



WIKIPEDIA  
The Free Encyclopedia

[Main page](#)  
[Contents](#)  
[Featured content](#)  
[Current events](#)  
[Random article](#)  
[Donate to Wikipedia](#)  
[Wikipedia store](#)

Interaction

[Help](#)  
[About Wikipedia](#)  
[Community portal](#)  
[Recent changes](#)  
[Contact page](#)

Tools

[What links here](#)  
[Related changes](#)  
[Upload file](#)  
[Special pages](#)  
[Permanent link](#)  
[Page information](#)  
[Wikidata item](#)  
[Cite this page](#)

Print/export

[Create a book](#)  
[Download as PDF](#)  
[Printable version](#)

Languages


[Deutsch](#)  
[Polski](#)  
[Suomi](#)

 [Edit links](#)

[Create account](#) [Log in](#)

Article [Talk](#)

[Read](#) [Edit](#) [View history](#)



# Sequitur algorithm

From Wikipedia, the free encyclopedia  
(Redirected from [SEQUITUR algorithm](#))

**Sequitur** (or *Nevill-Manning algorithm*) is a recursive algorithm developed by [Craig Nevill-Manning](#) and [Ian H. Witten](#) in 1997<sup>[1]</sup> that infers a hierarchical structure ([context-free grammar](#)) from a sequence of discrete symbols. The algorithm operates in linear space and time. It can be used in [data compression](#) software applications.<sup>[2]</sup>

**Contents** [\[hide\]](#)

- [1 Constraints](#)
- [2 Digram Uniqueness](#)
- [3 Rule Utility](#)
- [4 Method summary](#)
- [5 See also](#)
- [6 References](#)
- [7 External links](#)

## Constraints [\[edit\]](#)

Sequitur algorithm constructs grammar by substituting repeating phrases in the given sequence with new rules and therefore produces concise representation of the sequence. For example, if the sequence is  $S \rightarrow \text{ab cab}$ , sequitur algorithm will produce:  $S \rightarrow \mathbf{AcA}$ ,  $A \rightarrow \mathbf{ab}$ . While scanning the input sequence, sequitur algorithm follows two constraints for generating its grammar efficiently: **digram uniqueness** and **rule utility**.

## Digram Uniqueness [\[edit\]](#)

Whenever a new symbol is scanned from the sequence, it is appended with the last scanned symbol to form a new **digram**. If this digram has been formed earlier then a new rule is made to replace both the occurrences of the digrams. Therefore, it ensures that no digram occurs more than once in the grammar. For example, in sequence  $S \rightarrow \mathbf{abaaba}$ , when the first four symbols are already scanned, digrams formed are - **ab**, **ba**, **aa**. When the fifth symbol is read, a new digram 'ab' is formed which exists already. Therefore, both instances of 'ab' are replaced by a new rule (say, A) in S. Now, the grammar becomes  $S \rightarrow \mathbf{AaAa}$ ,  $A \rightarrow \mathbf{ab}$ , and the process continues until no repeated digram exists in the grammar.

## Rule Utility [\[edit\]](#)

This constraint ensures that all the rules are used more than once in right side of all the productions of the grammar, i.e., if a rule occurs just once, it should be removed from the grammar and its occurrence should be substituted with the symbols from which it is created. For example, in the above example, if we scan the last symbol and apply digram uniqueness for 'Aa', then grammar will produce:  $S \rightarrow \mathbf{BB}$ ,  $A \rightarrow \mathbf{ab}$ ,  $B \rightarrow \mathbf{Aa}$ . Now, rule 'A' occurs only once in the grammar in  $B \rightarrow \mathbf{Aa}$ . Therefore, A is deleted and finally grammar becomes:  $S \rightarrow \mathbf{BB}$ ,  $B \rightarrow \mathbf{aba}$ . This constraint helps in removing redundant rules from the grammar.

## Method summary [\[edit\]](#)

The algorithm works by scanning a sequence of **terminal symbols**, building a list of all the symbol pairs which it has read. Whenever a second occurrence of a pair is discovered, the two occurrences are replaced in the sequence by an invented **nonterminal symbol**, the list of symbol pairs is adjusted to match the new sequence, and scanning continues. If a pair's nonterminal symbol is used only in the just created symbol's definition, the used symbol is replaced by its definition and the symbol is removed from the defined nonterminal symbols. Once the scanning has been completed, the transformed sequence can be interpreted as the top-level rule in a grammar for the original sequence. The rule definitions for the nonterminal symbols which it contains can be found in the list of symbol pairs. Those rule definitions may themselves contain additional nonterminal symbols whose rule definitions can also be read from elsewhere in the list of symbol pairs.

## See also [edit]

- [Context-free grammar](#)
- [Straight-line grammar](#)
- [Lossless data compression](#)
- [Data compression](#)

## References [edit]

- ↑ Nevill-Manning, C.G.; Witten, I.H. (1997). "Identifying Hierarchical Structure in Sequences: A linear-time algorithm". *arXiv.cs/9709102*.
- ↑ Nevill-Manning, C.G.; Witten, I.H. (1997). "Linear-Time, Incremental Hierarchy Inference for Compression". *doi:10.1109/DCC.1997.581951*.

## External links [edit]

- [sequitur.info](http://sequitur.info)  - the reference Sequitur algorithm implementation in C++, Java, and other languages.
- [GrammarViz 2.0](#)  - Sequitur and parallel Sequitur implementations in Java, Sequitur-based time series patterns discovery.

Categories: [Lossless compression algorithms](#)

This page was last modified on 9 August 2015, at 07:02.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Mobile view](#)

