Descriptive Language as a Linguistic Tool

Mei-Hui Su* and Keh-Yih Su**

***BTC R&D Center**
2F, 28 R&D Road II
Science-Based Industrial Park
Hsinchu, Taiwan, R.O.C.

****Department of Electrical Engineering**
National Tsing Hua University
Hsinchu, Taiwan, R.O.C.

ABSTRACT

In developing a natural languages processing system, the linguist needs to transfer a large amount of linguistic knowledge into the system. Therefore, the transferring and the maintaining of the linguistic knowledge in a working system become a crucial issue. If the linguist transfers the linguistic knowledge indirectly through the computer engineer, the productivity of both the linguist and the computer engineer might be lowered. So it is better for the linguist to transfer the knowledge directly into the system.

In this paper, we propose a descriptive language that is tailored specifically for direct transferring and maintaining of the linguistic knowledge needed in our system by our linguists. We will also discuss the criterions for designing the descriptive language and the design experiences gained from a completed descriptive language and its environment. In addition to offering a simple and direct way to express linguistic knowledge, descriptive language unifies the knowledge into a clear and definite form. As a direct consequence, it is easier to build an environment that can maintain the global knowledge consistency needed in a natural language processing system.

Introduction

In a natural language processing (NLP) system, after a research topic is completed by the linguist, the knowledge obtained must be transferred into the working system in order to be utilized during the natural language processing. The traditional approach of transferring the knowledge to the system indirectly is depicted in the following figure.

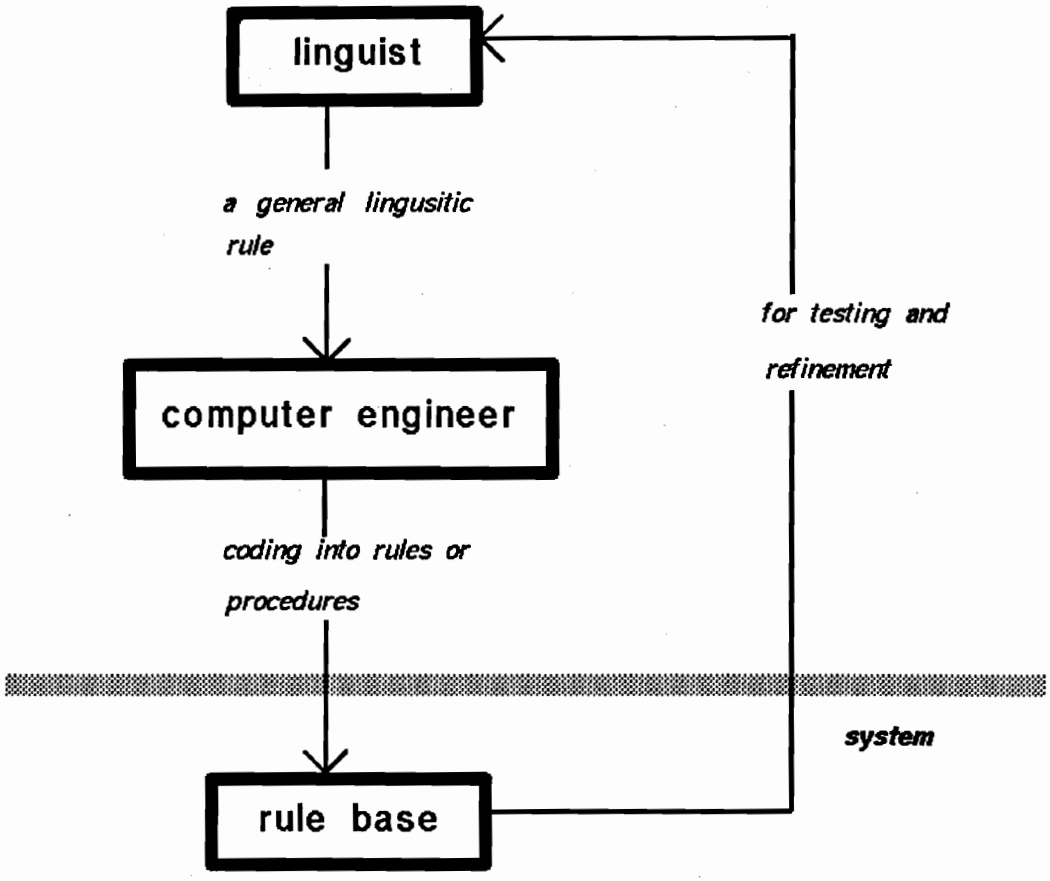

Figure 1. Indirect knowledge transferring approach

In this figure, the linguist passes the rule to the computer engineer after a research topic is completed. Then, after the rules are implemented by the computer engineer, the linguist will test the system to see if it works as expected. Several places can go wrong in this process. In the first place, the linguist might have concluded an incorrect knowledge and so the system will not work as expected. In the second place, the computer engineer might have misunderstood the request and implement the wrong rules. As a result, the linguist will get an incorrect response from the system. And in the third place, the computer engineer might implement the request erroneously and so the linguist will get a wrong feedback from the system.

This indirect approach introduces two extra error sources in comparison to that of a direct approach in which the linguist would interact directly with the system. Above all, the more

severe problem of using the indirect approach is that if an error occurred, it is hard to pinpoint the source of error. Therefore numerous looping through the flow is likely before the initial request is correctly implemented. As a result, the productivity of both computer engineer and linguist is lowered. Hence, it is essential to have the linguist interact directly with the system instead of through the computer engineer. This can be solved by providing a tool that is designed specifically for handling linguistic knowledge and so that the linguist can interact with the system directly.

This linguistic tool might be an user-friendly support environment or it might go as far as a complete high-level linguistics-oriented programming language like LINGOL [PRAT 73], MUIR [WINO 86] and PERIPHRASE [BEES 88]. For our machine translation system, we developed a descriptive language which is a high level declarative language for the linguist to develop and to maintain the linguistic modules we have, to minimize the computer engineer's involvement during the knowledge transferring process and to provide a friendly knowledge transferring environment for the linguist. The purpose of developing this descriptive language is not to create a metalanguage [SHIE 86] which is intended to handle different linguistic grammar formalisms but as a tool that meets the specific needs of our linguists and their knowledge.

In addition to the reasons mentioned above, descriptive language has another advantage. The advantage is that it allows the linguistic knowledge to be represented and maintained in a clear and definite way. This is very important in a natural language processing system with an on-going research in linguistic phenomena. The reason is that to maintain a consistent view of all linguistic knowledge for the NLP system is not a simple task. But a complete and consistent view is necessary if the output quality of the working system is to be at its best. Therefore, It will be of advantage if the linguistic knowledge are kept in a definite form. In this way the descriptive language will make it easier to build the environment for the computer to support the knowledge consistency needed by the working system.

Criterions of Descriptive Language

In designing a descriptive language it is important to keep the following three criterions in mind. The first criterion is that its interface must be user-friendly and the format should be declarative. The rule format should be declarative because declarative rule is easier to write and it screens the knowledge representation from the issues of the underlying system. This is important because the user will be the linguist instead of the computer engineer. Therefore the user should not be hindered with any detail of the system. The second criterion is that it should be linguistic felicitous [SHIE 85][SHIE 86]. This implies that the descriptive language should be flexible and powerful enough to let the linguist represent their knowledge and it should also be natural in the linguistic sense. This will make the maintaining of the knowledge easier for the linguist. The third criterion is that the linguistic information stored internally should be in a form such that a runtime efficient driving mechanism is possible to implement. This is a must if we do not want to slow down the speed of the natural language processing system.

Format of a Descriptive Language

In our system, we have a linguistic module called the condition and action. The linguistic rules in this module place condition tests on grammar rules being applied and adopt some

actions if the conditions are satisfied. It is similar in idea to the condition and action used in the ATN [WINO 83]. Following is a subset of the syntax of the descriptive language defined for the condition and action rules.

test2 : ASSIGN what TO where1
ftype : TERMINALWORD (where1 . < aRVARIABLE >) . wpropty
where1 : PARENT | CHILD | aCVARIABLE

In the above, the lower case terms are the non-terminals and the upper case terms are the terminals or the reserved words in our grammar. This syntax is a context-free grammar with a total of 57 grammar rules. Following are some of the condition and action rules in our system written according to the descriptive language defined above.

1: *[assign (parent.<R_Head>).<A_Aspt> to parent]*

This is an example of attribute percolation from a child to its parent. This rule assigns the aspect attribute of the child, which is the head in the current node's role register [WINO 83], to the current node.

2: *[terminalword(parent.<R_Head>).stem is ("more")]*

This is an example of a condition test that determines whether the terminal word governed by the head attribute of the current node is the word *"more"*. If the condition test fails, the current grammar rule being applied will be blocked and the next grammar rule will be tried.

3: *if [terminalword(parent.<R_Head>).Morf is ("er")]*
then [assign ("cprt") to parent]

This is an example of doing a conditional check before a direct assignment of an attribute to the current node.

Although each of the three rules mentioned above has different functions, their formats are similar. The rule format for this descriptive language was designed after several meeting with our linguist to make sure that it will give the expressive power they need and that it will be easy for them to understand and write the rules.

Design Experience

There are several linguistic modules that can be implemented using the descriptive language. They are the Condition and Action module, the Transfer Rule module, the Grammar Rule module, and the Knowledge module. Every module's knowledge is distinctly represented because their functions during the translation process are different. For example, the transfer rules are used in the transferring phase and the condition and action rules are used in the analyzing phase. Therefore the descriptive language will have different format for each module if we want to retain the properties that are best suited for their functions. As a result, each module will become an individual linguistic kernel within the descriptive language support system. A block diagram of this support system is shown in Figure 2.

In Figure 2 , each linguistic kernel in this support system will maintain their own independent rule base as their private data. If the linguists want to make changes to a kernel's rule base, they will have to interact directly with that kernel. At the present, one of

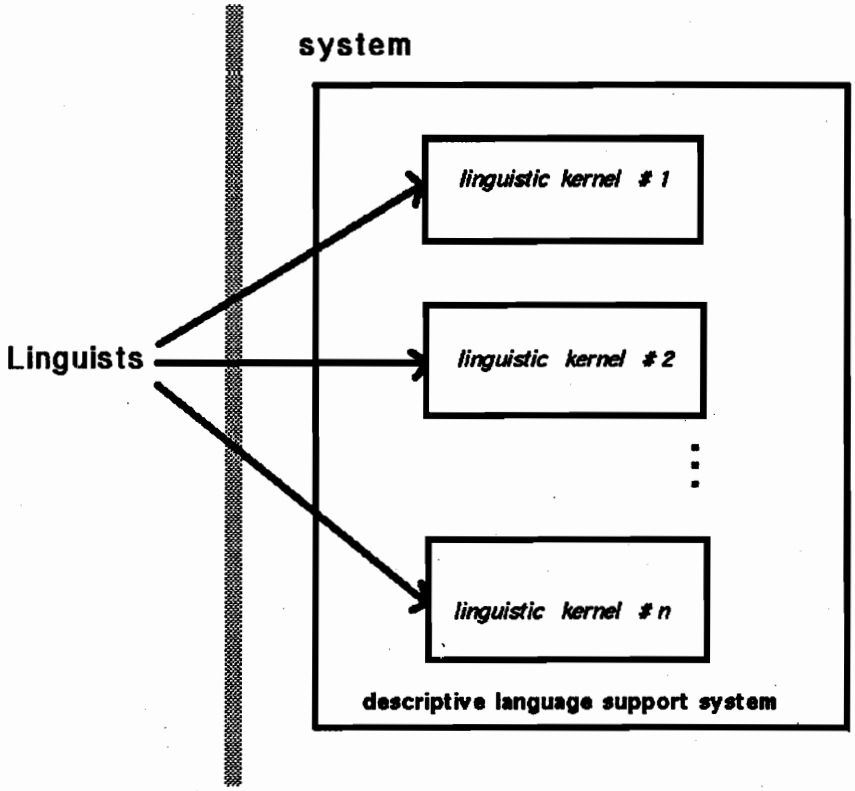

Figure 2. Descriptive language support system

the linguistic kernels is completed in our system and it is the condition and action kernel. A simple block diagram which shows how this kernel is implemented is in Figure 3.

The condition and action kernel is separated into two main distinct parts according to their basic intended purposes. The first is the user interface and rule base maintainer. The second is the code generator. The user interface will process the linguist's request to insert, update or delete rules. It will also make a quick syntactic check on the rule format and prompt the user if the rule is not in the correct format. This part also does the housekeeping chores like updating database's general information. The task of code generator is to take legal rules and generate procedures in C for each of the rules.

The condition and action rules written by the linguist through the descriptive language are very rigid in construction and rarely altered once they are written. Therefore developing a kernel for the Condition and Action module can be seen similar to that of developing a compiler which in this case will accept the linguistic knowledge rules as its input and produce codes recognizable by the system as its output.

Looking at the block diagram of Figure 3, if the user interface and the code generator blocks are combined, it is similar to the conventional compiler that takes in condition and action rules and produces the corresponding routines in C language. But there are several features in this kernel's approach that makes it different from the traditional compiler. The first feature is that the rules can be changed incrementally and are kept in a private rule base. The second feature is that this kernel has an on-line interactive user interface (it also has a batch-mode). The third feature is that this kernel will give immediate feedbacks to user on ill-formed rules. All three features mentioned above are intended to make this kernel user friendly and easy to use. And the last feature is that the object code is in C instead of in any lower level language. The reason for this is because the good readability of C procedures

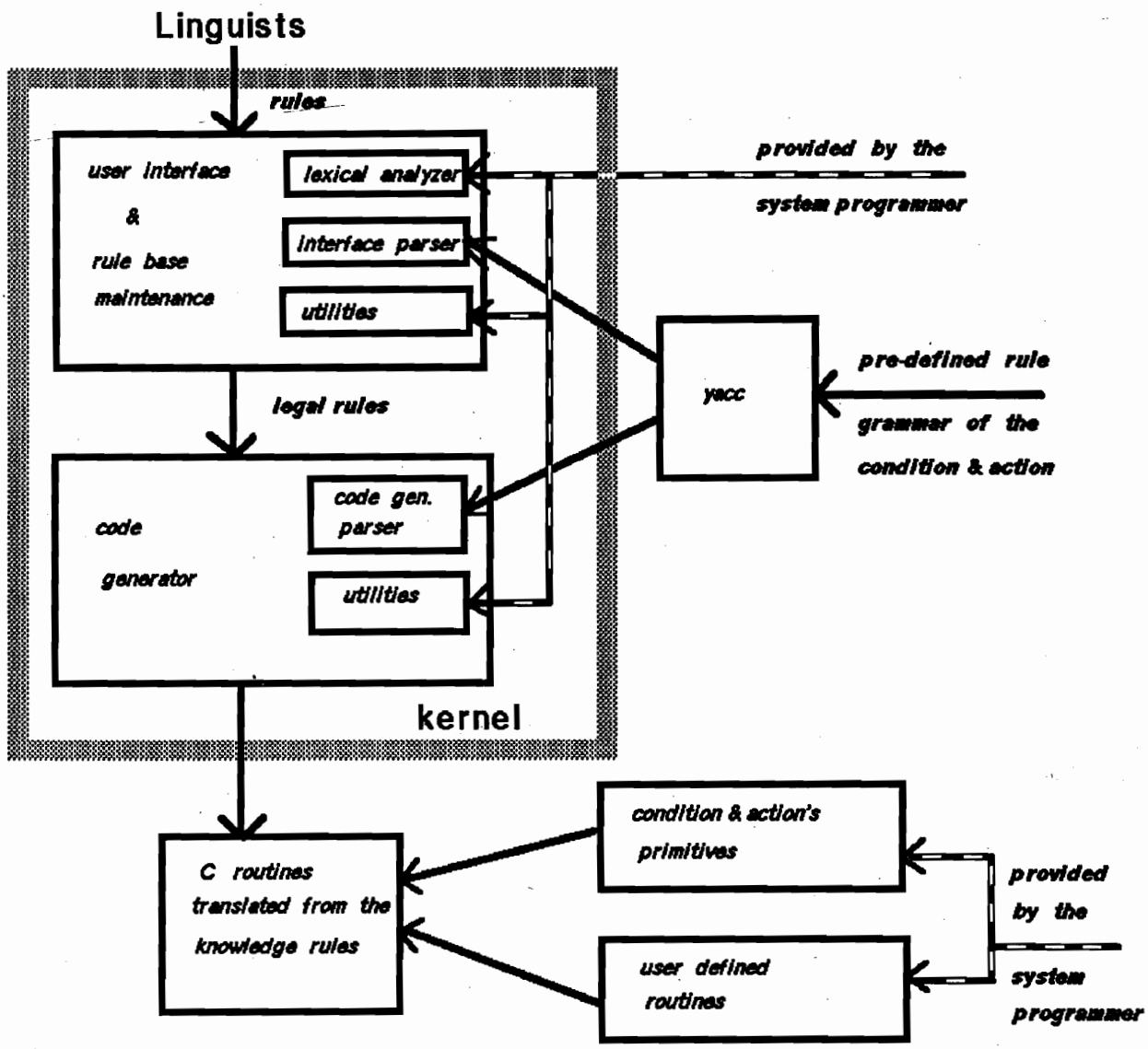

Figure 3. The condition and action kernel

which will be of advantage during debugging. The one extra step of compiling the C program will not matter much since it will not affect the working system at the runtime.

In this kernel, YACC is used in creating parsers for the user interface block and for the code generator block. The reason for having two parsers is because their functions are different and so their syntax rule grammars for the YACC are different. The parser for the user interface is for doing the syntax check and the parser for the code generator is for driving the code generation. With the rigid form of the condition and action rules, the parser produced by YACC is more than adequate. With the procedures translated automatically from the linguistic rules, this compiler solution frees the computer engineers from hand-coding each new procedure as rules are added or updated. Figure 4 shows how the condition and action module's knowledge is incorporated into the working system. In the figure, the C routines are first compiled into machine executable codes and then included into the working system.

Several design decisions were made in designing this kernel. First, at the user's level the condition and action rule remains in their declarative rule form for the linguistic friendliness.

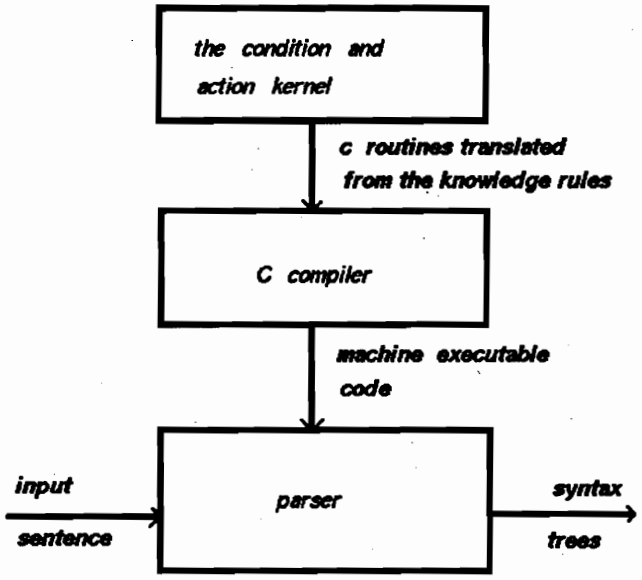

Figure 4. The incoporating of linguistic knowledge into the parser

But at the system's level, rules are translated into procedures to improve the run-time efficiency of the system. This is possible because of the stable nature of the condition and action rule. Rules of other kernels might not exhibit such property and so different design choice must be made. Another design decision is in allowing the inclusion of user-defined routines in contrast to expanding the rule grammar format to include exceptional and rare cases.

An actual session of this kernel accepting a condition and action rule is in appendix. Note, If a rule is ill-formed the system will let the user be aware of and immediately asks for a correct one.

Consistency Controller

With all the linguistic knowledge organized into rigid formats by using the descriptive language, it is simpler to construct the consistency controller, whose purpose is to maintain a consistent logical view of knowledge for our working system. Figure 5 is a general block diagram of the relation between the descriptive language support system and the consistency controller.

In Figure 5, The globally shared data and primitives are controlled by the consistency controller. For example, the category table which is referenced by every linguistic module will be kept here. When requested by the linguist through the descriptive language support system, the consistency controller will do all the global changes and updates the shared data. For example, if a grammar rule is altered, all the rules in other kernels that are dependent on that grammar rule will have to be checked for validity. It is the consistency controller's job to let the linguist know which one might need to be altered. The reason for this is to avoid missing any of the changes needed in other related kernels when the linguistic knowledge is altered in one of the kernels. With all linguistic modules integrated into one descriptive language support system, and with a centralized consistency controller the chance of inconsistency will be small.

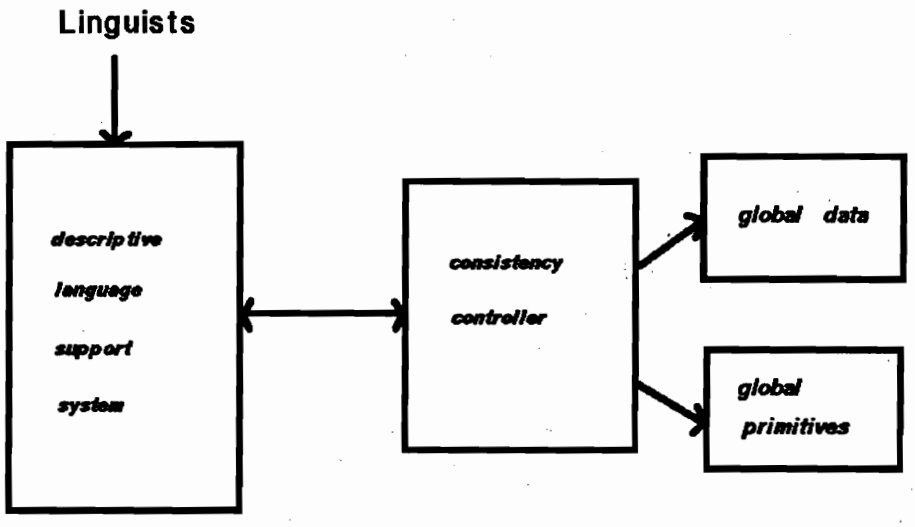

Figure 5. Consistency controller

Conclusion

In a natural language processing system, it is essential to have the linguist transfer and maintain the linguistic knowledge directly with the system. The descriptive language proposed is a linguistic tool that will support the linguists in transferring their linguistic knowledge directly into the system. Moreover, as a direct consequence of having a descriptive language and its support system, the linguistic knowledge will be organized into a clear and definite form. With this, a consistency controller can be constructed for maintaining the knowledge consistency needed in a natural language processing system.

For our machine translation system, a descriptive language is provided and the linguistic kernel for one linguistic module, the Condition and Action, is completed. This descriptive language will allow our linguist to write condition and action rules and transferring them into the system directly.

Appendix

This is an actual session of running the condition and action module.

```
=========================================================
||     WELCOME TO CONDITION AND ACTION MODULE           ||
=========================================================

your name :mei

Select one .. 0 - to exit,
              1 - to lookup,
              2 - to change condition and action,
              3 - to change user defined routine,
```

```
              4 - to initialize condition and action,
              5 - to initialize routines,
0 to 5 => 2


*************************************************************
**      Condition and Action                                **
*************************************************************

Select one .. 0 - to exit,
              1 - to add new condition and action,
              2 - to update condition and action,
              3 - to delete condition and action,
0 to 3 => 1

Name of the condition or action => test1
definition :        ( type "end" or return to exit )
>>[assign (parent.<R_head>) to parent]
>>

comment :         ( return to end )
>>test rule
>>

              /* If the input's format is incorrect, the */
              /* system will show the erroneous input and*/
              /* ask the user to try it again.           */

following is an erroneous input -- lets do it again
              /* system show the bad input to the user   */
-- temporary file --
[assign (parent.<R_head>) to parent]
--------------------



definition :        ( type "end" or return to exit )
>>[assign (parent.<R_head>).<A_aspt> to parent]
>>
accept !



Select one .. 0 - to exit,
              1 - to add new condition and action,
              2 - to update condition and action,
```

```
                3 - to delete condition and action,
0 to 3 => 0

are you sure of the changes? (y/n) y
```

References

[AHO 86] Aho, A.V., R. Sethi and J.D. Ullman, 1986. *Compilers, principles, techniques and tools*, Addison-Wesley, Massachusetts.

[BEES 88] Beesley, K.R. and D. Hefner, 1988. "PERIPHRASE: A High-Level Language for Linguistic Parsing", company communication note from *Automated Language Processing Systems*, Utah.

[BOGU 88] Boguraev, B., J. Carroll, E. Briscoe, and C. Grover, 1988. "Software Support for Practical Grammar Development," *Proceedings of the 12th International Conference on Computational Linguistics*, vol. 1, Budapest, Hungary. PP. 54-58.

[PERE 84] Pereira, F.C.N. and S.M. Shieber, 1984. "The Semantics of Grammar Formalisms seem as Computer Languages", *Proceedings of the Tenth International Conference on Computational Linguistics*, Stanford University, California. PP. 123–129.

[PRAT 73] Pratt, Vaughan R., 1973. "A Linguistics Oriented Programming Language", *Proceedings of the Third International joint Conference on Artificial Intelligence*, Stanford University, California.

[SHIE 84] Shieber, S.M., 1984. "The Design of a Computer Language for Linguistic Information", *Proceedings of the Tenth International Conference on Computational Linguistics*, Stanford University, California. PP. 362–366.

[SHIE 85] Shieber, S.M., 1985. "Criteria for designing computer facilities for linguistic analysis", *Linguistics*, pp. 189-211.

[SHIE 86] Shieber, S.M., 1986. *An Introduction to Unification-Based Approaches to Grammar*, CSLI, Stanford University, California.

[WINO 86] Winograd, T., 1986. "MUIR: A Tool for Language Design", technical report of *CSLI* , Report No. CSLI–87-81, CSLI, Stanford University, California.

[WINO 83] Winograd, T., 1983. *Language as a cognitive process*, Vol. 1, Addison-Wesley, Massachusetts.