Knowledge Acquisition from Technical Texts using Attribute Grammars

J. KONTOS AND J. C. CAVOURAS
Dept. of Informatics, Athens School of Economics and Business Science, 76 Patission St, Athens 104 34, Greece.

The work reported in the present paper concerns the development of a computer system which can acquire an attribute grammar knowledge base from a technical text. The use of attribute grammars is explored both in the implementation of the system and in the target-knowledge representation of the text content. Knowledge-based encoding of the text knowledge content is accomplished and inferential exploitation of the resulting knowledge base is possible using the same attribute grammar interpreter as our knowledge engineering tool. The specific problems involved in using attribute grammars for representing both the knowledge content of a text and the knowledge required for extracting and formalising that content are studied, and some solutions are proposed. The techniques proposed can be applied to technical texts that deliver knowledge about the structure of objects and the functional relations of their parts. The prerequisite knowledge problem for text analysis is addressed and faced using attribute grammars. The kinds of prerequisite knowledge considered include subject-related background knowledge, linguistic knowledge and representation knowledge.

Received August 1986, revised June 1987

1. INTRODUCTION

The work reported in the present paper aims at the development of a computer system which can acquire a knowledge base from an expository or instructional text. The use of attribute grammars is explored both in the implementation of the system and in the target-knowledge representation of the text content.

The development of systems with the ability to read texts and assimilate the knowledge contained in them is one of the long-range goals of artificial intelligence and computational linguistics.[7] One approach is to translate texts into some knowledge-representation formalism and use that representation to perform retrieval and inference with it. The first kind of texts translated to some formalism by computer were simple stories. Some early efforts used the Conceptual Dependency formalism.[10] A recent critical survey of language data processing was written by Garvin.[2] Norton[8] wrote a program that translated a technical text using PROLOG as the target-knowledge representation formalism.

The text that Norton considered concerns programming in the BASIC language. The subject matter being communicated by such a text covers both the syntax and the semantics of BASIC programs and their subcomponents. The connections between syntax and semantics must be elucidated during the text analysis process. We propose that attribute grammars may be a good target-representation formalism for such texts, since they are specifically designed for the representation of syntax and semantics. The use of attribute grammars as a knowledge-representation formalism has already been proposed by Papakonstantinou and Kontos (1986).[9] In the present paper we study the specific problems involved in using attribute grammars for representing both the knowledge content of a text and the knowledge required for extracting and formalising that content.

The example text that we use to illustrate our method is very similar to the one used by Norton. We believe, however, that our techniques can also be applied to other texts that deliver knowledge about any kind of object. It is only necessary that the knowledge delivered by the text concerns the structure of objects and the functional relations between the parts of that structure. Attribute-grammar techniques of narrative text analysis were proposed by Kontos some years ago[4,5] and summarised in Ref. 6.

2. THE TEXT BEING ANALYSED

The example text that we used to illustrate our method is very similar to the text used by Norton. Norton's example text has four sections or paragraphs, namely sections A, B, C and D. We have chosen to essentially omit section B, which merely describes the structure of numbers. We have only retained that part of section A that refers to the form of the variables. After these changes and some additional editing, our example text consists of the three sections α, β, γ and reads as follows.

(α1) The form of a variable is 'letter'.
(α2) The form of a variable is 'letter', 'digit'.
(β1) Expressions are formed according to the rules of arithmetic.
(β2) Expressions may use variables as operands.
(β3) Constants may be used as operands.
(β4) The operators +, −, *, / are available.
(γ1) The form of a command is 'variable' = expression.
(γ2) A command means evaluate the expression and insert the value in 'variable'.

In brief, section α refers to the syntax of variables, section β refers to the syntax and semantics of expressions and section γ refers to the syntax and semantics of LET commands.

3. PREREQUISITE KNOWLEDGE

As Norton emphasised,[8] there is a large amount of knowledge required to read a text on any subject. The first kind of knowledge required is subject-related background knowledge, the second is linguistic, and the third, which we call 'representation knowledge', is necessary whenever the result of the analysis of a text must be represented. Then some target-knowledge representation formalism is used which in the case of Norton is PROLOG and in our case is attribute grammars. This third kind of prerequisite knowledge, namely representation knowledge, is used for the actual conversion of the text being analysed into some representation formalism.

The subject-related knowledge required for the example-text analysis includes knowledge about digits, letters, constants, rules of arithmetic, evaluation, insertion and semantics. In Norton's system most of this knowledge is in the form of pre-existing PROLOG rules, which are added to the PROLOG rules synthesised by the text-analysis program. In our system this knowledge is in the form of attribute values of the text-analysis attribute grammar. These values are passed to synthesised attributes and combined with the attribute-grammar fragments that are synthesised from knowledge delivered by the text itself.

The prerequisite linguistic knowledge includes lexical, syntactic and semantic knowledge of English. This kind of knowledge is represented in two ways by Norton. Most of it is embodied in the PROLOG rules forming his program and the rest is encoded in a lexicon. In our system all linguistic knowledge is of course embodied in attribute-grammar BNF rules and associated semantic rules. The attribute-grammar interpreter does all the parsing of the text sentences, as does the PROLOG interpreter.

The prerequisite representation knowledge in Norton's system is represented by some portions of his text-analysis program. In our system the representation knowledge is embodied in attribute-grammar rules.

4. A BRIEF INTRODUCTION TO ATTRIBUTE GRAMMARS

Attribute grammars were proposed first by Knuth as a tool for formal language specification.[3] The basis of an attribute grammar is a context-free grammar. This context-free grammar is augmented with attributes and semantic rules. Attributes are associated with the nonterminal symbols of the grammar. We write $X(<nonterm>)$ to denote attribute X of the nonterminal *nonterm*. Semantic rules are associated with the grammar productions. These rules describe how the value of some attributes of nonterminals appearing in a production is defined in terms of the values of other attributes of the nonterminals or terminals of the production. Semantic rules may also impose conditions on the values of the attributes of a production. Whenever a condition is not fulfilled the grammar rejects the current analysis of the input string. See Kontos,[4,5] Watt and Madsen[12] and Yellin and Mueckstein[13] for the introduction and use of semantic conditions. Translation of strings can be specified using an attribute that collects the translation of an input string.

*p*1: <*S*> ::= *The form of* <*NOUN*> *is* <*STRING*>
  U(<*s*>) = *conc*(*U*(<*NOUN*>), "::=", *U*(<*STRING*>))
*p*2: <*NOUN*> ::= *variable*
  U(<*NOUN*>) = "<*variable*>"
*p*3: <*STRING*> ::= <*KNOWN*>
  U(<*STRING*>) = *U*(<*KNOWN*>)
*p*4: <*STRING*> ::= <*KNOWN*> − <*KNOWN*>
  U(<*STRING*> = *conc*(*U*(<*KNOWN*>1), *U*(<*KNOWN*>2))
*p*5: <*KNOWN*> ::= *letter*
  U(<*KNOWN*>) = "<*letter*>"
*p*6: <*KNOWN*> ::= *digit*
  U(<*KNOWN*>) = "<*digit*>"

Figure 1. A simple attribute-grammar example.

A small example of an attribute grammar is given in Fig. 1. This grammar describes the translation of simple-structure sentences to syntactic productions, in particular the translation of *The form of variable is letter* to <*variable*> ::= <*letter*>. The nonterminals of this grammar are <*S*>, <*NOUN*>, <*STRING*>, and <*KNOWN*>. All nonterminals have a single attribute called *U*. This attribute is used for the collection of the translation which finally resides in *U*(<*S*>). Each of the six productions *p*1 to *p*6 consists of a syntax rule and a semantic rule written below the syntax rule, e.g. in *p*1 <*S*> ::= *The form of a* <*NOUN*> *is* <*STRING*> is the syntax rule and *U*(<*S*>) = *conc*(*U*(<*NOUN*>), "::=", *U*(<*STRING*>)) is the semantic rule. The function *conc* used in the semantic rules means *concatenation* and it may be omitted whenever it is obvious. The quotation marks in the example mean that the constant string written between them will be copied in the string value of the attribute computed by this function.

5. THE SYSTEM IMPLEMENTATION

The system we developed is based on an attribute-grammar interpreter, and it can acquire an attribute-grammar knowledge base by translating a text. The attribute-grammar interpreter is written in UCSD Pascal and is based on a parsing algorithm proposed by Floyd.[1] The translation of the text into an attribute grammar is accomplished with the linguistic knowledge of English required for the translation. This grammar also contains representation knowledge in the form of semantic rules. The subject-related knowledge required is given to the system in the form of attribute values of this grammar. A simpler interpreter written in BASIC was used for some illustrative examples presented below.

Our Translation Attribute Grammar (TAG) recognises three kinds of sentence: structure sentences, meaning sentences and hybrid sentences. Structure sentences describe the structure of objects. These objects in the case of BASIC are commands, expressions, operands, operators, variables and constants. The structure sentences define the structure of such objects in terms of objects of lower order. Such sentences in the example text are the sentences $\alpha 1$, $\alpha 2$ and $\gamma 1$, which have the form *The form of* <*NOUN*> *is* <*NOUNSTRING*>. The nonterminal <*NOUN*> may correspond to any of the objects mentioned above. The nonterminal <*NOUNSTRING*> may correspond to a list of one or more objects.

Structure sentences are translated by TAG into syntax rules of the Object Attribute Grammar (OAG). The meaning sentences define the *meaning* of objects in terms of a series of one or more actions. These actions, in the case of BASIC, are evaluation and insertion actions. The sentence $\gamma 2$ is the only meaning sentence of the example text. Meaning sentences are translated by TAG into *semantic* rules of the OAG that define attribute manipulations. The hybrid sentences are the most complex sentences handled by TAG and have the properties of both structure and meaning sentences. Such sentences in the example text are the sentences $\beta 1$, $\beta 2$, $\beta 3$ and $\beta 4$. The sentences are further complicated by the fact that they refer to prerequisite knowledge, which in the case of the example text refers to letters, digits, constants and arithmetic expressions. This prerequisite knowledge is encoded as attribute-grammar fragments that are given as values to certain TAG attributes and are combined with the text translation to form the OAG.

6. THE TRANSLATION OF THE EXAMPLE TEXT

Our system translates a technical text into an attribute grammar, which can be used as a knowledge base. The example text of Section 2 is translated into the object-attribute grammar (OAG) as described below.

From *The form of a variable is a letter* ($\alpha 1$) we obtain:

$\alpha 1.1$ $<variable> ::= <letter>$
$\alpha 1.2$ $t(<variable>) := t(<letter>)$
$\alpha 1.3$ $v(<variable>) := t(<letter<)$
$\alpha 1.4$ $<letter> ::= A$
$\alpha 1.5$ $t(<letter>) := A$
$\alpha 1.6$ $<letter> ::= B$
$\alpha 1.7$ $t(<letter>) := B$
$\alpha 1.8$ $<letter> ::= C$
$\alpha 1.9$ $t(<letter>) := C$

where $t(<variable>) = t(<letter>)$ means *the name of a letter becomes the name of the corresponding variable.*

From *The form of a variable is letter, digit* ($\alpha 2$) we obtain:

$\alpha 2.1$ $<variable> ::= <letter><digit>$
$\alpha 2.2$ $t(<variable>) := t(<letter>)t(<digit>)$
$\alpha 2.3$ $v(<variable>) := t(<letter>)t(<digit>)$
$\alpha 2.4$ $<letter> ::= A$
$\alpha 2.5$ $t(<letter>) := A$
$\alpha 2.6$ $<letter> ::= B$
$\alpha 2.7$ $t(<letter>) := B$

.
.
.

$\alpha 2.56$ $<digit> ::= 0$
$\alpha 2.57$ $t(<digit>) := 0$
$\alpha 2.58$ $<digit> ::= 1$
$\alpha 2.59$ $t(<digit>) := 1$
etc.

The steps involved in the translation of the above two sentences can be described by using an attribute grammar that is based on Fig. 1. All productions except $p2$ must be augmented to include semantic rules that represent the prerequisite knowledge of letters and digits as well as translation knowledge. This augmented attribute grammar is shown in Fig. 2 with semantics written in BASIC as processed by the interpreter written in BASIC.

```
***—GRAMMAR—***
Rule (1)
————————————————
<STRUCTURE> = The form of <NOUN> is <STRING>$
.....................................
4110 X$ = V$(0+2)@Y$ = V$(0+1)
4120 Z$ = Y$&":=" &X$&"——V(" &Y$&") = V(" &X$&")"
4130 Z$ = Z$&"**" &T$(0+2)
4140 V$(0) = Z$
4198 RETURN
4199 ! ————————————————————————————————
Rule (2)
————————————————
<NOUN> = variable$
.....................................
4210 V$(0) = "<variable>"
4298 RETURN
4299 ! ————————————————————————————————
Rule (3)
————————————————
<STRING> = <KNOWN>—<KNOWN>/<KNOWN>$
.....................................
4300 ON A GOTO 4310, 4320
4310 V$(0) = V$(0+1)&V$(0+2)
4312 Y$ = SY$(0+1)&"——" &SE$(0+1)
4314 X$ = SY$(0+2)&"——" SE$(0+2)
4316 T$(0) = Y$&"**" X$
4318 RETURN
4319 ! ————————————————————————————————
4320 V$(0) = V$(0+1)
4322 T$(0) = SY$(0+1)&"——" &SE$(0+1)
4328 RETURN
4399 ! ————————————————————————————————
Rule (4)
————————————————
<KNOWN> letter/digit$
.....................................
4400 ON A GOTO 4410, 4420
4410 SY$(0) = "<letter> = A/B/C"
4412 SE$(0) = "V(A) = A,V(B) = B,V(C) = C"
4413 V$(0) = "<letter>"
4418 RETURN
4419 ! ————————————————————————————————
4420 SY$(0) = "<digit> = 1/2/3"
4422 SE$(0) = "V(1) = 1,V(2) = 2,V(3) = 3"
4423 V$(0) = "<digit>"
4428 RETURN
4429 ! ————————————————————————————————
```

Figure 2. An illustrative translation attribute grammar.

The steps involved in the translation of the sentence *The form of variable is letter, digit*, are summarised in Fig. 3. Five steps are shown, although other steps are involved that would complicate the explanation. The first column shows the nonterminal of the left-hand side (LHS) of the rule used at this step. For example step 1 uses the $<NOUN> ::= variable$ *rule*. The second column (POS) shows the position in the input string where the currently recognised substring ends. For example the word *variable*), which is recognised as $<noun>$, ends at the 20th position. The third column $V(LHS)$ shows the values taken by the attribute V of the left-hand side of the production currently used. At the 5th step $V(<STRUCTURE>)$ contains the full object-attribute grammar that represents the above sentence.

Some explanations are necessary for the understanding of the translation-attribute grammar of Fig. 2. First of

Rule, LHS	POS	V(LHS)
1. <*noun*>	20	<*variable*>
2. <*KNOWN*>	30	<*letter*>
3. <*KNOWN*>	36	<*digit*>
4. <*STRING*>	36	<*letter*><*digit*>
5. <*STRUCTURE*>	36	<*variable*>::=<*letter*><*digit*> *V*(<*variable*>)=*V*(<*letter*><*digit*>) <*letter*>::=*A*/*B*/*C* *V*(*A*)=*A*,*V*(*B*)=*B*,*V*(*C*)=*C* <*digit*>::=1/2/3 *V*(1)=1,*V*(2)=2,*V*(3)=3

Figure 3. Summary of the steps involved in the translation of the sentence *The form of variable is letter, digit.*

all, the syntax rules use the / convention for rules with the same left-hand side. For example the two rules:

<*KNOWN*>::=*letter* and <*KNOWN*>::=*digit*

are written as a single rule with two alternatives as follows:

<*KNOWN*>::=*letter*/*digit*

The *ON...GOTO* statements guide the interpreter to use the appropriate semantic rule depending on the value of *A* which stands for the position of the alternative involved at the right-hand side of the syntax rule. Such a statement is no. 4400, which points to statements 4410 and 4420 for $A = 1$ and $A = 2$ respectively.

The attributes used in Fig. 2 are *V*, *SY*, *SE* and *T*. The attribute *V* plays the same role as *U* in Fig. 1. The attribute *SY* contains syntax rule knowledge and *SE* semantic rule knowledge that originates from the prerequisite knowledge about letters and digits (Rule 4). *T* contains the syntax and semantics of <*STRING*> and is used in building up the grammatical representation of <*STRUCTURE*> in *V*. *V*(0+*n*) means 'the value of *V* corresponding to the *n*th non-terminal of the right-hand side of the syntax rule involved'.

It proved fairly straightforward to express the translation process using attribute grammars. However, a non-procedural semantic metalanguage would further simplify this task.

From *Expressions are formed according to the rules of arithmetic* ($\beta 2$) we get:

$\beta 1.1$ <*expression*>::=<*arithmetic*>
$\beta 1.2$ *v*(<*expression*>):=*v*(<*arithmetic*>)
$\beta 1.3$ <*arithmetic*>::=<*operand*>
$\beta 1.4$ *v*(<*arithmetic*>):=*v*(<*operand*>)
$\beta 1.5$ <*arithmetic*>::=<*operand*><*operator*><*arithmetic*>
$\beta 1.6$ *v*(<*arithmetic*>):=*f*(*v*(<*operand*>), *t*(<*operator*>),*v*(<*arithmetic*>))

where *f* is a function defined by the prerequisite knowledge.

From *Expressions may use variables as operands* ($\beta 2$) associated with <*expression*> are also:

$\beta 2.1$ <*operand*>::=<*variable*>
$\beta 2.2$ *v*(<*operand*>):=*v*(<*variable*>)

From *Constants may be used as operands* ($\beta 3$) we get:

$\beta 3.1$ <*operand*>::=<*constant*>
$\beta 3.2$ *v*(<*operand*>):=(<*constant*>)
$\beta 3.3$ <*constant*>::=<*digit*>
$\beta 3.4$ *v*(<*constant*>):=*v*(<*digit*>)
$\beta 3.5$ <*constant*>::=<*digit*><*constant*>
$\beta 3.6$ *v*(<*constant*>):=10**v*(<*digit*>)+*v*(<*constant*>)
$\beta 3.7$ <*digit*>::=1
$\beta 3.8$ *v*(<*digit*>):=1
$\beta 3.9$ <*digit*>::=2
$\beta 3.10$ *v*(<*digit*>:=2
etc.

From *The operators +, −, *, / are available* ($\beta 4$) we get:

$\beta 4.1$ <*operator*>::=+
$\beta 4.2$ *t*(<*operator*>):=+
$\beta 4.3$ <*operator*>::=−
$\beta 4.4$ *t*(<*operator*<):=−
$\beta 4.5$ <*operator*>::=*
$\beta 4.6$ *t*(<*operator*>):=*
$\beta 4.7$ <*operator*>::=/
$\beta 4.8$ *t*(<*operator*>):=/

From *The form of a command is variable=expression* ($\gamma 1$) we get:

$\gamma 1.1$ <*command*>::=<*variable*>=<*expression*>
$\gamma 1.2$ *t*(<*command*>):=*t*(<*variable*>)"="*t*(<*expression*>)
$\gamma 1.3$ *v*(<*command*>):=*t*( )

($\gamma 1.2$ and $\gamma 1.3$ are not used for the evaluation of BASIC expressions).

From *A command means evaluate the expression and insert the value in variable* ($\gamma 2$) we get:

$\gamma 2.1$ <*command*> *has semantics*:
v(<*variable*>):=*v*(<*expression*>)

When this OAG is used by the attribute-grammar interpreter, strings of BASIC commands presented to it are recognised and executed. Some examples are given below:

Ex1. String presented: `A2=2+3`
Interpreter answer: `The value of A2 is 5`
Ex2. String presented: `A=3-2; B=A`
Interpreter answer: `The value of B is 1`
Ex3. String presented: `A=3; B=A*2.`
Interpreter answer: `The value of B is 6`
Ex4. String presented: `A=3; B=A*2, C=B-1.`
Interpreter answer: `The value of C is 5`

In order to be able to recognise and execute a sequence of commands, the OAG must be augmented with the top production having the syntax part:

<*question*>:=<*command*>;/
<*command*>;<*question*>

where <*question*> corresponds to a simple BASIC program consisting of a sequence of LET commands.

It should be noted that in the OAG *t*(<*x*>)*t*(<*y*>) means *conc*(*t*(<*x*>), *t*(<*y*<)) and so on for more strings.

For illustrative purposes the actions of the interpreter when presented with the input string `A2=2+3` of Ex1 are given in Fig. 4. The three columns show the same information as that of Fig. 3, but for the object-attribute grammar corresponding to the text being analysed. The other column appearing second in Fig. 4 gives the

Analysis of: $A2=2+3.$——

Rule LHS	Alternative	String-position	V(LHS)
<letter>	1	1	0
<digit>	2	2	0
<variable>	1	2	0
<constant>	2	4	2
<operand>	2	4	2
<operator>	1	5	1
<constant>	3	6	3
<operand>	2	6	3
<constant>	3	6	3
<operand>	2	6	3
<arithmet>	2	6	3
<arithmet>	1	6	5
<express>	1	6	5
<command>	1	8	5

Figure 4. Actions of the interpreter using the grammar of Fig. 5 for A2=2+3.

```
***—GRAMMAR—***
Rule (1)
————————————
<command> = <variable> = <express>.—$
.............................
4100 V(0+1)=V(0+2)
4105 V(0)=V(0+2)
Rule (2)
————————————
<express> = <arithmet>$
.............................
4200 V(0)=V(0+1)
Rule (3)
————————————
<arithmet> = <operand> <operator> <arithmet> / <operand>$
.............................
4300 ON A GOTO 4310, 4320
4310 Q1=V(0+1)@Q2=V(0+2)@Q3=V(0+3)
4315 V(0)=FNOP(Q1,Q2,Q3)
4319 !
————————————————————————————————————
4320 V(0)=V(0+1)
Rule (4)
————————————
<operand> = <variable> / <constant>$
.............................
4410 V(0)=V(0+1)
Rule (5)
<constant> = 1/2/3/4/5/6/7/8/9$
.............................
4500 V(0)=A
Rule (6)
————————————
<operator> = +/-/*/:$
.............................
4600 V(0)=A
Rule (7)
————————————
<variable> = <letter> <digit> / <letter>$
.............................
Rule (8)
————————————
<letter> = A/B/C/D/E/F$
.............................
Rule (9)
————————————
<digit> = 1/2/3/4/5/6/7/8/9$
.............................
```

Figure 5. An illustrative object-attribute grammar.

alternative position being used in the right-hand side of the OAG, which is written in the / convention and is shown in Fig. 5 as used by the interpreter version written in BASIC. In Fig. 5 *FNOP* is the name of a predefined function corresponding to f above.

This function is included in the prerequisite knowledge and is used for the evaluation of the arithmetic expressions. Also in Fig. 5 the RETURN statements have been removed for simplification purposes. For example such statements are necessary after statements 4315 and 4320 as well as at the end of every semantic routine. The explanations of Fig. 2 apply here too, except for the fact that the attribute V now collects the arithmetic value of the *LHS* and *RHS* of a command.

7. CONCLUSION

With the work reported in the present paper it was shown that attribute grammars can be successfully used for the acquisition and representation of knowledge from a technical text. Attribute grammars were used both in the implementation of a text analysis system and as the target representation formalism of the text content. The interpretation of the attribute grammars used was easily performed by using an attribute-grammar interpreter. It is believed that attribute grammars are a good representation formalism for texts that describe the structure and function of objects. Further research is required in order to overcome the limitations of the present system.

Some of the main limitations existing due to the fact that our system is the first attempt to exploit attribute grammars for knowledge acquisition from texts are discussed below.

The semantics of the translation attribute grammar are at present written in the procedural language that the interpreter is written in. The adoption of an interpreter with a declarative semantic language will certainly speed up the expansion of the system.

The method has been applied to a special kind of text only, and other kinds must be tried. Here it must be noted that attribute grammars allow for the representation of only two main kinds of relations between sentences. The one kind is represented by the relation between two different productions, and the other is represented by the relation between a syntactic and a semantic rule. For those texts whose sentences exhibit relations that are not representable by these two grammatical relations, some extensions of the method may be needed.

The vocabulary used was very small. For practical applications with large vocabularies an interpreter must be used which is capable of communicating with a lexical database.

The attribute-grammar representations produced answer one kind of question only. These questions are of the form 'what is the meaning of A=B+3, where A=B+3 is a BASIC command and A and B are variables. The treatment of other kinds of questions such as 'why' questions and 'diagnostic' questions is a possible expansion that will render the attribute-grammar representations of the texts more useful. We are currently studying ways of performing diagnostic question-answering with attribute grammars. By this we mean the

analysis of strings with errors and the generation of an answer that classifies each error.[11]

Finally, the prerequisite knowledge is given at present in grammatical form. Further research is required to find ways of utilising prerequisite knowledge expressed in natural-language texts or dictionaries.

REFERENCES

1. R. W. Floyd, The syntax of programming languages – a survey. *IEEE Trans. Electronic Computers* **EC-13** (4), 346–353 (1964).
2. P. L. Garvin, The current state of language data processing. *Advances in Computers* **24**, 217–275 (1985).
3. D. E. Knuth, Semantics of context-free languages. *Mathematical Systems Theory* **2**, 127–145 (1968).
4. J. Kontos, Syntax-directed processing of texts with action semantics. *Cybernetica* **23** (2), 157–175 (1980).
5. J. Kontos, Syntax-directed plan recognition with a microcomputer. *Microprocessing and Microprogramming* **9**, 277–279 (1982).
6. J. Kontos, *Natural Language Processing of Scientific/Technical Data, Knowledge and Text Bases*. Artint Workshop, Luxembourg (1985).
7. J. Larkin, J. McDermott, D. P. Simon and H. A. Simon, Expert and novice performance in solving physics problems. *Science* **208**, 1335–1342 (1980).
8. L. M. Norton, Automated analysis of instructional text. *Artificial Intelligence* **20**, 307–344 (1983).
9. G. Papakonstantinou and J. Kontos, Knowledge representation with attribute grammars. *The Computer Journal* **29** (3), 241–245 (1986).
10. R. C. Schank, *Conceptual Information Processing*. Elsevier-North Holland, New York (1975).
11. C. D. Spyropoulos and J. Kontos, Automatic organization diagnosis based on behavioural modelling with grammars. International Conference on Modelling and Simulation, Karlsruhe (1987).
12. D. A. Watt and O. L. Madsen, Extended attribute grammars. *The Computer Journal* **26** (2), 142–153 (1983).
13. D. M. Yellin and E.-M. M. Mueckstein, The automatic inversion of attribute grammars. *IEEE Trans. on Software Engineering* **SE-12**, 590–599 (1986).