Property

- Given a valid prefix $v \in \Sigma^*$ of a word $vw \in L$, it is decidable in constant time for each $a \in \Sigma$ whether there is a $w' \in \Sigma^*$ such that $vaw' \in L$.
- Amounts to incremental computability.

## Flexibility

- Minimize requirements on what might constitute “musical structure”.

## Mutability

- Enable efficient runtime change of language structure.

# Finite-State Machines

Finite-State Acceptor (FSA) $A = \langle Q, \Sigma, \delta, q_0, F \rangle$

- $Q$ a finite set of states,
- $\Sigma$ a finite terminal alphabet,
- $\delta : Q \times \Sigma \to 2^Q$ a finite transition function,
- $q_0$ a distinguished initial state, and
- $F \subseteq Q$ a set of final states.

$\Rightarrow$ Rooted directed graph with edge labels in $\Sigma$.

Finite-State Transducer (FST) $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$

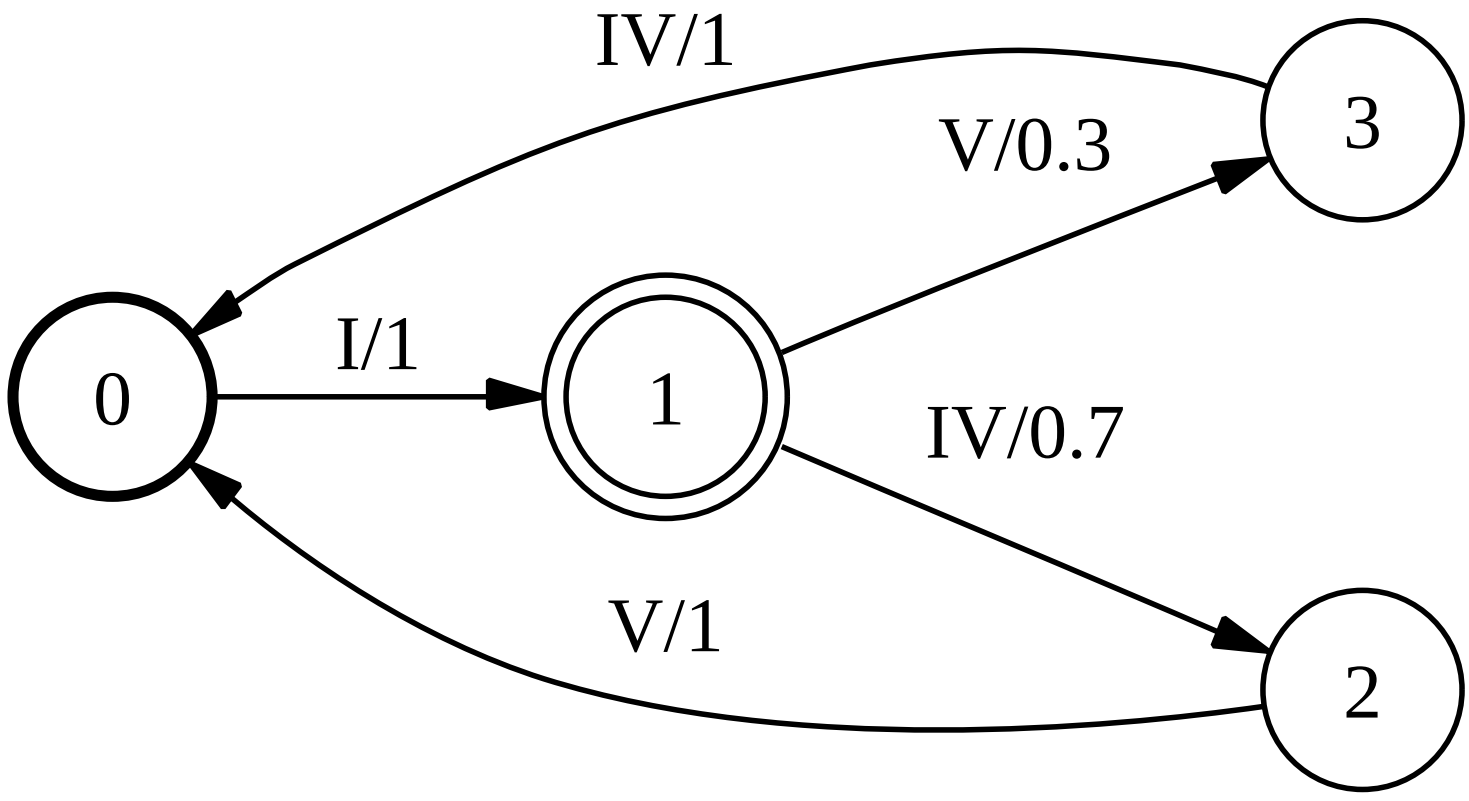
- Basically an FSA with edge labels in $\Sigma \times \Gamma$.

Weighted FSMs

- Extend $\delta$ by marking each arc with a weight $w \in \mathbb{R}$.

# Finite State Machines: Example

A Recognizer for Simple Cadences



# Finite State Machines: Applications

## Markov Chains

- $\equiv$ weighted FSAs
- incremental stochastic sequence generation

## Hidden Markov Models

- $\equiv$ weighted FSTs
- runtime quantization and classification

## Rational Transformations

- control message rewrite rules

## Regular Expressions

- user-interface specifications

# The pd-gfsm Library

`gfsm_alphabet NAME`

- Maps Pd atoms $\leftrightarrow$ integer “labels”
- AT&T text format I/O

`gfsm_automaton NAME`

- Weighted finite-state transducer
- AT&T text and native binary format I/O
- Suitable for large machines ($\geq$ 3 million states)

`gfsm_state NAME STATE_ID`

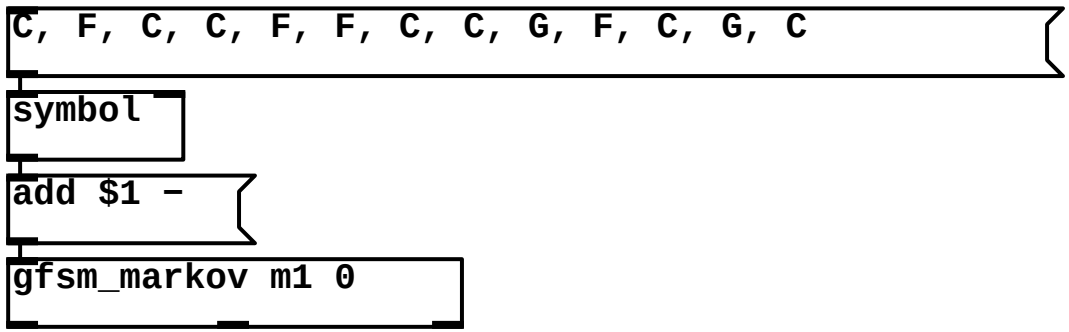
- Holds current state for incremental processing

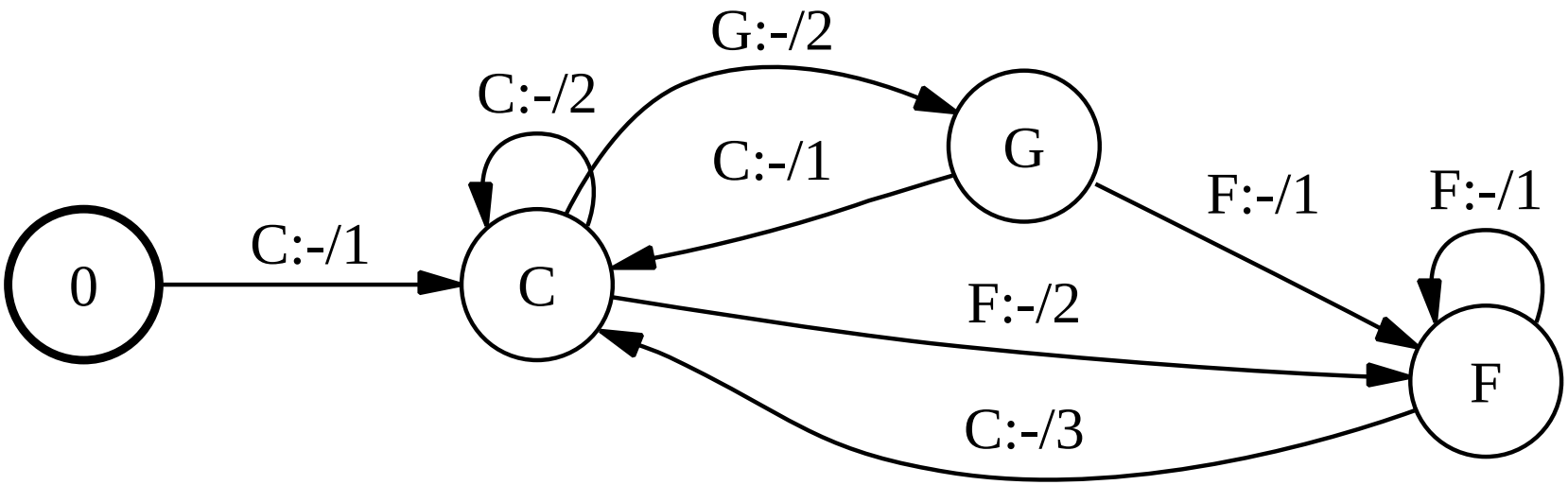
`gfsm_markov NAME`

- Trainable Markov chain patch

# gfsm_markov **Example 1**

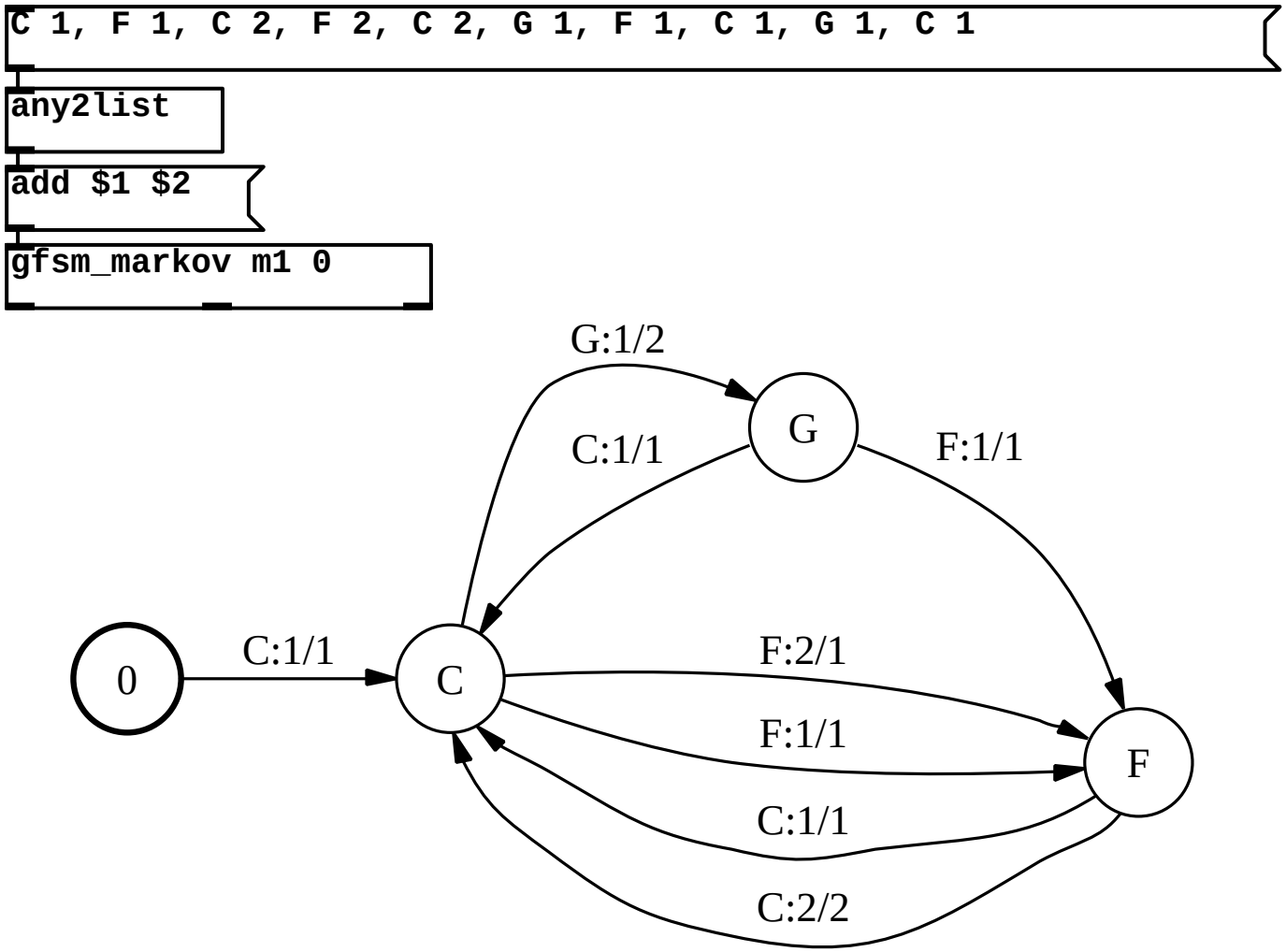
## Simple 12-Bar Blues



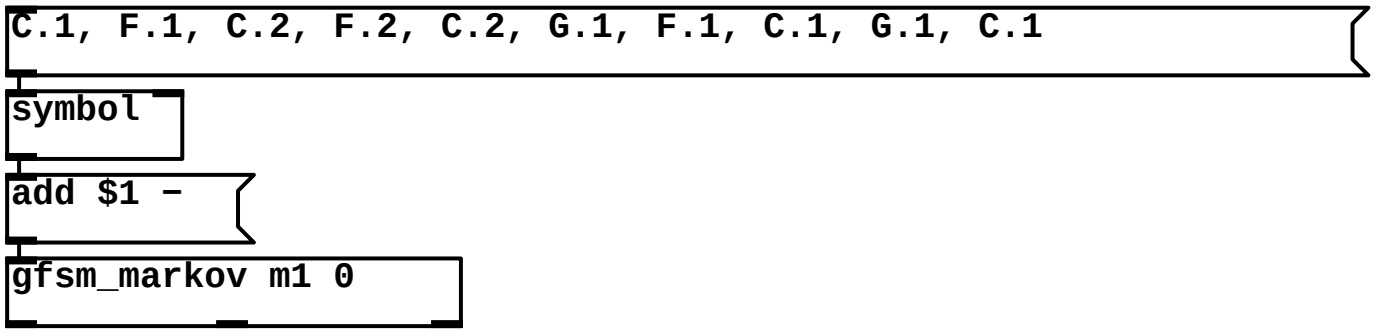
# gfsm_markov **Example 2**

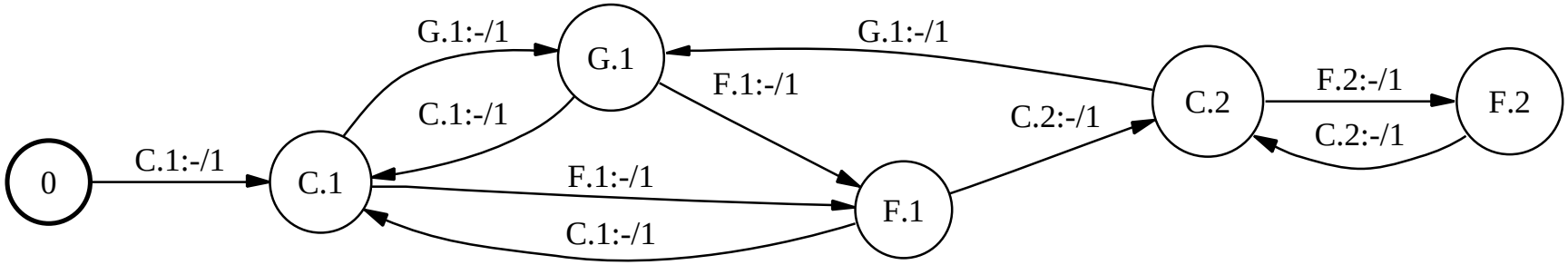
12-Bar Blues with upper duration labels



# gfsm_markov **Example 3**

## 12-Bar Blues with complex state labels



# Future Directions

## FSM Operations

- Algebra: compose, intersect, remove-$\varepsilon$
- Continuous FSMs
- Threading

## Compilers & Interfaces

- Regular Expressions
- 2-Level Rules
- Graphical Editor

## Stochastic Modelling

- Viterbi Algorithm
- Baum-Welch Re-estimation

# Concluding Remarks

## Expressive Power

- $\mathcal{RL} \subsetneq \mathcal{CFL} \subsetneq \mathcal{ML} \subseteq \mathcal{MCSL}$
- $\mathcal{PDL} = \mathcal{TL}$

## Practical Considerations

- Improvisability
- Flexibility
- Mutability

## Embedded Implementation `pd-gfsm`

- FSMs (weak) in Pd (strong)
- Best of both worlds

# The End

# Mild Context-Sensitivity

For every mildly context-sensitive $L \in \mathcal{MCSL}$:

## Contstant Growth

- There is a fixed upper bound $j \in \mathbb{N}$ on the minimum length difference between two words of $L$.
- Intuition: $j \sim$ length of a “measure”.

## Polynomial Parsing Time

- $w \in_? L$ is decidable in $O(|w|^k)$ time for some $k \in \mathbb{N}$.
- Too weak? $\Rightarrow$ *Improvisation property*

## Bounded Cross-Serial Dependencies

- There is an upper bound $m \in \mathbb{N}$ on the number of cross-serial structural dependencies exhibited by words of $L$.
- Too strong? $\Rightarrow$ *Theme and Variation*