# CSE 490 G
# Introduction to Data Compression
Winter 2006

Sequitur

## 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

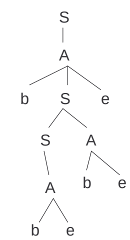
- S → SA
  S → A
  A → bSe
  A → be

Example: b and e matched as parentheses

derivation of bbebee

S
A
bSe
bSAe
bAAe
bbeAe
bbebee

hierarchical parse tree





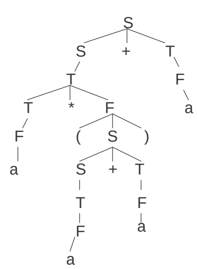

## Arithmetic Expressions

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

derivation of a * (a + a) + a

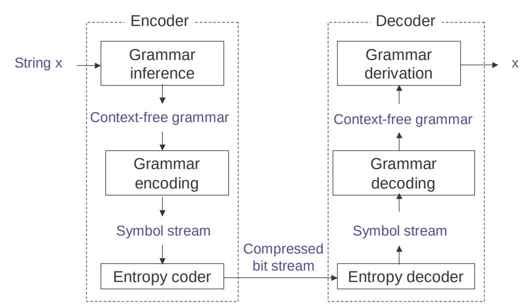
S
S+T
T+T
T*F+T
F*F+T
a*F+T
a*(S)+F
a*(S+F)+T
a*(T+F)+T
a*(F+F)+T
a*(a+F)+T
a*(a+a)+T
a*(a+a)+F
a*(a+a)+a

parse tree






## Overview of Grammar Compression



## 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)

bbebeebebebbebee

S → b



## Sequitur Example (2)

bbebeebebebbebee

S → bb



## Sequitur Example (3)

bbebeebebebbebee

S → bbe



## Sequitur Example (4)

bbebeebebebbebee

S → bbeb



## Sequitur Example (5)

bbebeebebebbebee

S → bbebe

Enforce digram uniqueness.
be occurs twice.
Create new rule A → be.

## Sequitur Example (6)

bbebeebebebbebee

S → bAA
A → be



## Sequitur Example (7)

bbebeebebebbebee

S → bAAe
A → be



## Sequitur Example (8)

bbebeebebebbebee

S → bAAeb
A → be



## Sequitur Example (9)

bbebeebebebbebee

S → bAAebe
A → be

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



## Sequitur Example (10)

bbebeebebebbebee

S → bAAeA
A → be



## Sequitur Example (11)

bbebeebebebbebee

S → bAAeAb
A → be

## Sequitur Example (12)

bbebeebebebbebee

S → bAAeAbe
A → be

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



## Sequitur Example (13)

bbebeebebebbebee

S → bAAeAA
A → be

Enforce digram uniqueness
AA occurs twice.
Create new rule B → AA.



## Sequitur Example (14)

bbebeebebebbebee

S → bBeB
A → be
B → AA



## Sequitur Example (15)

bbebeebebebbebee

S → bBeBb
A → be
B → AA



## Sequitur Example (16)

bbebeebebebbebee

S → bBeBbb
A → be
B → AA



## Sequitur Example (17)

bbebeebebebbebee

S → bBeBbbe
A → be
B → AA

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

## Sequitur Example (18)

bbebeebebebbebee

S → bBeBbA
A → be
B → AA



## Sequitur Example (19)

bbebeebebebbebee

S -> bBeBbAb
A -> be
B -> AA



## Sequitur Example (20)

bbebeebebebbebee

S → bBeBbAbe
A → be
B → AA

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



## Sequitur Example (21)

bbebeebebebbebee

S → bBeBbAA
A → be
B → AA

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



## Sequitur Example (22)

bbebeebebebbebee

S → bBeBbB
A → be
B → AA

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



## Sequitur Example (23)

bbebeebebebbebee

S → CeBC
A → be
B → AA
C → bB

## Sequitur Example (24)

bbebeebebebbebee

S → CeBCe
A → be
B → AA
C → bB

Enforce digram uniqueness.
Ce occurs twice.
Create new rule D → Ce.



## Sequitur Example (25)

bbebeebebebbebee

S → DBD
A → be
B → AA
C → bB
D → Ce

Enforce rule utility.
C occurs only once.
Remove C → bB.



## Sequitur Example (26)

bbebeebebebbebee

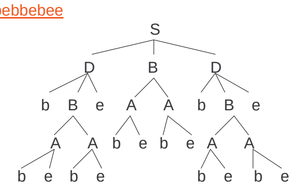
S → DBD
A → be
B → AA
D → bBe



## The Hierarchy

bbebeebebebbebee

S → DBD
A → be
B → AA
D → bBe




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



## Sequitur Algorithm

Input the first symbol s to create the production S → s;
repeat
  match an existing rule:
    A → ....XY...  ⟶  A → ....B....
    B → XY  ⟶  B → XY
  create a new rule:
    A → ....XY....  ⟶  A → ....C....
    B →....XY....  ⟶  B →....C....
    C → XY
  remove a rule:
    A → ....B....
    B → $X_1X_2...X_k$  ⟶  A → .... $X_1X_2...X_k$ ....
  input a new symbol:
    S → $X_1X_2...X_k$  ⟶  S → $X_1X_2...X_k$s
until no symbols left



## Exercise

Use Sequitur to construct a grammar for aaaaaaaaaa = $a^{10}$

## Complexity

- The number of non-input sequitur operations applied < 2n where n is the input length.
- Since each operation takes constant time, sequitur is a linear time algorithm



## Amortized Complexity Argument

- Let m = # of non-input sequitur operations. Let n = input length. Show $m \le 2n$.
- 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 > 0 at all times because each rule has at least 1 symbol on the right hand side.



## Sequitur Rule Complexity

- Digram Uniqueness - match an existing rule.

$A \to \ldots XY\ldots$, $B \to XY$ $\longrightarrow$ $A \to \ldots B\ldots$, $B \to XY$ | s r: -1 0 | 2s - r: -2

- Digram Uniqueness - create a new rule.

$A \to \ldots XY\ldots$, $B \to \ldots XY\ldots$ $\longrightarrow$ $A \to \ldots C\ldots$, $B \to \ldots C\ldots$, $C \to XY$ | s r: 0 1 | 2s - r: -1

- Rule Utility - Remove a rule.

$A \to \ldots B\ldots$, $B \to X_1X_2\ldots X_k$ $\longrightarrow$ $A \to \ldots X_1X_2\ldots X_k \ldots$ | s r: -1 -1 | 2s - r: -1

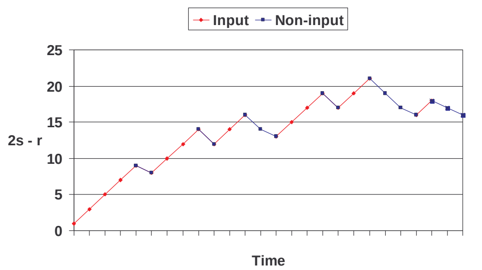


## Amortized Complexity Argument

- $2s - r \ge 0$ 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.
- n = number of input symbols
  m = number of non-input operations
- $2n - m \ge 0$. $m \le 2n$.



## Amortized Complexity Argument



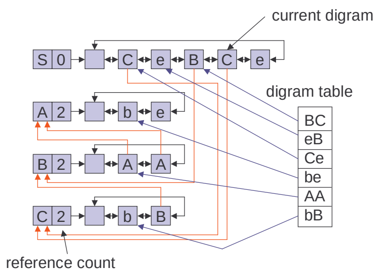



## Linear Time Algorithm

- 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 nonterminal points to its production rule.
  - Digrams stored in a hash table for quick lookup.

## Data Structure Example

S → CeBCe
A → be
B → AA
C → bB






## Basic Encoding a Grammar

Grammar
S → DBD
A → be
B → AA
D → bBe

| Symbol Code | | |
|---|---|---|
| b | 000 | No code for S needed |
| e | 001 | |
| A | 010 | |
| B | 011 | |
| D | 100 | |
| # | 101 | |

Grammar Code

| D | B | D | # | b | e | # | A | A | # | b | B | e | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 011 | 100 | 101 | 000 | 001 | 101 | 010 | 010 | 101 | 000 | 011 | 001 | 39 bits |

|Grammar Code| = $(s+r-1)\lceil \log_2(r+a) \rceil$

$r$ = 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 LZ77 ideas.
  - Send the right hand side of the S production.
  - The first time a nonterminal is sent, its right hand side is transmitted instead.
  - The second time a nonterminal is sent as a triple [*i*, *j*, *d*], which says the right hand side starts at position *j* in production rule *i* and is *d* long. A new production rule is then added to a dictionary.
  - Subsequently, the nonterminal is represented by the index of the production rule.



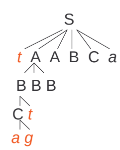
## Transmission Example

S → *t*AABC*a*
A → BBB
B → C*t*
C → *ag*

T = Transmitted

T *tagt*

$X_0$ *tagt* $l_0 = 4$





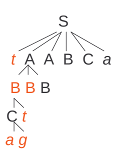

## Transmission Example

S → *t*AABC*a*
A → BBB
B → C*t*
C → *ag*

T = Transmitted

T *tagt* [0, 1, 3]

$X_0$ *t* $X_1X_1$ $l_0 = 3$
$X_1$ *agt* $l_1 = 3$





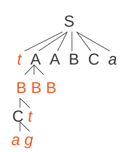

## Transmission Example

S → *t*AABC*a*
A → BBB
B → C*t*
C → *ag*

T = Transmitted

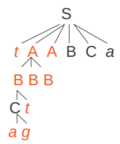
T *tagt* [0, 1, 3] 1

$X_0$ *t* $X_1X_1X_1$ $l_0 = 4$
$X_1$ *agt* $l_1 = 3$



## Transmission Example

$S \rightarrow tAABCa$
$A \rightarrow BBB$
$B \rightarrow Ct$
$C \rightarrow ag$




T = Transmitted

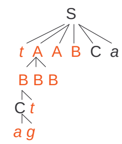
T *tagt* [0, 1, 3] 1 [0, 1, 3]

| | | |
|---|---|---|
| $X_0$ | $t\,X_2X_2$ | $l_0 = 3$ |
| $X_1$ | *agt* | $l_1 = 3$ |
| $X_2$ | $X_1X_1X_1$ | $l_2 = 3$ |



## Transmission Example

$S \rightarrow tAABCa$
$A \rightarrow BBB$
$B \rightarrow Ct$
$C \rightarrow ag$




T = Transmitted

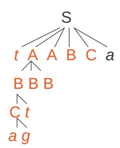
T *tagt* [0, 1, 3] 1 [0, 1, 3] 1

| | | |
|---|---|---|
| $X_0$ | $t\,X_2X_2X_1$ | $l_0 = 4$ |
| $X_1$ | *agt* | $l_1 = 3$ |
| $X_2$ | $X_1X_1X_1$ | $l_2 = 3$ |



## Transmission Example

$S \rightarrow tAABCa$
$A \rightarrow BBB$
$B \rightarrow Ct$
$C \rightarrow ag$




T = Transmitted

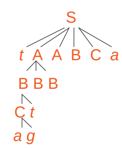
T *tagt* [0, 1, 3] 1 [0, 1, 3] 1 [1, 0, 2]

| | | |
|---|---|---|
| $X_0$ | $t\,X_2X_2X_1\,X_3$ | $l_0 = 5$ |
| $X_1$ | $X_3t$ | $l_1 = 2$ |
| $X_2$ | $X_1X_1X_1$ | $l_2 = 3$ |
| $X_3$ | *ag* | $l_3 = 2$ |



## Transmission Example

$S \rightarrow tAABCa$
$A \rightarrow BBB$
$B \rightarrow Ct$
$C \rightarrow ag$




T = Transmitted

T *tagt* [0, 1, 3] 1 [0, 1, 3] 1 [1, 0, 2] *a*

| | | |
|---|---|---|
| $X_0$ | $t\,X_2X_2X_1\,X_3\,a$ | $l_0 = 6$ |
| $X_1$ | $X_3t$ | $l_1 = 2$ |
| $X_2$ | $X_1X_1X_1$ | $l_2 = 3$ |
| $X_3$ | *ag* | $l_3 = 2$ |



## Kieffer-Yang Improvement

- Kieffer and Yang developed a theoretical framework for studying these types of grammars in 2000.
  - KY is universal; it achieves entropy in the limit
- Add to sequitur Reduction Rule 5:

| | | | |
|---|---|---|---|
| $S \rightarrow AB$ | | $S \rightarrow AA$ | Adding this constraint makes sequitur universal. |
| $A \rightarrow CD$ | | $A \rightarrow CD$ | |
| $B \rightarrow aE$ | $\Rightarrow$ | ~~$B \rightarrow aE$~~ | |
| $C \rightarrow ab$ | | $C \rightarrow ab$ | |
| $D \rightarrow cd$ | | $D \rightarrow cd$ | |
| $E \rightarrow bD$ | | ~~$E \rightarrow bD$~~ | |

<A> = <B> = *abcd*



## Compression Quality

- Neville-Manning and Witten 1997

| | size | comp | gzip | sequitur | PPMC | bzip2 |
|---|---|---|---|---|---|---|
| bib | 111261 | 3.35 | 2.51 | 2.48 | 2.12 | 1.98 |
| book | 768771 | 3.46 | 3.35 | 2.82 | 2.52 | 2.42 |
| geo | 102400 | 6.08 | 5.34 | 4.74 | 5.01 | 4.45 |
| obj2 | 246814 | 4.17 | 2.63 | 2.68 | 2.77 | 2.48 |
| pic | 513216 | 0.97 | 0.82 | 0.90 | 0.98 | 0.78 |
| progc | 38611 | 3.87 | 2.68 | 2.83 | 2.49 | 2.53 |

= First; = Second; = Third.

Files from the Calgary Corpus
Units in bits per character (8 bits)
compress - based on LZW
gzip - based on LZ77
PPMC - adaptive arithmetic coding with context
bzip2 – Burrows-Wheeler block sorting

## Notes on Sequitur

- Yields compression and hierarchical structure simultaneously.
- With clever encoding is competitive with the best of the standards.
- The grammar size is not close to approximation algorithms
  - Upper = $O((n/\log n)^{3/4})$; Lower = $\Omega(n^{1/3})$. (Lehman, 2002)
- *But!* Practical linear time encoding and decoding.



## Other Grammar Based Methods

- Longest Match
- Most frequent digram
- Match producing the best compression