BASEBALL: AN AUTOMATIC QUESTION-ANSWERER

Bert F. Green, Jr., Alice K. Wolf, Carol Chomsky, and Kenneth Laughery
Lincoln Laboratory*, Massachusetts Institute of Technology
Lexington 73, Massachusetts

Summary

Baseball is a computer program that answers questions phrased in ordinary English about stored data. The program reads the question from punched cards. After the words and idioms are looked up in a dictionary, the phrase structure and other syntactic facts are determined for a content analysis, which lists attribute-value pairs specifying the information given and the information requested. The requested information is then extracted from the data matching the specifications, and any necessary processing is done. Finally, the answer is printed. The program's present context is baseball games; it answers such questions as "Where did each team play on July 7?"

Introduction

Men typically communicate with computers in a variety of artificial, stylized, unambiguous languages that are better adapted to the machine than to the man. For convenience and speed, many future computer-centered systems will require men to communicate with computers in natural language. The business executive, the military commander, and the scientist need to ask questions of the computer in ordinary English, and to have the computer answer questions directly. *Baseball* is a first step toward this goal.

Baseball is a computer program that answers questions posed in ordinary English about data in its store. The program consists of two parts. The linguistic part reads the question from a punched card, analyzes it syntactically, and determines what information is given about the data being requested. The processor searches through the data for the appropriate information, processes the results of the search, and prints the answer.

The program is written in IPL-V[1], an information processing language that uses lists, and hierarchies of lists, called list structures, to represent information. Both the data and the dictionary are list structures, in which items of information are expressed as attribute-value pairs, e.g., Team = Red Sox.

The program operates in the context of baseball data. At present, the data are the month, day, place, teams and scores for each game in the American League for one year. In this limited context, a small vocabulary is sufficient, the data are simple, and the subject-matter is familiar.

Some temporary restrictions were placed on the input questions so that the initial program could be relatively straightforward. Questions are limited to a single clause; by prohibiting structures with dependent clauses the syntactic analysis is considerably simplified. Logical connectives, such as *and*, *or*, and *not*, are prohibited, as are constructions implying relations like *most* and *highest*. Finally, questions involving *sequential* facts, such as "Did the Red Sox ever win six games *in a row*?" are prohibited. These restrictions are temporary expedients that will be removed in later versions of the program. Moreover, they do not seriously reduce the number of questions that the program is capable of answering. From simple questions such as "Who did the Red Sox lose to on July 5?" to complex questions such as "Did every team play at least once in each park in each month?" lies a vast number of answerable questions.

Specification List

Fundamental to the operation of the baseball program is the concept of the *specification list*, or *spec list*. This list can be viewed as a canonical expression for the meaning of the question; it represents the information contained in the question in the form of attribute-value pairs, e.g., Team = Red Sox. The spec list is generated from the question by the linguistic part of the program, and it governs the operation of the processor. For example, the question "Where did the Red Sox play on July 7?" has the spec list:

Place = ?
Team = Red Sox
Month = July
Day = 7

Some questions cannot be expressed solely in terms of the main attributes (Month, Day, Place, Team, Score and Game Serial Number), but require some modification of these attributes. For example, on the spec list of "What teams won 10

*Operated with support from the U.S. Army, Navy, and Air Force.

From the collection of the Computer History Museum (www.computerhistory.org)

games in July?", the attribute Team is modified by Winning, and Game is modified by Number of, yielding

$$\text{Team}_{(\text{winning})} = ?$$

$$\text{Game}_{(\text{number of})} = 10$$

$$\text{Month} = \text{July}$$

Dictionary

The dictionary definitions, which are expressed as attribute-value pairs, are used by the linguistic part of the program in generating the spec list. A complete definition for a word or idiom includes a part of speech, for use in determining phrase structure; a meaning, for use in analyzing content; an indication of whether the entry is a question-word, e.g., *who* or *how many*; and an indication of whether a word occurs as part of any stored idiom. Separate dictionaries are kept for words and idioms, an idiom being any contiguous set of words that functions as a unit, having a unique definition.

The meaning of a word can take one of several forms. It may be a main or derived attribute with an associated value. For example, the meaning of the word *Team* is Team = (blank), the meaning of *Red Sox* is Team = Red Sox, and the meaning of *who* is Team = ?. The meaning may designate a subroutine, together with a particular value, as in the case of modifiers such as *winning*, *any*, *six*, or *how many*. For example, *winning* has the meaning Subroutine A1 = Winning. The subroutine, which is executed by the content analysis, attaches the modifier Winning to the attribute of the appropriate noun. Some words have more than one meaning; the word *Boston* may mean either Place = Boston or Team = Red Sox. The dictionary entry for such words contains, in addition to each meaning, the designation of a subroutine that selects the appropriate meaning according to the context in which the word is encountered. Finally, some words such as *the*, *did*, *play*, etc., have no meaning.

Data

The data are organized in a hierarchical structure, like an outline, with each level containing one or more items of information. Relationships among items are expressed by their occurrence on the same list, or on associated lists. The main heading, or highest level of the structure, is the attribute Month. For each month, the data are further subdivided by place. Below each place under each month is a list of all games played at that place during that month. The complete set of items for one game is found by tracing one path through the hierarchy, i.e. one list at each level. Each path contains values for each of six attributes, e.g.:

```
Month = July
   Place = Boston
      Day             = 7
      Game Serial No. = 96
      (Team = Red Sox, Score = 5)
      (Team = Yankees, Score = 3)
```

The parentheses indicate that each Team must be associated with its own score, which is done by placing them together on a sublist.

The processing routines are written to accept any organization of the data. In fact, they will accept a non-parallel organization in which, for example, the data might be as above for all games through July 31, and then organized by place, with month under place, for the rest of the season. The processing routines will also accept a one-level structure in which each game is a list of all attribute-value pairs for that game. The possibility of hierarchical organization was included for generality and potential efficiency.

Details of the Program

The program is organized into several successive, essentially independent routines, each operating on the output of its predecessor and producing an input for the routine that follows. The linguistic routines include question read-in, dictionary look-up, syntactic analysis, and content analysis. The processing routines include the processor and the responder.

Linguistic Routines

Question Read-in. A question for the program is read into the computer from punched cards. The question is formed into a sequential list of words.

Dictionary Look-up. Each word on the question list is looked up in the word dictionary and its definition copied. Any undefined words are printed out. (In the future, with a direct-entry keyboard, the computer can ask the questioner to define the unknown words in terms of words that it knows, and so augment its vocabulary.) The list is scanned for possible idioms; any contiguous words that form an idiom are replaced by a single entry on the question list, and an associated definition from the idiom dictionary. At this point, each entry on the list has associated with it a definition, including a part of speech, a meaning, and perhaps other indicators.

Syntax. The syntactic analysis is based on the parts of speech, which are syntactic categories assigned to words for use by the syntax

From the collection of the Computer History Museum (www.computerhistory.org)

routine. There are 14 parts of speech and several ambiguity markers.

First, the question is scanned for ambiguities in part of speech, which are resolved in some cases by looking at the adjoining words, and in other cases by inspecting the entire question. For example, the word *score* may be either a noun or a verb; our rule is that, if there is no other main verb in the question, then *score* is a verb, otherwise it is a noun.

Next, the syntactic routine locates and brackets the noun phrases, [ ] , and the prepositional and adverbial phrases, ( ). The verb is left unbracketed. This routine is patterned after the work of Harris and his associates at the University of Pennsylvania.[2] Bracketing proceeds from the end of the question to the beginning. Noun phrases, for example, are bracketed in the following manner: certain parts of speech indicate the end of a noun phrase; within a noun phrase, a part of speech may indicate that the word is within the phrase, or that the word starts the phrase, or that the word is not in the phrase, which means that the previous word started the phrase. Prepositional phrases consist of a preposition immediately preceding a noun phrase. The entire sequence, preposition and noun phrase, is enclosed in prepositional brackets. An example of a bracketed question is shown below:

[How many games] did

[the Yankees] play (in [July])?

When the question has been bracketed, any unbracketed preposition is attached to the first noun phrase in the sentence, and prepositional brackets added. For example, "Who did the Red Sox lose to on July 5?" becomes "(To [who]) did [the Red Sox] lose (on [July 5])?"

Following the phrase analysis, the syntax routine determines whether the verb is active or passive and locates its subject and object. Specifically, the verb is passive if and only if the last verb element in the question is a main verb and the preceding verb element is some form of the verb *to be*. For questions with active verbs, if a *free* noun phrase (one not enclosed in prepositional brackets) is found between two verb elements, it is marked *Subject*, and the first free noun phrase in the question is marked *Object*. Otherwise the first free noun phrase is the subject, the next, if any, is the object. For passive verbs, the first free noun phrase is marked *Object* (since it is the object in the active form of the question) and all prepositional phrases with the preposition *by* have the noun phrase within them marked *Subject*. If there is more than one, the content analysis later chooses among them on the basis of meaning.

Finally, the syntactic analysis checks to see if any of the words is marked as a question word. If not, a signal is set to indicate that the question requires a *yes/no* answer.

Content Analysis. The content analysis uses the dictionary meanings and the results of the syntactic analysis to set up a specification list for the processing program. First any subroutine found in the meaning of any word or idiom in the question is executed. The subroutines are of two basic types; those that deal with the meaning of the word itself and those that in some way change the meaning of another word. The first chooses the appropriate meaning for a word with multiple meanings, as, for example, the subroutine mentioned above that decides, for names of cities, whether the meaning is Team = A_t or Place = A_p. The second type alters or modifies the attribute or value of an appropriate syntactically related word. For example, one such subroutine puts its value in place of the value of the main noun in its phrase. Thus Team = (blank) in the phrase *each team* becomes Team = each; in the phrase *what team*, it becomes Team = ?. Another subroutine modifies the attribute of a main noun. Thus Team = (blank) in the phrase *winning team* becomes Team$_{\text{(winning)}}$ = (blank). In the question "Who beat the Yankees on July 4?", this subroutine, found in the meaning of *beat*, modifies the attribute of the subject and object, so that Team = ? and Team = Yankees are rendered Team$_{\text{(winning)}}$ = ? and Team$_{\text{(losing)}}$ = Yankees. Another subroutine combines these two operations: it both modifies the attribute and changes the value of the main noun. Thus, Game = (blank) in the phrase *six games* becomes Game$_{\text{(number of)}}$ = 6, and in the phrase *how many games* becomes Game$_{\text{(number of)}}$ = ?.

After the subroutines have been executed, the question is scanned to consolidate those attribute-value pairs that must be represented on the specification list as a single entry. For example, in "Who was the winning team..." Team = ? and Team$_{\text{(winning)}}$ = (blank) must be collapsed into Team$_{\text{(winning)}}$ = ?. Next, successive scans will create any sublists implied by the syntactic structure of the question. Finally, the composite information for each phrase is entered onto the spec list. Depending on its complexity, each phrase furnishes one or more entries for the list. The resulting spec list is printed in outline form, to provide the questioner with some intermediate feedback.

Processing Routine

Processor. The specification list indicates to the processor what part of the stored data is relevant for answering the input question. The

From the collection of the Computer History Museum (www.computerhistory.org)

processor extracts the matching information from the data and produces, for the responder, the answer to the question in the form of a list structure.

The core of the processor is a search routine that attempts to find a match, on each path of a given data structure, for all the attribute-value pairs on the spec list; when a match for the whole spec list is found on a given path, these pairs relevant to the spec list are entered on a found list. A particular spec list pair is considered matched when its attribute has been found on a data path and, either the data value is the same as the spec value, or the spec value is ? or each, in which case any value of the particular attribute is a match. Matching is not always straightforward. Derived attributes and some modified attributes are functions of a number of attributes on a path and must be computed before the values can be matched. For example, if the spec entry is Home Team = Red Sox, the actual home team for a particular path must be computed from the place and teams on that path before the spec value Red Sox can be matched with the computed data value. Sublists also require special handling because the entries on the sublist must sometimes be considered separately and sometimes as a unit in various permutations.

The found list produced by the search routine is a hierarchical list structure containing one main or derived attribute on each level of each path. Each path on the found list represents the information extracted from one or more paths of the data. For example, for the question "Where did each team play in July?", a single path exists, on the found list, for each team which played in July. On the level below each team, all places in which that team played in July occur on a list that is the value of the attribute Place. Each path on the found list may thus represent a condensation of the information existing on many paths of the search data.

Many input questions contain only one query, as in the question above, i.e., Place = ?. These questions are answered, with no further processing, by the found list produced by one execution of the search routine. Others require simple processing on all occurrences of the queried attribute on the generated found list. The question "In how many places did each team play in July?" requires a count of the places for each team, after the search routine has generated the list of places for each team.

Other questions imply more than one search as well as additional processing. For a spec attribute with the value every, a comparison with a list of all possible values for that attribute must be made after the search routine has generated lists of found values for that attribute. Then, since only those found list paths for which all possible values of the attribute exist should remain on the found list as the answer to the question, the search routine, operating on this found list as the data, is again executed. It now generates a new found list containing all the data paths for which all possible values of the attribute were found. Likewise, questions involving a specified number, such as 4 teams, imply a search for which teams, a count of the teams found on each path, and a search of the found list for paths containing 4 teams.

In general, a question may contain implicit or explicit queries. Since these queries must be answered one at a time, several searches, with intermediate processing, are required. The first search operates on the stored data while successive searches operate on the found list generated by the preceding search operation. As an example, consider the question "On how many days in July did eight teams play?" The spec list is

Day (number of) = ? ;
Month = July;
Team (number of) = 8 .

On the first pass, the implicit question which teams is answered. The spec list for the first search is

Day = Each;
Month = July;
Team = ? .

The found data is a list of days in July; for each day there is a list of teams that played on that date. Following this search, the processor counts the teams for each day and associates the count with the attribute Team. On the second search, the spec list is

Day = ? ;
Month = July;
Team (number of) = 8 .

The found data is a list of days in July on which eight teams played. After this pass, the processor counts the days, addsthe count to the found list and is finished.

Responder. No attempt has yet been made to respond in grammatical English sentences. Instead, the final found list is printed, in outline form. For questions requiring a yes/no answer, YES is printed along with the found list. If the search routine found no matching data, NO is printed for yes/no questions, and NO DATA for all other cases.

Discussion

The differences between Baseball and both automatic language translation and information retrieval should now be evident. The linguistic part of the baseball program has as its main goal the understanding of the meaning of the question as embodied in the canonical specification list. Syntax must be considered and ambiguities resolved in order to represent the meaning adequately. Translation programs have a different goal: transforming the input passage from one natural language to another. Meanings must be considered and ambiguities resolved to the extent that they effect the correctness of the final translation. In general, translation programs are concerned more with syntax and less with meaning than the Baseball program.

Baseball differs from most retrieval systems in the nature of its data. Generally the retrieval problem is to locate relevant documents. Each document has an associated set of index numbers describing its content. The retrieval system must find the appropriate index numbers for each input request and then search for all documents bearing those index numbers. The basic problem in such systems is the assignment of index categories. In Baseball, on the other hand, the attributes of the data are very well specified. There is no confusion about them. However, Baseball's derived attributes and modifiers imply a great deal more data processing than most document retrieval programs. (Baseball does bear a close relation with the ACSI-MATIC system discussed by Miller et al at the 1960 Western Joint Computer Conference.[3])

The concept of the spec list can be used to define the class of questions that the baseball program can answer. It can answer all questions whose spec list consists of attribute-value pairs that the program recognizes. The attributes may be modified or derived, and the values may be definite or queries. Any combination of attribute-value pairs constitutes a specification list. Many will be nonsense, but all can be answered. The number of questions in the class is, of course, infinite, because of the numerical values. But even if all numbers are restricted to two digits, the program can answer millions of meaningful questions.

The present program, despite its restrictions, is a very useful communication device. Any complex question that does not meet the restrictions can always be broken up into several simpler questions. The program usually rejects questions it cannot handle, in which case the questioner may rephrase his question. He can also check the printed spec list to see if the computer is on the right track, in case the linguistic program has erred and failed to detect its own error. Finally, he can often judge whether the answer is reasonable.

Next Steps

No important difficulty is expected in augmenting the program to include logical connectives, negatives, and relation words. The inclusion of multiple-clause questions also seems fairly straightforward, if the questioner will mark off for the computer the boundaries of his clauses. The program can then deal with the subordinate clauses one at a time before it deals with the main clause, using existing routines. On the other hand, if the syntax analysis is required to determine the clause boundaries as well as the phrase structure, a much more sophisticated program would be required.

The problem of recognizing and resolving semantic ambiguities remains largely unsolved. Determining what is meant by the question "Did the Red Sox win most of their games in July?" depends on a much larger context than the immediate question. The computer might answer all meaningful versions of the question (we know of five), or might ask the questioner which meaning he intended. In general, the facility for the computer to query the questioner is likely to be the most powerful improvement. This would allow the computer to increase its vocabulary, to resolve ambiguities, and perhaps even to train the questioner in the use of the program.

Considerable pains were taken to keep the program general. Most of the program will remain unchanged and intact in a new context, such as voting records. The processing program will handle data in any sort of hierarchical form, and is indifferent to the attributes used. The syntax program is based entirely on parts of speech, which can easily be assigned to a new set of words for a new context. On the other hand, some of the subroutines contained in the dictionary meanings are certainly specific to baseball; probably each new context would require certain subroutines specific to it. Also, each context might introduce a number of modifiers and derived attributes that would have to be defined in terms of special subroutines for the processor. Hopefully, all such occasions for change have been isolated in a small area of special subroutines, so that the main routines can be unaltered. However, until we have actually switched contexts, we cannot say definitively that we have been successful in producing a general question-answering program.

Acknowledgment

The Baseball program was conceived by Fredrick C. Frick, Oliver G. Selfridge, and Gerald P. Dineen, whose continued guidance is

From the collection of the Computer History Museum (www.computerhistory.org)

gratefully acknowledged.

References

1. A. Newell and F. Tonge, "An introduction to Information Processing Language V", Commun. Assoc. for Computing Mach., Vol. 3, pp. 205-211; April, 1960.

2. See the project summary, by Z. S. Harris, in Current Research and Development in Scientific Documentation No. 6, pp. 52-53, Nat'l Sciences Foundation, May, 1960.

3. L. Miller, J. Minker, W. G. Reed, and W. E. Shindle, "A multi-level file structure for information processing", Proceedings Western Joint Computer Conference, Vol. 17, pp. 53-59, May, 1960.

From the collection of the Computer History Museum (www.computerhistory.org)