Package ‘sumer’

February 18, 2026

Type Package

Title Sumerian Cuneiform Text Analysis

Version 1.1.0

Description Provides functions for converting transliterated Sumerian texts to sign names and cuneiform characters,
creating and querying dictionaries, and analyzing the structure of Sumerian words. Includes a built-in dictionary and supports both forward lookup (Sumerian to English) and reverse lookup (English to Sumerian).

License GPL-3

Encoding UTF-8

Depends R (>= 4.0.0)

Imports stringr, officer, xml2, cli, rlang, ggplot2, ragg

RoxygenNote 7.3.3

Maintainer Robin Wellmann <`ro.wellmann@gmail.com`>

NeedsCompilation no

Author Robin Wellmann [aut, cre]

Repository CRAN

Date/Publication 2026-02-18 13:10:02 UTC

Contents

sumer-package 2
as.cuneiform 5
as.sign_name 6
convert_to_dictionary 8
grammar_probs 10
info 11
look_up 13
make_dictionary 16
mark_ngrams 17

ngram_frequencies 19
plot_sign_grammar 20
prior_probs 22
read_dictionary 23
read_translated_text 24
save_dictionary 26
sign_grammar 28
skeleton 29
split_sumerian 31

Index **33**

`sumer-package` *Tools for Working with Sumerian Cuneiform Texts*

Description

Package sumer provides tools for translating and analyzing transliterated Sumerian cuneiform texts. It converts between transliterations, canonical sign names, and cuneiform Unicode characters, includes a dictionary lookup system for translation work, and offers statistical tools for analyzing the grammatical structure of signs in context.

Getting Started

Load the package, load a dictionary, and look up your first word:

```
library(sumer)

# Load the built-in dictionary
dic <- read_dictionary()

# Look up a Sumerian word
look_up("lugal", dic)

# Search for an English term
look_up("king", dic, "en")
```

Cuneiform Conversion

Sumerian text can be entered as transliteration (e.g. `"lugal"`), as sign names (e.g. `"LUGAL"`), or as cuneiform Unicode characters. The following functions convert between these representations:

`as.cuneiform` Converts transliteration or sign names to cuneiform Unicode characters.

```
as.cuneiform("lugal")
as.cuneiform("d-en-lil2")
```

`as.sign_name` Converts transliteration to canonical sign names.

```
as.sign_name("lugal")
as.sign_name("d-en-lil2")
```

info Shows all available information about a sign or compound: reading, sign name, cuneiform character, and alternative readings.

```
info("lugal")
info("jic-tukul")
```

Dictionary Lookup

The core workflow for translation: load a dictionary and look up words.

look_up Looks up a Sumerian expression in a dictionary. Forward lookup (Sumerian to translation) shows the cuneiform form, sign names, translations with grammatical types, and entries for individual signs and substrings. Reverse lookup searches for a term in the translations.

```
dic <- read_dictionary()

# Forward: Sumerian -> translation
look_up("d-suen", dic)

# Reverse: translation -> Sumerian
look_up("water", dic, "en")
look_up("Gilgamesh", dic, "en")
```

skeleton Generates a hierarchical translation template for a Sumerian sentence. Each word is broken down into syllables and individual signs, ready to be annotated with translations.

```
skeleton("a-ma-ru ba-ur3 ra")
```

Text Analysis

These functions help you analyze the statistical and grammatical structure of a Sumerian text.

N-gram Analysis: ngram_frequencies finds recurring sign combinations in a text.

```
# Use "Enki and the World Order" as an example text
path <- system.file("extdata", "enki_and_the_world_order.txt", package = "sumer")
text <- readLines(path, encoding="UTF-8")
freq <- ngram_frequencies(text, min_freq = 6)
head(freq)
```

mark_ngrams puts all these sign combinations in a text in curly brackets:

```
text_marked <- mark_ngrams(text, freq)
cat(text_marked[1:10], sep="\n")
```

Find all occurences of a pattern in the annotated text:

```
term    <- "IGI.DIB.TU"
(pattern <- mark_ngrams(term, freq))
result  <- text_marked[grepl(pattern, text_marked, fixed=TRUE)]
cat(result, sep="\n")
```

Grammatical Analysis: Each sign in a dictionary can have one or more grammatical types (e.g. S for noun, V for verb, A for attribute, or operators like Sx->V). The following functions analyze how signs are used grammatically.

`sign_grammar` counts how often each grammatical type occurs for each sign in a string, based on dictionary entries:

```
dic <- read_dictionary()
sg  <- sign_grammar("a-ma-ru ba-ur3 ra", dic)
sg
```

For a Bayesian estimate of the statistical distribution of the grammatical types use `prior_probs` and `grammar_probs`:

```
prior <- prior_probs(dic, sentence_prob = 0.25)
gp    <- grammar_probs(sg, prior, dic)
```

`plot_sign_grammar` visualizes the result as a stacked bar chart showing the grammatical type distribution for each sign in a sequence. It accepts output from either `sign_grammar` (raw counts) or `grammar_probs` (probabilities):

```
plot_sign_grammar(gp)
plot_sign_grammar(gp, output_file = "grammar.png")
```

Creating Your Own Dictionary

You can build a dictionary from annotated translation files. These files use a pipe format where each line starts with `|` and contains the sign name, grammatical type, and meaning separated by colons (e.g. `|lugal:S:king`).

`make_dictionary` Reads a translation file and converts it to dictionary format in one step:

```
filename   <- system.file("extdata", "text_with_translations.txt",
                          package = "sumer")
dictionary <- make_dictionary(filename)
```

This is equivalent to calling `read_translated_text` followed by `convert_to_dictionary`.

`save_dictionary` / `read_dictionary` Save a dictionary to a file and load it again later:

```
save_dictionary(dictionary, "my_dictionary.txt",
                author  = "My Name",
                year    = "2025",
                version = "1.0")

my_dic <- read_dictionary("my_dictionary.txt")
look_up("ki", my_dic)
```

Author(s)

Maintainer: Robin Wellmann <`ro.wellmann@gmail.com`>

See Also

Conversion: `as.cuneiform`, `as.sign_name`, `info`, `split_sumerian`

Dictionary lookup: `read_dictionary`, `look_up`, `skeleton`

Text analysis: `ngram_frequencies`, `mark_ngrams`, `sign_grammar`, `prior_probs`, `grammar_probs`, `plot_sign_grammar`

Dictionary creation: `read_translated_text`, `convert_to_dictionary`, `make_dictionary`, `save_dictionary`

`as.cuneiform` *Convert Transliterated Sumerian Text to Cuneiform*

Description

Converts transliterated Sumerian text to Unicode cuneiform characters. This is a generic function with a method for character vectors.

Usage

```
as.cuneiform(x, ...)

## Default S3 method:
as.cuneiform(x, ...)

## S3 method for class 'character'
as.cuneiform(x, mapping = NULL, ...)

## S3 method for class 'cuneiform'
print(x, ...)
```

Arguments

`x`	For `as.cuneiform`: An object to be converted to cuneiform. Currently, only character vectors are supported. For `print.cuneiform`: an object of class `"cuneiform"`.
`mapping`	A data frame containing the sign mapping table with columns `syllables`, `name`, and `cuneiform`. If `NULL` (the default), the package's internal mapping file '`etcsl_mapping.txt`' is loaded. Only used by the character method.
`...`	Additional arguments passed to methods.

Details

The function processes each element of the input character vector by:

1. Calling `info` to look up sign information for each transliterated sign.
2. Extracting the Unicode cuneiform symbols for each sign.
3. Reconstructing the cuneiform text using the original separators, but removing hyphens and periods which are only used in transliteration to indicate sign boundaries.

The default method throws an error for unsupported input types.

Value

`as.cuneiform` returns a character vector of class `cuneiform` with the cuneiform representation of each input element.

`print.cuneiform` displays a character vector of class `cuneiform`.

Note

The cuneiform output requires a font that supports the Unicode Cuneiform block (U+12000 to U+12500) to display correctly.

See Also

`info` for retrieving detailed sign information, `split_sumerian` for splitting Sumerian text into signs, `as.sign_name` for converting transliterated Sumerian text intos sign names

Examples

```
# Convert transliterated text to cuneiform
as.cuneiform(c("na-an-jic li-ic ma","en tarah-an-na-ke4"))

# Load transliterated text from a file
file <- system.file("extdata", "transliterated-text.txt", package = "sumer")
x <- readLines(file)
cat(x, sep="\n")

# Convert transliterated text to cuneiform
as.cuneiform(x)

# Using a custom mapping table
path <- system.file("extdata", "etcsl_mapping.txt", package = "sumer")
my_mapping <- read.csv2(path, sep=";", na.strings="")
as.cuneiform("lugal", mapping = my_mapping)
```

`as.sign_name` *Convert Transliterated Sumerian Text to Sign Names*

Description

Converts transliterated Sumerian text to canonical sign names in uppercase notation. This is a generic function with a method for character vectors.

Usage

```
as.sign_name(x, ...)

## Default S3 method:
as.sign_name(x, ...)
```

```
## S3 method for class 'character'
as.sign_name(x, mapping = NULL, ...)

## S3 method for class 'sign_name'
print(x, ...)
```

Arguments

`x`	For `as.sign_name`: An object to be converted to sign names. Currently, only character vectors are supported. For `print.sign_name`: An object of class `"sign_name"`.
`mapping`	A data frame containing the sign mapping table with columns `syllables`, `name`, and `cuneiform`. If `NULL` (the default), the package's internal mapping file '`etcsl_mapping.txt`' is loaded. Only used by the character method.
`...`	Additional arguments passed to methods.

Details

The function processes each element of the input character vector by:

1. Calling `info` to look up sign information for each transliterated sign.
2. Extracting the canonical sign names for each sign.
3. Reconstructing the text using the original separators, but replacing hyphens with periods to follow standard sign name notation.

The default method throws an error for unsupported input types.

Value

`as.sign_name` returns a character vector of class `c("sign_name", "character")` with the sign name representation of each input element.

`print.sign_name` displays a character vector of class `"sign_name"`.

See Also

`as.cuneiform` for converting to cuneiform characters, `info` for retrieving detailed sign information, `split_sumerian` for splitting Sumerian text into signs

Examples

```
# Convert transliterated text to sign names
as.sign_name(c("lugal-e", "an-ki"))

# Load transliterated text from a file
file <- system.file("extdata", "transliterated-text.txt", package = "sumer")
x <- readLines(file)
cat(x, sep="\n")
```

```
# Convert transliterated text to sign names
as.sign_name(x)

# Using a custom mapping table
path <- system.file("extdata", "etcsl_mapping.txt", package = "sumer")
my_mapping <- read.csv2(path, sep=";", na.strings="")
as.sign_name("lugal", mapping = my_mapping)
```

`convert_to_dictionary` *Convert Translation Data to a Sumerian Dictionary*

Description

Converts a data frame of Sumerian translations into a structured dictionary format, adding cuneiform representations and phonetic readings for each sign.

Usage

```
convert_to_dictionary(df, mapping = NULL)
```

Arguments

`df`	A data frame with columns `sign_name`, `type`, and `meaning`, typically produced by `read_translated_text`.
`mapping`	A data frame containing sign-to-reading mappings with columns `name`, `cuneiform` and `syllables`. If `NULL` (default), the package's built-in mapping file `etcsl_mapping.txt` is used.

Details

Processing Steps:

1. Aggregates translations and counts occurrences of each unique combination in `df`
2. Looks up phonetic readings and cuneiform signs for each sign component
3. Combines cuneiform, reading, and translation rows into a single data frame
4. Sorts the result by sign name and row type

Reading Format: Phonetic readings are formatted as follows:

- Multiple possible readings are enclosed in braces: `{a, dur5, duru5}`
- For compound signs, readings of individual components are joined with hyphens
- If a sign has more than three possible readings in a compound, only the first three are shown followed by `...`
- Unknown readings are marked with ?

Value

A data frame with the following columns:

sign_name The normalized Sumerian text (e.g., "A", "AN", "A2.TAB")

row_type Type of entry: "cunei." (cuneiform character), "reading" (phonetic readings), or "trans." (translation)

count Number of occurrences for translations; NA for cuneiform and reading entries

type Grammatical type (e.g., "S", "V", "A") for translations; empty string for other row types

meaning The cuneiform character(s), phonetic reading(s), or translated meaning depending on row_type

The data frame is sorted by sign_name, row_type, and descending count.

See Also

read_translated_text for reading translation files, make_dictionary for creating a complete dictionary with cuneiform representations and readings in a single step.

Examples

```
# Read translations from a single text document
filename     <- system.file("extdata", "text_with_translations.txt", package = "sumer")
translations <- read_translated_text(filename)

# View the structure
head(translations)

#Make some custom unifications (here: removing the word "the")
translations$meaning <- gsub("\\bthe\\b", "", translations$meaning, ignore.case = TRUE)
translations$meaning <- trimws(gsub("\\s+", " ", translations$meaning))

# View the structure
head(translations)

#Convert the result into a dictionary
dictionary   <- convert_to_dictionary(translations)

# View the structure
head(dictionary)

# View entries for a specific sign
dictionary[dictionary$sign_name == "EN", ]

# With custom mapping
path  <- system.file("extdata", "etcsl_mapping.txt", package = "sumer")
mapping <- read.csv2(path, sep=";", na.strings="")
translations <- read_translated_text(filename, mapping = mapping)
dictionary <- convert_to_dictionary(translations, mapping = mapping)
head(dictionary)
```

grammar_probs   *Posterior Probabilities of Grammatical Types for Each Sign*

Description

For each cuneiform sign in a sentence, computes Bayesian posterior probabilities for all grammatical types, combining prior beliefs from `prior_probs` with observed dictionary frequencies. The dictionary counts are corrected for verb underrepresentation using the `sentence_prob` stored in the prior.

Usage

```
grammar_probs(sg, prior, dic, alpha0 = 1)
```

Arguments

`sg`	A data frame as returned by `sign_grammar`.
`prior`	A named numeric vector as returned by `prior_probs`, with a `sentence_prob` attribute.
`dic`	A dictionary data frame as returned by `read_dictionary`.
`alpha0`	Numeric (>= 0). Strength of the prior (pseudo sample size). Larger values pull the posterior towards the prior. When `alpha0 = 0`, the result is purely data-driven. Default: 1.

Details

For each sign at position i in the sentence, the function computes:

1. The raw dictionary counts n_k for each grammar type k.
2. A correction factor $x_k = 1/\text{sentence\_prob}$ for verb-like types, $x_k = 1$ otherwise. The corrected counts are $m_k = n_k \cdot x_k$ with total $M = \sum_k m_k$.
3. The posterior probability (Dirichlet-Multinomial model):

$$\theta_k = \frac{\alpha_0\, p_k + m_k}{\alpha_0 + M}$$

   where p_k is the prior probability from `prior_probs()`.

For signs not in the dictionary ($M = 0$), the posterior equals the prior. For signs with many observations ($M \gg \alpha_0$), the posterior is dominated by the data.

Value

A data frame with columns:

position Integer. Position of the sign in the sentence.

sign_name Character. The sign name.

cuneiform Character. The cuneiform character.

type Character. The grammar type (e.g., `"S"`, `"V"`, `"Sx->S"`).

prob Numeric. Posterior probability for this type at this position.

n Numeric. Number of counts in the dictionary.

See Also

`prior_probs` for computing the prior, `sign_grammar` for the input data, `plot_sign_grammar` for visualisation.

Examples

```
dic   <- read_dictionary()
sg    <- sign_grammar("a-ma-ru ba-ur3 ra", dic)
prior <- prior_probs(dic, sentence_prob = 0.25)
gp    <- grammar_probs(sg, prior, dic, alpha0 = 1)
print(gp)
```

`info` *Retrieve Information About Sumerian Signs*

Description

Analyzes a transliterated Sumerian text string and retrieves detailed information about each sign, including syllabic readings, sign names, cuneiform symbols, and alternative readings.

The function `info` computes the result and returns an object of class `"info"`. The `print` method displays a summary of different text representations in the console.

Usage

```
info(x, mapping = NULL)

## S3 method for class 'info'
print(x, flatten = FALSE, ...)
```

Arguments

`x`	For `info`: a character string of length 1 containing transliterated Sumerian text. For `print.info`: an object of class `"info"`.
`mapping`	A data frame containing the sign mapping table with columns `syllables`, `name`, and `cuneiform`. If `NULL` (the default), the package's internal mapping file '`etcsl_mapping.txt`' is loaded.
`flatten`	Logical. If `TRUE`, grammar indicators in the text are removed (such as parentheses, brackets, braces, and operators). If `FALSE` (the default), the original separators are preserved.
`...`	Additional arguments passed to the print method (currently unused).

Details

The function `info` performs the following steps:

1. Splits the input string into signs and separators using `split_sumerian`.
2. Standardizes the signs.
3. Looks up each sign in the mapping table based on its type:
   - Type 1 (lowercase): Searches for a matching syllable reading.
   - Type 2 (uppercase): Searches for a matching sign name.
   - Type 3 (cuneiform): Searches for a matching cuneiform character.
4. Returns a data frame with the results, along with the separators stored as an attribute.

The mapping table must contain the following columns:

syllables Comma-separated list of possible syllabic readings for the sign. The first reading is used as the default.

name The canonical sign name in uppercase.

cuneiform The Unicode cuneiform character.

The `print` method displays each sign with its name and alternative readings, followed by three text representations: syllables, sign names, and cuneiform text.

Value

`info` returns a data frame of class `c("info", "data.frame")` with one row per sign and the following columns:

`reading`	The syllabic reading of the sign. For lowercase input, this is the standardized input; for other types, this is the default syllable from the mapping.
`sign`	The Unicode cuneiform character corresponding to the sign.
`name`	The canonical sign name in uppercase.
`alternatives`	A comma-separated string of all possible syllabic readings for the sign.

The data frame has an attribute `"separators"` containing the separator characters between signs.

`print.info` prints the following to the console and returns `x` invisibly:

Sign table Each sign with its cuneiform symbol, name, and alternative readings.

syllables The text with syllabic readings, using hyphens as separators within words.

sign names The text with sign names, using periods as separators within words.

cuneiform text The text rendered in Unicode cuneiform characters, with hyphens and periods removed.

Note

If no custom mapping is provided, the function loads the internal mapping file included with the **sumer** package.

See Also

`split_sumerian` for splitting Sumerian text into signs,

Examples

```
library(stringr)

# Basic usage - compute and print
info("lugal-e")

# Store the result for further processing
result <- info("an-ki")
result

# Access the underlying data frame
result$sign
result$name

# Print with and without flattened separators
result <- info("(an)na")
print(result)
print(result, flatten = TRUE)

# Using a custom mapping table
path <- system.file("extdata", "etcsl_mapping.txt", package = "sumer")
my_mapping <- read.csv2(path, sep=";", na.strings="")
info("an-ki", mapping = my_mapping)
```

`look_up` *Look Up Sumerian Signs or Search for Translations*

Description

Searches a Sumerian dictionary either by sign name (forward lookup) or by translation text (reverse lookup).

The function `look_up` computes the search results and returns an object of class `"look_up"`. The `print` method displays formatted results with cuneiform representations, grammatical types, and translation counts.

Usage

```
look_up(x, dic, lang = "sumer", width = 70)

## S3 method for class 'look_up'
print(x, ...)
```

Arguments

`x`	For `look_up`: A character string specifying the search term. Can be either: • A Sumerian sign name (e.g., `"AN"`, `"AN.EN.ZU"`) • A cuneiform character string • A word or phrase to search in translations (e.g., `"Gilgamesh"`, `"heaven"`) For `print.look_up`: An object of class `"look_up"` as returned by `look_up`.
`dic`	A dictionary data frame, typically created by `make_dictionary` or loaded with `read_dictionary`. Must contain columns `sign_name`, `row_type`, `count`, `type`, and `meaning`.
`lang`	Character string specifying whether `x` is a Sumerian expression (`"sumer"`) or an English expression (`"en"`).
`width`	Integer specifying the text width for line wrapping. Default is 70.
`...`	Additional arguments passed to the print method (currently unused).

Details

Search Modes: The function operates in two modes depending on the input:

Forward Lookup (Sumerian input detected):

1. Converts the sign name to cuneiform
2. Retrieves all translations for the exact sign combination
3. Retrieves translations for all individual signs and substrings

Reverse Lookup (non-Sumerian input):

1. Searches for the term in all translation meanings
2. Retrieves matching entries with sign names and cuneiform

Output Format: The print method displays results with:

- Sign names with cuneiform representations
- Occurrence counts in brackets (e.g., `[29]`)
- Grammatical type abbreviations (e.g., `S`, `V`)
- Translation meanings with automatic line wrapping
- Search term highlighting in blue for reverse lookups (only for ANSI-compatible terminals)

Value

`look_up` returns an object of class `"look_up"`, which is a list containing:

`search`	The original search term.

`lang`	The language setting used for the search.
`width`	The text width for formatting.
`cuneiform`	The cuneiform representation (only for Sumerian searches).
`sign_name`	The canonical sign name (only for Sumerian searches).
`translations`	A data frame with translations for the exact sign combination (only for Sumerian searches).
`substrings`	A named list of data frames with translations for individual signs and substrings (only for Sumerian searches).
`matches`	A data frame with matching entries (only for non-Sumerian searches).

`print.look_up` prints formatted dictionary entries to the console and returns `x` invisibly.

See Also

`read_dictionary` for loading dictionaries, `make_dictionary` for creating dictionaries, `as.cuneiform` for cuneiform conversion.

Examples

```
# Load dictionary
dic <- read_dictionary()

# Forward lookup: search by phonetic spelling
look_up("d-suen", dic)

# Forward lookup: search by Sumerian sign name
look_up("AN", dic)
look_up("AN.EN.ZU", dic)

# Forward lookup: search by cuneiform character string
AN.NA <- paste0(intToUtf8(0x1202D), intToUtf8(0x1223E))
AN.NA
look_up(AN.NA, dic)

# Reverse lookup: search in translations
look_up("Gilgamesh", dic, "en")

# Adjust output width for narrow terminals
look_up("water", dic, "en", width = 50)

# Store results for later use
result <- look_up("lugal", dic)
result$cuneiform
result$translations

# Print stored results
print(result)
```

make_dictionary *Create a Sumerian Dictionary from Annotated Text Files*

Description

Parses Word documents (.docx) or plain text files containing annotated Sumerian translations and creates a structured dictionary data frame. The function extracts sign names, their cuneiform representations, possible readings, and translations with grammatical types.

Usage

```
make_dictionary(file, mapping = NULL)
```

Arguments

`file`	A character vector of file paths to .docx or text files. Files must contain translation lines that are formatted as described below.
`mapping`	A data frame containing sign-to-reading mappings with columns `name`, `cuneiform` and `syllables`. If `NULL` (default), the package's built-in mapping file `etcsl_mapping.txt` is used.

Details

Input Format: The input files must contain lines starting with `|` in the following format:

`|sign_name: TYPE: meaning`

or

`|equation for sign_name: TYPE: meaning`

For example:

```
|a2-tab: S: the double amount of work performance
|me=ME: S: divine force
|AN: S: god of heaven
|na=NA: Sx->A: whose existence is bound to S
```

Lines not starting with `|` are ignored. Only the first entry in an equation of sign names is used for the dictionary. The following notation is suggested for grammatical types:

- `S` for substantives and noun phrases, (e.g., "the old man in the temple")
- `V` for verbs and decorated verbs (e.g., "to go", "to bring the delivery into the temple")
- `A` for adjectives, attributes and subordinate clauses that further define the subject (e.g., "who/which is weak", "whose resource for sustaining life is grain")
- `Sx->A` for a symbol that transforms the preceding noun phrase into an attribute (e.g., "whose resource for sustaining life is `S`"). Other transformations are denoted accordingly.
- `N` for numbers,
- `D` for everything else.

Processing Steps:

1. Extracts text from .docx files or reads plain text
2. Filters lines starting with `|`
3. Normalizes sign names and looks up possible readings from the mapping table
4. Aggregates translations and counts occurrences

Output Structure: For each unique sign, the output contains:

- One `cunei.` row with the cuneiform character(s)
- One `reading` row with possible phonetic readings
- One or more `trans.` rows with translations, sorted by frequency

Value

A data frame with the following columns:

sign_name The normalized Sumerian sign name (e.g., "A", "AN", "ME")

line_type Type of entry: `"cunei."` (cuneiform), `"reading"` (phonetic readings), or `"trans."` (translation)

count Number of occurrences for translations; `NA` for cuneiform and reading entries

type Grammatical type (e.g., "S", "V", "Sx->A") for translations; empty for other line types

meaning The cuneiform character(s), reading(s), or translated meaning depending on line_type

See Also

`as.cuneiform`, `split_sumerian`

Examples

```
# Create a dictionary from a single text document
filename  <- system.file("extdata", "text_with_translations.txt", package = "sumer")
dict <- make_dictionary(filename)

# Use the dictionary
look_up("an", dict)
```

`mark_ngrams` *Mark N-gram Combinations in Cuneiform Text*

Description

Takes a character vector of Sumerian text and marks all n-gram combinations (from `ngram_frequencies`) with curly braces. Longer combinations are marked first, shorter ones afterwards (including inside already-marked regions).

Usage

```
mark_ngrams(x, ngram)
```

Arguments

`x`	A character vector of Sumerian text (transliteration, sign names, or cuneiform). Will be converted to cuneiform internally.
`ngram`	A data frame as returned by `ngram_frequencies`, with at least columns `combination` and `length`.

Details

The function first converts `x` to cuneiform (if not already) and removes spaces and brackets `()[]{}`.

Then it sorts `ngram` descending by `length` and replaces each occurrence of a combination with `{combination}` (space, open brace, combination, close brace, space).

Shorter n-grams may be marked inside already-marked longer n-grams (nesting is allowed).

Value

A character vector of cuneiform text with n-gram combinations enclosed in curly braces and surrounded by spaces.

See Also

`ngram_frequencies`

Examples

```
# Load the example text of "Enki and the World Order"
path  <- system.file("extdata", "enki_and_the_world_order.txt", package = "sumer")
text <- readLines(path, encoding="UTF-8")
cat(text[1:10],sep="\n")

# Find combinations that appear at least 6 times in the text
freq <- ngram_frequencies(text, min_freq = 6)
freq[1:10,]

# Mark these combinations in the text
text_marked <- mark_ngrams(text, freq)
cat(text_marked[1:10], sep="\n")

# You can enter transliterated text
x <- "kij2-sig unu2 gal d-re-e-ne-ka me-te-ac im-mi-ib-jal2"
mark_ngrams(x, freq)

# Find all occurences of a pattern in the annotated text
term     <- "IGI.DIB.TU"
(pattern <- mark_ngrams(term, freq))
result   <- text_marked[grepl(pattern, text_marked, fixed=TRUE)]
cat(result, sep="\n")
```

`ngram_frequencies` *Frequency Analysis of Cuneiform Sign Combinations (N-grams)*

Description

Analyzes a Sumerian text for frequently occurring cuneiform sign combinations (n-grams). The input can be either cuneiform text or transliterated text (which is automatically converted to cuneiform via `as.cuneiform`). The analysis starts with the longest combinations and works down to single signs, masking already-counted occurrences to avoid reporting subsequences that are only frequent because they are part of a longer frequent combination. N-grams are searched within lines only (not across line boundaries).

Usage

```
ngram_frequencies(x, min_freq = c(6, 4, 2))
```

Arguments

`x`	Character vector whose elements are the lines of a Sumerian text. The input can be either cuneiform characters or transliterated text. If no cuneiform characters (U+12000 to U+1254F) are detected, the input is automatically converted using `as.cuneiform`. Lines starting with # are treated as comments and ignored. Optional line numbers at the beginning of a line (e.g., `"42)\t"`) are automatically removed. Spaces are removed before tokenization.
`min_freq`	Integer vector specifying minimum frequencies (default: `c(6, 4, 2)`). The i-th value specifies the minimum frequency for combinations of length i. For lengths beyond the vector's length, the last value is used. The default `c(6, 4, 2)` means: single signs must occur at least 6 times, pairs at least 4 times, and all longer combinations at least 2 times.

Details

A “sign” is defined as either a single cuneiform Unicode character (U+12000 to U+1254F) or a character sequence enclosed in mathematical angle brackets (U+27E8 ... U+27E9), which is treated as a single token. All other characters (spaces, X, numbers, punctuation, etc.) are skipped during tokenization.

The maximum n-gram length is automatically determined as the length of the longest tokenized line in the input.

The analysis proceeds from the longest combinations down to single signs. When a combination is identified as frequent (i.e., meets the minimum frequency threshold), all occurrences except the first are masked before continuing with shorter combinations. This prevents subsequences from being reported as frequent when their frequency is solely due to a longer frequent combination.

Value

A data frame with four columns, sorted by descending length, then descending frequency:

`frequency`	Integer. The number of occurrences of the combination.
`length`	Integer. The number of signs in the combination.
`sign_names`	Character. The sign names of the combination (e.g., `"AN.EN.KI"`).
`combination`	Character. The cuneiform sign combination (e.g., `"\U0001202D\U00012097\U000120A0"`).

See Also

`as.sign_name` for converting cuneiform to sign names, `as.cuneiform` for converting transliterations to cuneiform, `split_sumerian` for tokenizing transliterated text.

Examples

```
# Read the text "Enki and the World Order"

path  <- system.file("extdata", "enki_and_the_world_order.txt", package = "sumer")
text <- readLines(path, encoding="UTF-8")

cat(text[1:10],sep="\n")

# Find combinations that appear at least 6 times in the text
freq <- ngram_frequencies(text, min_freq = 6)

freq[1:10,]
```

`plot_sign_grammar` *Stacked Bar Chart of Grammatical Type Frequencies*

Description

Creates a stacked bar chart from the output of `sign_grammar` or `grammar_probs`. Each bar represents one sign position in the sentence. The colours indicate the relative frequency or posterior probability of each individual grammatical type.

Usage

```
plot_sign_grammar(sg,
                  output_file = NULL,
                  width       = 10,
                  height      = 5,
                  sign_names  = FALSE,
                  font_family = NULL)
```

Arguments

`sg`	A data frame as returned by `sign_grammar` (with column `n`) or `grammar_probs` (with column `prob`).
`output_file`	Character. File path for saving the plot (PNG or JPG). If `NULL` (default), the plot is displayed on the current device.
`width`	Numeric. Plot width in inches. Default: 10.
`height`	Numeric. Plot height in inches. Default: 5.
`sign_names`	Logical. Whether sign names or cuneiform characters should be used as labels of the x-axis. Default: FALSE.
`font_family`	Character. Font family for cuneiform x-axis labels. If `NULL` (default), a cuneiform-capable font is detected automatically.

Details

When the input comes from `sign_grammar()` (column `n`), absolute frequencies are converted to percentages so that bars sum to 100%. When the input comes from `grammar_probs()` (column `prob`), posterior probabilities are used directly.

Colours are assigned per grammatical type, grouped by class:

- Red shades: Verbs (`V`) and operators returning verbs
- Blue shades: Operators returning attributes `A`
- Orange: Adjectives and other signs with grammatical type (`Sx->S`)
- Green: Nouns
- Grey/other shades: All other types

Value

Invisibly returns the **ggplot2** plot object.

See Also

`sign_grammar` for generating raw frequency data, `grammar_probs` for Bayesian posterior probabilities, `prior_probs` for computing the prior.

Examples

```
dic   <- read_dictionary()
sg    <- sign_grammar("a-ma-ru ba-ur3 ra", dic)

# Plot raw frequencies
file <- file.path(tempdir(), "test.png")
plot_sign_grammar(sg, file)

# Plot probabilities
prior <- prior_probs(dic, sentence_prob = 0.25)
gp    <- grammar_probs(sg, prior, dic, alpha0 = 1)
file  <- file.path(tempdir(), "test2.png")
plot_sign_grammar(gp, file)
```

prior_probs *Prior Probabilities of Grammatical Types*

Description

Computes prior probabilities for each grammatical type (e.g., `S`, `V`, `Sx->S`, `xS->A`, etc.) from a dictionary. The priors can be corrected for verb underrepresentation in the dictionary data.

Usage

```
prior_probs(dic, sentence_prob = 1.0)
```

Arguments

`dic`	A dictionary data frame as returned by `read_dictionary`.
`sentence_prob`	Numeric in (0, 1]. The estimated proportion of complete sentences (as opposed to noun phrases) in the training data from which the dictionary was created. Verbs appear in complete sentences, so a value less than 1 upweights verb-like types. Default: 1.0.

Details

The function proceeds in three steps:

1. For each single-sign dictionary entry with at least one count, the counts per grammatical type are normalised to sum to 1.
2. The prior probability of each type is the mean of these normalised frequencies across all signs.
3. A correction is applied: counts of verb-like types (`V` and all operators with return type `V`, such as `Vx->V` or `xV->V`) are multiplied by `1/sentence_prob`, then all probabilities are renormalised. This compensates for the fact that verbs are underrepresented when most dictionary entries are obtained from noun phrases rather than complete sentences.

When `sentence_prob` = `1`, no correction is applied.

Value

A named numeric vector with one element per grammatical type found in the dictionary, summing to 1. The names are the type strings as they appear in the dictionary (e.g., `"S"`, `"V"`, `"Sx->S"`). The `sentence_prob` parameter is stored as an attribute.

See Also

`sign_grammar` for per-sign grammatical type frequencies.

Examples

```
dic   <- read_dictionary()

# Default usage
prior_probs(dic)

# Applying correction (only 25% sentences in training data)
prior_probs(dic, sentence_prob = 0.25)
```

`read_dictionary` *Read a Sumerian Dictionary from File*

Description

Reads a Sumerian dictionary from a semicolon-separated text file, optionally displaying the metadata header with author, version, and update information.

Usage

```
read_dictionary(file = NULL, verbose = TRUE)
```

Arguments

`file`	A character string specifying the path to the dictionary file. If `NULL` (default), the package's built-in dictionary `sumer-dictionary.txt` is loaded.
`verbose`	Logical. If `TRUE` (default), the metadata header (author, year, version, URL) is printed to the console.

Details

File Format: The function expects a semicolon-separated file with a metadata header. Lines starting with # are treated as comments. The expected format is:

```
###---------------------------------------------------------------
###                Sumerian Dictionary
###
### Author:  Robin Wellmann
### Year:    2026
### Version: 0.5
### Watch for Updates:
###   https://founder-hypothesis.com/en/sumerian-mythology/downloads/
###---------------------------------------------------------------
sign_name;row_type;count;type;meaning
A;cunei.;;;<here would be the cuneiform sign for A>
A;reading;;;{a, dur5, duru5}
A;trans.;3;S;water
```

Encoding: The file is read with UTF-8 encoding to properly handle cuneiform characters.

Value

A data frame with the following columns:

sign_name The Sumerian sign name (e.g., "A", "AN", "ME")

row_type Type of entry: "cunei." (cuneiform character), "reading" (phonetic readings), or "trans." (translation)

count Number of occurrences for translations; NA for cuneiform and reading entries

type Grammatical type (e.g., "S", "V") for translations; empty string for other row types

meaning The cuneiform character(s), phonetic reading(s), or translated meaning depending on row_type

See Also

save_dictionary for saving dictionaries to file, make_dictionary and convert_to_dictionary for creating dictionaries.

Examples

```
# Load the built-in dictionary
dic <- read_dictionary()

# Load a custom dictionary
filename <- system.file("extdata", "sumer-dictionary.txt", package = "sumer")
dic <- read_dictionary(filename)

# Look up an entry
look_up("d-suen", dic)
```

---

read_translated_text *Read Annotated Sumerian Translations from Text Files*

Description

Reads Word documents (.docx) or plain text files containing annotated Sumerian translations and extracts sign names, grammatical types, and meanings into a structured data frame.

Usage

```
read_translated_text(file, mapping=NULL)
```

Arguments

file	A character vector of file paths to .docx or text files. Files must contain translation lines that are formatted as described below.
mapping	A data frame containing sign-to-reading mappings with columns name, cuneiform and syllables. If NULL (default), the package's built-in mapping file etcsl_mapping.txt is used.

Details

Input Format: The input files must contain lines starting with `|` in the following format:

`|sign_name: TYPE: meaning`

or

`|equation for sign_name: TYPE: meaning`

For example:

```
|a2-tab: S: the double amount of work performance
|me=ME: S: divine force
|AN: S: god of heaven
|na=NA: Sx->A: whose existence is bound to S
```

Lines not starting with `|` are ignored. Only the first entry in an equation of sign names is extracted. The following notation is suggested for grammatical types:

- `S` for substantives and noun phrases, (e.g., "the old man in the temple")
- `V` for verbs and decorated verbs (e.g., "to go", "to bring the delivery into the temple")
- `A` for adjectives, attributes and subordinate clauses that further define the subject (e.g., "who/which is weak", "whose resource for sustaining life is grain")
- `Sx->A` for a symbol that transforms the preceding noun phrase into an attribute (e.g., "whose resource for sustaining life is `S`"). Other transformations are denoted accordingly.
- `N` for numbers,
- `D` for everything else.

Processing Steps:

1. Reads text from .docx files or plain text files
2. Filters lines starting with `|`
3. Parses each line into sign name, type, and meaning components
4. Normalizes transliterated text by removing separators and looking up the sign names from the `mapping`
5. Cleans meaning field by removing content after `;` or `|` delimiters
6. Issues a warning for entries with missing type annotations
7. Excludes empty sign names from the result

Value

A data frame with the following columns:

sign_name The normalized sign name with components separated by hyphens (e.g., `"A"`, `"AN"`, `"X-NA"`)

type Grammatical type (e.g., `"S"`, `"V"`, `"A"`, `"Sx->A"`)

meaning The translated meaning of the sign

Note

If any translations have missing type annotations, the function prints a warning message listing the affected entries.

See Also

convert_to_dictionary for converting the result into a dictionary, make_dictionary for creating a complete dictionary with cuneiform representations and readings in a single step.

Examples

```
# Read translations from a single text document
filename     <- system.file("extdata", "text_with_translations.txt", package = "sumer")
translations <- read_translated_text(filename)

# View the structure
head(translations)

# Filter by grammatical type
nouns <- translations[translations$type == "S", ]
nouns

#Make some custom unifications (here: removing the word "the")
translations$meaning <- gsub("\\bthe\\b", "", translations$meaning, ignore.case = TRUE)
translations$meaning <- trimws(gsub("\\s+", " ", translations$meaning))

# View the structure
head(translations)

#Convert the result into a dictionary
dictionary   <- convert_to_dictionary(translations)

# View the structure
head(dictionary)
```

---

save_dictionary *Save a Sumerian Dictionary to File*

Description

Saves a Sumerian dictionary data frame to a semicolon-separated text file with a metadata header containing author, year, version, and URL information.

Usage

```
save_dictionary(dic, file, author = "", year = "", version = "", url = "")
```

Arguments

`dic`	A dictionary data frame, typically created by make_dictionary or convert_to_dictionary. Must contain columns `sign_name`, `row_type`, `count`, `type`, and `meaning`.
`file`	A character string specifying the output file path.

`author`	A character string with the author name(s) for the metadata header.
`year`	A character string with the year of creation for the metadata header.
`version`	A character string with the version number for the metadata header.
`url`	A character string with a URL where updates can be found.

Details

Output Format: The output file consists of two parts:

1. A metadata header with lines starting with ###, containing author, year, version, and URL information
2. The dictionary data in semicolon-separated format with columns: `sign_name`, `row_type`, `count`, `type`, `meaning`

Example output:

```
###----------------------------------------------------------------
###                Sumerian Dictionary
###
### Author:  Robin Wellmann
### Year:    2026
### Version: 1.0
### Watch for Updates: https://founder-hypothesis.com/sumer/
###----------------------------------------------------------------
sign_name;row_type;count;type;meaning
A;cunei.;;;<cuneiform sign for A>
A;reading;;;{a, dur5, duru5}
A;trans.;3;S;water
```

Value

No return value. The function is called for its side effect of writing the dictionary to a file.

See Also

`make_dictionary` and `convert_to_dictionary` for creating dictionaries, `read_dictionary` for reading saved dictionaries.

Examples

```
# Create and save a dictionary

filename  <- system.file("extdata", "text_with_translations.txt", package = "sumer")
dictionary <- make_dictionary(filename)

save_dictionary(
  dic     = dictionary,
  file    = file.path(tempdir(), "sumerian_dictionary.txt"),
  author  = "John Doe",
  year    = "2026",
  version = "1.0",
  url     = "https://example.com/dictionary"
)
```

---

sign_grammar *Grammatical Type Frequencies for Each Sign in a Sumerian Sentence*

Description

For each cuneiform sign in a Sumerian sentence, looks up the dictionary to determine the frequency of each individual grammatical type (e.g., S, V, Sx->S, xS->A). Returns a data frame with one row per sign per grammatical type.

Usage

```
sign_grammar(x, dic)
```

Arguments

x	A single character string containing a Sumerian sentence (cuneiform, sign names, or transliteration).
dic	A dictionary data frame as returned by read_dictionary.

Details

The function converts the input to cuneiform, splits it into individual signs, and looks up each sign in the dictionary. For each sign, the translations are grouped by their individual type string (e.g., "S", "V", "Sx->S", "xS->A").

For each type the dictionary count values are summed. If a translation entry has no count, it is treated as 1.

The set of types returned is the union of all types found across all signs in the sentence. Each sign gets one row per type, even if the count is 0 for that type.

Value

A data frame with columns:

position Integer. Position of the sign in the sentence.

sign_name Character. The sign name (e.g., "KA").

cuneiform Character. The cuneiform character.

type Character. The grammar type string (e.g., "S", "V", "Sx->S").

n Integer. Sum of dictionary counts for this sign and this type.

See Also

grammar_probs for Bayesian posterior probabilities, plot_sign_grammar for visualising the result, read_dictionary for loading a dictionary, as.cuneiform for cuneiform conversion.

Examples

```
dic <- read_dictionary()

# Analyse a sentence
sg <- sign_grammar("a-ma-ru ba-ur3 ra", dic)
print(sg)

# Use with cuneiform input
x<-"\U00012000\U000121AD"
print(x)
sg <- sign_grammar(x, dic)
print(sg)
```

`skeleton` *Create a Translation Template for Sumerian Text*

Description

Creates a structured template (skeleton) for translating Sumerian text. The template displays each word and syllable with its sign name and cuneiform representation, providing a framework for adding translations.

The function `skeleton` computes the template and returns an object of class `"skeleton"`. The `print` method displays the template in the console.

Usage

```
skeleton(x, mapping = NULL)

## S3 method for class 'skeleton'
print(x, ...)
```

Arguments

`x`	For `skeleton`: A character string of length 1 containing transliterated Sumerian text. Words are separated by spaces, syllables within words by hyphens or other separators. For `print.skeleton`: An object of class `"skeleton"` as returned by `skeleton`.
`mapping`	A data frame containing the sign mapping table with columns `syllables`, `name`, and `cuneiform`. If `NULL` (the default), the package's internal mapping file '`etcsl_mapping.txt`' is loaded.
`...`	Additional arguments passed to the print method (currently unused).

Details

The function generates a hierarchical template with different levels of detail depending on the input type:

Multiple words The template includes a header line with the original text, followed by entries for each word, its syllables (indented with one tab), and sub-signs for multi-sign syllables (indented with two tabs).

Single word (multiple syllables) The word equation serves as the header, followed by syllable entries (one tab) and sub-sign entries (two tabs). No redundant header line is generated.

Single syllable Only the syllable equation is shown (no indentation), with sub-sign entries indented by one tab if applicable.

Each line in the template follows the format

`|[tabs]reading=SIGN.NAME=cuneiform::`

The template should be filled in as follows:

- Between the two colons: the grammatical type of the expression (e.g., `S` for noun phrases, `V` for verbs, etc.). See `make_dictionary` for details.
- After the second colon: the translation

For example, a filled-in line might look like:

`|an=AN=<cuneiform sign for AN>:S: god of heaven`

Redundant lines are automatically omitted: if a word consists of only one syllable, no separate syllable line is generated.

This function is intended to be used together with `look_up` for translating Sumerian texts: first create a template with `skeleton`, then use `look_up` to find the meanings of words and signs, and fill in the template accordingly.

The template format is designed to be saved as a text file (.txt) or Word document (.docx), filled in manually, and can then be used as input for `make_dictionary` to create a custom dictionary.

Value

`skeleton` returns a character vector of class `c("skeleton", "character")` containing the template lines.

`print.skeleton` prints the template to the console and returns `x` invisibly.

See Also

`look_up` for looking up translations of Sumerian signs and words, `make_dictionary` for creating a dictionary from filled-in templates, `info` for retrieving detailed sign information

Examples

```
# Create a template for a multi-word phrase
skeleton("e-ta-na an-ce3 ba-ed3-de3")

# Create a template for a single word
skeleton("lugal-e")

# Create a template for a single syllable
skeleton("an")

# Store the template for further use
tmpl <- skeleton("lu2 du")
tmpl

# Typical workflow: create template, then look up meanings
dic <- read_dictionary()
tmpl <- skeleton("lugal kur-ra-ke4")
print(tmpl)
look_up("lugal", dic)
look_up("kur", dic)
```

`split_sumerian` *Split a String into Sumerian Signs and Separators*

Description

Splits a transliterated Sumerian text string into its constituent signs and the separators between them. The function recognizes three types of Sumerian sign representations: lowercase transliterations, uppercase sign names, and Unicode cuneiform characters.

Usage

```
split_sumerian(x)
```

Arguments

`x`	A character string containing transliterated Sumerian text.

Details

The function identifies Sumerian signs based on three patterns:

1. **Lowercase transliterations** (type 1): Sequences of lowercase letters (a-z) including special characters (ĝ, š, ...) and accented vowels (á, é, í, ú, à, è, ì, ù), optionally followed by a numeric index.
2. **Uppercase sign names** (type 2): Sequences starting with an uppercase letter, optionally followed by additional uppercase letters, digits, or the characters `+`, `/`, and `×`.

3. **Cuneiform characters** (type 3): Unicode characters in the Cuneiform block (U+12000 to U+12500).

The function returns the signs and separators in a format that allows exact reconstruction of the original string using `paste0(c("", signs), separators, collapse = "")`.

Value

A list with three components:

`signs`	A character vector containing the extracted Sumerian signs.
`separators`	A character vector of length `length(signs) + 1` containing the separators. The first element contains any text before the first sign, subsequent elements contain text between consecutive signs, and the last element contains any text after the final sign. Empty strings indicate no separator at that position.
`types`	An integer vector of the same length as `signs` indicating the type of each sign: `1` for lowercase transliterations, `2` for uppercase sign names, and `3` for cuneiform characters.

Examples

```
# Example 1

set.seed(4)

x <- "en-tarah-an-na-ke4"

result <- split_sumerian(x)

result

# Example 2

x <- "en-DARA3.AN.na-ke4"

result <- split_sumerian(x)

result

# Reconstruct the original string
paste0(c("", result$signs), result$separators, collapse = "")
```

Index

∗ **character**
    as.cuneiform, 5
    as.sign_name, 6
    info, 11
    skeleton, 29
    split_sumerian, 31
∗ **database**
    look_up, 13
∗ **hplot**
    plot_sign_grammar, 20
∗ **methods**
    as.cuneiform, 5
    as.sign_name, 6
∗ **univar**
    ngram_frequencies, 19
∗ **utilities**
    as.cuneiform, 5
    as.sign_name, 6
    grammar_probs, 10
    info, 11
    look_up, 13
    ngram_frequencies, 19
    prior_probs, 22
    sign_grammar, 28
    skeleton, 29
    split_sumerian, 31

as.cuneiform, *2*, *5*, 5, *7*, *15*, *17*, *19*, *20*, *28*
as.sign_name, *2*, *5*, *6*, 6, *20*

convert_to_dictionary, *4*, *5*, 8, *24*, *26*, *27*

grammar_probs, *4*, *5*, 10, *20*, *21*, *28*

info, *3*, *5–7*, 11, *30*

look_up, *3*, *5*, 13, *30*

make_dictionary, *4*, *5*, *9*, *14*, *15*, 16, *24*, *26*, *27*, *30*
mark_ngrams, *3*, *5*, 17

ngram_frequencies, *3*, *5*, *17*, *18*, 19

plot_sign_grammar, *4*, *5*, *11*, 20, *28*
print.cuneiform *(as.cuneiform)*, 5
print.info *(info)*, 11
print.look_up *(look_up)*, 13
print.sign_name *(as.sign_name)*, 6
print.skeleton *(skeleton)*, 29
prior_probs, *4*, *5*, *10*, *11*, *21*, 22

read_dictionary, *4*, *5*, *10*, *14*, *15*, *22*, 23, *27*, *28*
read_translated_text, *4*, *5*, *8*, *9*, 24

save_dictionary, *4*, *5*, *24*, 26
sign_grammar, *4*, *5*, *10*, *11*, *20–22*, 28
skeleton, *3*, *5*, 29
split_sumerian, *5–7*, *12*, *13*, *17*, *20*, 31
sumer *(sumer-package)*, 2
sumer-package, 2