Python-Modeling Generalized Trochee in Wergaia

Joseph Lin
Hsinchu County American School, Hsinchu, Taiwan

Emory Linguistics Conference (EULC6), April 18, 2025

Wergaia Stress

1) Words with an even number of syllables:

(ˈLL)
(ˈLH)
(ˈLH)(ˌLH)
(ˈLH)(ˌLL)

2) Odd-parity words with a final light syllable:

(ˈLH)L
(ˈHL)L

3) Odd-parity words with a final heavy syllable:

(ˈLL)(ˌH)
(ˈLH)(ˌH)

Traditional Solutions

1) Parallelism
2) FTBIN » PARSE(σ) ALLFTL » PARSE(σ) » ALLFTR: Pruitt (2010, 2012)

We propose a Directional Harmonic Serialism (DHS) analysis:
1. No over generation problems
2. More Parsimonious (without the need of FTBIN & ALIGN)

Directional Harmonic Serialism

/LLH/ → [('LL)('H)]

/LLH/	TROCHEE⇒	PARSE(σ)⇒	IAMB⇒	*CLASH⇒	PARSE(σ)⇐
☞ a. ('LL)H		001	110		001
b. (L'L)H	W 110	001	L		001
c. L('LH)		W 100	L 010		L 100
d. ('LL)H		W 001	110		W 001
☞ e. ('LL)(ˌH)			110		

(Convergence at 3rd iteration not shown)

Directional Harmonic Serialism

Serial maximal parsing with initial/medial heavy syllable in odd-parity words

/HLLLL/ *1st step*	TROCHEE⇒	PARSE(σ)⇒	IAMB⇒	*CLASH⇒	PARSE(σ)⇐
a. ('H)LLLL		W 01111	L		W 01111
☞ b. ('HL)LLL		00111	01000		00111
/('LL)HLL/ *2nd step*	TROCHEE⇒	PARSE(σ)⇒	IAMB⇒	*CLASH⇒	PARSE(σ)⇐
☞ c. ('LL)('HL)L		00001	11010		00001
d. ('LL)('H)LL		W 00011	L 11000		W 00011

Python Coding Example

1. **Goals**

- Simulate Foot Structure and Stress Assignment with Serial OT
- Output Foot Structure based on Input Given

Python Coding Example

1. **Goals**

- Simulate Foot Structure and Stress Assignment with Serial OT
- Output Foot Structure based on Input Given

2. **Key Components**

- Generator -> a function that generates candidates
- Evaluator
  - Takes in a candidate, output violation vector for each of the constraints
  - Rank violations based on priority

Representing Each Candidate

1. **Encode as a String Representation**

- Ex: “(‘LR)(R’L)L
- Difficult to identify where stresses and foots are.

2. **Encode as a Class Structure**

- A candidate consists of many syllables
- Syllables act as Basic Units
- Encodes information such as weight, hasLeftParenthesis, hasRightParenthesis, hasStress.

Representing Each Candidate

```
4   # Structure representing a syllable with optional stress and parentheses
5   class Syllable:
6       def __init__(self):
7           self.weight = ''  # 'L' = light, 'H' = heavy
8           self.hasLeftParenthesis = False
9           self.hasRightParenthesis = False
10          self.hasStress = False
```

Each candidate is represented by a list of syllables.

```
def Trochee(word: List[Syllable])
```

Generating Possible Candidates

1. According to Serialism

- Generators can only make one “operation at a time”
- I.e. Add a foot (including a stress) somewhere

Therefore, we use a for loop to enumerate possible locations of left and right parenthesis (enclose to make a foot) as well as enumerate positions to add stresses.

Generating Possible Candidates

```
134     for leftParens in range(wordLen):
135         for rightParens in range(leftParens, wordLen):
136             if rightParens - leftParens + 1 > 2:
137                 continue
138 
139             copy = deepcopy(word)
140             if (copy[leftParens].hasLeftParenthesis or copy[leftParens].hasRightParenthesis or
141                 copy[rightParens].hasLeftParenthesis or copy[rightParens].hasRightParenthesis):
142                 continue
143 
144             copy[leftParens].hasLeftParenthesis = True
145             copy[rightParens].hasRightParenthesis = True
146 
147             footSize = rightParens - leftParens + 1
148             for mask in range(1, 1 << footSize):  # skip 0 (no stress)
149                 stressedCopy = deepcopy(copy)
150                 for i in range(footSize):
151                     stressedCopy[leftParens + i].hasStress = (mask & (1 << i)) != 0
152 
153                 candidateScore = []
154                 for constraint in constraints:
155                     violation = constraint(stressedCopy)
156                     candidateScore.append(score(violation))
157 
158                 candidates.append((stressedCopy, candidateScore))
159 
```

Functioning Constraints

1. **Trochee**

Assign one violation for a monomoraic syllable that is i) a foot-initial non-head, i.e., *(L'H), *(L'L), *(L), or ii) a foot-final head, i.e., *(L'L), *('L), *(H'L).

2. **Parse Right**

For Every Syllable that is not footed, assign a violation mark to it.

Same **for Iamb and Parse Left**. Eventually, each constraint will take in a candidate and return a violation vector. (1 represents violation, where 0 represents no violation.)

Functioning Constraints

```
41  def Trochee(word: List[Syllable]) -> List[int]:
42      wordLen = len(word)
43      violation = [0] * wordLen
44      parensLocation = findLocation(word)
45
46      for parens in parensLocation:
47          leftParens, rightParens = parens
48          length = rightParens - leftParens + 1
49
50          if length >= 3:
51              assignViolation(leftParens, rightParens, violation)
52          if length == 2:
53              if word[rightParens].hasStress or not word[leftParens].hasStress:
54                  assignViolation(leftParens, rightParens, violation)
55          if length == 1:
56              if word[leftParens].weight == 'L':
57                  assignViolation(leftParens, rightParens, violation)
58      return violation
```

Ranking Constraints

To rank violation vectors, we encode them into binary numbers.

Ex: 10010 -> 18, 00101 -> 5, 01111 -> 15

```
167     # Sort by lexicographically smallest violation vector
168     candidates.sort(key=lambda x: x[1])
169
170     # Print all candidates and their scores
171     optionNum = 1
172     for candidate in candidates:
173         print(f"Option {optionNum}: {printInfo(candidate[0])} | Scores: ", end="")
174         for i, scoreVal in enumerate(candidate[1]):
175             constraintName = ["Trochee", "ParseLeft", "Iamb", "ParseRight"][i]
176             print(f"{constraintName}={scoreVal}", end=", " if i != len(candidate[1]) - 1 else "")
177         print()
178         optionNum += 1
179
180     print(f"✅ Selected Best Candidate: {printInfo(candidates[0][0])}")
181     print("====================================")
182
183     return candidates[0][0]  # Return best candidate
```

(Line 168) sorts the candidates based on the violation number.

Main Function

Serialism OT Function takes in one candidate, and outputs the optimal candidate afterwards. Each time in the while loop, we compare the initial candidate and the final candidate.

If they are the same -> Serialism has reached its end
Otherwise -> Feed the output as input for the next layer.

Main Function

```
186  if __name__ == "__main__":
187      inputSequence = input()
188      v = parseString(inputSequence)
189
190      # Repeat SerialOT until no improvement occurs
191      while True:
192          ret = SerialOT(v, constraints)
193          if printInfo(ret) == printInfo(v):
194              break
195          else:
196              v = ret
```

Conclusion

This paper provides a python coding example to the previous **directional Harmonic Serialism** evaluation of trochaic languages like Wergaia.

We also provide a flexible framework for adjusting and re-ordering constraints, massively scaling derivation speeds for traditional directional Harmonic Serialism derivations.

Final Code is in Github (both Python and C++ available): https://github.com/JosephtheUnbelievable/WergaiaCode/tree/main

Acknowledgements

I would like to thank Mr. Kuo-Chiao Lin for mentoring this project.

Selected References

1. **Lin**, Joseph. & **Lin**, K.-C. 2025. Serial Directional Evaluation of Generalized Trochee in Wergaia (GLOW47).
2. **Lamont**, A., 2022. A restrictive, parsimonious theory of footing in directional Harmonic Serialism.
3. **Lin**,K.-C. & S.-F.**Wang**. 2024. Serial directional evaluation of rhythmic reversal in Axininca.
4. **Pruitt**, K, 2010. Serialism and locality in constraint-based metrical parsing.