CONVERSATIONS WITH A COMPUTER - AN EXAMPLE OF NETWORK PROGRAMMING IN RELATIONAL DATA BASE ENVIRONMENT

M. Nalbach, K. Studzinski and S. Waligorski

Institute of Informatics
Warsaw University
00901 Warsaw, PKiN 850
Poland

Any efficient human-computer conversation in an ethnic language needs rather large ammount of information, which generally may be classified as follows:

1/ Script: rules governing entire exchange of messages between computer and human which generally determine how computer reacts and responds in various possible situations.

2/ Grammar and vocabulary of the language: rules of recognition of words, phrases and entire messages, including algorithms of lexical and syntactical analysis.

3/ Semantics: rules of understanding of words and messages, including methods of determining how the words and messages relate to data stored in memory and what should be specific reactions of the computer according to the recognized meaning messages.

This data may be presented in various forms, e.g. as dictionaries, transition networks for lexical analysis, augmented transition networks (ATN) for syntactic analysis, semantic networks, relations, and so on. They may be also included into programs which control and run conversations.

It is evident that this information must be easily modifiable in case of debugging or improvement. It is very useful if the form of all data created by a user complies with certain standards and the data are stored in a uniform way, so as to make understanding and modification as easy as possible. For this reason we use as a standard storage a relational data base.

We may consider any real conversation with a computer as a sequence of atomic units, each comprising one exchange of messages between human and computer, with all accompanying actions. Every dialogue determines one sequence of such units and transitions. All such sequences of units and transitions form a network. It turns out that it is very useful to introduce recursion into such networks; since it is possible to store or to fetch data during a dialogue, this concept resembles augmented transition networks (ATN). In fact, these conversation control networks may be transformed into ATN. However, their original form provides better protection against undesired indeterminism and backtracking. ATN in their original form are applied for syntactical analysis.

Obviously, such networks are nothing else as programs in a special programming language (or languages). Quite naturally, networks may be comfortably described by relations. This paper presents results of an implementation of these ideas. Data base management programs are in Fortran, but user access to the data base is entirely conversational. All networks stored in the base have form of relations. For example, elements of a relation for an ATN may have the form shown in Fig. 1. This relation is translated into a simpler one in which all conditions and actions are replaced by links to Fortran subroutines obtained as results of translation of corresponding expressions, and names of states in 'to' field are replaced by identifiers of appropriate tuples in the new relation. Networks in this form may be interpreted or compil-

ed to Fortran. Compilation is usually made for large ATN's for which simple interpretation would be too slow. We obtained in this way good speeds even for very complicated syntactic grammars. Conversation control networks are always interpreted, for in this case no speedup was necessary. The conversation control allows bootstrapping, i.e. an appropriately created network may control process of conversational creation or modification of any CCN.

Our implementation was made in Fortran, in spite of all its disadvantages, mainly for Portability reasons. It is still much more easy to transfer to other computer or mini software written in Fortran than in any other language, including LISP.

STATE	ARCNAM	ARGUMENT	CONDITION	ACTIONS	TO
S/	TST	ONTEST	EQ(LEX QUOTE(ON))	SETR(WHERE ON)	S1/
S5/	PUSH	NPS/	MEMQ(GETF(CAT) QUOTE(ADJ PRON))	SETR(ADJ GETF(F))	S4/
S10/	POP	APPEND(GETR(N) GETR(PRED))	T	-	-

Fig. 1