The Logic of Kanji Lookup in a Japanese↔English Hyperdictionary

Harvey Abramson[1], Subhash Bhalla, Kiel Christianson, James Goodwin, Janet Goodwin[2], John Sarraille[3] & Lothar Schmitt[2]

[1] *Unisys Corp., G 140-9, 2476 Swedesford Road, Paoli, PA 19301, USA*
[2] *University of Aizu, Aizu Wakamatsu 96580, Japan*
[3] *California State University, Stanislaus, Turlock, California 95381, USA*

KEYWORDS: computational lexicography; hyperdictionary; language study

E-MAIL: harvey@tr.unisys.com
{bhalla, kiel, jamesg, janetg}@uaizu.ac.jp
john@ishi.csustan.edu
FAX NUMBER: +81-242-37-2735 [2]
PHONE NUMBER: +81-242-37-2713 [2]

Extended abstract

1 Kanji and traditional kanji lookup methods

In another paper submitted to this conference [Abramson et al, 1995] we discuss the general notion of "hyperdictionary", and in particular, a Japanese ↔English multimedia hyperdictionary based on modern large memory technology such as CD-ROMs or magneto-optical (MO) disks. By hyperdictionary we mean a relational and deductive database containing the words of a language or languages, together with an open-ended set of access and display methods so as to present at least their orthography, pronunciation, signification, part of speech, and use, their history, synonyms, homonyms, antonyms, derivation, relationships to one another, and any other aspect of the words which may be necessary for reference, teaching or study purposes. Additional information about the language or languages, including grammar, morphology, semantics, pragmatics, machine tractable representations, etc., as well as relevant information concerning geography, names, literature, society, culture, history and so on, is not excluded from the database. In this paper we concentrate on one aspect of the Japanese↔English hyperdictionary project, generalized kanji (Chinese character) lookup. We shall use some simple logic programming notation to describe the methods but even readers not familiar with Prolog or logic programming notation should be able to follow the discussion without trouble.

A student who is learning to read Japanese requires not only the familiar bilingual dictionaries, but also a character dictionary which gives the various pronunciation and definitions in English of each character, as well as pronunciations and definitions in English of compounds, words which are written with two or more characters. Characters in such a dictionary are classically arranged according to a system of 214 radicals or kanji components which divide the whole set of characters into subsets which are more or less semantically related. The traditional radical system is a formalization of the practice of creating characters from two simpler ones, one corresponding to a semantic notion, and one to a word that "rhymed" (so to speak) with the word being represented. 言 can mean "word" or "language" or "speak" and is one of the 214 radicals. Characters with this radical usually have some connection with speaking, words, or language use. Some of the characters which have this radical as a constituent, usually on the left side, but possibly on the bottom, are listed here:

言 – word, phrase, speech, statement
計 – plan, scheme, trick, meter, gauge
訂 – correct, decide
訃 – obituary, report of a death
記 – account, narrative, history
訓 – Japanese reading of a character, lesson, regulation
譽 – praise, honor, glory
警 – commandment, admonition
讖 – foretelling, presage, omen
讙 – convey, assign, deed, bequeath, postpone
讃 – praise, title or brief inscription on a picture

Many characters are composed of constituents each of which, or many of which, as far as the learner knows, might be *the* radical of the character. 訓 , the character for *kun*, the Japanese pronunciation of characters (as opposed to *on*, the pronunciation derived from Chinese), is composed of 言 the "word" element just mentioned on the left, and 川 , the element for "river", on the right. Both of these elements are radicals in the system, but which is *the* radical of the character? There are methods or algorithms for deciding the radical of a character in terms of the structure of a kanji when there are several constituents to the character (e.g., prefer left over right, so in the kanji for *kun* the radical must be the "speaking" constituent). Radicals are ordered according to the number of strokes needed to write them and there is a traditional ordering of radicals with the same number of strokes. In the total ordering of characters in such a dictionary, there is an entry for each radical followed in some order (e.g., stroke count)

by entries for characters containing that radical. Clearly, character dictionaries implemented as books require a linearization of the set of all characters and the notion of "radical of a character" coupled with associated stroke counts of the radicals and of the characters serve this purpose. Hyperdictionaries, essentially multilinear, do not require such an ordering of the characters to facilitate lookup. Rather, any identifiable component of a character may serve as a hyperindex to that character, generalizing the method of looking up a character. The next section deals with this topic.

2 The logic of kanji structure and generalized kanji lookup

It is obvious that the number of characters in the Japanese writing system is several orders of magnitude greater than the size of alphabets for most other languages. Although there is an official set of about 2,000 characters for everyday use, the number of characters actually in use depending on educational level, is anywhere from 5,000 to 20,000, and historically over 45,000 characters have been used. A recently published dictionary[1], for example, lists 21,084 characters. In order to represent these characters there are several two byte coding schemes. Most implementations of Prolog limit themselves to a one byte encoding scheme and are implicitly tailored to applications with the familiar Roman alphabet. This means that if kanji are to be used either as data or within programs, they must occur as quoted atoms, e.g., '漢字' with the underlying representation as

name ('漢字', [138, 191, 142, 154]).

Here the four integers in the list represent the four bytes needed to encode the two characters in the "Shift JIS" coding scheme used in Macintosh systems[2] .

As mentioned above, characters are traditionally ordered according to a scheme which classifies characters according to their radicals, a constituent of the character which usually carries some minimal semantic information. We can represent a character with the ternary predicate kanji where the first argument is the character, the second is its radical, and the third is a structural representation of the constituents of the character. (In the actual hyperdictionary application, there will be additional places for other information about the character, pronunciation, meaning, stroke number, calligraphic details, etc. Here we just want to demonstrate a more general character lookup method.)

kanji('寺',radical('寸'),tb('土','寸')).

This describes the character for "temple", showing its traditional radical, and an indication of its structure consisting of a top and bottom part. A few other structure indications are "lr" for left and right parts, "enc" for a part of the character which encloses another part as in:

kanji('持',radical('扌'),enc('扌','寺')).

There are several other structural descriptions which are not shown.

The components of the structural description may be characters, or a structural description of something which does not exist as an independent character as in:

kanji('峠',radical('山'),lr('山',tb('上','下'))).

The right hand constiuent of the character consists of a top and bottom arrangement of the characters for "above" and "below" which is not an independent character. The specification of "in" for such a case is:

in('山','峠').
in(tb('上','下'),'峠').
in('上',tb('上','下')).
in('下',tb('上','下')).

We have adapted a method originally given in [Dürst,1993] for representing characters graphically rather than by their pronunciation or by mnemonics (e.g., "ji" or "tera" or "temple" for '寺'). Dürst introduced this kind of representation largely for applications in font design, but it is very convenient in our hyperdictionary application.[3]

There are some problems in representing characters this way in that certain radicals may not always appear in the particular encoding scheme used. For example, the radical in

kanji('持',radical('扌'),lr('扌','寺')).

is a variant form of the character "手" (meaning hand) which itself does not appear as an independent character in the official scheme. This is sometimes solved by using a range of the two byte scheme which is undefined to represent characters (known as gaiji) which are needed in some particular application. Special action is required to make the characters printable.

In addition to the specification of "kanji" we also define what might be called a graph representation of the character and its constituents. The nodes of the graph are characters (including variant forms of characters which appear as radicals), and there is an arc connecting one node to another if one character is a direct constituent of the other. Thus

corresponding to:

```
kanji('寺',radical('寸'),tb('土','寸')).
```

we also have:

```
in('土','寺').        in('寸','寺').
```

The specification of "in" can be automatically derived from the structural definition in "kanji". Students of Japanese as a second language often find it hard at first to identify the radical of a character. However they often see patterns in kanji or parts of kanji other than the radical which could be used for character lookup if another part of the character had been chosen as the basis of the linear ordering of characters. For example, each of the following characters has '寺' as a constituent, and indirectly the constituents '土' and '寸'.

```
kanji('寺',radical('寸'),tb('土','寸')).
kanji('持',radical('扌'),lr('扌','寺')).
kanji('時',radical('日'),lr('日','寺')).
kanji('蒔',radical('艹'),tb('艹','時')).
kanji('侍',radical('亻'),lr('亻','寺')).
kanji('峙',radical('山'),lr('山','寺')).
kanji('侍',radical('亻'),lr('亻','寺')).
kanji('恃',radical('忄'),lr('忄','寺')).
kanji('塒',radical('土'),lr('土','時')).
kanji('痔',radical('疒'),enc('疒','寺')).
```

The specification of the graph for these characters is:

```
in('土','寺').        in('寸','寺').
in('扌','持').        in('寺','持').
in('日','時').        in('寺','時').
in('時','蒔').        in('艹','蒔').
in('寺','侍').        in('亻','侍').
in('山','峙').        in('寺','峙').
in('寺','侍').        in('亻','侍').
in('寺','恃').        in('忄','恃').
in('土','塒').        in('時','塒').
in('寺','痔').        in('疒','痔').
```

We define a predicate "in_deep" which relates constituents and characters which are connected by one or more "in" links[4]:

```
in_deep(X,Y) : in(X,Y).
in_deep(X,Y) : in(Z,Y),Z=..[_|List],member(X,List).
in_deep(X,Z) : in(X,Y),in_deep(Y,Z).
```

The second clause deals with artificial constructs such as

```
in(tb('上','下'),'峠').
```

We then define "is_part_of" which finds the characters which contain a given character directly or indirectly:

```
is_part_of (Kanji,PartOfList) :-
    setof (KanjiPlus,in_deep(Kanji,
    KanjiPlus),PartOfList).
```

And we define a predicate "components_of" which lists all the direct and indirect constituents of a character:

```
components_of(Kanji,ComponentsList) :-
    setof(KanjiMinus,in_deep(KanjiMin
us,
    Kanji),ComponentsList).
```

Thus:

```
is_part_of( '寺' , X).
```

yields

```
X = ['持', '時', '蒔', '侍', '峙', '侍', '恃', '塒', '痔']
```

and

```
components_of( '痔' , X).
```

yields

```
X = ['寺', '寸', '土', '疒']
```

while

```
components_of( '峠' , X)
```

yields

```
X = ['下', '山', '上', tb('上','下')]
```

Although there are some specialized[5] Japanese dictionaries which do group together characters with similar non-radical constitutents (for example, characters containing 寺), this kind of information about character structure is generally not accessible in traditional dictionaries other than by a tedious linear search through the book. Furthermore, this sort of information is not available in standard Japanese word processors. It must be said, however, that students of Japanese as a second language and particularly those who wish to learn to read tend to ask questions which native speakers do not.

A complete decomposition of a character into all its constituents is not necessary. There are some characters where the only easily identifiable part may be its radical. We are prototyping a Japanese-English CD-ROM character dictionary with about 6,000 characters. Part of the project will involve making a very user-friendly interface since we do not expect or want all users to be Prolog programers.

We have been using LPA MacProlog 32 [Johns, 92] running on Macintosh Quadras and PowerBooks, and also SICStus Prolog 2.1 [Andersson et al, 1993] running on various Sparc workstations under Unix. Porting to other Prologs and platforms should not be a problem.

I am indebted to one of the reviewers for suggesting that the metod introduced here may be ported to other logographic systems. For instance, in cuneiform, the reviewer comments, very similar problems arise, heightened by the unfortunate circumstance that many characters in actual "documents", i.e., clay tablets, are incomplete in that there are broken edges, smudges, etc., so that only subparts of a character can be seen. Having a retrieval system that is able to come up with ranked guesses would be highly useful in this field. There might also be applications in dealing with other kinds of fragmentary documents.

References

Abramson, H., Bhalla, S., Christianson, K., Goodwin, J., Goodwin, J., Sarraille, J., Schmitt, L. (1995) *Towards CD-ROM based Japanese↔English Dictionaries: Justification and Some Implemenation Issues*. Submitted to Joint International Conference ALLC-ACH '96.

Andersson, J. et al (1993), SICStus Prolog User's Manual Version 2.1 #8, SICS Technical Report T93:01, SICS, Kista, Sweden.

Dürst, M.J. (1993) Coordinate-independent font description using Kanji as an example, Electronic Publishing vol. 6, no. 3, pp. 133–143, Sept. 1993.

Johns, N. MacPROLOG (1992) Reference Manual version 4.1, Logic Programming Associates Ltd.

Notes

[1] Shindaijiten (新大字典) published by Kodansha.

[2] On the other hand, SICStus Prolog does permit use of the Extended Unix Code (EUC) scheme which permits kanji to be displayed without the use of quotes: 漢字 .

[3] Dürst's original representation should prove useful in supplying calligraphic instruction as to the order in which the pieces of the character should be written.

[4] "in_deep" is the transitive closure of "in".

[5] Such as Kanji no yomikata (漢字の読み方) published by Kadokawa Shoten.