Sequitur

- Nevill-Manning and Witten, 1996.
- Uses a context-free grammar (without recursion) to represent a string.
- The grammar is inferred from the string.
- If there is structure and repetition in the string then the grammar may be very small compared to the original string.
- Clever encoding of the grammar yields impressive compression ratios.
- Compression plus structure!

Context-Free Grammars

- Invented by Chomsky in 1959 to explain the grammar of natural languages.
- Also invented by Backus in 1959 to generate and parse Fortran.
- Example:
  - terminals: b, e
  - non-terminals: S, A
  - Production Rules:
    - S → SA, S → A, A → bSe, A → be
    - S is the start symbol

Context-Free Grammar Example

```
S → SA
S → A
A → bSe
A → be
```

derivation of bbebee

```
Example: b and e matched as parentheses
```

```
hierarchical
```
```
parse tree
```

Arithmetic Expressions

```
S → S + T
T → T * F
F → a
F → (S)
```

derivation of a "(a + a) + a"

```
parse tree
```

Sequitur Principles

- Digram Uniqueness:
  - no pair of adjacent symbols (digram) appears more than once in the grammar.
- Rule Utility:
  - Every production rule is used more than once.
  - These two principles are maintained as an invariant while inferring a grammar for the input string.
Sequitur Example (1)

\[ S \rightarrow b \]

Sequitur Example (2)

\[ S \rightarrow bb \]

Sequitur Example (3)

\[ S \rightarrow bbe \]

Sequitur Example (4)

\[ S \rightarrow bbeb \]

Sequitur Example (5)

\[ S \rightarrow bbbe \]

Sequitur Example (6)

\[ S \rightarrow bAA \]

A \rightarrow be

Enforce digram uniqueness. be occurs twice. Create new rule A \rightarrow be.
Sequitur Example (7)

```
bbbeebbobbebee
S → bAAe
A → be
```

Enforce diagram uniqueness.

Sequitur Example (8)

```
bbbeebbobbebee
S → bAAeb
A → be
```

Sequitur Example (9)

```
bbbeebbobbebee
S → bAAebe
A → be
```

Enforce diagram uniqueness. be occurs twice. Use existing rule A → be.

Sequitur Example (10)

```
bbbeebbobbebee
S → bAAeAeA
A → be
```

Sequitur Example (11)

```
bbbeebbobbebee
S → bAAeAb
A → be
```

Enforce diagram uniqueness. be occurs twice. Use existing rule A → be.

Sequitur Example (12)

```
bbbeebbobbebee
S → bAAeAeA
A → be
```

Enforce diagram uniqueness. be occurs twice. Use existing rule A → be.
**Sequitur Example (13)**

- **Input:** `bbbeebbeebbeee`
- **Rules:**
  - `S → bAAeAA`
  - `A → be`
  - Enforce digram uniqueness
  - `AA` occurs twice.
  - Create new rule `B → AA`.

---

**Sequitur Example (14)**

- **Input:** `bbbeebbeebbeee`
- **Rules:**
  - `S → bBleB`
  - `A → be`
  - `B → AA`.

---

**Sequitur Example (15)**

- **Input:** `bbbeebbeebbeee`
- **Rules:**
  - `S → bBleBb`
  - `A → be`
  - `B → AA`.

---

**Sequitur Example (16)**

- **Input:** `bbbeebbeebbeee`
- **Rules:**
  - `S → bBleBbBb`
  - `A → be`
  - `B → AA`.

---

**Sequitur Example (17)**

- **Input:** `bbbeebbeebbeee`
- **Rules:**
  - `S → bBleBbBb`
  - `A → be`
  - `B → AA`.
  - Enforce digram uniqueness.
  - `be` occurs twice.
  - Use existing rule `A → be`.

---

**Sequitur Example (18)**

- **Input:** `bbbeebbeebbeee`
- **Rules:**
  - `S → bBleBbBbA`
  - `A → be`
  - `B → AA`.
Sequitur Example (19)

- bbeebbeebbeebbeeb e

- S → bBeB: Ab
- A → be
- B → AA

Sequitur Example (20)

- bbeebbeebbeebbeeb e

- S → bBeB: Ab
- A → be
- B → AA

Enforce diagram uniqueness.  be occurs twice.
Use existing rule A → be.

Sequitur Example (21)

- bbeebbeebbeebbeeb e

- S → bBeB: AA
- A → be
- B → AA

Enforce diagram uniqueness.  AA occurs twice.
Use existing rule B → AA.

Sequitur Example (22)

- bbeebbeebbeebbeeb e

- S → bBeB: bB
- A → be
- B → AA

Enforce diagram uniqueness.  bB occurs twice.
Create new rule C → bB.

Sequitur Example (23)

- bbeebbeebbeebbeeb e

- S → CbBC
- A → be
- B → AA
- C → bB

Sequitur Example (24)

- bbeebbeebbeebbeeb e

- S → CbBC: e
- A → be
- B → AA
- C → bB

Enforce diagram uniqueness.  Ce occurs twice.
Create new rule D → Ce.
Sequitur Example (25)

բբեբեբեբեբեբեբե

\[ S \rightarrow DBD \]  Enforce rule utility.
\[ A \rightarrow be \]  C occurs only once.
\[ B \rightarrow AA \]  Remove C \( \rightarrow bB \).
\[ C \rightarrow bb \]  D \( \rightarrow bB \).
\[ D \rightarrow Ce \]  E

Sequitur Example (26)

բբեբեբեբեբեբեբե

\[ S \rightarrow DBD \]  A
\[ A \rightarrow be \]  B
\[ B \rightarrow AA \]  D \( \rightarrow bB \).
\[ C \rightarrow bb \]  E

The Hierarchy

բբեբեբեբեբեբե

\[ S \rightarrow DBD \]  A
\[ A \rightarrow be \]  B \( \rightarrow bB \).
\[ B \rightarrow AA \]  D \( \rightarrow bB \).
\[ C \rightarrow bb \]  E

Is there compression? In this small example, probably not.

Sequitur Algorithm

Input the first symbol s to create the production \( S \rightarrow s \);
repeat
match an existing rule:
\[ A \rightarrow ...XY... \]  \( A \rightarrow ...B... \)
\[ B \rightarrow XY \]  \( B \rightarrow XY \)
create a new rule:
\[ A \rightarrow ...XY... \]  \( A \rightarrow ...C... \)
\[ B \rightarrow ...XY... \]  \( B \rightarrow ...C... \)
remove a rule:
\[ A \rightarrow ...B... \]  \( C \rightarrow XY \)
\[ B \rightarrow ...X_1X_2...X_k \]  \( A \rightarrow ...X_1X_2...X_k \)
input a new symbol:
\[ S \rightarrow X_1...X_k \]  \( S \rightarrow X_1...X_k \)
until no symbols left

Complexity

• The number of non-input sequitur operations applied \( \leq 2n \) where \( n \) is the input length.
• Amortized Complexity Argument
  – Let \( s \) = the sum of the right hand sides of all the production rules. Let \( r \) = the number of rules.
  – We evaluate \( 2s - r \).
  – Initially \( 2s - r = 1 \) because \( s = 1 \) and \( r = 1 \).
  – \( 2s - r \geq 0 \) at all times because each rule has at least 1 symbol on the right hand side.
  – \( 2s - r \) increases by 2 for every input operation.
  – \( 2s - r \) decreases by at least 1 for each non-input sequitur rule applied.

Sequitur Rule Complexity

• Digram Uniqueness - match an existing rule.
\[ A \rightarrow ...XY... \]  \( A \rightarrow ...B... \)  \( s \)  \( t \)  \( 2s - r \)
\[ B \rightarrow XY \]  \( B \rightarrow XY \)  \( -1 \)  \( 0 \)  \( -2 \)

• Digram Uniqueness - create a new rule.
\[ A \rightarrow ...XY... \]  \( A \rightarrow ...C... \)  \( s \)  \( t \)  \( 2s - r \)
\[ B \rightarrow ...XY... \]  \( B \rightarrow ...C... \)  \( 0 \)  \( 1 \)  \( -1 \)
\[ C \rightarrow XY \]  \( C \rightarrow XY \)  \( -1 \)  \( -1 \)

• Rule Utility - Remove a rule.
\[ A \rightarrow ...B... \]  \( A \rightarrow ...X_1X_2...X_k \)  \( s \)  \( t \)  \( 2s - r \)
\[ B \rightarrow ...X_1X_2...X_k \]  \( A \rightarrow ...X_1X_2...X_k \)  \( -1 \)  \( -1 \)  \( -1 \)
Linear Time Algorithm

- There is a data structure to implement all the sequitur operations in constant time.
  - Production rules in an array of doubly linked lists.
  - Each production rule has reference count of the number of times used.
  - Each non-terminal points to its production rule.
  - Digrams stored in a hash table for quick lookup.

Basic Encoding a Grammar

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Symbol Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>S \rightarrow DBD</td>
<td>b 000</td>
</tr>
<tr>
<td>A \rightarrow be</td>
<td>e 001</td>
</tr>
<tr>
<td>B \rightarrow AA</td>
<td>A 010</td>
</tr>
<tr>
<td>D \rightarrow bBe</td>
<td>B 011</td>
</tr>
<tr>
<td>#</td>
<td>D 100</td>
</tr>
</tbody>
</table>

Grammar Code

\[ (\text{Grammar Code}) = (s + r - 1) \lfloor \log_2 (r + a) \rfloor \]

- \( r \) = number of rules
- \( s \) = sum of right hand sides
- \( a \) = number in original symbol alphabet

Better Encoding of the Grammar

- Nevill-Manning and Witten suggest a more efficient encoding of the grammar that uses L77 ideas.
  - Send the right hand side of the S production.
  - The first time a non-terminal is sent, its right hand side is transmitted instead.
  - The second time a non-terminal is sent as a tuple \(<i,j,k>\) which says the right hand side starts occurs in production \(i\), at position \(j\) and is \(k\) long. A new production rule is then added to a dictionary.
  - Subsequently, the non-terminal is represented by the index of the production rule.

Compression Quality

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>comp</th>
<th>gzip</th>
<th>sequitur</th>
<th>PPMC</th>
<th>bzip2</th>
</tr>
</thead>
<tbody>
<tr>
<td>bib</td>
<td>111261</td>
<td>3.35</td>
<td>2.51</td>
<td>2.48</td>
<td>2.12</td>
<td>1.98</td>
</tr>
<tr>
<td>book</td>
<td>788771</td>
<td>3.46</td>
<td>3.36</td>
<td>2.82</td>
<td>2.52</td>
<td>2.42</td>
</tr>
<tr>
<td>geo</td>
<td>102450</td>
<td>6.08</td>
<td>5.34</td>
<td>4.74</td>
<td>5.01</td>
<td>4.45</td>
</tr>
<tr>
<td>sp2</td>
<td>246814</td>
<td>4.17</td>
<td>2.62</td>
<td>2.68</td>
<td>2.77</td>
<td>2.48</td>
</tr>
<tr>
<td>pic</td>
<td>513216</td>
<td>0.97</td>
<td>0.82</td>
<td>0.92</td>
<td>0.96</td>
<td>0.76</td>
</tr>
<tr>
<td>prog</td>
<td>38611</td>
<td>3.87</td>
<td>2.58</td>
<td>2.83</td>
<td>2.49</td>
<td>2.53</td>
</tr>
</tbody>
</table>

Notes on Sequitur

- Very new and different from the standards.
- Yields compression and hierarchical structure simultaneously.
- With clever encoding is competitive with the best of the standards.
- Practical linear time encoding and decoding.
- Alternatives
  - Off-line algorithms – (i) find the most frequent digram, (ii) find the longest repeated substring
Other Grammar Based Methods

- YK Algorithm
  - Kiefer, Yang 2000
  - Like Sequitur, but does not allow different non-terminals to generate the same string
  - Slower, but has some better theoretical properties
- Longest Match
- Most frequent digram
- Match producing the best compression