

A Formal Syntax of Natural Languages and the Deductive Grammar

Yingxu Wang*

Visiting Professor, Dept. of Computer Science, Stanford University

Stanford, CA 94305-9010, USA

yingxuw@stanford.edu

International Center for Cognitive Informatics (ICfCI)

Theoretical and Empirical Software Engineering Research Centre (TESERC)

Dept. of Electrical and Computer Engineering, Schulich School of Engineering

University of Calgary, 2500 University Drive, NW, Calgary, Alberta, Canada T2N 1N4

yingxu@ucalgary.ca

Streszczenie. This paper presents a formal syntax framework of natural languages for computational linguistics. The abstract syntax of natural languages, particularly English, and their formal manipulations are described. On the basis of the abstract syntax, a universal language processing model and the deductive grammar of English are developed toward the formalization of Chomsky's universal grammar in linguistics. Comparative analyses of natural and programming languages, as well as the linguistic perception on software engineering, are discussed. A wide range of applications of the deductive grammar of English have been explored in language acquisition, comprehension, generation, and processing in cognitive informatics, computational intelligence, and cognitive computing.

Keywords: Cognitive informatics, linguistics, computational linguistics, formal languages, universal grammar, deductive grammar, formal syntax, formal semantics, EBNF, RTPA, comparative linguistics, software engineering, programming languages

*Address for correspondence: International Center for Cognitive Informatics (ICfCI), Theoretical and Empirical Software Engineering Research Centre (TESERC), Dept. of Electrical and Computer Engineering, Schulich School of Engineering, University of Calgary, 2500 University Drive, NW, Calgary, Alberta, Canada T2N 1N4

1. Introduction

Linguistics is a discipline that studies human or natural languages. Languages are an oral and/or written symbolic system for thought, self-expression, and communications. Lewis Thomas highlighted that “The gift of language is the single human trait that marks us all genetically, setting us apart from the rest of life [19].” This is because functions of languages can be identified as for memory, instruction, communication, modeling, thought, reasoning, problem-solving, prediction, and planning [2, 17, 33].

Linguists commonly agree there is a universal language structure known as the *universal grammar* [3, 4, 5, 6, 7, 8, 15, 17]. However, a grammar may be precise and explicit as in formal languages, or ambiguous and implied as in natural languages. Although a language string is symbolically constructed and for reading sequentially, all natural languages have the so called *metalinguistic ability* to reference themselves out of the sequences. That is, the ability to construct strings that refer to other strings in the language.

From a linguistic point of view, software science is the application of information technologies in communicating between a variety of stake holders in computing, such as professionals and customers, architects and software engineers, programmers and computers, as well as computing systems and their environments. Therefore, linguistics and formal language theories play important roles in computing theories, without them computing and software engineering theories would not be considered as complete.

It is noteworthy that, historically, *language-centered programming* had been the dominate methodology in computing and software engineering. However, this should not be taken as granted as the only approach to software development, because the expressive power of programming languages is inadequate to deal with complicated software systems. In addition, the extent of rigorousness and the level of abstraction of programming languages are too low to model the architectures and behaviors of software systems. This is why bridges in mechanical engineering or buildings in civil engineering were not modeled or described by natural or artificial languages. This observation leads to the recognition of the need for *mathematical modeling* of both software *system architectures* and *static/dynamic behaviors*, supplement with the support of automatic code generation systems [24].

This paper comparatively studies fundamental theories of natural and artificial languages. Formal languages and applications of mathematics in computational linguistics are investigated. This paper analyzes how linguistics may improve the understanding of programming languages and their work products - software, and how formal language theories may enhance the study on natural languages. In the remainder of this paper, formal syntaxes and semantics of natural languages are explored and the abstract syntax of English and its formal manipulations are described in Section 2, which results in a universal language processing model. Based on formal language theories, a deductive grammar of English is developed in Section 3 toward the formalization of the universal grammar proposed by Chomsky [4, 7]. Comparative analyses of natural and programming languages, as well as the linguistic perceptions on software engineering, are discussed in Section 4.

2. Formal Syntaxes and Semantics of Natural Languages

Syntaxes deal with relations and combinational rules of words in sentences, while semantics embody the meaning of words and sentences. This section presents a formal treatment of syntaxes and semantics

of natural languages, which forms a foundation of the universal language processing model and the deductive grammar of English.

2.1. The Formal Syntactic Model of Natural Languages

The syntactic rules of languages that underlie natural languages form the domains of formal linguistics and grammars. One of the most influential linguistic framework known as the theory of *universal grammar* (UG) was proposed by Noam Chomsky [4, 7]. UG and its modern version, the *Government and Binding Theory* [8], have become a linguistic premise on grammatical analyses in linguistics.

Definition 1. A *syntax* is a domain of linguistics that studies sentence formation and structures.

Definition 2. An *abstract syntax* is the abstract description of a syntax system where concrete strings of tokens and their grammatical relations are symbolically represented and analyzed.

Hierarchical tree schemas are conventionally adopted to denote sentence structures in syntactic analyses. In a syntactic perspective, any human language, natural or artificial, is a sequential or one-dimensional (1-D) symbol stream of syntactic blocks, which can be decomposed into paragraphs, sentences, phrases, words, and letters from the top-down. Although a sentence in a language is 1-D, its grammar is recursively structured in a 2-D space. For instance, the abstract productions, $A \rightarrow (a, Aa, B)$ and $B \rightarrow b$, can be denoted as:

$$\begin{array}{c}
 A \\
 / \quad | \quad \backslash \\
 a \quad Aa \quad B \\
 \quad \quad | \\
 \quad \quad b
 \end{array} \tag{1}$$

Observing Eq. 1 and then Table 3, the basic properties of syntaxes of natural languages can be formally described below.

Theorem 1. The *syntax of natural languages* is two-dimensionally composable and decomposable.

Proof:

Let p be an arbitrary production in a grammar G , and T_p^2 a 2-D syntax tree. Because $\forall p \in G: T_p^2$, and $\forall T_p^2: p \in G$, thus $G \Leftrightarrow T_p^2$. \square

However, the semantics of languages expressed and implied by the 2-D syntaxes can be more complicated, which are non-sequential and multi-dimensional in most cases, such as the branch, parallel, embedded, concurrent, interleaved, and interrupt structures as given in Table 1.

All semantic relations of sentences in natural languages can be rigorously treated by Real-Time Process Algebra (RTPA) [20, 24, 29, 30] in denotational mathematics [27, 32].

Theorem 2. The *semantic relations* of sentences, \mathcal{R} , in natural languages are a finite set of semantic connectors which obey the formal semantics of RTPA, i.e.:

Table 1. Semantic Relations of Sentences

No.	Relation	Symbol	Description
1	Sequential	\rightarrow	and, then
2	Branch		or, however, but
3	Parallel		and, simultaneously (action by the same subject)
4	Embedded	\rightsquigarrow	that, which, if, whether
5	Concurrent	⌘	and, simultaneously (action by different subjects)
6	Interleave		alternatively
7	Interrupt	\nrightarrow	when, while, during

$$\mathcal{R} = \{\rightarrow, |, ||, \rightsquigarrow, \lrcorner, |||, \nrightarrow\} \subseteq \mathfrak{R}_{RTPA} \quad (2)$$

As given in Theorem 1 and Table 1, the semantic relations of sentences \mathcal{R} are a set of important connectors, which formally models phrase and sentence compositions and their joint meaning in complex sentence structures. The seven semantic relations in Theorem 2 are a subset of the 17 process relations \mathfrak{R}_{RTPA} as defined in RTPA [29].

Definition 3. The set of syntactic elements \mathcal{S} in natural languages can be classified into the categories of *lexical*(\mathcal{L}), *functional*(\mathcal{F}), *phrasal*(\mathcal{P}), and *relational*(\mathcal{R}), i.e.:

$$\begin{aligned} \mathcal{S} &\triangleq (\mathcal{L}, \mathcal{F}, \mathcal{P}, \mathcal{R}) \\ &= \{N, V, A, \Lambda, P\} \\ &|| \{\tau, \delta, \kappa, \alpha, \gamma, \neg\} \\ &|| \{NP, VP, AP, \Lambda P, PP, CP\} \\ &|| \{\rightarrow, |, ||, \rightsquigarrow, \lrcorner, |||, \nrightarrow\} \end{aligned} \quad (3)$$

where further details of each syntactic elements may be referred to Table 2.

A set of four categories and 25 lexical and syntactic elements of languages is summarized in Table 2, where an element in angular brackets is optional. In Table 2, there is a special category of lexical components known as *complement phrases* (CPs). CPs can be a supplemental part of N/NP, V/VP, A/AP, or P/PP. The rules for defining relations between CPs and other lexical categories of sentences may be referred to [15].

2.2. Formal Means for Syntactic and Semantic Analyses

A Backus-Naur form is a recursive notation for describing the productions of a context-free grammar. It is developed based on the work of John Backus with contributions by Peter Naur [Naur, 13, 14].

Table 2. Definition of Lexical and Syntactic Categories of Languages

Category	Subcategory	Symbol	Description
Lexical (\mathcal{L})	Noun	N	Entities and abstract objects
	Verb	V	Actions, states, and possessions
	Adjective	A	Properties of a noun
	Adverb	Λ	Properties of a verb
	Preposition	P	Designated relations in space or time
Functional (\mathcal{F})	Determiner	τ	the, a, this, these, etc.
	Degree word	δ	too, so, very, more, quite, etc.
	Qualifier	κ	almost, always, often, perhaps, never, etc.
	Auxiliary	α	will, can, may, must, should, could, etc.
	Conjunction word	γ	and, or, that, which, if, whether, etc.
	Negative	\neg	not
Phrasal (\mathcal{P})			<i>A syntactic unit with one or more words as a lexical category</i>
	Noun phrase	NP	τ N [PP]
	Verb phrase	VP	V NP etc.
	Adjective phrase	AP	[δ] A [PP]
	Adverb phrase	Λ P	[Λ] V V [Λ]
	Prepositional phrase	PP	[δ] P [NP]
	Complement phrase	CP	Supplemental part of N/NP, V/VP, A/AP, or P/PP
Relational (\mathcal{R})		\mathcal{R}	<i>A set of connectors</i>
	Sequential	\rightarrow	and, then
	Branch		or
	Parallel		and, simultaneously (action by the same subjects)
	Embedded	\dashrightarrow	that, which, if, whether
	Concurrent	⌘	and, simultaneously (action by different subjects)
	Interleave		alternatively
Interrupt	\Leftarrow	when, while, during	

Definition 4. A Backus-Naur Form (BNF) is defined by a 5-tuple:

$$BNF \hat{=} (\Sigma, T, V, P, S) \tag{4}$$

where

- (i) Σ is a finite nonempty set of alphabet;
- (ii) T is a finite set of terminals, $T \subseteq \Sigma$;
- (iii) V is a finite set of nonterminals, $V \subseteq \Sigma \wedge V = \Sigma \setminus T$;
- (iv) P is a finite set of production rules denoted by $\alpha ::= \beta$;

- (v) S is a finite set of metasympols that denote relations of the multiple derived products β s separated by alternative selection $|$.

For example, the BNF counterparts of Eq. 1 can be recursively denoted by:

$$\begin{aligned} A &::= a|Aa|B \\ B &::= b \end{aligned} \quad (5)$$

BNF is found useful to define context-free grammars of programming languages, because of its simple notations, recursive structures, and widely available support by many compiler generation tools such as YACC [9], LEX [10], and ANTLR [16].

However, it is realized in applications that the descriptive power of BNF may be greatly improved by introducing a few extended metasympols, particularly those for *repetitive* and *optional* structures of grammar rules. There are a variety of extended BNFs proposed for grammar description and analysis [34]. A typical EBNF is given below.

Definition 5. An *extended Backus-Naur Form* (EBNF) is defined by a similar 5-tuple, i.e.:

$$EBNF \hat{=} (\Sigma, T, V, P, S') \quad (6)$$

with an extended set of metasympols $S = \{(), ()^+, []\}$ beyond BNF, where:

- (i) The metasympol β^* and β^+ are adopted to denote the recursive structures of derived products, which can be described by Wang's big-R notation [31], i.e.:

$$\beta^* = \mathbf{R}_{i=0}^n \beta_i \quad (7)$$

$$\beta^+ = \mathbf{R}_{i=0}^n \beta_i \quad (8)$$

- (ii) The metasympol $[\beta]$ is adopted to denote optional structures of a derived product.

Corresponding to the EBNF description of the syntactic structures of a natural or programming language, a flow diagram known as *syntax diagram* can be used to illustrate the rules and syntactic structures. Syntax diagrams form the second approach to denoting syntactic structures of languages.

The descriptive power of EBNF can be further extended by RTPA's algebraic process relations, particularly the big-R notation and more complicated sentence composing structures such as concurrent, interrupt, and causal relations.

Definition 6. The *big-R notation* is a mathematical operator that is used to denote: (a) a finite set of *repetitive* behaviors, or (b) a finite set of recurring architectural constructs in computing, in the following forms:

Table 3. Typical Syntactic Entities and Structures in EBNF and their Mathematical Models in RTPA

No	Syntactic structure	EBNF notation	Syntax diagram	RTPA notation
1	Serial	$S ::= S_1 S_2 \dots S_n$		$S = S_1 \rightarrow S_2 \rightarrow \dots \rightarrow S_n$
2	Serial with option	$S ::= S_1 [S_2] \dots S_n$		$S = S_1 \rightarrow S_2 \rightarrow S_3 \dots \rightarrow S_n$ $ S_1 \rightarrow S_3 \rightarrow \dots S_n$
3	Repeat serial for 0 or more times	$S ::= (S_1 S_2 \dots S_n)^*$		$R^*(S_1 \rightarrow S_2 \rightarrow \dots \rightarrow S_n)$
4	Repeat serial for 1 or more times	$S ::= (S_1 S_2 \dots S_n)^+$		$R^+(S_1 \rightarrow S_2 \rightarrow \dots \rightarrow S_n)$
5	Alternative	$S ::= S_1 S_2 \dots S_n$		$S = S_1 S_2 \dots S_n$
6	Alternative with option	$S ::= [S_1 S_2 \dots S_n]$		$S = S_1 S_2 \dots S_n \emptyset$
7	Repeat alternative for 0 or more times	$S ::= (S_1 S_2 \dots S_n)^*$		$R^*(S_1 S_2 \dots S_n)$
8	Repeat alternative for 1 or more times	$S ::= (S_1 S_2 \dots S_n)^+$		$R^+(S_1 S_2 \dots S_n)$

$$(a) \quad \overset{\mathbf{F}}{R} \underset{\text{exp } \mathbf{BL}=\mathbf{T}}{P} \quad (9.1)$$

$$(b) \quad \overset{n}{R} \underset{i \in \mathbf{N}=1}{P(i)} \quad (9.2)$$

where **BL** and **N** are the type suffixes of Boolean and natural numbers, respectively, as defined in RTPA.

Table 3 contrasts the three syntactical description techniques for typical syntactic entities and structures in EBNF, syntax diagrams, and RTPA. In the syntax diagrams, a terminal and a nonterminal are represented by an oval and a square, respectively.

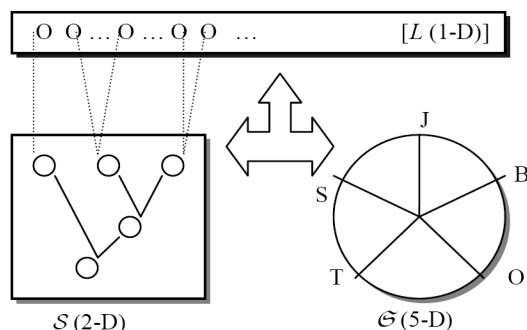


Figure 1. The Universal Language Processing (ULP) model

2.3. The Formal Semantic Model of Natural Languages

Definition 7. *Semantics* is a domain of linguistics that studies the interpretation of words and sentences, and analyses of their meanings.

Semantics deals with how the meaning of a sentence in a language is obtained and comprehended. Studies on semantics explore mechanisms in the understanding of language and the nature of meaning where syntactic structures play an important role in the interpretation of sentence and the intension and extension of word meaning [3, 4, 5, 6, 7, 8, 18, 26].

Theorem 3. The *mathematical model of semantics* of natural languages, \mathfrak{S} , is a 5-tuple, i.e.:

$$\mathfrak{S} \triangleq (J, B, O, T, S) \quad (10)$$

where

- J is the subject of the sentence.
- B is a behavior or action.
- O is the object of the sentence.
- T is the time when the action is occurring.
- S is the space where the action is occurring.

According to Theorem 3 (\mathfrak{S}) and Definition 3 (S), the relationship between a language and its syntaxes and semantics can be illustrated as shown in Fig. 1. Fig. 1 explains that linguistic analyses are a deductive process that maps the 1-D language into the 5-D semantics via the 2-D syntactical analyses.

Corollary 1. The semantics of a sentence is comprehended *iff*:

- (a) The logical relations of parts of the sentence are clarified;
- (b) All parts of sentence are reduced to the terminal entities, which are either a real-world image or a primitive abstract concept.

Semantic analysis and comprehension are a deductive cognition process. Further discussions on the theoretical foundations on language cognition, comprehension, and concept algebra may be referred to [21,22,24,25,28].

3. Formalization of UG by the Deductive Grammar

Syntactic and semantic analyses in linguistics rely on a set of explicitly described rules known as the grammar of a language. Therefore, contemporary linguistic analyses focus on the study of grammars, which is centered in language acquisition, understanding, and interpretation.

Definition 8. The *grammar* of a language is a set of common rules that integrates phonetics, phonology, morphology, syntax, and semantics of the language.

The grammar governs the articulation, perception, and patterning of speech sounds, the formation of phrases and sentences, and the interpretation of utterance. This section analyzes the basic properties of Grammars of natural languages and introduces the Universal Grammar (UG). Then, the deductive grammar and its formal descriptions are developed.

3.1. Properties of Grammars and UG

O.Grady and Archibald identified five basic properties of grammars known as the generality, parity, universality, mutability, and inaccessibility [15].

Lemma 1. Natural languages share the following five fundamental properties:

- Property 1: *Generality* - All languages have a grammar;
- Property 2: *Parity* - All grammars are equivalent in terms of their expressive capacity;
- Property 3: *Universality* - Grammars are commonly alike, or basic principles and properties are shared in all languages;
- Property 4: *Mutability* - Grammars of all languages are constantly changing over time;
- Property 5: *Inaccessibility* - Grammatical knowledge of the mother tongue is built at the subconscious layer of the brain.

The above basic properties of grammars form an important part of the foundations of human intelligence. An important discovery in modern linguistics is the existence of the universal grammar among human languages [8].

Definition 9. The *Universal Grammar* (UG) is a system of categories, mechanisms, and constraints shared by all human languages.

UG is perceived as innate based on recent neurolinguistic and psycholinguistic studies [8, 15]. UG treats all languages with the same generic type of syntactic mechanisms, which include the *merge* and *transformation* operations. The former is a syntactic operation that combines words in accordance with

their syntactic categories and properties; while the latter is a syntactic operation that puts words and phrases in an appropriate structure.

Based on Property 2 of natural languages in Lemma 1, the expressive parity of language grammars can be formally expressed below.

Theorem 4. The *principle of expressive parity* states that all grammars of natural languages are equivalent.

Proof:

The top-level sentence in any language, S , can be formalized as:

$$S ::= [Subject] Predicate \mid S\gamma S \quad (11)$$

where $[]$ represents a term in it is optional, $|$ stands for or, and γ a set of conjunction words as identified in Table 1, i.e., $\gamma \in \mathcal{R}$.

Thus, Theorem 4 is hold because Eq. 11 is generally inherited by any grammar of natural languages. \square

Based on Theorem 4, it is perceived that, in computing and software engineering, all programming languages are equivalent. In other words, no programming language may claim a primitive status over others as long as they implement the core expressive power known as the 17 meta-processes and 17 process relations as identified in RTPA [20, 29, 32], which form the essential set of fundamental operations in computing [24].

3.2. The Deductive Grammar of English

An instance of UG is the English grammar. Formal language theories of computing science and software engineering perceive that the grammar of any programming language or professional notation system may be rigorously defined by the EBNF notation [13, 14]. The author found that the formal language theory can be extended to describe and analyze the grammars of natural languages such as that of English.

Definition 10. The *deductive grammar* is an abstract grammar that formally denotes the syntactic rules of a language based on which, as a generic formula, valid language sentences can be deductively derived.

On the basis of the definitions of the syntactic elements as given in Table 2, the English grammar can be formally described in EBNF known as the Deductive Grammar of English (DGE). A rigorous definition of DGE at the sentence level is given in Fig. 2. Some aspects of DGE are simplified at the bottom level, particularly on person rules of nouns, time rules of verbs, and the matching of nouns and verbs in sentences.

According to DGE, the schema of the most complicated sentence in English that consists of all possible and legal syntactic components of DGE is shown in Fig. 3. The generic schema of DGE can be used as a universal formula to deductively derive any sentence in English. For example, the shortest possible sentence is given in Example 1 in Fig. 3. The longest possible sentence is presented in Example 3, i.e.:

“The unregistered new student all in the class [and another phrase] will not get the expected comprehensive handbook directly from the teacher [or another sentence].”

<p> $S ::= [\text{Subject}] \text{ Predicate}$ $S \ \gamma \ S$ $\text{Subject} ::= \text{NP}$ $\text{NP} ::= \tau[\text{AP}] \text{ N } [\text{PP}]$ τN^* $\text{NP} \ \gamma \ \text{NP}$ $\text{AP} ::= [\Lambda] \text{A}$ $\text{AP} ::= [\delta] \Lambda$ $[\kappa] \Lambda$ $\text{PP} ::= [\Lambda] \text{P} [\text{NP}]$ $\text{VP} ::= \text{VP} \ \gamma \ \text{VP}$ $[\alpha] [\neg] \text{V } [\text{Object}]^*$ $[\text{AP}] \text{V } [\neg] [\text{Object}]^*$ $\text{V} \neg [\text{Object}]^* [\text{AP}]$ $\neg ::= \langle \text{not} \rangle$ $\text{N} ::= \langle \text{nouns} \rangle$ $\text{V} ::= \langle \text{to be} \rangle$ $\langle \text{to have} \rangle$ $\langle \text{to do} \rangle$ $\text{P} ::= \langle \text{propositions} \rangle$ $\text{A} ::= \langle \text{adjective} \rangle$ $\Lambda ::= \langle \text{adverbs} \rangle$ $\delta ::= \langle \text{degree words} \rangle$ $\kappa ::= \langle \text{qualifier words} \rangle$ $\alpha ::= \langle \text{auxiliary words} \rangle$ $\tau ::= \langle \text{determiner words} \rangle$ $\gamma ::= \langle \text{conjunction words} \rangle$ $\langle \rangle$ $\langle ; \rangle$ </p>

Figure 2. The Deductive Grammar of English (DGE)

As provided in Fig. 3, Example 3 is an instance that uses almost all possible syntactic components. Obviously, natural sentences in practical usages are always a subset of the DGE schema. Therefore, their syntaxes are rather simple and short as shown in the first two examples in Fig. 3.

The 1-D structured sentences as shown in Fig. 3 can be modeled in a 2-D graphical form as shown in Fig. 4.

Observing Figs. 2 through 4, it is noteworthy that the syntactic structure of the DGE schema is highly recursive. The recursive characteristics in Fig. 4 are repetitively represented by the none phrases (NP) and verb phrases (VP).

No.	S (Sentence)																						
	Subject										Action												
	NP										VP												
	NP					γ	NP	VP					γ	VP									
	τ	AP	N	PP	γ	NP	α	\neg	V	Object *					γ	VP							
Λ	A		Λ	P	τ	N	...		τ	Λ	A	N	Λ	P	τ	N	...						
1																							
2																							
3	a	b	b	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w

Note: Words in Sentence 3 are given as follows:
 a – The, b – unregistered, c – new, d – student, e – all, f – in, g – the, h – class, i – and,
 j – ..., k – will, l – not, m – get, n – the, o – expected, p – comprehensive, q – handbook,
 r – directly, s – from, t – the, u – teacher, v – or, w – another sentence.

Figure 3. The schema of a generic sentence based on DGE

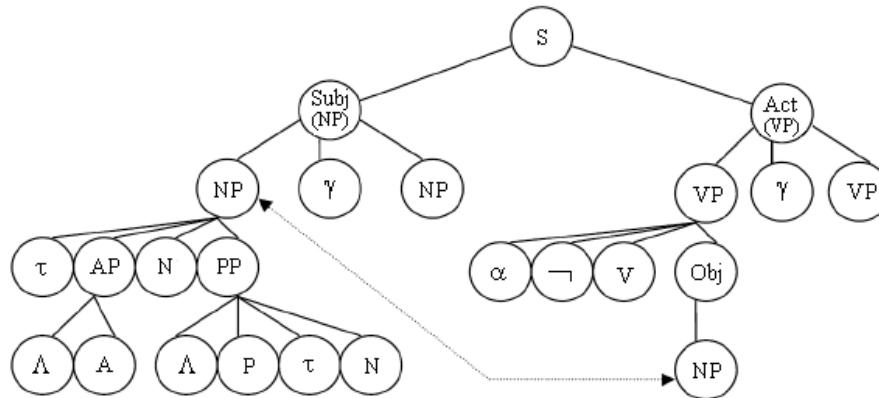


Figure 4. The syntax structure of the generic sentence schema in DGE

4. Comparative Analysis of Natural and Programming Language Theories

On the basis of the formal syntaxes, semantics, and DGE, a comparative study can be conducted between the linguistic properties of natural and programming languages in this section.

The fundamental expressiveness of natural languages can be classified as shown in Table 4. It is observed [Wang, 2007a] that although natural languages can be rich, complex, and powerfully descriptive, they do share three common basic structures known as the meta-expressivenesses of ‘to be ($=$)’, ‘to have (\subset)’, and ‘to do (\supset)’. as shown in Table 4.

The formal models of UG and DGE provide linguists, particularly language analyzers, implementers, and recognizers, for a powerful tool to formally describe and process natural language documents. Perspective applications of DGE may be in the development of Internet searching engines, semantic analysis of natural languages, speech recognitions, and intelligent systems for natural language parsing and word processing.

Table 4. Fundamental Elements in Natural Languages

Function	Category	Notation	Example
Identify <i>objects</i> and <i>attributes</i>	To be	$ =$	$A = B \Rightarrow (A \text{ is } B)$
Describe <i>relations</i> and <i>possession</i>	To have	$ <$	$A < B \Rightarrow (A \text{ has } B)$
Describe <i>status</i> and <i>behaviors</i>	To do	$ >$	$A > B \Rightarrow (A \text{ does } B)$

A summary of the comparative analysis of programming and natural languages is provided in Table 5. Intuitively, it is expected that a programming language would be a small subset of natural languages. Surprisingly, this hypothesis is only partially true at the morphology (lexicon) and semantic levels. However, the syntax of programming languages is far more complicated than those of natural languages.

Table 5. Comparative Analysis of Natural and Programming Language Properties

No	Category	Natural language	Programming language
1	Phonetics	Small	N/A
2	Phonology	Complex	N/A
3	Morphology (lexis)	Very large (> 60,000 words)	Small (< 1,000 instructions/reserved words)
4	Syntax	Simple (< 100 rules)	Very complicated (> 1,000 rules)
5	Semantics	Very complex (5-D)	Simple (2-D)
6	Grammar	Context sensitive	Context free
7	Applications	Thought, communications	Computing, system control

It is noteworthy in Table 5 that the semantics of programming languages is much simpler than that of natural languages, which is determined by the basic objectives of applications that should be suitable for limited machine intelligence. However, for achieving such simple and precise semantics in programming languages, a set of very complex and rigorous syntax and grammatical rules has to be adopted. Further discussion on semantics of programming languages may be referred to [12, 23, 24, 30].

More generally, it is noteworthy that there is no clear-cut between syntax and semantics in both natural and programming languages as formally stated below.

Theorem 5. The *principle of tradeoff between syntaxes and semantics* states that in the DGE system, the complexities of the syntactic rules (or grammar) C_{syn} and of the semantic rules C_{sem} are inversely proportional, i.e.:

$$C_{syn} \propto \frac{1}{C_{sem}} \quad (12)$$

Theorem 5 indicates that the simpler the syntactic rules or the grammar, the richer or complicated the semantics, and vice versa. Because UG or DGE for natural languages as formally defined in Fig. 2 are relatively simple, its semantics are much richer, complicated, and more ambiguity. In contrary, because programming languages adopt very detailed and complicated grammars, their semantics are relatively concise, simple, and rigor.

The finding in Theorem 5 indicates that syntactic and semantic rules are equivalent and interchangeable in linguistics. A simple syntax will require for a complex semantics, while a complex syntax will result in a simple semantics.

It is noteworthy that a natural language is usually *context sensitive*. However, almost all programming languages, no matter at machine level or higher level, are supposed to be *context free*. Therefore, it is interesting to query if a real-world problem and its solution(s), in a context-dependent manner, can be described by a context-free programming language in software engineering without losing any information. Automata and compiler theories [1, 11] indicate a context-sensitive language may be transformed into a corresponding context-free language. But the costs to do so are really expensive, because the context cannot be freely removed. A common trick to do so is to hide (imply) the context of software in data objects and intermediate data structures in programming. However, the drawbacks of this convention, or the limitations of conventional compiling technologies, make programming hard and complicated, because the computational behaviors and their data objects were separated or incoherent in the language's descriptive power. This observation suggests that a much natural and context-dependent programming language and related compiling technology are yet to be sought. Actually, Abstract Data Types (ADTs), object-oriented programming technologies, and software patterns are paradigms of those context-dependent programming languages, because the context (in the form of a set of data objects) has been encapsulated into a set of individual classes and the whole class hierarchy of a software system.

5. Conclusion

This paper has comparatively analyzed fundamental theories and formal models of natural and artificial languages. As a result, a formal syntax of natural languages and the deductive grammar of English (DGE) have been rigorously modeled. This work has explained how formal linguistics may improve the understanding of programming languages and their work products - software, as well as how formal language theories may extend the study on natural languages. The findings on features of natural and programming languages on morphologies, syntaxes, semantics, and grammars have been formally described, which lead to the development of the Universal Language Processing (ULP) model and DEG as a formalized model of the universal grammar. Applications of DEG have been identified in language acquisition, comprehension, generation, and processing in software and intelligent systems, as well as cognitive informatics.

Acknowledgement

The author would like to acknowledge the Natural Science and Engineering Council of Canada (NSERC) for its partial support to this work. The author would like to thank anonymous reviewers for their valuable comments and suggestions to the earlier versions of this work.

Literatura

- [1] Aho, A.V., R. Sethi, and J.D. Ullman: *Compilers: Principles, Techniques, and Tools*, Addison-Wesley Publication Co., New York, 1985.
- [2] Casti J.L. and A. Karlqvist eds.: *Complexity, Language, and Life: Mathematical Approaches*, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1986.
- [3] Chomsky, N.: Three Models for the Description of Languages, *I.R.E. Transactions on Information Theory*, **2**(3), 113-124, 1956.
- [4] Chomsky, N.: *Syntactic Structures*, Mouton, the Hague, 1957.
- [5] Chomsky, N.: On Certain Formal Properties of Grammars, *Information and Control*, **2**, 137-167, 1959.
- [6] Chomsky, N.: Context-Free Grammar and Pushdown Storage, *Quarterly Progress Report*, MIT Research Laboratory, **65**, 187-194, 1962.
- [7] Chomsky, N.: *Aspects of the Theory of Syntax*, MIT Press, Cambridge, MA, 1965.
- [8] Chomsky, N.: *Some Concepts and Consequences of the Theory of Government and Binding*, MIT Press, Cambridge, MA, 1982.
- [9] Johnson, S.C.: Yacc - Yet Another Compiler Compiler, AT&T Bell Laboratories, *Computing Science Technical Report No.32*, AT&T Bell Labs., Murray Hill, NJ, 1975.
- [10] Lesk, M.E.: Lex - A Lexical Analyzer Generator, AT&T Bell Laboratories, *Computing Science Technical Report No.39*, Murray Hill, NJ, 1975.
- [11] Lewis, H. R. and Papadimitriou, C. H.: *Elements of the Theory of Computation*, 2nd ed., Prentice-Hall International, Englewood Cliffs, NJ, 1998.
- [12] McDermid, J. ed.: *Software Engineer's Reference Book*, Butterworth Heinemann Ltd., Oxford, UK, 1991.
- [13] Naur, P. ed.: Revised Report on the Algorithmic Language Algol 60, *Communications of the ACM*, **6**(1), 1-17, 1963.
- [14] Naur, P.: The European Side of the Last Phase of the Development of Algol, *ACM SIGPLAN Notices*, **13**, 15-44, 1978.
- [15] O'Grady, W. and J. Archibald: *Contemporary Linguistic Analysis: An Introduction*, 4th ed., Pearson Education Canada Inc., Toronto, 2000.
- [16] Parr, T.: *ANTLR Reference Manual*, <http://www.antlr.org/>, 2000.
- [17] Pattee, H.H.: Universal Principles of Measurement and Language Functions in Evolving Systems, in J.L. Casti and A. Karlqvist eds. (1986), *Complexity, Language, and Life: Mathematical Approaches*, Springer-Verlag, Berlin, 268-281, 1986.
- [18] Tarski, A.: The Semantic Conception of Truth, *Philosophic Phenomenological Research*, **4**, 13-47, 1944.
- [19] Thomas, L.: *The Lives of a Cell: Notes of a Biology Watcher*, Viking Press, NY, 1974.
- [20] Wang, Y.: The Real-Time Process Algebra (RTPA), *Annals of Software Engineering*, Springer, USA, **14**, 235-274, 2002.
- [21] Wang, Y.: On Cognitive Informatics, *Brain and Mind: A Transdisciplinary Journal of Neuroscience and Neurophilosophy*, **4**(2), 151-167, 2003.

- [22] Wang, Y.: Keynote: Cognitive Informatics - Towards the Future Generation Computers that Think and Feel, *Proc. 5th IEEE International Conference on Cognitive Informatics (ICCI'06)*, Beijing, China, IEEE CS Press, July, 3-7, 2006.
- [23] Wang, Y.: On the Informatics Laws and Deductive Semantics of Software, *IEEE Transactions on Systems, Man, and Cybernetics (C)*, **36**(2), March, pp.161-171, 2006.
- [24] Wang, Y.: *Software Engineering Foundations: A Software Science Perspective*, CRC Series in Software Engineering, Vol. II, Auerbach Publications, USA, July, 2007.
- [25] Wang, Y.: The Theoretical Framework of Cognitive Informatics, *Int'l Journal of Cognitive Informatics and Natural Intelligence*, USA, **1**(1), 1-27, 2007.
- [26] Wang, Y.: The OAR Model of Neural Informatics for Internal Knowledge Representation in the Brain, *Int'l Journal of Cognitive Informatics and Natural Intelligence*, USA, **1**(3), 64-75, 2007.
- [27] Wang, Y.: On Contemporary Denotational Mathematics for Computational Intelligence, *Transactions of Computational Science*, Springer, August, **2**, 6-29, 2008.
- [28] Wang, Y.: On Concept Algebra: A Denotational Mathematical Structure for Knowledge and Software Modeling, *Int'l Journal of Cognitive Informatics and Natural Intelligence*, USA, **2**(2), 1-19, 2008.
- [29] Wang, Y.: RTPA: A Denotational Mathematics for Manipulating Intelligent and Computational Behaviors, *Int'l Journal of Cognitive Informatics and Natural Intelligence*, USA, **2**(2), 44-62, 2008.
- [30] Wang, Y.: Deductive Semantics of RTPA, *Int'l Journal of Cognitive Informatics and Natural Intelligence*, USA, **2**(2), 95-121, 2008.
- [31] Wang, Y.: On the Big-R Notation for Describing Iterative and Recursive Behaviors, *Int'l Journal of Cognitive Informatics and Natural Intelligence*, USA, **2**(1), 17-28, 2008.
- [32] Wang, Y.: Mathematical Laws of Software, *Transactions of Computational Science*, Springer, Aug., **2**, 46-83, 2008.
- [33] Wang, Y.: On Abstract Intelligence: Toward a Unified Theory of Natural, Artificial, Machinable, and Computational Intelligence, *Int'l Journal of Software Science and Computational Intelligence*, USA, **1**(1), 1-17, 2009.
- [34] Wirth, N.: *Algorithm + Data Structures = Programs*, Prentice Hall, Englewood Cliffs, NJ, 1976.