The Predictive Mechanism

This is a difficult text on the mathematical background of text prediction as present in Htyp.

Author: j.verrips@planet.nl

Date: 1997, revised 2003.

Introduction

Much effort has been devoted to the study of word prediction that is now present in many mainstream computer applications such as most of the Windows family. It goes back at least to the Cornell synthesizer for programmers in 1971 –and many others- has inspired critical studies as those by Koester and Levine (1997), efforts to help dyslectics, efforts to combine with syntax (as by Hunnicut and Carlberger, 2001) and new applications as when combined with a stylus on an organiser (Dasher, 2002). This rather technical text only describes the peculiar techniques used in Htyp. It does not contain overview of relevant literature. For more background information please contact the author.

Word prediction in Htyp is a forgiving combination of several predicates and is combined with phrase prediction that uses First Letters Upper Case (FLUC) as well as the other predicates. If a scan matrix is used it may also be combined with letter-prediction. Mathematical background can be found in logic, a technique called dovetailing, and in recursion theory, where the mu-operator selects the smallest element of a set that has a certain property. Other analogies are so-called boolean short cut evaluation (programming) and analysis of ironic utterances (see Levinson, 1983 and others) and programming by demonstration using agents in the user interface (Cypher Ed. 1993). See figures 1 to 3 for an overview of Htyp with prediction of stored text.

```
1  .more-text
2  +more-poems
3  *menu
4  .some-phrases      -Some-Phrases
5  @
6  Let us go for a walk
```

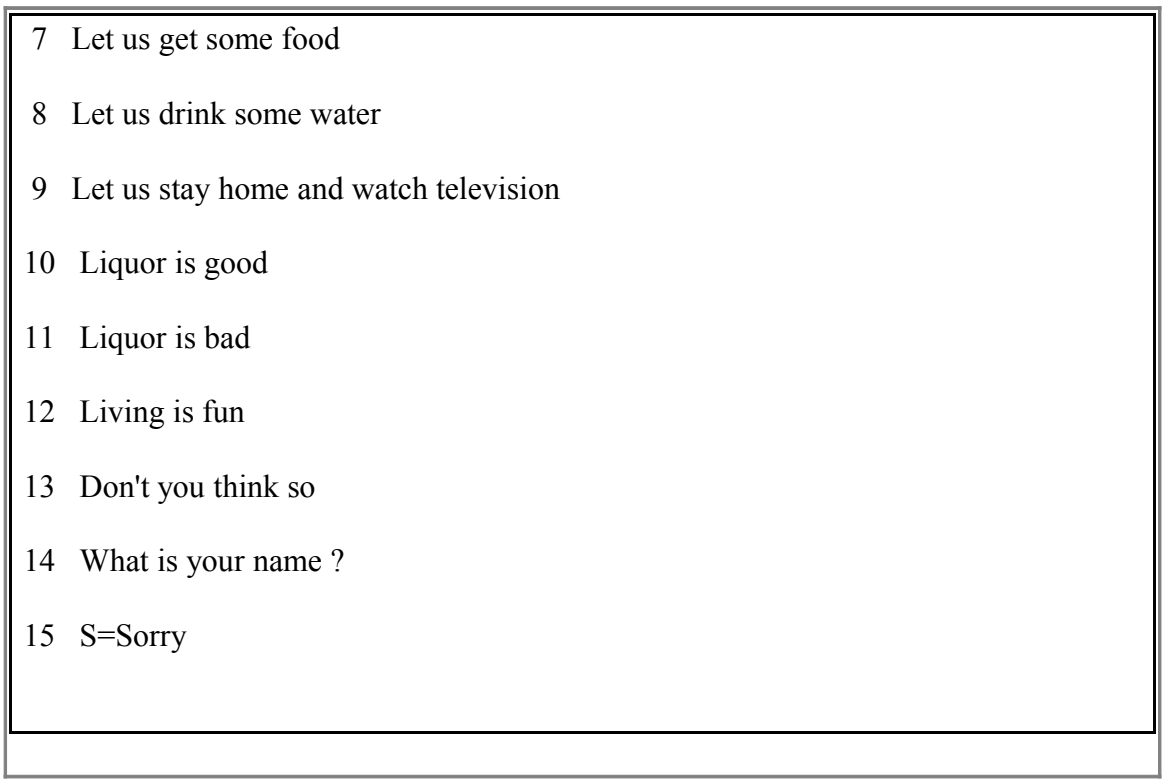
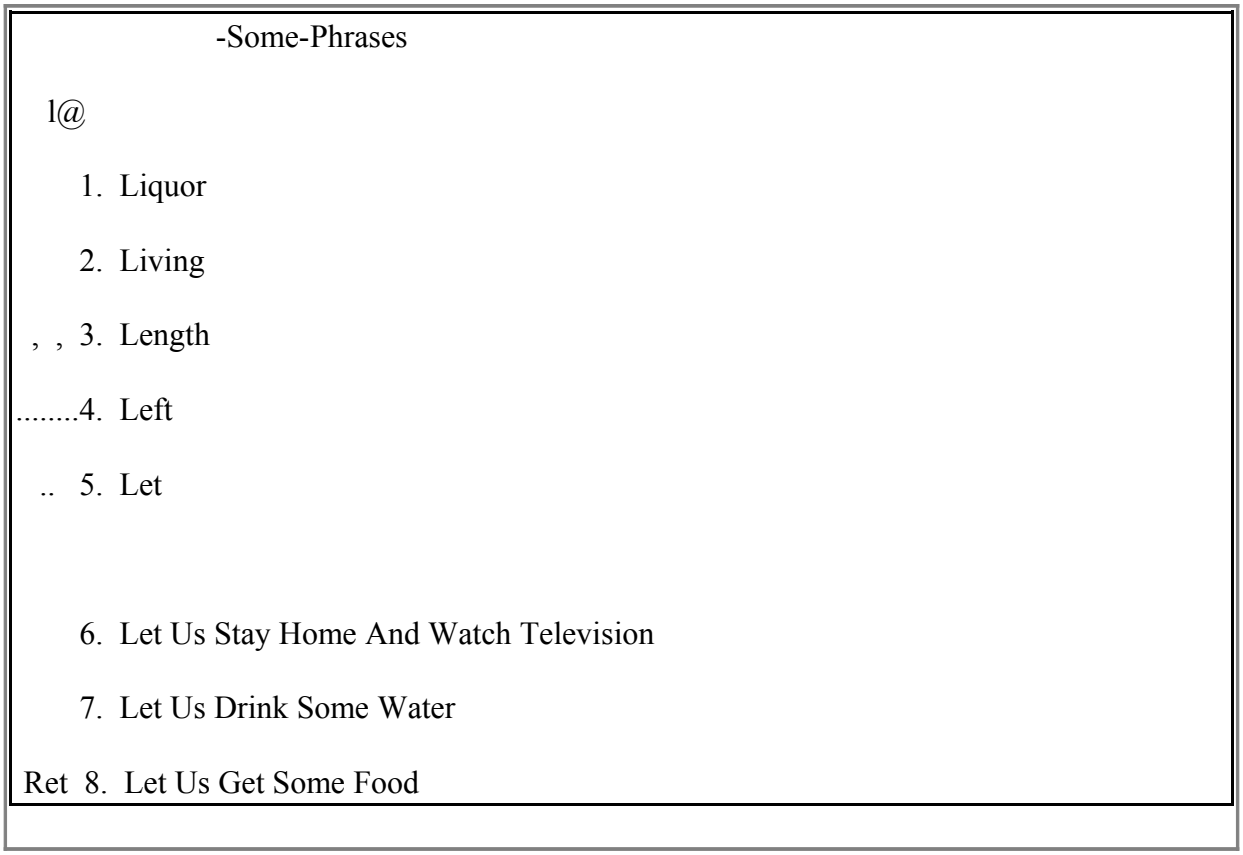


Figure 1. Screen image with margin menu. Text may be entered at the cursor @, located in paragraph -Some-Phrases. Phrases are voiced through the menu in the left margin, if a user hits *6* to *15*. *'1 '* reads in paragraph -More-Text, *2* reads in file more-poems.spr and discards the current text. The line *Menu is accessed by *3* and represents a hypertext link to a paragraph -Menu-. Paragraphs are also selected by word prediction. Underneath one sees uppercase *S* defined as "Sorry "; if a user enters S the machine will speak Sorry. Meaning of function keys is displayed at the bottom.



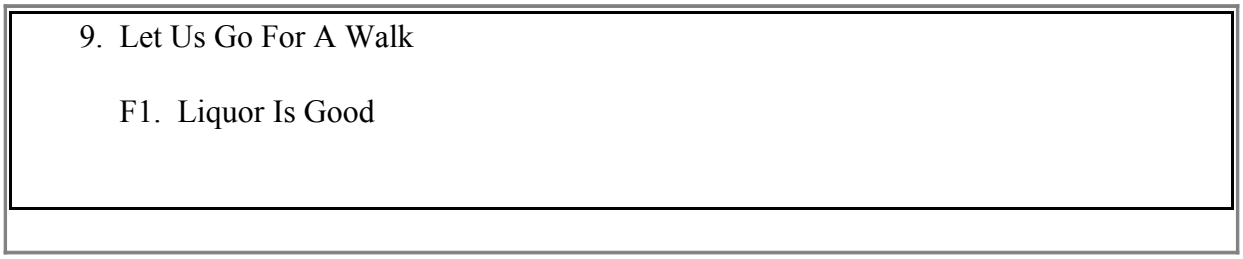


Figure 2. Same screen as figure 1, a user entered *l* and text prediction started up automatically. Two lists of suggestions are shown, the words above the phrases, long words first. *1* to *5* now select a word and period or comma select a recently used word. *Ret* selects the same phrase as *8*, the one that was most recently used or was most recently read in.

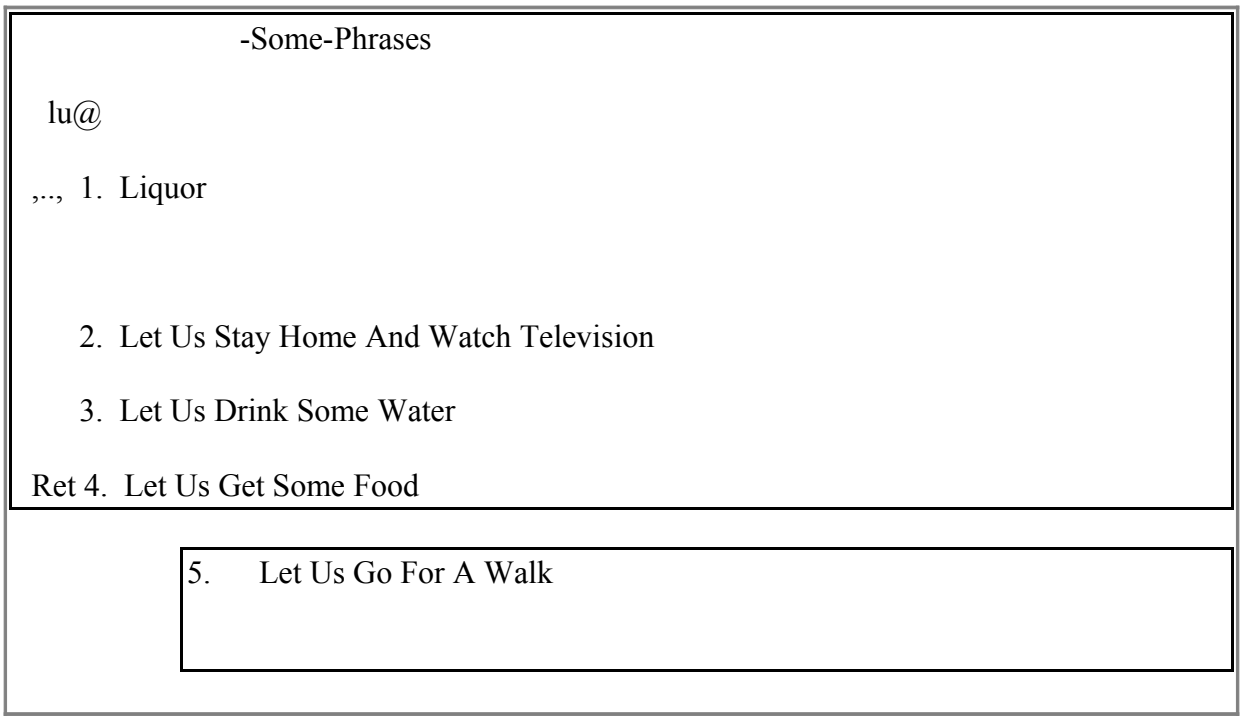


Figure 3. After *lu* only one word is shown and all phrases that start with L and contain a U. In this screen F10 selects "Let Us ", two words from the phrase bound to Ret. "lugF10" will select "Let us get ". Space will select "lu ".

Much literature exists on abbreviations and predictive systems for speech disabled subjects (VanderHeiden & Kelso, 1987; Darragh & Witten, 1992; Vanderheiden Lloyd, 1986; Stum, 1991; Demasco & McCoy, 1992). Abbreviations are used in other contexts and guidelines suggest a simple rule that is communicated to the operators and fixed length abbreviations (Ehrenreich & Porcu, 1982). Examples of rules are truncation and truncation plus selective vowel deletion (trn pls slc vwl dlt). Such rules propose an abbreviation like "trn" when a text is given such as "truncation".

Combining several predicates

The predictive mechanism of Htyp computes a set of texts when an abbreviation is proposed, almost the opposite mechanism. To do so, several sets of texts are computed using predicates, functions that accept

some items in a set and reject all others. A predicate defines a subset of all items with a certain property. If {"Hello", "How are you?", "Nice to see you"} is a set of texts, then the predicate 'starts with "H"' defines a subset of the first two items. 'Contains an "a"' now defines the subset with only the second item: {"How are you?"}. A subset is empty if the predicate defines a property that no item satisfies; sets or subsets of texts are displayed on screen as numbered lists.

We shall call a suggested text-string a_string and what the user typed in shall be called some_abbreviation. When a first letter of a word or a phrase is typed in the software initialises a list of suggestions from a database internal to the program. In our terminology it is a list of a_strings. When a second or a third letter is typed in, six different elementary predicates compute which suggestions comply and store results in six different lists. The first nonempty list from those six lists is displayed on screen.

Predicates p_1 to p_6 are used:

p_1. A_string contains all upper case letters from some_abbreviation and ordered from left to right and consecutively.

p_2. A_string contains all upper case letters from some_abbreviation ordered from left to right.

p_3. A_string contains all upper case letters from some_abbreviation in any ordering.

p_4. A_string contains some_abbreviation as a left substring.

p_5. A_string contains all letters, either upper case or lower case, from some_abbreviation and ordered from left to right.

p_6. A_string contains all letters, either upper case or lower case, from some_abbreviation in any ordering.

Predicates can be thought of as advisers. If the earlier one has nothing to say, the next one may speak: advisers are ordered hierarchically. If P_e defines a predicate that accepts no items and P_x, P_y, P_z are predicates the following rules apply.

. $P_x\ P_y$ means:

if the result of P_x is not empty then $P_x\ P_y = P_x$ else $P_x\ P_y = P_y$

. $P_x\ P_x = P_x$

. $P_x = P_x\ P_e$ (if no item is accepted, the empty set is proposed)

. $(P_x\ P_y)\ P_z = P_x\ (P_y\ P_z)$

The ordering $p_4\ p_5\ p_6$ (1a) is used for words but will not do to select phrases using FLUC; that would require something like $p_1\ p_2\ p_3\ p_4\ p_5\ p_6$ (1b). Some_abbreviations that consist of first letters of words rarely are left sub-strings of words as is required by p_4 and therefore 1b will generally result in short lists of suggestions on the screen. However, to type part of the first word of a phrase can be an effective way to filter a list of phrases using p_4 and need not exclude use of p_1 for other phrases. For words 1a is good enough and for phrases the implemented ordering is (p_4 or p_1 or p_2) $p_3\ p_5\ p_6$ (1c). This will work for a small database. In my experience, if p_2 is moved outside the brackets, beginning users will experience confusion. If a user types on presumably the word is not found and a larger database is searched, using p_5

after three keystrokes followed by p_4 p_5. This could be a two step process, searching a small database of recent and frequent words after two and a larger database after three or four keystrokes.

Users need only rarely mentally represent the interaction of individual predicates. In rare cases however one may be disappointed. If one types "exm" to select "example" the database might contain "exmarried". In this case "example" will NOT be suggested! This will usually be corrected if one types on, "exmp" will do.

See Table 9 for all the some_abbreviations that one might use to suggest the a_string, "Dag Goeden Avond" (Hello good evening). All other predicates are ignored as well as all other texts. Predicates p_5 and p_6 have very long lists of some_abbreviations but will only be used infrequently as mostly p_1 and p_4 will do. No matter how selective the predicates are, users will need a numbered list on the screen to pick one from pairs such as "I Will Not Go There " and "I Will Never Go There ". One may select the last one after "iwnv" using p_1 and then p_5.

predicate	p_1	p_2	p_3	p_4	p_5
abbreviations	d	d	d	d	d
	dg	da	da	da	da
	dga	dg	dag	dg	dg
		dga		dv	dov
		dag		dov	dvo
				..	...

Table 1. Texts one could type using a single predicate to suggest "Dag goeden avond".

Studies of effectiveness

To get an impression of the effectiveness of the FLUC- algorithm, an English article was processed. Variable length abbreviations were determined of every line that consist of first letters of every word. These abbreviations were sorted and counted. 120 of 541 lines had unique two letter abbreviations, 389 of 494 lines had unique three letter abbreviations (541 - 494 = 47 lines did consist of two words only and were ignored), 433 of 443 sentences had unique four letter abbreviations, ignoring 98 (541 - 443 = 98) lines with less than four words. A unique three letter abbreviation means that to type in three first letters will create a list of a single suggestion. I conclude that to select individual lines from this short article the first predicate would be remarkably effective. Selection from long lists of known phrases using FLUC will require few keystrokes.

To compare word prediction with and without recency enhancement another study was done. A simple text was typed in either with word prediction only or word prediction plus recency marking and only short words after the first keystroke, a property that is rarely used. In both cases, no other rate enhancements were used and maximum listlength was eight. The text was taken from (Higginbotham 1992), fourth grade example with average word length at 5.4, simple words and rich in punctuation. With word prediction-only 406 keystrokes selected a total length of 625 (116 words, 35 % saving). Word prediction-plus-recency achieved 39% keystroke reduction, 30 out of 81 word predictions ended through recency selections. Results would have been slightly better if use of other rate enhancing techniques would have been allowed as in (Higginbotham, 1992), but not much.

The forgiving implementation of word prediction might allow to rapidly decrease list length. To test this hypothesis the second chance algorithm was set off and a file was used with about 630 English words. A set of 20 words was prepared with a vowel as a second letter, picking the longest such word for each letter in the alphabet, if present. List length was then compared after either the first two characters or first character, next consonant. Thus "de" and "dt" would be used to find "determination". In the second condition list length was equal three times, longer two times and shorter fifteen times. This is a significant difference (token test; alfa = 0.05; Wilcoxon rank test alfa = 0.005). Total list length after twenty times two keystrokes was 87 in the first and 40 in the second condition. However, in three out of twenty cases the second condition required another keystroke to display the word one needed. In those cases other words interfered with selection. For example, "argument ", that starts with "ar" might interact with selection of "aeroplanes " in the second condition. Note that actual users may use the system in very different ways.

Manipulating recency marking

One might search for improvements to word prediction in semantics. Disabled subjects may experience difficulty to indicate persons or objects in a new communicative context labeled frame by (Higginbotham & Wilkins 1996). Traditionally frame means 'perceived context' as in Tannen (1993), Goffman (1974) or Givón (1989). Communicative context might influence word prediction through adaptation of recency information. As an example, consider the phrase "John went into the restaurant. He ordered a hamburger and a coke. He asked the waitress for the check and left." This phrase has often been quoted to illustrate the concept of a frame: the term restaurant makes one infer there must be a specific (the) waitress. If copied twice, to see the effect of quasi optimal adjusting of recency marking, 18 out of 21 words can be selected the second time with just the first character then either '.' or ','. To exploit this effect, recency may be updated when one reads in a new paragraph. It will not simplify things for the users, however.

Combining word prediction with scanning

Some report that with very low input rates word prediction can be valuable. However, these input techniques, such as Morse code or scanning, compete with word prediction for the attention of the user. In many small studies continued training and adapting of all parameters to the user showed little advantage for word prediction in a typical copy task, or even showed two-bit quartering faster without it. In conversations where specific names are often repeated and for some individual users word prediction may be faster and certainly reduces the number of keystrokes for people with rapid muscle-fatigue.

Combining letter prediction with scanning

Scanning, especially two-bit quartering, can easily be combined with letterprediction, for instance using relative frequencies of bigrams, using the last character as a parameter, or using bigrams or trigrams combined with wordlength. Though this is implemented in Htyp, can be bound to a shortcut too and allow

about 40% first hits, I refer to other sources for further data on the subject of letterprediction as it has nothing to do with the mechanisms described above.

References

Beattie, W., McKinlay, A., Arnott, J. & Gregor, P. (1994). An investigation of the use of recency in word prediction algorithms. *Isaac 1994,* 496-498.

Card, S., Moran, T.P. & Newell, A. (1983). *The psychology of human computer interaction.* Lawrence Erlbaum.

Cypher, A. (Ed.) (1993). *Watch what I do, programming by demonstration.* MIT Press, Cambridge, Massachusetts

Dasher, www.inference.phy.cam.ac.uk/dasher/.

Darragh, J.J & Witten, I.H. (1992). *The reactive keyboard.* Cambridge Series on HCI.

Ehrenreich, S.L. & Porcu, T. (1982). Abbreviations for automated systems. *Directions in Human Computer Interaction.* Albert Badre, Ben Shneiderman, eds. 111-135.

Givón, T. (1989). *Mind, code and context. Essays in Pragmatics.* London, Hillsdale, Lawrence Erlbaum Associates.

Goffman, E. (1974). Frame analysis. An essay on the Organisation of Experience. Harvard University Press.

Higginbotham, D.J. & Wilkins, D.P. (1996). A communication frame based language system for AAC. *Paper, Max Planck Institute,* Nijmegen, Netherlands.

Horstmann Koester, H.M., & Levine, S.P. (1996). Effect of a word prediction feature on user performance. *Augmentative and Alternative Communication, 12*, 155-168.

Horstmann Koester, H.M., & Levine, S.P. (1997). Keystroke-Level Models for User Performance. *Augmentative and Alternative Communication, 13*, 239-257.

Hunnicutt, S. & Carlberger, J. (2001). Improving word prediction using Markov models and heuristic methods. *AAC, Augmentative and Alternative Communication,* 17, 255-264.

Tannen, D. Ed. (1993). *Framing in Discourse.* Oxford: Oxford University Press.