Enriching Partially-Specified Representations for Text Realization Using an Attribute Grammar *

Songsak Channarukul **Susan W. McRoy** **Syed S. Ali**
{songsak, mcroy, syali}@cs.uwm.edu

Natural Language and Knowledge Representation Research Group
http://tigger.cs.uwm.edu/~nlkrrg
Electrical Engineering and Computer Science Department
University of Wisconsin-Milwaukee

Abstract

We present a new approach to enriching under-specified representations of content to be realized as text. Our approach uses an attribute grammar to propagate missing information where needed in a tree that represents the text to be realized. This declaratively-specified grammar mediates between application-produced output and the input to a generation system and, as a consequence, can easily augment an existing generation system. End-applications that use this approach can produce high quality text without a fine-grained specification of the text to be realized, thereby reducing the burden to the application. Additionally, representations used by the generator are compact, because values that can be constructed from the constraints encoded by the grammar will be propagated where necessary. This approach is more flexible than defaulting or making a statistically good choice because it can deal with long-distance dependencies (such as gaps and reflexive pronouns). Our approach differs from other approaches that use attribute grammars in that we use the grammar to enrich the representations of the content to be realized, rather than to generate the text itself. We illustrate the approach with examples from our template-based text-realizer, YAG.

1 Introduction

Typically, a text realization system requires a great deal of syntactic information from an application in order to generate a high quality text; however, an application might not have this information (unless it has been built with text generation in mind). This problem has been referred to as the *Generation Gap* (Meteer, 1990). Meteer first identified the generation gap problem as arising at the text planning stage. A *text planner* must decide what content needs to be expressed and creates a corresponding text plan for generating it. A *sentence planner* is then used to select an appropriate syntactic structure for a given plan. Typically, neither a text planner nor a sentence planner is concerned with fine-grained syntactic issues, such as whether the subject of the sentence is a singular or plural noun. Thus, it becomes the responsibility of a text realizer to infer the missing information and to generate the best possible text from a given input.

Most generation systems (such as FUF/SURGE (Elhadad, 1992), Penman (Mann, 1983), RealPro (Lavoie and Rambow, 1997), TG/2 (Busemann, 1996), and YAG (Channarukul, 1999; McRoy et al., 1999)) alleviate this problem by using ***defaulting***, in which a grammar writer specifies a default for each syntactic constraint. This approach is inflexible and prone to errors, because there might not be one default that suits all applications or situations. Another approach that has been proposed is to fill in the missing information on the basis of word co-occurrence data collected from a large corpus of text (see Nitrogen (Knight and Hatzivassiloglou, 1995)). However, statistical approaches have difficulty when there are long-distance dependencies among constituents in a text.

In this paper, we present a new approach to resolving the so-called generation gap that uses an *Attribute Grammar* (Knuth, 1968) to enrich partially-specified inputs to a realization system to produce high quality texts. Attribute Grammars are a declarative formalism for defining rules for attribute propagation (see Section 3). They have been used primarily for specifying the semantics of programming languages, although a few researchers have also used them to drive a text generator (see (Levison and Lessard, 1990), for example). The main advantage of our approach is that it allows a generator to enjoy the computational efficiency of a template-based realization system, while reducing the linguistic burden on an application and increasing the quality of the generated texts.

Our work differs from previous uses of attribute grammars in natural language generation, which are similar to Levison and Lessard (Levison and Lessard, 1990) in that they apply attribute grammars directly to text realization. For example, Lev-

[0]This work was supported by a gift from Intel Corporation; and by the National Science Foundation, under grants IRI-9701617 and IRI-9523666.

```
((template CLAUSE)
 (process-type MENTAL)
 (process "want")
 (processor ((template NOUN-PHRASE)
             (head ((template CONJUNCTION)
                    (first ((template NOUN-PHRASE)
                            (head "Jack")
                            (np-type PROPER)
                            (gender MASCULINE)
                            (definite NOART)))
                    (second ((template PRONOUN)) )))
             (person SECOND)
             (number PLURAL)) )
 (phenomenon ((template NOUN-PHRASE)
              (head "dog")
              (definite NOART)
              (possessor ((template NOUN-PHRASE)
                          (head "sister")
                          (gender FEMININE)
                          (definite NOART)
                          (possessor ((template NOUN-PHRASE)
                                      (head "Jack")
                                      (np-type PROPER)
                                      (gender MASCULINE)
                                      (pronominal YES)
                                      (definite NOART)) )))))
 (rear-circum ((template CLAUSE)
               (mood TO-INFINITIVE)
               (process-type MATERIAL)
               (process "swim")) ) )
```

Figure 1: A Feature Structure for the Sentence *"Jack and I want his sister's dog to swim."*.

ison and Lessard extend a context-free grammar with attributes and semantic rules similar to classical attribute grammars presented by Knuth (Knuth, 1968). Attributes in their system assist the realization by propagating information down a tree that specifies the complete syntactic structure of the output text. By contrast, our work employs attribute grammars, not to realize a text, but to perform a generation gap analysis prior to actual realization. We use both inherited and synthesized attributes (*i.e.*, propagating information both down and up a tree) to share information and to determine appropriate values for any missing features.

2 An Overview of YAG

YAG (Yet Another Generator) (Channarukul, 1999; McRoy et al., 1999) is a template-based text-realization system that generates text in real-time. YAG uses templates to express text structures corresponding to fragments of the target language. Templates in YAG are declarative and modular. Complex texts can be generated by embedding templates inside other templates.

Values for the templates are provided by an application; inputs can include either a conceptual representation of content or a feature structure. When an input is only partially specified, defaults defined in a template will be applied. Figure 1 shows an example of YAG's feature-structure based input; YAG would realize this example as *"Jack and I want his sister's dog to swim."*. This input is partially specified, and thus is more compact and easier for an application to specify, than a complete specification. Figure 2 shows the features that have been omitted and the defaults used by YAG to realize the sentence from the input.

Although the input is already more compact than a full specification, further simplification of the input provided from an application would have been possible, if certain inferences could be made. For example, Figure 3 shows an input structure that could replace the one given in Figure 1. In Figure 3, it was not necessary for the application to specify that the conjunction of two noun phrases is a plural noun phrase, nor that component noun phrases (proper nouns, pronouns, and possessives) should not contain an article. In the case of conjunctions, there is no default that would provide the correct outputs in all cases, because the same conjunction template is used to conjoin adjectives and clauses. Instead, our approach uses an attribute grammar to make the appropriate inferences and enrich the feature struc-

Template Name	Template Slot	Default	Allowed Values
CLAUSE	sentence	YES	YES, NO
	mood	DECLARATIVE	DECLARATIVE, YES-NO, WH, IMPERATIVE, TO-INFINITIVE
	process-type	ASCRIPTIVE	ASCRIPTIVE, MENTAL, MATERIAL, COMPOSITE, POSSESSIVE, LOCATIVE, TEMPORAL, VERBAL, EXISTENTIAL
	mode	nil	ATTRIBUTIVE, EQUATIVE, CAUSATIVE
	tense	PRESENT	PRESENT, PAST
	future	NO	YES, NO
	progressive	NO	YES, NO
	perfective	NO	YES, NO
	voice	ACTIVE	ACTIVE, PASSIVE
	quality	POSITIVE	POSITIVE, NEGATIVE
NOUN-PHRASE	np-type	COMMON	COMMON, PROPER
	person	THIRD	FIRST, SECOND, THIRD
	number	SINGULAR	SINGULAR, PLURAL
	gender	NEUTRAL	NEUTRAL, MASCULINE, FEMININE
	definite	NO	YES, NO, NOART
	regular-noun	YES	YES, NO
	countable	YES	YES, NO
	inflected	YES	YES, NO
	pronominal	NO	YES, NO
POSSESSOR	pronominal	YES	YES, NO
PRONOUN	type	PERSONAL	PERSONAL, OBJECTIVE, REFLEXIVE, POSSESSIVE-PRONOUN, POSSESSIVE-DETERMINER, RELATIVE, DEMONSTRATIVE
	person	FIRST	FIRST, SECOND, THIRD
	number	SINGULAR	SINGULAR, PLURAL
	gender	NEUTRAL	NEUTRAL, MASCULINE, FEMININE
CONJUNCTION	sentence	NO	YES, NO

Figure 2: Some Defaults from YAG's Syntactic Templates.

```
((template CLAUSE)
 (process-type MENTAL)
 (process "want")
 (processor ((template CONJUNCTION)
             (first ((template NOUN-PHRASE)
                     (head "Jack")
                     (np-type PROPER)
                     (gender MASCULINE) ))
             (second ((template PRONOUN)) )))
 (phenomenon ((template NOUN-PHRASE)
              (head "dog")
              (possessor ((template NOUN-PHRASE)
                          (head "sister")
                          (gender FEMININE)
                          (possessor ((template NOUN-PHRASE)
                                      (head "Jack")
                                      (np-type PROPER)
                                      (gender MASCULINE)
                                      (pronominal YES)) )))))
 (rear-circum ((template CLAUSE)
               (mood TO-INFINITIVE)
               (process-type MATERIAL)
               (process "swim")) ) )
```

Figure 3: A (shorter) Feature Structure of the Sentence *"Jack and I want his sister's dog to swim."*.

ture input so that neither the application, nor the templates need to be altered to handle dependencies, like conjunctions, correctly.

3 Attribute Grammars

An attribute grammar consists of a context-free grammar, a finite set of attributes, and a set of semantic rules. The *Context-Free Grammar* (CFG) specifies the syntax of a language by expressing how to construct a syntax tree from non-terminal and terminal symbols defined in a language. The *Attributes* and *Semantic Rules* specify the semantics. A finite set of attributes is associated with each non-terminal symbol. Each of these sets is divided into two disjoint subsets, namely *Inherited Attributes* and *Synthesized Attributes.* Inherited attributes propagate down a syntax tree whereas synthesized attributes propagate upward. A semantic rule specifies how to compute the value of an attribute from others. This specification implicitly defines dependencies among attributes in an attribute grammar, locally (within a production) and globally (among productions). *Attribute Evaluation* is the process of computing values for every attribute instance in the tree according to the semantic rules defined for each production.

An example of an attribute grammar and its components is given in Figure 4 (adapted from (Alblas, 1991)). This attribute grammar consists of two non-terminals, two terminals, and three production rules. The inherited attributes of the non-terminal A are a and b. Its synthesized attributes are x and y. No attributes are assigned to the non-terminal S.

```
nonterminals: S, A.
terminals: s, t.
start symbol: S.

description of attributes:
a, b: integer, inh of A;
x, y: integer, syn of A;

productions and semantic rules:
 1) S -> A.
    A.a := A.x
 2) A0 -> A1 s.
    A1.a := A0.a; A1.b := A1.y;
    A0.x := A1.x; A0.y := 1
 3) A -> t.
    A.y := A.a; A.x := A.b
```

Figure 4: An Example Attribute Grammar.

As mentioned earlier, semantic rules define dependencies among attributes. Figure 5 shows dependency graphs corresponding to the semantic rules of Figure 4. In the graphs, a dotted line represents a derivation of a production rule, while an arrow denotes an attribute dependency. Thus, $A \rightarrow B$ means B is dependent on A, but not the other way around. In other words, we cannot know B before we know A.

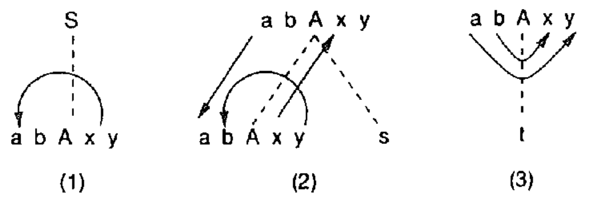

Figure 5: Dependency Graphs.

4 Extending a Grammar to Enable Generation Gap Analysis

To make a generation gap analysis possible, a grammar writer must first extend the grammar of his or her existing generator to capture the propagation semantics of a target language. This extension involves defining attributes (synthesized and inherited) and associated semantic rules. Next, a small program must be built to construct a tree from a given input and retrieve semantic rules and attributes from associated grammar units.

Attribute evaluation begins by instantiating each inherited attribute with values from the input and then the remaining attributes are evaluated. This process is incremental in the sense that new information gained from previous evaluations might lead to the discovery of additional information. When all attributes remain unchanged, or there is a conflict detected in the input, the process terminates. The generator then passes the enriched input to the realization component.

Consider the following fragment of input from Figure 3 that uses the CONJUNCTION template to join a noun phrase and a pronoun.

```
((template CONJUNCTION)
 (first ((template NOUN-PHRASE)
         (head "Jack")
         (np-type PROPER)
         (gender MASCULINE) ))
 (second ((template PRONOUN)) ))
```

This fragment is the subject of the sentence, therefore features such as **person** and **number** would be required to enforce the *subject-verb agreement* of English. Figure 6 shows a dependency graph[1] for this

[1] The notation used in the dependency graph is the following:
The oval represents a template slot that is bound to an atomic value. The rectangle denotes a slot that is bound to another feature structure. The top text in a rectangle specifies a slot name, and the bottom text is the name of a template assigned to this slot. A value with an underline means a default of the above slot. The bold font represents a value yielded from attribute evaluations.

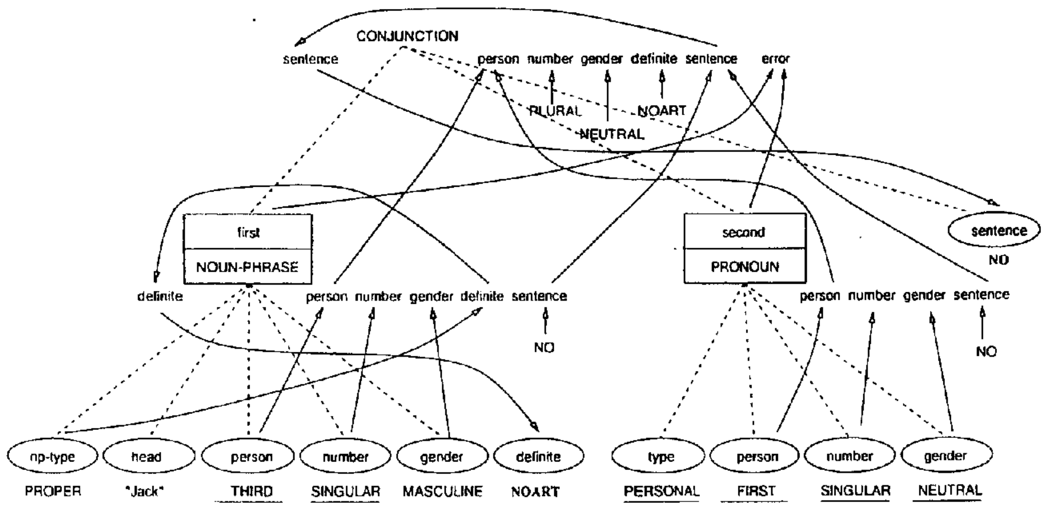

Figure 6: A Dependency Graph of the CONJUNCTION Template corresponding to the text *"Jack and I"*.

fragment. The dependencies are based on the semantic rules given in Figure 7 (Section 6 describes syntax of these rules.).

The semantic rules in Figure 7 give constraint information for the CONJUNCTION template, the NOUN-PHRASE template, and the PRONOUN template. For the CONJUNCTION template, the grammar will:

- Use the **sentence** feature of the current template (which is NO by default).
- Pass up the **person** feature found by comparing the **person** features associated with the two conjuncts (*i.e.*, pass up second person whenever the conjuncts combine either first person and second or third person, or they combine second person and third person; pass up third person if both conjuncts use third person; otherwise pass up nil);
- Constrain the **number** feature to be PLURAL, the **gender** feature to be NEUTRAL, the **definite** feature to be NOART, and the **sentence** feature to the same as the sentence feature of the conjuncts.

For the NOUN-PHRASE template, the grammar will

- Require this template to enforce the inherited values of the **definite**, **number**, and **np-type** features.
- Require the (embedded) DETERMINER template enforce the **number** feature of the current template.
- Pass up four features (**definite**, **number**, **person**, and **np-type**) to any templates that use this noun phrase, where the following constraints apply:

The definiteness feature that is passed is YES whenever the current template has inherited YES for this value or there is a possessor or a determiner and one of them passes up YES for this feature. (If there is neither possessor nor determiner then the grammar considers the **np-type**: if it is COMMON, it uses NO (for indefinite) and if it is PROPER, it uses NOART)

The number feature passed is the value passed from the determiner, if there is one, or the value from the current template.

The person feature passed is the one from the current template.

The np-type feature passed is COMMON if the value of definite is NO and PROPER if the value is NOART.

For the PRONOUN template, the grammar will:

- Pass up the **person**, **number**, and **gender** values from the current template (possibly using default values), along with the constraint that the string realized for it not be a sentence and not be preceded by an article.

In the example shown in Figure 6, inherited attributes[2] have been initialized to the associated values given in an input. If the input does not specify a value for an inherited attribute, then the value nil is used.

The attribute evaluation is *depth-first*, and requires multiple traversals. Here, the NOUN-PHRASE sub-tree is evaluated twice, as we discover that the **definite** feature must be NOART. Since the PRONOUN

[2] Inherited attributes are placed on the left side of each node. Synthesized attributes are on the right.

Template Name	Semantic Rules

CONJUNCTION	`((this sentence) (this inh sentence))`
	`((this syn person) (CASE (UNION (first syn person) (second syn person)) OF ((first nil) (second nil) ((first second) second) ((first third) second) ((second third) second) (third third)) ))`
	`((this syn number) PLURAL)`
	`((this syn gender) NEUTRAL)`
	`((this syn definite) NOART)`
	`((this syn sentence) (UNION (first syn sentence) (second syn sentence)))`

NOUN-PHRASE	`((this definite) (this inh definite))`
	`((this number) (this inh number))`
	`((this np-type) (this inh np-type))`
	`((determiner inh number) (this inh number))`
	`((this syn definite) (IF (AND (NULL (this possessor)) (NULL (this determiner))) THEN (UNION (this definite) (CASE (this np-type) OF ((common NO) (proper NOART))) ) ELSE (UNION (this definite) (possessor syn definite) (determiner syn definite)) ))`
	`((this syn number) (UNION (determiner syn number) (this number)) )`
	`((this syn person) (this person))`
	`((this syn np-type) (CASE (this definite) OF ((NO COMMON) (NOART PROPER)) ))`

PRONOUN	`((this syn person) (this person))`
	`((this syn number) (this number))`
	`((this syn gender) (this gender))`
	`((this syn sentence) NO)`
	`((this syn definite) NOART)`

Figure 7: Semantic Rules of the CONJUNCTION, NOUN-PHRASE, and PRONOUN template.

template has no inherited attributes, a single evaluation would be sufficient. The CONJUNCTION sub-tree is also traversed twice because the **sentence** feature is re-assigned once (from nil to NO).

Figure 8 shows the tree and dependencies, for the fragment, *"his sister's dog"*. It shows how the definiteness of a noun phrase is dependent on the existence of a possessor. For example, if a possessor (such as *"his"* or *"Jack's"*) is specified, a noun phrase will not need an article.

Note that this feature structure can be generated differently as *"Jack's sister's dog"*, *"her dog"*, *"the dog of Jack's sister"*, *"the dog of his sister"*, and *"the dog of hers"*. While some of these variations require further investigation to determine how to transform a tree so that it reflects a new ordering of constituents, some can be implemented using semantic rules. For example, to avoid an awkward construction such as *"Jack's sister's dog"* in the sen-

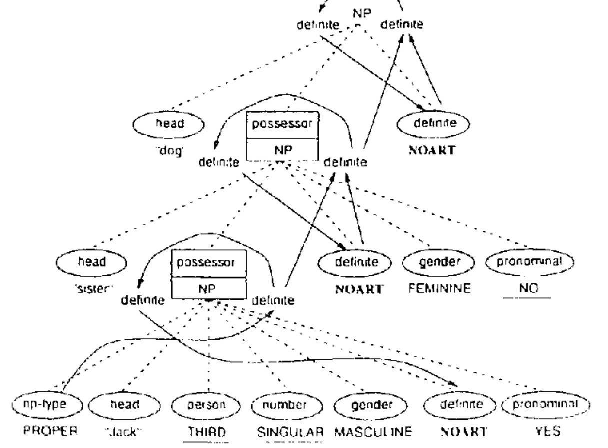

Figure 8: A Dependency Graph of the NOUN-PHRASE Template corresponding to the text *"his sister's dog"*.

tence *"Jack and I want Jack's sister's dog to swim."*, in favor of *"his sister's dog"*, without the application having to request a pronoun explicitly, as in the example shown above, we could add a rule to force the **pronominal** feature of the inner most possessor to be YES, whenever a (repeated) noun phrase is a possessor of a possessor of the primary noun.

5 The Use of the Generation Gap Analysis to Resolve Conflicting Information

One side benefit of the use of attribute grammars is that they can help resolve inconsistencies in the input provided from an application. Previously, a generation system might not be able to recognize such conflicts, and therefore might generate a text that is ungrammatical, or it might simply fail to produce an output at all.

The following is an example input that has a conflict; the values of the **number** feature in the NOUN-PHRASE and PRONOUN templates are inconsistent.

```
((template NOUN-PHRASE)
 (head "book")
 (number PLURAL)
 (determiner ((template PRONOUN)
              (type DEMONSTRATIVE)
              (distance NEAR)
              (number SINGULAR)) )
)
```

Executed literally, a generator would produce the phrase *"this books"*, rather than *"this book"* or *"these books"*. Figure 9 shows a dependency graph corresponding to the above input.

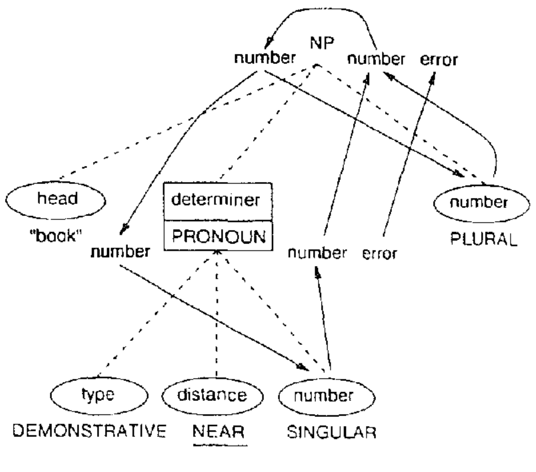

Figure 9: A Dependency Graph corresponding to the text *"this book"* or *"these books"*.

With the use of an appropriate attribute grammar, an analysis of this structure would detect a conflict when the value SINGULAR of the **number** feature propagates upward and conflicts with the value PLURAL of the **number** feature of the NOUN-PHRASE template. In this case, a generator can choose to override one of the conflicting features and generate a text from the revised input.

6 Implementation

Grammars in a template-based system differ sufficiently from phrase-based systems so that traditional attribute grammars specifications cannot be used without changes. In particular, grammars in a template-based system are not restricted to syntactic text structure as they are in phrase-based systems, but may include either syntactic specifications, semantic specifications, or a mixture of both. Therefore template-based grammars do not restrict derivations on the right side of a production to some specific non-terminals, as they would be in a phrase-based grammar.

In our approach, a template is equivalent to the non-terminal on the left side of a production. Template slots are equivalent to terminals and non-terminals on the right side depending on their value at the time of generation. Slots that are bound to a simple value are considered terminals, while those that are bound to a feature structure are considered non-terminals. The evaluation function of terminals is actually a constant function whose return value is the value to which the terminal has been bound.

We have defined a small language sufficient to specify attribute grammars in a template as given in Figure 10. Additional keywords are also defined. The keyword **this** refers to the current template. The keywords **inh** and **syn** indicate an inherited attribute and a synthesized attribute, respectively.

We have implemented an attribute grammar-based propagation analysis program in Lisp as an extension to YAG. Some templates have been augmented with semantic propagation rules. It was not necessary to define attributes for YAG's template-based grammar because template slots already served as attributes. The program has been able to identify missing information (using the defined semantic propagation rules) and to reject inputs that have a conflict.

Other generation systems that intend to use an attribute grammar approach to enrich their partially-specified input will need to analyze the characteristics of their grammar formalism. Basically, one needs to identify the smallest unit of a grammar (*e.g.*, a category (cat) in FUF/SURGE), and then define semantic rules similar to those presented in this paper for each grammar unit. From a given input, a generator should be able to pick semantic rules associated with information provided in an input. An attribute evaluation is then executed as described.

```
AttributeGrammar ::- EvalRules
EvalRules ::- "(" EvalRule EvalRules ")" | nil
EvalRule ::- "(" Attribute Stmt ")"

Stmt ::- Expr | CaseStmt | IfStmt

Expr ::- Attribute | constant |
         "(" "UNION" Stmt Stmt ")" |
         "(" "INTERSECTION" Stmt Stmt ")"
Attribute ::- inherited | synthesized

CaseStmt ::- "(" "CASE" Expr "OF" Alters ")"
Alters ::- "(" Alter Alters ")" | nil
Alter ::- "(" value result ")"
result ::- Expr

IfStmt ::- "(" "IF" Cond "THEN" Stmt ")" |
           "(" "IF" Cond "THEN" Stmt
                "ELSE" Stmt ")"
Cond ::- "(" "NULL" Expr ")" |
         "(" "EQUAL" Expr Expr ")" |
         "(" "NOT" Cond ")" |
         "(" "AND" Cond Cond ")" |
         "(" "OR" Cond Cond ")"
```

Figure 10: The Syntax of YAG's Attribute Grammars Specification.

7 Conclusion

We have presented a new approach to enriching under-specified representations of content to be realized as text using attribute grammars with semantic propagation rules. Our approach is not intended to replace defaulting mechanisms used in the current generation systems. Instead it improves the quality of input to the generator for better realization. Defaults are still used if the analysis fails to discover useful information.

Acknowledgement

The authors are indebted to John T. Boyland for his helpful comments and suggestions.

References

Henk Alblas. 1991. Introduction to attribute grammars. In Henk Alblas and Bořivoj Melichar, editors, *Attribute Grammars, Applications and Systems*, volume 545 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, New York-Heidelberg–Berlin. June. Prague.

Stephan Busemann. 1996. Best-first surface realization. In Donia Scott, editor. *Proceedings of the Eighth International Workshop on Natural Language Generation*. pages 101–110.

Songsak Channarukul. 1999. YAG: A Natural Language Generator for Real-Time Systems. Master's thesis, University of Wisconsin-Milwaukee, December.

Michael Elhadad. 1992. *Using argumentation to control lexical choice: A functional unification-based approach.* Ph.D. thesis, Computer Science Department, Columbia University.

Keven Knight and Vasileios Hatzivassiloglou. 1995. Two-level, many-paths generation. In *Proceedings of ACL.*

Donald E. Knuth. 1968. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, June. Correction: *Mathematical Systems Theory* 5, 1, pp. 95-96 (March 1971).

Benoit Lavoie and Owen Rambow. 1997. A fast and portable realizer for text generation systems. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 265–268, Washington.

Michael Levison and Gregory Lessard. 1990. Application of attribute grammars to natural language sentence generation. In Pierre Deransart and Martin Jourdan, editors, *Attribute Grammars and their Applications (WAGA)*, volume 461 of *Lecture Notes in Computer Science*, pages 298–312. Springer-Verlag, New York–Heidelberg–Berlin, September. Paris.

William C. Mann. 1983. An overview of the Penman text generation system. In *Proceedings of the Third National Conference on Artificial Intelligence (AAAI-83)*, pages 261–265, Washington, DC, August 22-26,. Also appears as USC/Information Sciences Institute Tech Report RR-83-114.

Susan W. McRoy, Songsak Channarukul, and Syed S. Ali. 1999. A Natural Language Generation Component for Dialog Systems. In *Working Notes of the AAAI Workshop on Mixed-Initiative Intelligence, at the 1999 Meeting of the American Association for Artificial Intelligence*, Orlando, FL.

Marie W. Meteer. 1990. *The "Generation Gap" The Problem of Expressibility in Text Planning*. Ph.D. thesis, University of Massachusetts.