Package 'sumer'

October 20, 2026

Type Package

Title Sumerian Cuneiform Text Analysis

Version 1.7.0

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

Author Robin Wellmann [aut, cre]

License GPL-3

Encoding UTF-8

Depends R (>= 4.0.0)

Imports stringr, officer, xml2, cli, rlang, ggplot2, ragg, shiny, utils, stats, tools, grDevices

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation yes

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

Repository CRAN

Date/Publication 2026-10-20 21:10:02 UTC

Contents

apply_translation_rules 2
as.cuneiform 3
as.sign_name 5
classify 6
convert_to_dictionary 9
fill_substr_info 11
grammar_probs 13

grammatical_structure 14
guess_substr_info 16
info 18
look_up 20
make_dictionary 23
mark_ngrams 25
merge_dictionaries 26
ngram_frequencies 28
plot_sign_grammar 29
prior_probs 31
read_dictionary 32
read_translated_text 34
save_dictionary 36
sign_grammar 37
skeleton 39
split_sumerian 41
translate 42
translate_line 45

Index **48**

`apply_translation_rules`

Apply Translation Rules to a Bracketed Structure String

Description

Translates a bracketed structure string into English by evaluating sumerian operators (substituting arguments into translation templates) and composing adjacent elements according to grammatical rules. The input is a structure string as produced by `add_brackets`, together with vectors of types and translations for each tag.

Usage

```
apply_translation_rules(s, type, translation)
```

Arguments

`s`	Character string showing the order of evaluation, as produced by `add_brackets`.
`type`	Character vector of grammatical types. `type[k]` is the type string for tag `#k` (e.g. `"S"`, `"V"`, `"Sx->V"`).
`translation`	Character vector of translations. `translation[k]` is the translation for tag `#k`. Operator translations contain placeholders (e.g. `"to deliver S"`).

Details

Nested `{...}` groups are evaluated from the inside out. Within each group, an operator (if present) binds its arguments and produces a typed result. Groups without an operator are composed according to the rules described in `eval_operator`.

Value

A character vector of length 2: `c(result_type, result_translation)`.

On error (e.g. incompatible types), a character vector of length 1 containing the error message.

See Also

`eval_operator` for the evaluation rules, `add_brackets` for the previous pipeline step, `compose_skeleton_entry` which calls this function

Examples

```
x <- "mec3-ki-aj2-ga-ce-er"
x <- as.cuneiform(x)
x

meaning <- rbind( c("S",      "a man who relies on his own strength"),
                  c("S",      "place {earth}"),
                  c("Sx->A",  ", whose allocated resource is S"),
                  c("xS->A",   ", whose sustenance is S"),
                  c("S",      "grain"),
                  c("Sx->S",  "lamented S"))

df <- data.frame(
    type  = meaning[,1],
    translation = meaning[,2],
    expr  = split_sumerian(x)$signs)

s <- x
for(i in 1:nrow(df)){
 s <- sub(df$expr[i], paste0("#", i), s)
}
s

s_bracketed <- sumer:::add_brackets(s, df$type)
s_bracketed

apply_translation_rules(s_bracketed$string, df$type, df$translation)
```

`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. Hyphens and periods between signs are removed since cuneiform signs are written without separators. Hyphens before numbers are replaced by spaces so that numeric tokens remain distinguishable from adjacent signs.

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 to 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)
```

---

classify *Classify Dictionary Readings into Grammatical Equivalence Classes*

Description

For each sumerian token of a string, `classify` looks up the token's readings in the dictionary and assigns each reading a grammatical equivalence class (`0S0`, `0S1`, `1S0`, ...). Depending on `Format` it returns either the dictionary subset with an added `class` column, or a frequency table (counts or percentages) with one column per token.

Usage

```
classify(x, dic, Format = c("percent", "count", "dictionary"),
       header = c("cuneiform", "sign_name"), sign_grammar = NULL, mapping = NULL)
```

Arguments

x	A single character string of sumerian signs (cuneiform or sign names). Each sumerian token becomes one column. Signs enclosed in angle brackets <...> are treated as a single token (e.g. "<U.U.U>"); unbalanced or nested angle brackets raise an error. If x is the empty string or NA, the whole dictionary is used.
dic	A dictionary data frame as returned by read_dictionary or make_dictionary, with columns sign_name, row_type, count, type and meaning.
Format	Output format. "percent" and "count" return a frequency table (rounded percentage of N, or raw summed counts; N itself is always a count). "dictionary" returns the dictionary subset of all readings for the tokens, with an added class column.
header	Column-header style: "cuneiform" (the default) names each column by its cuneiform sign, "sign_name" by its sign name.
sign_grammar	Path to the sign-grammar table that drives the Sx->S -> 1S0/0A0 decision (columns name and Adj). If NULL (the default), the table 'extdata/sign_grammar.txt' shipped with the package is used.
mapping	A sign mapping data frame (columns name, cuneiform, syllables), passed on to as.sign_name and as.cuneiform. If NULL (the default), the package's 'etcsl_mapping.txt' is used.

Details

The equivalence classes capture the different ways an expression can be used, even when – because the sign is rare – only a single use is securely attested. An expression formed from a transitive verb (class 1A0), for example, can yield either a verb or a relative clause, and the expression itself does not reveal which: what is meant depends on whether it stands at the end or in the middle of a sentence. Likewise a noun (S) can be used as a verb "to be S"; again this cannot be read off the expression and depends on its position in the sentence. Knowing the class therefore makes the full range of admissible readings available even from sparse evidence.

The classes are equivalence classes of grammatical meanings. A class name nXm encodes n arguments to the left and m arguments to the right of the sign, with result type X (S or A) for the canonical representative. In the table below, the Type column gives the type of that canonical representative:

Name	Type	Criteria	Example
0S0	S	type S	"king"
		xS->S ending in "of S"/"for S"	"king of S"
		V of the form "to be <noun>"	"to be king"
		xS->V of the form "to be <noun> of S"	"to be king of S"
		A of the form "who is <noun>"	"who is king"
		Sx->S of the form "who is <noun>"	"S, who is king"

0S1	xS->S	xS->S not ending in "of S"/"for S"	"great one who creates S"
1S0	Sx->S	Sx->S, not in 0S0, head-changing (changes the head noun)	"city of S"
0S2	xSS->S	xSS->S	"life-fire that binds S1 to S2"
1S1	SxS->S	SxS->S not beginning with "S1"	"one who is defended from S1 by S2"
2S0	SSx->S	SSx->S not beginning with "S1"	"binding S1 to S2"
0A0	A	type A, not in 0S0 type V, not in 0S0 Sx->S, not in 0S0, used adjectivally (head unchanged)	"who is good" "to be good" "good S"
0A1	xS->A	xS->A (introduced relative clause) SxS->S beginning with "S1"	"who relies on S" "S1 who relies on S2"
1A0	Sx->A	Sx->A (relative clause) Sx->V or Vt (transitive verb) SSx->S beginning with "S1"	"that is defined by S" "to be defined by S" "S1 that is defined by S2"
0A2	xSS->A	xSS->A (introduced complex relative clause)	"who binds S1 and S2 together"
1A1	SxS->A	SxS->A (for completeness only)	
2A0	SSx->A	SSx->A (complex relative clause) SSx->V (complex transitive verb)	"that equips S1 with S2"
VP	xV->V	xV->V (verb prefix)	"to not V"
VS	Vx->V	Vx->V (verb suffix, for completeness only)	
SEN	SEN	a whole-sentence reading (lexicalised compound)	"The king is a catalyst for change"
R		everything else (not matched above)	"Which SEN?"

Readings of type SEN form the class SEN. Meanings containing "invisible force", "no one", or "misidentified" inside an angle-bracket note (<... misidentified ...>) are not classified: they keep class = NA in the dictionary subset and do not count towards the frequency table. In the frequency table an entry without a count value counts as 1.

For Sx->S that is not in 0S0, the split between 1S0 (head-changing) and 0A0 (adjectival) is taken from the per-sign table given by sign_grammar (column Adj: 0 head-changing, 1 adjectival, 2 undecidable), by default the bundled 'extdata/sign_grammar.txt'. When the sign is undecidable or absent from the table, a fallback heuristic decides per reading: head-changing (1S0) when the meaning embeds S as an object or complement ("X of S", a relative clause with its own head, or a nominalisation), and adjectival (0A0) when S stays the head (the meaning begins with S or is a bare "<*adjective*> S"). A few readings that the heuristic misses are corrected directly in the function.

Whether a "to be X" reading of type V is a noun predication (0S0) or an intransitive verb/adjective (0A0) is decided by comparing X against the sign's own noun readings (type S and the core of "X of S").

Value

For Format = "dictionary": the subset of dic (rows of type trans.) for the tokens of x, with an added column class holding the equivalence class of each reading (NA for ignored readings).

For Format = "percent" or "count": a data frame with one column per token (named by its cuneiform sign, or by its sign name when header = "sign_name"; a single column "all" when x selects the whole dictionary) and the rows, in this order: N, 0S0, 0S1, 1S0, 0S2, 1S1, 2S0, 0A0, 0A1, 1A0, 0A2, 1A1, 2A0, VP, VS, SEN, R.

See Also

read_dictionary and make_dictionary for dictionaries, as.sign_name for tokenisation, look_up.

Examples

```
dic <- read_dictionary()

# Class profile per sign, as rounded percentages
classify("AN.LUGAL", dic)

# Raw counts
classify("AN.LUGAL", dic, "count")

# Treat several signs as one token via angle brackets
classify("LUGAL.<U.U.U>", dic)

# Dictionary subset with the class of every reading
classify("AN.LUGAL", dic, "dictionary")

# Class distribution of the whole dictionary
classify(NA, dic, "count")
```

---

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)
```

---

fill_substr_info *Read Back a Saved Translation into a Substring Data Frame*

Description

Reads a saved skeleton file (or character vector) and reconstructs the substring data frame that can be passed as `fill` to `translate` or `skeleton`. This allows resuming work on a translation that was saved earlier with `writeLines`.

The counterpart to this function is `guess_substr_info`, which fills the substring data frame from dictionaries. In contrast, `fill_substr_info` fills it from a skeleton that already contains manually entered types and translations.

Usage

```
fill_substr_info(skeleton)
```

Arguments

skeleton	A file path to a saved skeleton text file, or a character vector as returned by skeleton or translate.

Details

A typical workflow for translating Sumerian texts spans multiple sessions:

1. Call translate to interactively translate a line.
2. Save the result with writeLines(result, "Line_29.txt").
3. In a later session, call fill_substr_info("Line_29.txt") to reload the saved translation.
4. Pass the result as fill to translate to continue editing.

The function parses the skeleton format (header line "Structure: ..." followed by entry lines starting with |), extracts the type and translation from each entry, and places them at the correct positions in a substring data frame as created by init_substr_info.

Value

A data frame with $N(N+1)/2$ rows (where N is the number of cuneiform tokens) and the following columns:

start Integer. The 1-based position of the first token in the substring.

n_tokens Integer. The number of tokens in the substring.

expr Character. The concatenated cuneiform signs of the substring.

type Character. The grammatical type (e.g. "S", "V", "Sx->V"), or "" if not yet specified.

translation Character. The translation, or "" if not yet specified.

Rows without a corresponding skeleton entry have empty type and translation fields. The row order matches init_substr_info, so indices can be computed with substr_position.

See Also

translate for the interactive translation tool, skeleton for creating and displaying translation templates, guess_substr_info for filling a substring data frame from dictionaries, init_substr_info for the underlying data frame structure

Examples

```
skeleton_file  <- system.file("extdata", "project/lines/Line_29.txt", package = "sumer")
the_skeleton   <- readLines(skeleton_file)

#Get the cuneiform text of the line:
x <- sub("^Structure: ", "", the_skeleton[1])
x

#See the whole file:
cat(the_skeleton, sep="\n")
```

```
df_fill <- fill_substr_info(skeleton_file)


## Not run:

#Use the result of the function to revise the translation:

dict_file <- system.file("extdata", "sumer-dictionary.txt", package = "sumer")
text_file <- system.file("extdata", "project", "enki_and_the_world_order.txt", package = "sumer")

result <- translate(x,
                text = text_file,
                dic = dict_file,
                fill = df_fill,
                min_freq = c(6, 4, 2))
print(result)
# Now you may save the result with writeLines.

## End(Not run)
```

`grammar_probs` *Posterior Probabilities of Grammatical Types or Classes for Each Sign*

Description

For each sign in a sentence, computes Bayesian posterior probabilities by combining the frequencies in `sg` with a prior distribution. Whether grammatical types or equivalence classes (see `classify`) are returned is taken from the format of `sg`.

Usage

```
grammar_probs(sg, dic = NULL, prior = NULL, alpha0 = 1, mapping = NULL,
              sign_grammar = NULL)
```

Arguments

`sg`	A data frame as returned by `sign_grammar`: with a `type` column (types) or a `class` column (classes, from `sign_grammar(unit = "class")`), plus `position`, `sign_name`, `cuneiform` and `n`.
`dic`	A dictionary data frame as returned by `read_dictionary`, used to compute the prior when `prior` is `NULL`.
`prior`	A named numeric vector of priors. If `NULL` (the default), it is computed internally with `prior_probs` for the same unit (type or class) as `sg`. Supply your own vector to override it.
`alpha0`	Numeric (>= 0). Strength of the prior (pseudo sample size). Larger values pull the posterior towards the prior; `alpha0 = 0` gives a purely data-driven result. Default: 1.

`mapping` A sign mapping table passed through to `prior_probs` when the prior is computed internally (only for class priors). If `NULL` (the default), the package's internal file '`etcsl_mapping.txt`' is used.

`sign_grammar` The path to a sign-grammar file passed through to `prior_probs` when the prior is computed internally (only for class priors). If `NULL` (the default), the package's internal file '`sign_grammar.txt`' is used.

Details

For each sign at position i, the counts n_k per type/class are taken from `sg`, with total $M = \sum_k n_k$, and the posterior probability is (Dirichlet-Multinomial model):

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

where p_k is the prior probability. For signs with no readings ($M = 0$) the posterior equals the prior; for signs with many observations ($M \gg \alpha_0$) it is dominated by the data.

Either `dic` or `prior` must be supplied.

Value

A data frame with columns `position`, `sign_name`, `cuneiform`, `prob` (posterior probability), `n` (counts), and either `type` (when `sg` carries types) or `class` (when `sg` carries classes).

See Also

`sign_grammar` for the input frequencies, `prior_probs` for the prior, `classify` for the equivalence classes, `plot_sign_grammar` for visualisation.

Examples

```
dic <- read_dictionary()

# Equivalence-class posteriors (prior computed internally)
sg <- sign_grammar("a-ma-ru ba-ur3 ra", dic, unit = "class")
gp <- grammar_probs(sg, dic)
print(gp)

# Type posteriors (sg carries types)
grammar_probs(sign_grammar("a-ma-ru ba-ur3 ra", dic), dic)
```

`grammatical_structure` *Grammatical Structure of a Sumerian Expression*

Description

Determines and visualizes the grammatical structure of a Sumerian expression. The function groups sub-expressions according to operator binding and composition rules and returns a bracketed string in which each bracket type indicates the grammatical role of the group:

- `()` – substantive (S)
- `<>` – verb (V)
- `[]` – attribute (A)
- `{}` – sentence (SEN)

The result has class `"grammatical_structure"` and comes with a `print` method that displays the bracket tree with color-coded groups in the console (requires ANSI color support).

Usage

```
grammatical_structure(s, type, expr = NULL)

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

Arguments

`s`	Character string. A Sumerian expression in cuneiform characters.
`type`	Character vector of grammatical types, one per sub-expression. Each entry is either a base type (`"S"`, `"V"`, `"A"`) or an operator type (e.g. `"Sx->V"`, `"xS->A"`).
`expr`	Character vector of sub-expressions (e.g. the individual signs or sign groups that make up `s`). The function matches each `expr[k]` in `s`, determines the grouping, and returns the result with the original expressions in place.
`x`	An object of class `"grammatical_structure"`.
`...`	Further arguments (currently unused).

Details

The grouping is performed in two stages. First, `add_brackets` inserts bracket groups based on operator binding strength and pairwise composition rules. Then each group is assigned a bracket type that reflects its grammatical role, as determined by the operator it contains or by the types of its elements.

The `print` method displays the resulting string with ANSI colors in the console. Each bracket type and its direct content (nesting level 0) are shown in a distinct color: green for `()`, blue for `[]`, red for `<>`, and yellowish-brown for `{}`. Bracket pairs that contain only nested sub-groups (no bare symbols at nesting level 0) are shown in light gray.

Value

A character string of class `"grammatical_structure"` with typed brackets showing the grammatical grouping. On error, a plain character string containing the error message.

See Also

apply_translation_rules for translating (instead of visualizing) the structure, add_brackets for the underlying grouping algorithm, split_sumerian for obtaining the sub-expressions from a Sumerian string

Examples

```
# Example 1
x    <- "mec3-ki-aj2-ga-ce-er ce du"
expr <- split_sumerian(x)$signs
type <- c("S", "S", "Sx->A", "xS->A",  "S",  "Sx->S", "S", "Sx->V")

grammatical_structure(x, type, expr)

grammatical_structure(as.cuneiform(x), type, as.cuneiform(expr))

# Example 2: An example with a proper name and verb prefixes
x    <- "an-en-ki-en-gan-ig-la"
expr <- c("an-en-ki", "en", "gan", "ig", "la")
type <- c("S", "S", "xV->V", "xV->V", "Vt")
grammatical_structure(as.cuneiform(x), type, as.cuneiform(expr))
```

---

guess_substr_info *Look Up Translations for All Substrings of a Sumerian Text*

Description

Converts a Sumerian text string into cuneiform tokens, generates all contiguous substrings, and looks up the most frequent translation for each substring in one or more dictionaries.

Usage

```
guess_substr_info(x, dic, mapping = NULL)
```

Arguments

x	A character string of length 1 containing Sumerian text (transliteration, sign names, or cuneiform characters). May contain brackets as used by skeleton.
dic	A dictionary, a list of dictionaries, or a character vector of file paths to dictionary files. If file paths are given, each file is loaded with read_dictionary. Dictionaries are tried in order: the first dictionary that contains a translation for a given substring wins.
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.

Details

The function performs the following steps:

1. If `dic` is a character vector of file paths, the dictionaries are loaded with `read_dictionary`. If `dic` is a single data frame, it is wrapped in a list.
2. The input string `x` is converted to cuneiform with `as.cuneiform` and split into individual tokens with `split_sumerian`.
3. A data frame of all contiguous substrings is created with `init_substr_info`.
4. A `sign_name` column is added by converting each substring expression with `as.sign_name`.
5. For each substring, the dictionaries are searched in order. The most frequent translation (highest `count` among rows with `row_type == "trans."`) from the first dictionary that contains a match is used to fill in the `type` and `translation` columns.
6. Single-token entries of type 4 (numbers and `N`) receive type `"S"` and their numeric value as translation, regardless of dictionary content.

Value

A data frame with one row per substring and the following columns:

`start`	Integer. The token position of the first token in the substring (1-based).
`n_tokens`	Integer. The number of tokens in the substring.
`expr`	Character. The concatenated cuneiform tokens of the substring.
`type`	Character. The grammatical type of the most frequent translation (e.g. `"S"`, `"V"`), or `""` if no translation was found.
`translation`	Character. The most frequent translation from the dictionaries, or `""` if no translation was found.
`sign_name`	Character. The sign name representation of the substring.

The rows are ordered as in `init_substr_info` (by `n_tokens` descending, then `start` ascending), so that row indices can be computed with `substr_position`.

See Also

`init_substr_info` for creating the substring data frame, `substr_position` for computing row indices, `read_dictionary` for loading dictionaries, `look_up` for interactive dictionary lookup, `skeleton` for creating translation templates

Examples

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

# Look up translations for all substrings
x <- "lugal kur-ra-ke4"
df <- guess_substr_info(x, dic)

# Show rows that have a translation
```

```
df[df$translation != "", ]

# Use multiple dictionaries (ordered by reliability -> first match wins)
file1 <- system.file("extdata", "sumer-dictionary.txt", package = "sumer")
df <- guess_substr_info(x, file1)
```

`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.
- Type 4 (numbers and N): Passed through unchanged.
- Type 5 (unknown X): Passed through unchanged.

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, `as.cuneiform` for converting to cuneiform characters, `as.sign_name` for converting to sign names

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, mapping = NULL, aggregate = FALSE)

## 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.
`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.
`aggregate`	Logical. If `TRUE`, the dictionary entries are aggregated before the lookup, so that as far as possible only the canonical representative of each entry is shown; its other usage forms as a noun, a relative clause or a verb can be derived from it (see 'Aggregation'). Default is `FALSE`. The argument name can be abbreviated (e.g. `agg = TRUE`).
`...`	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)

Aggregation: With `aggregate = TRUE`, as far as possible only the canonical representative of each dictionary entry is shown. Entries that are merely other usage forms of the same entry are removed, and their counts are added to the count of the representative. For example, an entry `X` attested as a noun (type `S`) can also be used as the verb `"to be X"` or as the relative clause `", who is X"`, so such entries are merged into `X`; likewise, relative clauses such as `"who X"` or `"those who X"` are merged into the verb `X`. Finally, entries of the same type that differ only in a short part are merged into one entry that lists the differing parts in parentheses, e.g. `"supervisors of (those/actors) who need supervision"`; an empty part is shown as the empty set symbol, and entries with a large share of the counts of a sign are left unchanged by this step.

The total of the counts of a sign name is not changed by the aggregation, and the entries are sorted by decreasing count. For reverse lookups (`lang = "en"`), the aggregation is applied to the whole dictionary before the search. The exact rules are documented in the source files 'R/aggregate_dictionary.R' and 'R/merge_similar_entries.R'.

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.
`aggregate`	Whether the dictionary entries were aggregated (see section 'Aggregation').
`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)

# Merge grammatical variants (e.g. "X of S", "who X") into their base entries
look_up("A", dic, aggregate = TRUE)
look_up("A", dic, agg = TRUE)   # the argument name can be abbreviated
look_up("water", dic, "en", aggregate = TRUE)

# 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. Excludes lines containing the unknown-sign placeholder X
4. Replaces standalone numbers in sign names with N (suffix digits like the 2 in jal2 are not affected)
5. Normalizes sign names and looks up possible readings from the mapping table
6. 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")

row_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

read_translated_text for reading translation files, convert_to_dictionary for the aggregation step, read_dictionary for loading a saved dictionary, save_dictionary for saving a dictionary to file, look_up for searching a dictionary

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, mapping = NULL)
```

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`.
`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.

Details

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

Single-sign n-grams (`length == 1`) are excluded from marking. The remaining n-grams are sorted descending by `length` and each occurrence of a combination is replaced 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", "project", "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")
```

---

merge_dictionaries *Merge Two or More Sumerian Dictionaries*

Description

Combines two or more dictionaries (as produced by make_dictionary) into a single dictionary. Translation entries that agree in sign name, grammatical type, and meaning are merged by summing their counts. Cuneiform and reading rows are taken from the first dictionary that contains them.

Usage

```
merge_dictionaries(...)
```

Arguments

...	Two or more dictionaries, each either a data frame (as returned by make_dictionary or read_dictionary) or a file path to a dictionary file. A single list of dictionaries is also accepted.

Details

The function processes the three row types differently:

`"cunei."` **and** `"reading"` Each sign name has at most one row of each type. If multiple input dictionaries contain a cuneiform or reading row for the same sign, the row from the first dictionary (in argument order) is kept.

`"trans."` Rows that share the same `sign_name`, `type`, and `meaning` are merged into a single row whose `count` is the sum of the individual counts. Rows that differ in type or meaning are kept as separate entries.

The result is sorted by `sign_name`, then `row_type`, then descending `count`, matching the order produced by `make_dictionary`.

Note: Merging dictionaries is most meaningful when the underlying text corpora come from comparable periods and regions of Mesopotamia. Combining dictionaries from widely different epochs or dialects may produce misleading frequency counts, since the same sign can carry different meanings across time and place.

Value

A data frame with five columns in the standard dictionary format:

sign_name Character. The sign name (e.g. `"A"`, `"LUGAL"`).

row_type Character. One of `"cunei."`, `"reading"`, or `"trans."`.

count Numeric. The (merged) occurrence count for translation rows; `NA` for cuneiform and reading rows.

type Character. The grammatical type for translation rows (e.g. `"S"`, `"V"`); empty for cuneiform and reading rows.

meaning Character. The cuneiform glyph, the reading notation, or the translation text, depending on `row_type`.

See Also

`make_dictionary` for creating a dictionary from a translated text, `read_dictionary` for loading a dictionary from file, `save_dictionary` for saving a dictionary to file, `convert_to_dictionary` for the aggregation step inside `make_dictionary`

Examples

```
# Load the built-in dictionary
dic1 <- read_dictionary()

# Make a dictionary from some translations of "Enki and the World Order"
path <- system.file("extdata", "project/lines", package = "sumer")
dic2 <- make_dictionary(list.files(path, full.names=TRUE)[1:10])

# Merge both dictionaries
dic <- merge_dictionaries(dic1, dic2)
```

```
#Test the function
look_up("IL2", dic1)
look_up("IL2", dic2)
look_up("IL2", dic)
```

`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), mapping = NULL)
```

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.
`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.

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 three 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.
`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", "project", "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 Class or 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. Whether grammatical equivalence classes (see `classify`) or raw grammatical types are shown is taken from the format of `sg`: a `class` column selects classes, a `type` column selects types. Each case uses its own fixed colour palette.

Usage

```
plot_sign_grammar(sg,
                  output_file = NULL,
                  width       = 10,
                  height      = 5,
```

```
                sign_names  = FALSE,
                font_family = NULL,
                mapping     = NULL)
```

Arguments

sg	A data frame as returned by grammar_probs or sign_grammar. It must contain either a class column (classes) or a type column (types), and either a prob column (probabilities) or an n column (counts).
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), "Segoe UI Historic" is used.
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.

Details

When the input carries posterior probabilities (column prob), they are used directly. When it carries raw class counts (column n), absolute frequencies are converted to percentages so that the bars of a position sum to 100%.

Colours of the equivalence classes are grouped by result type (S or A) and binding direction; within each pair the left class is the lighter shade:

- Yellow: left-binding S classes 1S0, 2S0.
- Green: balanced S classes 0S0, 1S1.
- Blue: right-binding S classes 0S1, 0S2.
- Orange: left-binding A classes 1A0, 2A0.
- Red: balanced A classes 0A0, 1A1.
- Violet: right-binding A classes 0A1, 0A2.
- Grey: VP, VS, SEN and R.

VS and SEN appear in the legend only when they have a positive value for at least one sign.

When a type column is present, the fixed colour palette of the type-frequency view is used instead; types not in that palette are shown in grey.

Value

Invisibly returns the **ggplot2** plot object.

See Also

sign_grammar for the type or class frequency table, grammar_probs for the Bayesian posterior probabilities, classify for the equivalence classes.

Examples

```
dic <- read_dictionary()

# Smoothed class probabilities (grammar_probs; prior computed internally)
sg <- sign_grammar("a-ma-ru ba-ur3 ra", dic, unit = "class")
gp <- grammar_probs(sg, dic, alpha0 = 1)
plot_sign_grammar(gp, sign_names = TRUE)

# Raw class frequencies (sign_grammar with unit = "class")
plot_sign_grammar(sign_grammar("a-ma-ru ba-ur3 ra", dic, unit = "class"))

# Raw type frequencies (sign_grammar, the default)
plot_sign_grammar(sign_grammar("a-ma-ru ba-ur3 ra", dic))

# Save to file
file <- file.path(tempdir(), "grammar.png")
plot_sign_grammar(gp, output_file = file)
```

---

prior_probs *Prior Probabilities of Grammatical Types or Equivalence Classes*

Description

Computes prior probabilities for each grammatical type (e.g. S, Sx->S) or, alternatively, for each grammatical equivalence class (0S0, 1A0, ...; see classify) from a dictionary.

Usage

```
prior_probs(dic, unit = c("type", "class"), mapping = NULL,
            sign_grammar = NULL)
```

Arguments

dic	A dictionary data frame as returned by read_dictionary.
unit	Whether to compute priors over grammatical "type" (the default) or over equivalence "class" (the classes of classify).
mapping	Only relevant for unit = "class": a sign mapping table (data frame) passed through to classify. If NULL (the default), the package's internal file 'etcsl_mapping.txt' is used.
sign_grammar	Only relevant for unit = "class": the path to a sign-grammar file passed through to classify. If NULL (the default), the package's internal file 'sign_grammar.txt' is used.

Details

The function proceeds in two steps:

1. For each single-sign dictionary entry, the readings are tabulated by grammatical type – or assigned to equivalence classes with the same rules as `classify` when `unit = "class"` – and the resulting counts are normalised to sum to 1.
2. The prior probability of each type/class is the mean of these normalised distributions across all single signs that have at least one count.

Value

A named numeric vector summing to 1, with one element per grammatical type (when `unit = "type"`) or per equivalence class (when `unit = "class"`).

See Also

`classify` for the equivalence classes, `grammar_probs` for per-sign posteriors, `sign_grammar` for per-sign frequencies.

Examples

```
dic <- read_dictionary()

prior_probs(dic)            # over grammatical types
prior_probs(dic, "class")   # over equivalence classes
```

`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. Cleans meaning field by removing content after ; or | delimiters
5. Issues a warning for entries with missing type annotations
6. Excludes lines containing the unknown-sign placeholder X
7. Replaces standalone numbers in sign names with N (suffix digits like the 2 in jal2 are not affected)
8. Normalizes transliterated text by removing separators and looking up the sign names from the mapping
9. 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 or Class Frequencies for Each Sign in a Sumerian Sentence*

Description

For each cuneiform sign in a Sumerian sentence, looks up the dictionary and counts how often the sign occurs with each grammatical type (e.g. `S`, `V`, `Sx->S`) or, alternatively, with each grammatical equivalence class (`0S0`, `1A0`, ...; see `classify`).

Usage

```
sign_grammar(x, dic, unit = c("type", "class"), mapping = NULL,
             sign_grammar = NULL)
```

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.
unit	Whether to count by grammatical "type" (the default) or by equivalence "class" (the classes of classify).
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.
sign_grammar	Only relevant for unit = "class": the path to a sign-grammar file (sign name to adjectival/head-changing value) used by classify. If NULL (the default), the package's internal file 'sign_grammar.txt' is used.

Details

The function converts the input to cuneiform, splits it into individual signs, and looks up each sign in the dictionary. Signs enclosed in angle brackets (<. . .>) are kept together as a single token; other brackets and separators are ignored. Unbalanced or nested angle brackets raise an error.

For unit = "type", the dictionary count values are summed per grammatical type (a missing count is treated as 1); each sign gets one row per type with a positive count.

For unit = "class", the readings are classified with classify (readings whose meaning contains "invisible force", "no one" or "misidentified" inside an angle-bracket note are ignored; a missing count is treated as 1) and the counts are summed per class.

Signs not found in the dictionary are returned with a few base types (S, A, V) or base classes (0S0, 0A0, 1A0) at count 0, so that the sign still appears in a plot.

Value

A data frame with columns position (integer), sign_name, cuneiform, n (integer; summed dictionary counts), and either type (when unit = "type") or class (when unit = "class").

See Also

classify for the equivalence classes, grammar_probs for Bayesian posterior probabilities, plot_sign_grammar for visualising the result, read_dictionary for loading a dictionary.

Examples

```
dic <- read_dictionary()

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

# Equivalence-class frequencies (can be passed to plot_sign_grammar)
sgc <- sign_grammar("a-ma-ru ba-ur3 ra", dic, unit = "class")
print(sgc)
```

skeleton *Create a Translation Template for Sumerian Text*

Description

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

The input may contain three types of brackets to control how the template is generated (see Details). Optionally, the template can be pre-filled with translations from one or more dictionaries using guess_substr_info.

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, fill = NULL, space = FALSE)

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

Arguments

x	For skeleton: A character string of length 1 containing transliterated Sumerian text (transliteration, sign names, or cuneiform characters). Tokens may be grouped with brackets to control template generation (see Details). 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.
fill	A data frame as returned by guess_substr_info, containing translations and grammatical types for all substrings of x. If provided, the template lines are pre-filled with the corresponding type and translation. If NULL (the default), the template lines are left empty.
space	Logical. If TRUE, an empty line is inserted before each entry at nesting depth 1, visually separating top-level groups. Defaults to FALSE.
...	Additional arguments passed to the print method (currently unused).

Details

The function generates a hierarchical template from a Sumerian text string. The input is first converted to cuneiform with as.cuneiform. The input string may contain three types of brackets that control how entries in the template are generated:

Angle brackets < > The enclosed token sequence is treated as a fixed term. No individual skeleton entries are generated for the tokens inside. For example, <d-nu-dim2-mud> is treated as a single unit.

Round brackets ( ) The enclosed token sequence is a coherent term for which a single skeleton entry is generated, in addition to entries for its individual tokens. Nesting is allowed.

Curly braces { } Ignored during skeleton generation. They can be used in the input to indicate which tokens serve as arguments to an operator, but this information is not needed for the skeleton.

In addition, a skeleton entry is generated for every individual token that does not appear inside angle brackets.

Each line in the resulting template follows the format:

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

When `fill` is not provided, the type and translation fields are left empty:

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

The template should then be filled in as follows:

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

The indentation level (number of tabs) reflects the nesting depth: top-level entries have no indentation, their sub-entries have one tab, and so on.

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

If `fill` is provided, the function validates that `fill` matches `x`: the cuneiform tokens of the first row in `fill` must be identical to the tokens of `x`, and the number of rows must equal $N(N+1)/2$ where N is the number of tokens.

Value

`skeleton` returns a character vector of class `c("skeleton", "character")` containing the template lines. The first line is the header with the full reading of the input, followed by one line per skeleton entry. If `space = TRUE`, empty strings are inserted as separator lines.

`print.skeleton` prints the template to the console (one line per element) and returns `x` invisibly.

See Also

`guess_substr_info` for generating the `fill` data frame, `mark_skeleton_entries` for the bracket normalization step, `extract_skeleton_entries` for the hierarchical extraction step, `substr_position` for computing row indices in the fill data frame, `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 an empty template
x <- "<d-nu-dim2-mud> ki a. jal2 (e2{kur}) ra. gaba jal2. an ki a"
skeleton(x)

# Pre-fill the template with dictionary translations
dic <- read_dictionary()
fill <- guess_substr_info(x, dic)
skeleton(x, fill = fill)

# Use spacing to visually separate top-level groups
skeleton(x, fill = fill, space = TRUE)
```

`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 five types of Sumerian sign representations: lowercase transliterations, uppercase sign names, Unicode cuneiform characters, numbers (including the placeholder `N`), and the unknown-sign placeholder `X`.

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).
4. **Numbers** (type 4): Integer or decimal numbers (e.g. `4`, `3.5`), and the standalone letter `N` which serves as a placeholder for an arbitrary number.
5. **Unknown signs** (type 5): The standalone letter `X`, which serves as a placeholder for an unreadable sign.

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, `3` for cuneiform characters, `4` for numbers and the placeholder `N`, and `5` for the unknown-sign placeholder `X`.

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 = "")
```

`translate` *Interactive Translation Tool for Sumerian Text*

Description

Opens an interactive Shiny gadget for translating a single line of Sumerian cuneiform text. The page displays four sections on a single scrollable page: n-gram patterns, context with neighbouring lines, grammar probabilities, and an interactive skeleton with dictionary lookup. When the user clicks "Done", the function returns a `skeleton` object with the updated translations.

Usage

```
translate(x, text = NULL, dic = NULL, mapping = NULL, fill = NULL,
          min_freq = c(6, 4, 2), sign_grammar = NULL,
          viewer = shiny::paneViewer())
```

Arguments

x	A single Sumerian text string (transliteration, sign names, or cuneiform), or an integer line number indexing into text.
text	A character vector containing the full text being translated (one line per element), a file path to load with readLines(), or NULL. Lines may start with numbering like "12)\t..." or "12. ...". Required when x is an integer; optional otherwise. If a single string that is an existing file path, it is loaded automatically.
dic	A dictionary (data.frame), a list of dictionaries, or a character vector of file paths to dictionary files. If file paths are given, each is loaded with read_dictionary. If NULL, the built-in dictionary is loaded via read_dictionary().
mapping	A data frame containing the sign mapping table with columns syllables, name, and cuneiform, or a file path to a semicolon-separated mapping file. If NULL (the default), the package's internal mapping file 'etcsl_mapping.txt' is loaded.
fill	A pre-computed substring info data frame (as from init_substr_info or guess_substr_info). If NULL, it is computed automatically via guess_substr_info.
min_freq	Minimum frequency thresholds passed to ngram_frequencies. A numeric vector where the *i*-th element is the minimum frequency for n-grams of length *i*. Default is c(6, 4, 2).
sign_grammar	The path to a sign-grammar file used (via sign_grammar and prior_probs) to compute the grammar-probability chart. If NULL (the default), the package's internal file 'sign_grammar.txt' is used.
viewer	A Shiny viewer function that controls where the gadget window is opened. The default is shiny::paneViewer() which uses the RStudio Viewer pane. Use shiny::browserViewer() to open in the system browser, or shiny::dialogViewer("Translate", width = 842, height = 900) to open a fixed-size dialog in RStudio

Details

The gadget opens in the viewer specified by the viewer parameter (by default the RStudio Viewer pane) and displays four sections on a single scrollable page. The first three sections (N-grams, Context, Grammar) can be collapsed individually. A sticky navigation menu at the top allows jumping to each section.

N-gram Patterns Displays a merged table of n-gram combinations that appear in the current line: n-grams of length 2 or more from the full text (controlled by min_freq), combined with shared n-grams found in neighbouring lines. A "Theme" column marks n-grams shared with the context. Frequencies refer to the full text.

Context Shows neighbouring lines (up to 2 before and after) with frequent n-grams marked. Only available when text is provided and the line index is known.

Grammar Probabilities Displays a bar chart of grammar probabilities for each sign in the line, computed via grammar_probs.

Translation The main interactive section with dictionary selection checkboxes, a bracket input field for editing the skeleton structure, an interactive skeleton display with type and translation fields, and a dictionary lookup panel. Clicking a dictionary row adopts its type and translation into the selected skeleton entry.

When the line contains multiple sentences (separated by dots in the transliteration), skeleton entries belonging to different sentences are displayed with alternating background colours.

The bracket input field allows the user to add or modify brackets (), <>, {} to control the grouping structure of the skeleton. Pressing "Update Skeleton" rebuilds the skeleton display while preserving all translations in the fill data frame.

Value

A skeleton object (character vector of class c("skeleton", "character")), generated by calling skeleton with the final bracket string and updated fill data frame. Returns invisible(NULL) if the user closes the window without clicking "Done".

Note

Requires **shiny** (listed in Imports). By default, the gadget opens in the RStudio Viewer pane. To get a resizable window or a more stable connection (e.g. when the computer may enter standby), use viewer = shiny::browserViewer().

See Also

skeleton for creating translation templates, guess_substr_info for pre-computing substring translations, look_up for interactive dictionary lookup, ngram_frequencies for n-gram analysis, grammar_probs for grammar probability computation, prior_probs for prior probability computation, mark_ngrams for marking n-grams in text

Examples

```
## Not run:
# Basic usage with a transliterated string
result <- translate("lugal kur-ra-ke4")


# Full example with package data
x <- "<d-nu-dim2-mud> ki a. jal2 (e2-kur) ra. gaba jal2. an ki a"

dict_file <- system.file("extdata", "sumer-dictionary.txt", package = "sumer")
text_file <- system.file("extdata", "project", "enki_and_the_world_order.txt", package = "sumer")

result <- translate(x,
                text = text_file,
                dic = dict_file,
                min_freq = c(6, 4, 2))
print(result)
```

```
# Open in system browser (resizable, survives standby)

x <- 9

result <- translate(x,
                text = text_file,
                dic = dict_file,
                min_freq = c(6, 4, 2),
                viewer = shiny::browserViewer())
print(result)


## End(Not run)
```

---

translate_line *Line-by-Line Translation of a Sumerian Text*

Description

translation_context bundles the settings for a translation project: the project directory (with a 'lines/' subfolder for saving line files), the dictionaries, the source text, and parameters for the interactive translator.

translate_line opens the interactive translation tool (translate) for a single line. If a saved translation exists for this line, it is loaded so that previous work can be continued. When the user clicks "Done", the result is saved back to the line file.

Usage

```
translation_context(project_dir, dic = NULL, text = NULL,
                    mapping = NULL, min_freq = c(6, 4, 2))

translate_line(n, context)
```

Arguments

project_dir	Character string. Path to the project directory. Translated lines are saved in the subdirectory 'lines/' as 'Line_1.txt', 'Line_2.txt', etc. If text or dic are given as filenames without a path, they are assumed to reside in project_dir.
dic	Dictionaries to use for translation lookup. A character vector of file paths (or filenames relative to project_dir), a single data frame, or a list of data frames. The first dictionary serves as the primary source. Additional dictionaries serve as fallback references. If NULL, the built-in dictionary is used.
text	The full text being translated. A file path (or filename relative to project_dir) or a character vector (one line per element). Needed for n-gram analysis and context display in translate.

mapping	A data frame with the sign mapping table, or a file path to a semicolon-separated mapping file. If NULL (the default), the package's internal mapping is used.
min_freq	Minimum frequency thresholds passed to translate. Default is c(6, 4, 2).
n	An integer line number. If a file 'Line_<n>.txt' exists in the 'lines/' subdirectory of project_dir, the saved translation is loaded for revision. Otherwise, the most frequent translation for each substring is looked up in the dictionaries.
context	A context object as created by translation_context.

Details

translate_line performs the following steps:

1. If 'Line_<n>.txt' exists in the 'lines/' subdirectory, the cuneiform text and previous translations are loaded via fill_substr_info.
2. A project dictionary is built from saved line files using make_dictionary. Line n itself is excluded to avoid confirmation bias.
3. The interactive translator (translate) is opened with the primary dictionary, the project dictionary, and any additional dictionaries as fallback references.
4. If the user clicks "Done", the result is saved to 'Line_<n>.txt'. If the window is closed in another way, the result is not saved.

Value

translation_context returns a list of class "translation_context".

translate_line is called for its side effect (opening the interactive translator and saving the result). Returns NULL invisibly.

See Also

translate for the underlying interactive translation tool, fill_substr_info for reading back a saved translation, make_dictionary for creating a dictionary from translated lines

Examples

```
# Note: The folder containing the built-in project
#       is copied to a temporary directory.
#       This prevents you from altering the package files.

path  <- system.file("extdata", package = "sumer")

file.copy(
  from = file.path(path, "project"),
  to   = tempdir(),
  recursive = TRUE
)

ctx <- translation_context(
  project_dir = file.path(tempdir(), "project"),
  text        = "enki_and_the_world_order.txt",
```

```
  dic         = file.path(path, "sumer-dictionary.txt")
)
ctx

## Not run:
# Translate line 29 (opens interactive translator)
translate_line(29, ctx)

# Confirm that your changes are still there:
translate_line(29, ctx)

# Continue with the next line:
translate_line(30, ctx)

## End(Not run)
```

Index

∗ **character**
  as.cuneiform, 3
  as.sign_name, 5
  guess_substr_info, 16
  info, 18
  skeleton, 39
  split_sumerian, 41
∗ **database**
  classify, 6
  look_up, 20
∗ **hplot**
  plot_sign_grammar, 29
∗ **methods**
  as.cuneiform, 3
  as.sign_name, 5
∗ **univar**
  ngram_frequencies, 28
∗ **utilities**
  as.cuneiform, 3
  as.sign_name, 5
  classify, 6
  fill_substr_info, 11
  grammar_probs, 13
  guess_substr_info, 16
  info, 18
  look_up, 20
  merge_dictionaries, 26
  ngram_frequencies, 28
  prior_probs, 31
  sign_grammar, 37
  skeleton, 39
  split_sumerian, 41
  translate, 42
  translate_line, 45

add_brackets, *2, 3*, *15, 16*
apply_translation_rules, 2, *16*
as.cuneiform, 3, *6, 7*, *17*, *19*, *22*, *28, 29*, *39*
as.sign_name, *4*, 5, *7*, *9*, *17*, *19*, *29*

classify, 6, *13, 14*, *29*, *31, 32*, *37, 38*
compose_skeleton_entry, *3*
convert_to_dictionary, 9, *24*, *27*, *33*, *35–37*

eval_operator, *2, 3*
extract_skeleton_entries, *40*

fill_substr_info, 11, *46*

grammar_probs, 13, *29–32*, *38*, *44*
grammatical_structure, 14
guess_substr_info, *11, 12*, 16, *39, 40*, *43, 44*

info, *4*, *6*, 18, *40*
init_substr_info, *12*, *17*, *43*

look_up, *9*, *17*, 20, *24*, *40*, *44*

make_dictionary, *7*, *9, 10*, *21, 22*, 23, *26, 27*, *33*, *35–37*, *40*, *46*
mark_ngrams, 25, *44*
mark_skeleton_entries, *40*
merge_dictionaries, 26

ngram_frequencies, *25*, 28, *43, 44*

plot_sign_grammar, *14*, 29, *38*
print.cuneiform *(*as.cuneiform*)*, 3
print.grammatical_structure *(*grammatical_structure*)*, 14
print.info *(*info*)*, 18
print.look_up *(*look_up*)*, 20
print.sign_name *(*as.sign_name*)*, 5
print.skeleton *(*skeleton*)*, 39
prior_probs, *13, 14*, 31, *43, 44*

read_dictionary, *7*, *9*, *13*, *16, 17*, *21, 22*, *24*, *26, 27*, *31*, 32, *37, 38*, *43*
read_translated_text, *10*, *24*, 34

save_dictionary, *24*, *27*, *33*, 36

sign_grammar, *13, 14*, *29–32*, 37, *43*
skeleton, *11, 12*, *16, 17*, 39, *42*, *44*
split_sumerian, *4*, *6*, *16–19*, *29*, 41
substr_position, *12*, *17*, *40*

translate, *11, 12*, 42, *45, 46*
translate_line, 45
translation_context *(*translate_line*)*, 45