

Where does a Japanese word end?

How the engine inside KaiwaLens cuts a line of Japanese into words, traces each form to its dictionary word, and names its grammar, with the mathematics behind the cut and the rewrite.

By Lavrentii Ivanov

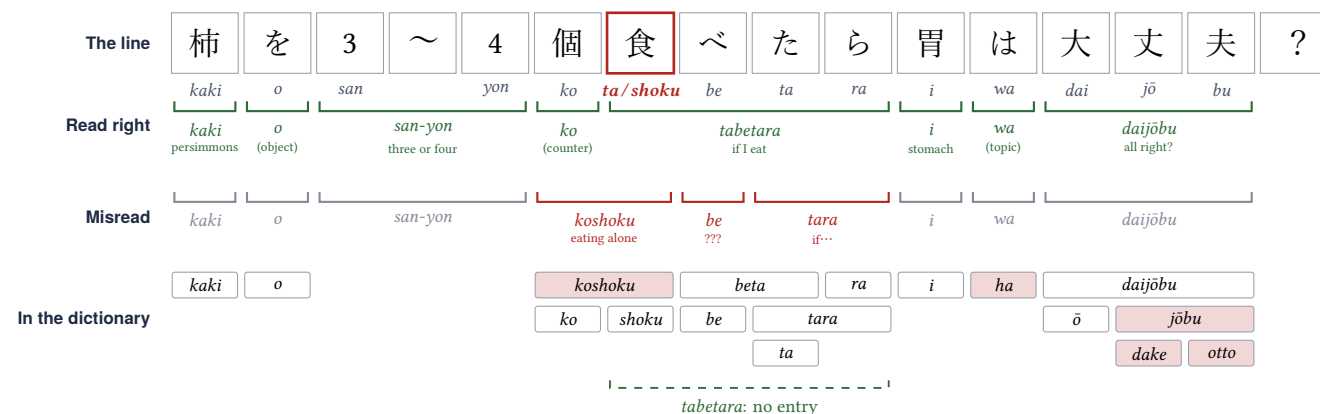


Figure 1. Sixteen characters, no spaces. The line's 16 characters leave 15 gaps, each cut or not, so a program can cut it $2^{15} = 32\,768$ ways. JMdict, the learner's dictionary, lists 17 words inside it, among them three inside *daijōbu* (*dake*, "only"; *jōbu*, "healthy"; *otto*, "husband") and *ha*, "tooth". Read right: "If I eat three or four persimmons, will my stomach be all right?" Misread: "Persimmons, three or four, eating alone, be, if ... stomach all right?" The word the sentence turns on, *tabetara*, is in no dictionary. It is a form of *taberu*, "to eat"; finding it takes a grammar engine that rebuilds the whole word, not only its units.

TAKE the line in Figure 1, from a chat: *Kaki o san-yon-ko tabetara i wa daijōbu?* ("If I eat three or four persimmons, will my stomach be all right?") Japanese writes it in kanji, the Chinese characters, and in kana, its two scripts of syllables, with no spaces between words. One of its kanji, 個 *ko*, is a counter word, used when counting things. It sits next to 食 *ta*-, the first character of 食べた *tabetara*, "if I eat". Together they spell a real word, 個食 *koshoku*, "eating alone", in which the same 食 is read *shoku*. A reader who grabs that word leaves *be* with no word to belong to and *tara*, "if", with no verb. That reader then reads a sentence about solitary dinners. A program is that reader: it has a dictionary and no sense of what is likely.

Before a tool can tell a learner what a word means, it must decide where the word is and which word it is. KaiwaLens, the reading tool I build for learners of Japanese, now makes both decisions with an engine of its own, Kaiwa Core. What follows traces the persimmon line through it.

Two questions, two readers

Take *tabesaserarete imasen deshita*, "I had not been made to eat". A linguist cuts it into short units, each roughly a morpheme, the smallest piece that carries meaning or grammar: *tabe|sase|rare|te|i|mase|n|deshi|ta*. That is the unit of UniDic, the dictionary built by Japan's National Institute for Japanese Language and Linguistics [9]. A learner wants one word, its dictionary form 食べる *taberu*, "to eat", and the grammar stacked on it: causative ("made to"), passive, negative, polite and past. The learner's dictionary, JMdict [2], lists *taberu*; it does not list *sase*. Kaiwa Core uses both dictionaries: UniDic to find where words may end, and JMdict to decide which words they are.

Five parts, one database

Five parts handle a line (Figure 2). The "scan" walks it from the left and cuts words. To cut, it asks two other parts. The "analyser" says where a word may end. The "grammar engine" says whether a stretch of text is a form of some dictionary word. Then the "ranker" picks the dictionary word that leads, and the "patterns" name the grammar in the line. All five read one database, which holds JMdict, UniDic's units and the costs between them.

Every rule about Japanese lives in one place, the engine's Lua code (Lua is a small scripting language). That code holds 95 groups of 681 rewrite rules, the classes of verbs, adjectives and characters, and the grammar patterns. When the rules load, Lua checks them. It refuses any chain of rules that loops back on itself, any malformed rule, any unknown grammar value and any rule that cites no reference book. The program around them, written in C++, carries text, adds dictionary frequencies to the ranking and draws the result; it decides nothing about Japanese.

A lattice of costs

The analyser answers the first question: where may a word end? It writes down every UniDic unit that could start at every position of the line. Laid out, the units form a graph whose paths are the possible ways to cut the line: a lattice. Each unit u has a word cost $w(u)$, for how rare it is. Each unit also has a left and a right context, a class for how it joins its neighbours. Each pair of contexts has a joining cost $c(r, \ell)$. The higher it is, the less often a unit that ends in context r comes before one that starts in context ℓ . A cut into units $u_1 \dots u_m$, where u_i has the contexts ℓ_i and r_i , costs

$$\sum_{i=1}^m [w(u_i) + c(r_{i-1}, \ell_i)] + c(r_m, 0), \quad (1)$$

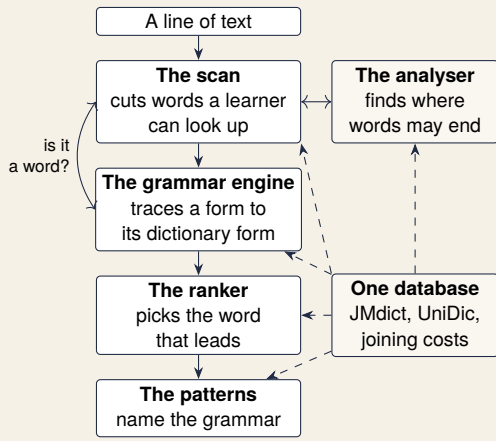


Figure 2. Five parts and one database: a line's path through Kaiwa Core

where $r_0 = 0$ stands for the beginning of the text and the last term joins the end. The rarer the units and the more awkward their joins, the higher the cost. The idea comes from MeCab, the standard open-source analyser for Japanese [6], and Kaiwa Core computes it over UniDic's data in its own database. UniDic learned its costs from hand-annotated text so that adding costs behaves like multiplying probabilities: the cheapest path is, roughly, the likeliest cut.

The analyser solves the problem for the line's first character, then its first two, and so on to the end, reusing every answer. The method is dynamic programming. Let $D(j, r)$ be the cheapest cost of covering the first j characters with a path whose last unit ends in context r . Then

$$D(j, r) = \min_{u, r'} [D(s(u), r') + w(u) + c(r', \ell(u))]. \quad (2)$$

Here u runs over the units that end at j in context r . The unit starts at $s(u)$, and $\ell(u)$ is its left context. The context r' is the one the unit before it ends in. In words, the cheapest way to reach j is the cheapest way to reach the start of the last unit, plus that unit and its join. D needs the context as well as the position, because the next join depends on how the last unit ends. Ties go to the path with fewer, longer units. A character no unit covers becomes a unit of its own at a cost of 100 000, so any dictionary unit wins and every line still has a path.

Joining costs are where grammar enters. With word costs alone, a common one-kanji noun beats a verb: the analyser cuts 来た *kita*, "came", as the prefix 来 *rai*-, "next", and *ta*. With joins, 来 *ki*- becomes the stem of 来る *kuru*, "to come", followed by *ta*, because the past ending follows a verb's stem more readily than a prefix.

Squeezing the joins

UniDic's full table of joining costs is a matrix M of 458 MB, one cell for each pair of contexts. Kaiwa Core keeps the exact cost where it matters and an average elsewhere:

$$c(r, \ell) = \begin{cases} M_{r\ell} & r \in K_R \text{ and } \ell \in K_L, \\ \bar{M}_{G(r)G(\ell)} & \text{otherwise.} \end{cases} \quad (3)$$

K_R and K_L are the 1 000 contexts on each side that real sentences use most. G maps a context to its group: the contexts that share UniDic's part of speech and conjugation. There

are 1 418 groups on one side and 1 317 on the other. $\bar{M}$ holds the mean cost over each pair of groups. Together the two tables take 5.5 MB, an eighty-third of the full table.

Speed is a Japanese problem too. With no spaces, the analyser must ask at every position which units start there. Kaiwa Core keeps in memory one sorted index that holds the lowest cost of each of UniDic's 674 927 spellings. The analyser walks it a character at a time and stops as soon as nothing starts that way. The index brought a call to the analyser down from 620 μ s to 138 μ s. On my PC, an Intel Core i7-7700, the analyser now takes 0.254 ms for a sentence of the treebank, about 3 900 sentences a second. The scan, which also looks every word up, takes 3.85 ms for a subtitle-sized line.

Words no dictionary has

Gurezuri is a made-up name in katakana, the script for foreign words. It has no entry, but its first part *gurezu* does. Left alone, the analyser would take the start as that word and cut the rest one character at a time. So every character belongs to a class (kanji, hiragana, katakana, Latin, digits and six more, all UniDic's own), and each class has flags. "Invoke" tries an unknown word even where a dictionary word starts. "Group" offers the whole run of the class, up to 24 characters, as one word. "Length" also offers the run's first characters, up to a length UniDic sets for each class. UniDic's table for unknown words prices each offer, and the analyser weighs it like any other unit. *Gurezuri*, offered whole, wins on cost; KaiwaLens tells the reader there is no entry instead of offering "grease".

Kaiwa Core sets two classes differently from UniDic. Hiragana, the script for grammar endings, does not group; if it did, one unknown hiragana character would swallow the rest of the line. Digits are "whole": no unit may start or end inside a run of them, so 2010 is one number, not four.

The scan

The analyser cuts too finely for a learner, so the scan builds a learner's words out of its units. Let B be the set of positions where a unit of the cheapest path ends, and let $t_{i..j}$ be the text from position i to position j . At each position i , the scan looks at the stretches $t_{i..j}$ up to 16 characters long. It takes the longest that is "worth looking up" and ends in B . If none qualifies, it takes the analyser's unknown word that starts at i , and failing that, the longest stretch worth looking up, then starts again at j .

A stretch is worth looking up when JMdict has it, as written or as pronounced, or when the grammar engine traces it to a word whose class JMdict confirms. *Tabetara* counts: it leads to *taberu*, which JMdict lists as a verb of the right class.

Back to the persimmons. 個食 *koshoku* is in JMdict, but the analyser takes 食 $\prec$ *tabe* as one unit, so no word may end after 食 *ta*-. The scan takes 個 *ko*, then *tabetara* whole, and looks it up as *taberu*.

People also type the way they speak. Where a lookup finds nothing as written, the scan has two last resorts, in order. It drops the stretch mark (*nagāi* for *nagai*, "long"), then collapses emphatic repeats (*sugōōi* for *sugōi*, "amazing"). Nothing rewrites the text itself.

Rules as rewrites

In Japanese, endings stack, and each attaches to the stem the one before it leaves; linguists call such inflection ag-

Box 1. One form, six rewrites

tabesaserarete imasen deshita

→ <i>tabesaserarete imasu</i>	- <i>imasen deshita</i> → - <i>masu</i>
→ <i>tabesaserarete iru</i>	- <i>masu</i> → - <i>ru</i>
→ <i>tabesaserarete</i>	- <i>te iru</i> → - <i>te</i>
→ <i>tabesaserareru</i>	- <i>te</i> → - <i>ru</i>
→ <i>tabesaseru</i>	- <i>rareru</i> → - <i>ru</i>
→ <i>taberu</i>	- <i>saseru</i> → - <i>ru</i>

Each step strips one ending. The chain says causative, potential, negative, polite and past; the potential has the same form as the passive, which is the reading here. Of the 19 chains the search finds for this form, the dictionary confirms 2, and the ranker keeps *taberu*, an *ichidan* verb.

glutinative. The verb’s class decides how a stem is made. *Ichidan* verbs such as *taberu* keep theirs. *Godan* verbs such as 書く *kaku*, “to write”, change their last sound. *Suru*, “to do”, and *kuru*, “to come”, are irregular.

The grammar engine runs that backwards. It tracks a “state”: the set of classes of form a text may belong to. A class is, for example, an *ichidan* verb, a *-te* form (the connective form, as in *tabete*) or a polite *-masu* form. A rule rewrites an ending and moves the text from one state to another,

$$(\sigma_{in} \rightarrow \sigma_{out}, C_{in} \rightarrow C_{out}), \quad (4)$$

where C_{in} and C_{out} are states. A rule fires on a text t in state C when t ends in σ_{in} and C is either empty (raw text) or shares a class with C_{in} . It gives the text with σ_{out} in place, in state C_{out} . The search runs breadth-first, trying every one-rule rewrite before any chain of two. It never visits the same text in the same state twice, and it stops at eight rules deep or at 192 results. A chain ends at a dictionary form: a state with a class JMdict confirms for that text. Box 1 shows one chain. A chain’s rules tell what grammar the form carries: its tense, polarity (positive or negative), formality and voice.

The rules propose more forms than exist, on purpose, and the dictionary prunes them. One rule turns *taberareru* back to *taberu*. The engine names it the potential, “can eat”; for an *ichidan* verb the passive, “is eaten”, has the same form, and only the sentence tells the two apart.

Spoken forms are rules like any other. *Shiranē* rewrites to 知らない *shiranai*, “don’t know”, an entry in its own right, and on to 知る *shiru*, “to know”.

Choosing the word

The rules often leave several chains, and the ranker first drops every one the dictionary does not confirm. For 書いた *kaita*, “wrote”, the rules propose both *kaku* and *kairu*, and JMdict has no *kairu*. For *tabesaserarete imasen deshita*, 2 of the 19 chains remain (Box 1). The ranker orders the rest by two tests. First, the least stripped chain leads. Second, the commoner word wins. Take 行った *itta*: it is the past of 行く *iku*, “to go”. The same spelling, read *okonatta*, is the past of 行う *okonau*, “to conduct”. UniDic’s word cost, which falls as a word grows commoner, is 2 229 for *iku* and 5 524 for *okonau*, so *iku* leads. A word nothing confirms gets no answer: 本 *hon*, “book”, stays a noun, and the false verb *honru* never appears.

One rule overrides both tests: a word leads with the dictionary form a grammar book gives. *Kita* leads with *kuru*; *sareta*, “was done”, with *suru*; and *shitari*, “doing things like”,

Box 2. The engine on other people’s test cases

Case set	Passed / run
Yomitan (pop-up dictionary): inflections	1 330 / 1 330
10ten Reader (pop-up dictionary): inflections	233 / 233
JMdictDB (JMdict’s editor): conjugations	1 794 / 1 794
GSD forms, all: lemma found	5 805 / 5 805
kamiya-codec (library): conjugations	197 / 197
jp-verb-deconjugator (library): deconjugations	194 / 194
GSD forms, test: lemma first	592 / 625
Five books’ families: covered fully	52 / 59
Yomitan’s set also holds 34 invalid forms; none leads to a dictionary word. GSD forms are its verb and adjective forms. Of the test forms, 24 more are read as grammar books read them; of the families, 7 more are covered in part.	

with *suru* rather than the rare verb *shitaru*. To decide, the ranker asks whether UniDic cuts the word the same way. It takes the cheapest cut and every near cut whose cost is within $\Delta = 8\,000$ of it. I tuned that margin on GSD’s other sentences and kept its test sentences for scoring.

Naming the grammar

Seventy-eight grammar patterns sit in one catalogue: *-te iru* (“is doing”), *-tara* (“if, when”) and *-nakereba naranai* (“must”), among others. Matching letters is not enough: *tara* hides inside *atarashii*, “new”, and *tame* hides at the start of *tamerau*, “to hesitate”. So patterns carry guards. Some check where grammar attaches. Temporal *mae ni*, “before”, follows a verb. So the patterns name it in *neru mae ni*, “before sleeping”, but not in *eki no mae ni*, “in front of the station”, or in *omae ni*, “to you”. The strongest guard asks the grammar engine and the dictionary together. A bare *tara* counts only if the text ending there rewrites, through the conditional rule itself, to a word JMdict confirms. *Tabetara* passes. *Atarashii* fails: its stretch *atara* rewrites through *-tara* only to *aru* as an *ichidan* verb, which JMdict refutes. Where a guard cannot run, the patterns name nothing: better no name than a wrong one.

How well it does

I score the analyser against GSD, a Japanese treebank: sentences whose words linguists marked by hand [1]. GSD comes from Universal Dependencies, a shared project of such texts in many languages. Let P be the percentage of the analyser’s boundaries that are right, and R the percentage of the hand-marked boundaries it found. The score is $F_1 = 2PR/(P + R)$, where 100 is perfect. On GSD’s 543 test sentences the analyser scores 99.65, and in 481 of them it cuts exactly as the linguists did.

The rewrite rules pass every case other projects wrote, from pop-up dictionaries to conjugation libraries (Box 2). On real text, the ranker leads with the treebank’s own dictionary form, its lemma, for 592 of GSD’s 625 test verb and adjective forms. In 24 more it differs the way grammar books differ: they read a potential such as 奪える *ubaeru*, “can snatch”, as its verb 奪う *ubau*.

I also check the engine against five reference grammars [3–5, 7, 8]. A grammar corpus, a file of facts, lists the families of forms each book describes. A family is all the forms of one construction, such as the *-tara* conditional. Each entry carries the page, an example and a one-line rule of mine. The engine covers 52 of the 59 families in full and 7 in part, and

the corpus records the cause of each gap.

Why its own engine

KaiwaLens once shipped MeCab, and MeCab remains the better general analyser: more than twice as fast, retrainable, with several dictionaries and two decades of use (the table). If you are cutting a large body of text for research, use MeCab; I still do, to check my scores.

MeCab and Kaiwa Core		
	MeCab with UniDic	Kaiwa Core
Built for	General analysis, research and industry	A learner reading one line
Returns	Short units, each with part of speech, lemma and conjugation	Dictionary words, short units underneath
GSD units, F_1 (sentences exact, of 543)	98.88 (421)	99.65 (481) ^a
GSD long words, F_1	82.58 ^b	82.58
Analyser (ms a sentence) ^c	0.11	0.254
Joining costs (MB)	458	5.5
Grammar of a whole word	Each unit's conjugation type and form	Dictionary form, tense, polarity, formality, voice, pattern
Flexibility	Retrainable; several dictionaries; usable from most programming languages	UniDic only; one application
Track record	Two decades; the field's yardstick	Young; measured against it

^aPart of it from writing numbers whole, as GSD does. ^bMy scan over MeCab's units.

^cOn my PC, an Intel Core i7-7700.

Still, Kaiwa Core exists for three reasons. First, its unit is the learner's word, built in one pass in which the scan, analyser and grammar engine answer each other over one set of data. Second, its joining costs take 5.5 MB, so every copy of KaiwaLens carries all of them and cuts a line the same way. Third, it needs two things a general analyser can skip: near cuts, to choose between words, and numbers and unknown words kept whole for the reader.

One caution about the scores: part of Kaiwa Core's margin on GSD is convention. GSD writes 2010 as one unit; UniDic, which MeCab follows, writes it as four. Before Kaiwa Core kept numbers whole, it scored 99.01 to MeCab's 98.88. On GSD's long words the two tie at 82.58.

What it still gets wrong

The engine still handles four kinds of text differently from how a learner would want them. The analyser cuts kanji numerals a character at a time: 三百万 *san/hyaku/man*, "three million", where GSD cuts 三百|万 *sanbyaku/man*. The scan returns long compounds such as 研究開発 *kenkyū kaihatsu*, "research and development", in the parts a dictionary lists, so the learner sees two words, not the term. A katakana name with a middle dot, such as Paku Chiwon, stays whole, where GSD cuts it in three. And a word that is an entry of its own, such as *dekiru*, "can do", leads with that entry rather than with *suru*, whose potential it is.

And the persimmons? The scan now cuts the line *ko/tabetara*, and KaiwaLens offers *taberu*, conditional. No-body is eating alone.

References

- [1] M. Asahara et al. Universal Dependencies Version 2 for Japanese. *LREC*, 2018.
- [2] J. Breen. JMdict: A Japanese-Multilingual Dictionary. *COLING Workshop on Multilingual Linguistic Resources*, 2004.
- [3] Group Jamashii. *Nihongo bunkei jiten, kaiteiban* [A dictionary of Japanese sentence patterns, revised]. Kuroshio, 2023.
- [4] I. Iori, S. Takanashi, K. Nakanishi et al. *Chūjōkyū o oshieru hito no tame no Nihongo bupō handobukku* [A handbook of Japanese grammar for teachers of intermediate and advanced learners]. 3A Network, 2001.
- [5] S. Kaiser, Y. Ichikawa, N. Kobayashi et al. *Japanese: A Comprehensive Grammar*. 2nd ed. Routledge, 2013.
- [6] T. Kudo, K. Yamamoto, Y. Matsumoto. Applying Conditional Random Fields to Japanese Morphological Analysis. *EMNLP*, 2004.
- [7] B. P. Lavrent'ev. *Prakticheskaya grammatika yaponskogo yazyka* [A practical grammar of Japanese]. 3rd ed. Zhivoi yazyk, 2002.
- [8] S. Makino, M. Tsutsui. *A Dictionary of Advanced Japanese Grammar*. The Japan Times, 2008.
- [9] National Institute for Japanese Language and Linguistics. *UniDic*, unidic-cwj version 3.1.0. n.d.