BetaText: An Event Driven Text Processing and Text Analyzing System

Benny Brodda

Department of Linguistics
University of Stockholm
S-106 91 Stockholm, Sweden

Abstract. BetaText can be described as an event driven production system, in which (combinations of) text events lead to certain actions, such as the printing of sentences that exhibit certain, say, syntactic phenomena. The analysis mechanism used allows for arbitrarily complex parsing, but is particularly suitable for finite state parsing. A careful investigation of what is actually needed in linguistically relevant text processing resulted in a rather small but carefully chosen set of "elementary actions" to be implemented.

1. Introduction. The field of computational linguistics seems, roughly speaking, to comprise two rather disjoint subfields, one in which the typical researcher predominantly occupies himself with problems such as "concordance generation", "backward sorting", "word frequencies" and so on, whereas the prototypic researcher in the other field has things like "parsing strategies", "semantic representations" on top of his mind.

This division into almost disjoint subfields is to be regretted, because we all are (or should be) students of one and the same thing - language as it is. The responsibility for this sad state of affairs can probably be divided equal by the researchers in these two subfields: the "concordance makers" because they seem so entirely happy with rather unsophisticated computational tools developed already in the sixties (and which allow the researcher to look at words or word forms only, and their distribution), and the theoreticians because they seem so obsessed with the idea of developing their fantastic models of language in greater and greater detail, a model that at a closer scrutiny is found to comprise a lexicon of, at best, a couple of hundred words, and covering, at best, a couple of hundred sentences or so. No wonder that the researchers in these two camps think so little of each other.

One way of closing the gap can be to develop more sophisticated tools for the investigation of actual texts; there is a need for the theoreticians to test to what extent their models actually cover actual language (and to get impulses from actual language), and there is a need for the "practicioners" to have simple tools for investigating more complex structures in texts than mere words and word forms. BetaText is an attempt to provide tools for both those needs.

2. Text events and text operations. BetaText is a system intended both for scientific investigations (or analyses) of texts, and text processing in a more technical sense, such as reformatting, washing spurious characters away, and so on. Due to the internal organisation of the system, even large texts can be run at a reasonable cost (cf. Brodda-Karlsson 1981). In this section we give some general definitions, and show their consequences for BetaText.

An elementary (text) event consists of the observation of one specified, concrete string in the text. The system records such an observation through the introduction of a specific internal state (or through a specific change of the internal state), the internal state being an internal variable that can take arbitrary, positive integral values.

Arbitrarily chosen states (sets of states, in fact) can be tied to specific activities (or processes), and each time such a state is introduced (i.e. the internal state becomes equal to that state) the corresponding process is activated. Such states are called action states.

A complex event (or just event, even elementary events can be complex in the sense used here) is the combined result of a sequence of interconnected elementary events, possibly resulting in an action state.

In BetaText all this is completely controlled by a set of production rules (cf. Smullyan 1961) of the type:

(<string>, <set of states>) ->
(<new string>, <move>, <new state>, <action(s)>)

where <string> is the string that is to be observed, <set of states> a condition for applying the rule, viz. that the current internal state belongs to this set; it is via such conditions that the chaining of several elementary events into one complex event is achieved. <new string> is a string that is substituted for the observed string (the default is that the original string is retained), <move> is a directive to the system where (in the text) it shall continue the analysis; the default is immediately to

the right of the observed string. <new state> is the state that the application of the rule results in. <action(s)>, finally, is the set of actions that are invoked through the application of the rule; the action part of a rule is only indirectly present, as the actions are invoked if the resulting state of the rule belongs to the corresponding action sets.

The actual rule format also allows for context conditions (and not only state conditions as is indicated above), but it is the way state conditions are evaluated that makes the Beta formalism as strong as it is; cf. Brodda-Karlsson 81 and Brodda 86.

3. Internal organization. The text corpus to be analyzed is assumed to adhere to a format that more or less has become the international standard, where each line of the running text is assumed to be preceded by a fixed length line head, usually containing some kind of line identifier. (Typically a document identifier + a running line enumeration.) The running text is presented to the user (well, the program) as if consisting of one immensely long string (without the line heads) and in which the original line divisions are represented by number signs (or some other unique symbol not appearing otherwise in the text). The original line heads are also lined up in an internal queue, and the correspondence between lines and line heads is retained via pointers. (This is completely hidden for the user.)

The system has (or can be thought to have) a cursor that is moved to and fro inside the text. At start up, the cursor is placed at the beginning of the text, and the internal state is initiated to 1; from there on, the user has complete control (via the application of rules) of the cursor and the internal state. (The the cursor is, however, automatically moved rightwards in the text as long as there is no rule applicable.)

Output is again organized in the line head, text line format, but now the line head may be given an internal structure, viz. as

<-kwoc-field-><-id-field-><-enum-field->

where the id-field corresponds to the line head of the input text, the kwoc-field may be filled with material from the text itself (e.g. words if one is making a word concordance of the KWOC-type), and the enum(eration)-field, if defined, contains a running enumeration. These fields - if defined - must be explicitly filled with corresponding material, through the application of action rules, which we describe in the next section.

4. Actions. The actions that can be invoked through the applications of rules can be divided into four different groups: i) analysis actions, actions that control in detail how the analysis proceedes internally; ii) block and line head actions, actions through which one can move material from the text into the line head (and vice versa); iii) output (or print) actions, actions which result in some kind of output, and, finally, iv) count actions.

The analysis actions control how the analysis is to proceed internally. In an accumulating rule the resulting state is added to (or subtracted from) the current internal state, rather than assigned to it (which is the default case). In stack rules some important internal parameters (internal state and the present positions of the cursor and the flag; cf. below) are pushed onto or popped from an internal stack. Through the use of stack actions ATN-like grammars can be writtern very conveniently in the Beta formalism (cf. Brodda 86.)

Block and line head actions: A flag setting action implies that an internal pointer is set to the present position of the cursor. The flag can later be the target of move directives (i.e. the cursor can be moved back to the flag). The area from the flag to the current position of the cursor can also be moved out into the kwoc-field as one block in a kwoc action.

In output actions the output can be formatted in many convenient ways. In kwic-format, for instance, always exactly one line at a time is output, and in such a way that the cursor is positioned in a fixed column.

BetaText has not in itself any advanced statistical apparatus, but one can at least count things, and perhaps in a little bit more advanced way than is usually the case. Arbitrary sets of states can be assigned specific registers (up to 128 such sets can presently be defined), and whenever any of these states is introduced, the corresponding register is raised by one. The content of the registers are then presented in a log file that accompanies all sessions with BetaText.

Several examples of actual analyses will be shown at the conference.

REFERENCES:

Brodda, B. & Karlsson, F. "An Experiment with Automatic Morphological Analysis of Finnish", Department of Linguistics, University of Helsinki, Helsinki 1981.

Brodda B. "An Experiment with Heuristic Parsing of Swedish" in Papers from the Seventh Scandinavian Conference of Linguistics, Publications No. 10, Department of Linguistics, University of Helsinki, Helsinki 1983.

Brodda, B. "BetaText: An event Driven Text Processing System and Text Analyzing System", to appear in Papers From the English Language and Literature department, University of Stockholm, Stockholm 1986.

Smullyan, R.M. "Theory of Formal Systems", Annals of Math. Studies, New York 1961.