PARSING KOREAN BASED ON REVISED PTQ

Key-Sun Choi & Gil Chang Kim

Department of Computer Science
Korea Advanced Institute of Science and Technology
P.O.Box 150 Cheongryang, Seoul 131 Korea

Abstract

The natural language parsing has two aspects: the linguistic aspect and the computational aspect. The interaction between two aspects should be also considered, and a model is required to preserve two structures representing them. In this paper, a model called SPN (Structure Preserved Net), which is an extended version of the Petri net, is proposed for the purpose of preserving those two structures. In the linguistic aspect, SPN is based on the RPTQ (Revised PTQ) to handle several phenomena of Korean. In the computational aspect, the Determinism Hypothesis is considered to parse korean deterministically. It is shown that SPN is a model which represents the syntax of RPTQ and also satisfies the Determinism Hypothesis.

1. Introduction

This paper proposes a model called SPN (Structured Preserved Net) for parsing Korean. We concentrate on representing the syntax to recognize the well-formed sentences. The parsing problem of natural languges generally has two aspects: the linguistic aspect and the computational aspect (Robinson 1981).

To handle the linguistic aspect of Korean, SPN is based on a grammatical framework, called RPTQ (Revised PTQ) (Lee 1981a, 1981b, 1982). By using the case assignment, Lee (1981a) argues that RPTQ explains phenomena of free word-order languages such as Korean, Japanese, Latin and so on. SPN is designed to represent the syntactic rules of RPTQ naturally.

In the computational aspect, the Determinism Hypothesis (Marcus 1980) is considered. He claims that natural languages are designed to be deterministically parsed from left to right and that writing a grammar should consist in finding out local clues which enable the parser to select properly what to do next. Bień and Szpakowicz (1982) also says that this idea seems even more advantageous for free word-order languages. SPN is designed such that Korean can be parsed deterministically.

We considered two aspects, which are the RPTQ in the linguistic aspect and the Determinism Hypothesis in the computational aspect. The interaction between two aspects should be also considered, and there is the need to bridge gaps between two structures representing each aspect. SPN reflects both structures, and preserves both characteristics of them. The SPN (Structure Preserved Net) is named in this sense.

In section 2, we introduce RPTQ briefly, and describe the characteristics of Korean. Section 3 shows the motivation and the definition of SPN. SPN is an extended version of the Petri net (Peterson 1981). The fundamental definitions of the Petri net is introduced for the sake of completeness. Next, the SPN realization of RPTQ will be shown, and simple parsing example be presented. Section 4 gives a proof that SPN satisfies the Determinism Hypothesis for Korean.

2. Introduction to Revised PTQ

RPTQ is a revised version of Montague's (1974) PTQ. RPTQ accomodates case assignment and free concatenation in Korean (Lee 1981a, 1981b, 1982). The word order of Korean is relatively free, resulting from free concatenation of a verb phrase with its argument term phrases. This concatenation is, however, strictly constrained in RPTQ by case indexing of verb phrases. RPTQ consists of three devices for case indexing, case marking, and case shifting. By the case indexing, both basic and derived verb phrases are subcategorized such that what cases are known to be admissible for concatenation. Case marking assigns cases to term phrases, and case shifting reassigns case indices to derived complex verb phrases.

By these devices, RPTQ provides a simple base for treating some complex constructions in Korean: (1) double case constructions, (2) derivation of adjectival phrases or relativization, and (3) passive and causative formations. We introduce basic concepts of RPTQ, which are handled in this paper.

The syntactic part of RPTQ contains (1) categories, (2) basic expressions, and (3) syntactic rules. The semantic part includes translation rules, which corresponds to syntactic rules in the one-to-one manner. RPTQ defines denumerably many basic categories and derives other categories recursively.

Definition 2.1 (Categories)

(i) 0 is a category.

(ii) Any natural number n is a category.

(iii) If A is a category, then A' is a category.

(iv) If A and B are categories, then (A,B) is a category.

The natural number n and 0 are basic categories. The category 0 corresponds to the category t (=truth value) of PTQ, and the natural number n corresponds to the category e (=entity). The derived category (A,B) corresponds to B/A in PTQ. Some of the typical categories of RPTQ are listed in Table 2.1 (Lee 1982).

RPTQ	Abbreviation	PTQ	Description
0		t	sentence
n		e	entity category
(n,0)	IV	t/e	intransitive verb
(c,0)	CN	t//e	common noun
(n',0)	DV		description verb
((n,0),0)	T or n*	t/(t/e)	term phrase
(m*,(n,0))	TV	IV/T	transitive verb

Table 2.1 Categories in RPTQ

Basic expressions of RPTQ may be lexical items in a lexicon:

(2-1) Basic Expressions of Korean

B_{n*}={John,Mary,Seoul,...}

B_{10}={čuk-die, nol-play,...}

$B_{1'0}$={hiri-be cloudy, yep'i-be pretty,...}

B_{c0}={namča-male, yəca-female, holangi-tiger,...}

B_{2*0}={mək-eat, čap-catch, salangha-love,...}

B_{3*2*10}={ču-give,...}

The case indexing of RPTQ uses categories of the natural number. Term phrases without the case marking belongs to the category n*. If they are assigned the specific case marking, n* takes a value of the natural number. For example, the term phrase 'Mary' of the category n* can be postpositioned by a nominative case particle 'ka'; then, n* becomes 1*, the value of the nominative case. The following is such a case marking rule:

(2-2) Case Marking

S1: $F_{1,m}([\alpha]_{n*})=[\alpha\text{-}K]_{m*}$

where K=ka (Nominative case) if m=1,

=lɨl (Accusative case) if m=2,

=eke (Dative case) if m=3.

This case marking rule simply adjoins a case particle to a term phrase of the category n*, replacing the parameter n with a case index. This process may be represented by the following tree:

(2-3) [Mary-ka]$_{1*}$
|
[Mary]$_{n*}$

Concatenated with an appropriate type of verb phrase, case-marked term phrases form more complex phrases. For example, a verb phrase of the category (1,0) may concatenate with a term phrase of the category 1* and form a sentence of the category 0. The following is an example representing such a derivation:

(2-4) [[Mary-ka]$_{1*}$ [nonta]$_{10}$]$_{0}$ 'Mary plays'
| play
[Mary]$_{n*}$

Concatenating a verb phrase with a term phrase is strictly constrained in RPTQ. It operates only when one of the case indices contained in the category of a verb phrase is erased by the same case index of a term phrase. Thus the following derivations are ill-formed:

(2-5) *[[Mary-lɨl]$_{2*}$ [nonta]$_{10}$]

However, the followng concatenations are both acceptable:

(2-6) [[John-ɨl]$_{2*}$ [salanghanta]$_{2*10}$]$_{10}$ 'love John'
[[Mary-ka]$_{1*}$ [salanghanta]$_{2*10}$]$_{20}$ 'Mary loves'

These concatenations are well-formed because the case index of each term phrase occurs in the category of the verb phrase and erases the corresponding case index in the verb category. These in turn derive the following well-formed sentences:

(2-7) [[Mary-ka]$_{1*}$ [John-ɨl salanghanta]$_{10}$]$_{0}$
'Mary loves John'.
[[John-ɨl]$_{2*}$ [Mary-ka salanghanta]$_{20}$]$_{0}$
'Mary loves John'.

RPTQ contains the following rules of concatenation:

(2-8) *Sentence Formation*

S4: $F_4([\alpha]_{n*},[\beta]_{n0})=[\alpha\beta'']_0$

where β'' is the present tense declarative form of β.

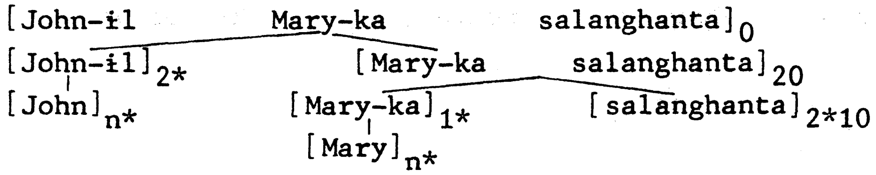
(2-9) *IV-Derivation*

S5: (i) $F_{5,1}([\alpha]_{1*},[\beta]_{k*10})=[\alpha\beta]_{k0}$

(ii) $F_{5,k}([\alpha]_{k*},[\beta]_{k*10})=[\alpha\beta]_{10}$

These rules along with the case marking rule derive the following analysis tree:

(2-10)



3. Structure Preserved Net

Structured Preserved Net (SPN) is an extended version of Petri net. SPN *structure* and SPN *graph* follow definitions and terminologies of Petri net structure and graph. The *Parsing* in SPN is an application of the execution rules for Petri net.

3.1 Introduction to Petri Net

In this section, we introduce the basic concept of Petri net which is used to define the SPN. Petri net is a four-tuple (P,T,I,O): a set of places P, a set of transitions T, an input function I, and an output function O. Input and output functions relate places with transitions. Input function I is a mapping from a transition t_j to a collection of places $I(t_j)$, and output function O is a mapping from a transition t_j to a collection of places $O(t_j)$. The definition of Petri net is as follows (Peterson 1981):

Definition 3.1 A Petri net structure, C, is a four-tuple C=(P,T,I,O). $P=\{p_1,p_2,\ldots,p_n\}$ is a finite set of *places*, $n\geq 0$.

$T=\{t_1,t_2,\ldots,t_m\}$ is a finite set of transitions, $m \geqslant 0$. The set of places and the set of transitions are disjoint, $P \cap T=\emptyset$. $I{:}T \to P^{\infty}$ is the input function, a mapping from transitions to bags of places. $O{:}T \to P^{\infty}$ is the output function, a mapping from transitions to bags of places.

While a set allows only one occurrence of each element in the set, a bag allows multiple occurrences. The use of bags allows one place to be a multiple input or a multiple output of a transition.

Example 3.1 Consider the next Petri net which has three places p_1, p_2 and p_3, and two transitions t_1 and t_2. p_2 is the multiple output of t_1.

$P=\{p_1,p_2,p_3\}$	$T=\{t_1,t_2\}$
$I(t_1)=\{p_1\}$	$O(t_1)=\{p_2,p_2\}$
$I(t_2)=\{p_2,p_3\}$	$O(t_2)=\{p_1,p_3\}$

The SPN parsing is based on the execution rules of Petri net. The execution is carried out on marked Petri nets. The following is quoted from Peterson (1981):

"A marking μ is an assignment of tokens to the places of a Petri net. A token is a primitive concept of Petri nets (like places and transitions). Tokens are assigned to, and can be thought to reside in the places of a Petri net. The number and position of tokens may change during the execution of a Petri net. The marking μ is defined as an n-vector, $\mu=(\mu_1,\mu_2,\ldots,\mu_n)$, where $n=|P|$ and each $\mu_i \in N$, $i=1,\ldots,n$. The vector μ gives for each place p_i in a Petri net the number of tokens in that place. A Petri net executes by firing transitions. A transition may fire if it is enabled. A transition is enabled if each of its input places has at least as many tokens in it as arcs from the place to the transition. The tokens in the input places which enable a transition are its enabling tokens. A transition fires by removing all of its enabling tokens from its input places and then depositing into each of its output places one token for each arc from the transition to the place."

A Petri net structure has the corresponding graphical version, Petri net graph. A Petri net graph has two types of nodes. A *circle* ○ represents a place, and a *bar* | represents a transition. *Directed arcs* → connect the places and the transitions. Input function for a transition is represented by the arrows directed from places to the transition. Similarly, output function is the directed arcs from a transition to places. A token is represented by a dot ● in a circle place.

Example 3.2 In the Petri net structure of example 3.1, if each of places p_1 and p_3 has one token, its marking $\mu=(1,0,1)$. In this case, t_1 is enabled and fires; then one token is removed from p_1, and since the output bag is $O(t_1)=\{p_2,p_2\}$, two tokens are deposited in p_2, that is, $\mu=(0,2,1)$. Since each of places p_2 and p_3 has at least one token, t_2 fires removing one token from each of p_2 and p_3 and then depositing one token in p_1 and one token again in p_3. In that case, the corresponding marking is $\mu=(1,1,1)$. The resulting marked Petri net structure is represented by the Petri net graph of Figure 3.1.

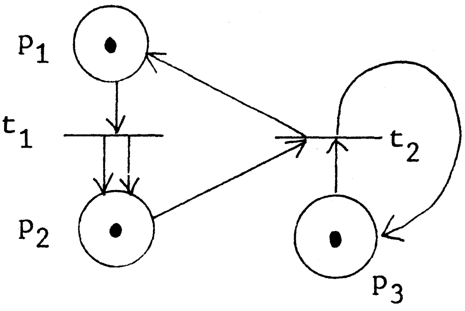


Figure 3.1 A marked Petri net graph as a result of example 3.2.

3.2 Motivation of SPN

In PTQ, a syntactic rule forming a complex expression contains three sorts of information (Dowty et al. 1979): (1) a bag of *input categories*, that is, the category or categories of expression(s) that serve as "input" to the rule, (2) the *output category*, that is, the category of the complex expression that is the "output" of the rule, and (3) the structural operation for the rule.

We consider an analogy between the syntactic rules of RPTQ and Petri net. Each category of RPTQ can be considered to be a place of Petri net, and each rule of RPTQ is a transition of Petri net. The number of categories and rules equals to that of places and transitions respectively. A place corresponds to a category in the one-to-one manner, and also a transition uniquely corresponds to a rule of RPTQ. Next, we can easily make an analogy between input categories of a rule and input places of the transition corresponding to the rule, and similarly between output categories and output places. Tokens of Petri net correspond to expressions of RPTQ. An expression is either a word (or lexical item) or a constituent (or phrase). While tokens of Petri net are simply one kind of marker (i.e., dots), each expression belongs to one of categories. SPN is constructed based on Petri net, with using these analogies between RPTQ and Petri net.

3.3 Definition of Structure Preserved Net

SPN is an extended version of Petri net. The basic definition of SPN follows that of Petri net introduced in section 3.1. Places, transitions, and tokens of SPN represent categories, rules, and expressions of RPTQ respectively. The nomenclature for transitions and places of SPN follows the names of rules and categories of RPTQ as it is.

Consider the rule S1: $F_{1,m}([\alpha]_{n*})=[\alpha\text{-}K]_{m*}$. This rule generates a case marked term phrase m* from a term phrase n*. Although functional words like K do not belong to any syntactic category (Table 2.1), we assign a place to each type of functional words. Such a place is called a functional place, and is represented by a square □. For parsing a case marked term phrase m*, input places of S1 should be a term phrase n* and a case particle K, and output place be m*. A bag with a single element may be represented without braces; for example, we write m* instead of $\{m*\}$. SPN represents S1 as follows:

(3-1) $I(S1)=\{n*,K\}$ $O(S1)=m*$

K represents a case particle. Since it is postpositioned after an input string of term phrase n*, there is a fixed word-order relation between n* and K. However, since elements in a bag do not have the fixed order relation between them, {n*,K} and {K,n*} make no difference. In order to describe the fixed word-orderness between places in a bag, we introduce the concept of ordered bag. The ordered bag (n*,K) is the representation for a term phrase n* postpositioned by a case particle K. In the SPN graph, the order of each two neighboring input places is represented by a dotted directed arc --> between them.

(3-1') $I(S1)=(n^*,K)$ $O(S1)=m^*$

Consider the rule S4: $F_4([\alpha]_{m*},[\beta]_{m0})=[\alpha\beta'']_0$. Since m is a natural number variable, and its value is restricted to 1, 2, or 3 due to the rule S1 (2-2), S4 can be described as follows:

(3-2) $I(S4,1)=\{1^*,10\}$ $O(S4,1)=0$
$I(S4,2)=\{2^*,20\}$ $O(S4,2)=0$
$I(S4,3)=\{3^*,20\}$ $O(S4,3)=0$

However if every variable must be realized to its possible values, it is inconvenient and reduces the expression power of SPN. We solve this problem to prepare a special place, called the testing place. Its notation is to be x=y, whose meaning is the question whether x=y or not. Its graphical representation is the triangle △. If the question of x=y is true, that testing place has a token (i.e., a dot). Hence we can simply describe the rule S4 in SPN as follows:

(3-2') $I(S4)=\{m^*,k0,m=k\}$ $O(S4)=0$

Consider the rule S5 (2-9) and its SPN representation (3-3):

(2-9)   S5: (i) $F_{5,1}([\alpha]_{1*},[\beta]_{k*10})=[\alpha\beta]_{k0}$
(ii) $F_{5,k}([\alpha]_{k*},[\beta]_{k*10})=[\alpha\beta]_{10}$

(3-3) $I(S5,1)=\{1^*,k^*10\}$ $O(S5,1)=k0$
$I(S5,k)=\{k^*,k^*10\}$ $O(S5,k)=10$

In this case, since k is a variable, 1* and k* places can be combined to one place m*, and m* is tested whether m=1 or m=k. Hence,

(3-3') $I(S5,1)=\{m^*,k^*10,m=1\}$ $O(S5,1)=k0$
$I(S5,k)=\{m^*,k^*10,m=k\}$ $O(S5,k)=10$

Consider the following rule S2 for the NP-derivation from a common noun:

(3-4) S2: $F_2([\alpha]_{(c,0)})=[\alpha]_{n*}$ or
$[\text{modun } \alpha]_{n*}$
where 'modun' is a universal determiner.

In Korean, a common noun without a determiner can become a noun. 'modun' is optional in S2. An optional place is represented by a bracket [] in the SPN structure and by dotted lines in the SPN graph. Since 'modun' is an optional functional word, then it is drawed by a dotted square ⬚. The following is the SPN representation of (3-4):

(3-5) I(S2)=(['modun'],c0)
O(S2)=n*

Consider the following rule S3 for the adjectival phrase (AP) modification or relativization:

(3-6) S3: $F_3([\delta]_{k0},[\alpha]_{c0})=[\delta'\alpha]_{c0}$
where δ' is the present tense adjectival form of δ.

This rule says that a common noun is composed of a verb phrase k0, its present tense adjectival ending form 'nun', and another common noun c0. Its SPN representation is as follows:

(3-7) I(S3)=(k0,'nun',c0)
O(S3)=c0

In this stage, we can make the full SPN structure (3-8) for the rules S1, S2, S3, S4 and S5 of RPTQ to combine (3-1'), (3-2'), (3-3'), (3-5) and (3-7). Figure 3.2 shows its corresponding SPN graph.

(3-8)	I(S1,m)=(n*,K)	O(S1,m)=m
	I(S2)=(['modun'],c0)	O(S2)=n*
	I(S3)=(k0,'nun',c0)	O(S3)=c0
	I(S4)= m*,k0,m=k	O(S4)=0
	I(S5,1)= m*,k*10,m=1	O(S5,1)=k0
	I(S5,k)= m*,k*10,m=k	O(S5,k)=10

3.4 Parsing in SPN

The parsing method in SPN resembles the execution rules for Petri net. A token in a place corresponds to an expression which is either a lexical item or a constituent of input data. First,

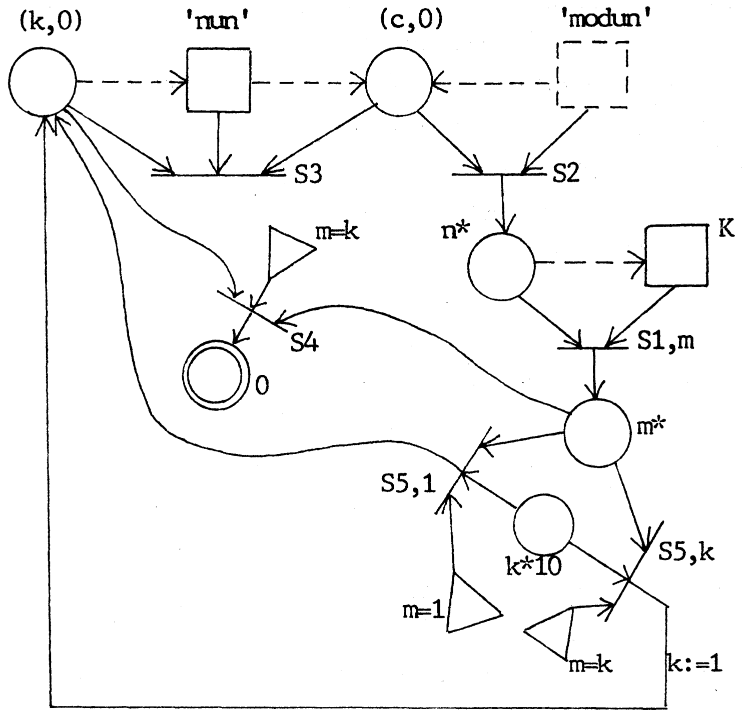


Figure 3.2 A SPN graph for the syntactic rules S1,S2,S3,S4, and S5 of RPTQ. ":=" is the assignment operator. Hence "k:=1" means that O(S5,k) is the place (1,0).

the parser recognizes a lexical item from input string with reference to a lexicon. This means that the lexical item is assigned a syntactic category. That recognized word resides in the corresponding category place in SPN. Hence, there are two types of places. One is the places which accept lexical items. Those places are called lexical places. The other is non-lexical places where complex expressions reside. Those non-lexical places are just output places of transitions. For example, a common noun (c,0) and an intransitive verb (n,0) are categories which are lexical places in SPN. However, because (n,0) is the output place of the rule S5, (n,0) may be a non-lexical place; in that case, (n,0) represents a verb phrase which is the concatenation of a term phrase n* and a transitive verb n*m0 (or m*n0).

More than one token may reside in one place. In this case, that place acts like a stack. For example, if m* place accepts two tokens, say, first 'Mary-ka' of 1* and the next, 'John-ɨl' of 2*, then the notation is [John-ɨl(2*),Mary-ka(1*)] . The last accepted token 'John-ɨl' resides at the top of the stack place m*. If the above place m* belongs to I(S) given a rule S and the rule S fires, the token 'John-ɨl' is removed from m*.

For instance, we show steps for parsing a simple sentence. Consider the SPN structure and graph in (3-8) and Figure 3.2. We can draw the parsing tree (Figure 3.3) and make a table of parsing steps (Table 3.1) for the below sentence:

(3-9) John-ɨl Mary-ka salanghanta.

However, since the input places of transitions S4 and S5 are not in ordered bags, the SPN structure (3-8) can parse the following sentences besides (3-9):

(3-10) Mary-ka John-ɨl salanghanta.
(3-11) Mary-ka salanghanta John-ɨl.
(3-12) John-ɨl salanghanta Mary-ka.
(3-13) salanghanta Mary-ka John-ɨl.
(3-14) salanghanta John-ɨl Mary-ka.

This shows that SPN can handle the complete free word-order languages. The SPN graph of Figure 3.2 can parse the following sentence (3-15), and Figure 3.4 is its parsing tree.

(3-15) John-ɨl salangha-nun yəča-ka yep'ɨta.
John love who woman is pretty
'A woman who loves John is pretty.'

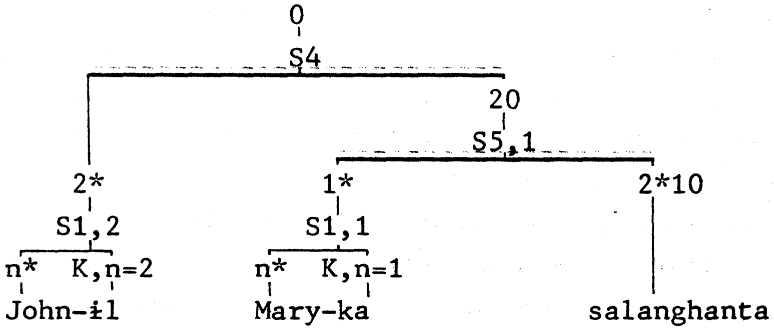


Figure 3.3 Parsing tree for (3-9).

Cat								fired
Step	n*	K	m*	k*10	k0	m=k	0	rule
1	J							
2	J	ɨl						S1,2
3			J-ɨl(2*)			m=2		
4	M		J-ɨl			m=2		
5	M	ka	J-ɨl			m=2		S1,1

6			M-ka(1*)
			J-ɨl(2*)
						m=1		

7			M-ka(1*)
			J-ɨl(2*)
				s
				(2*10)
						m=1		S5,1

8			J-ɨl(2*)		M-ka s (20)	m=2		S4
9							J-ɨl M-ka s.	

Table 3.1 Parsing steps for (3-9). In steps 6 and 7, 'M-ka' is at the top of the stack place m*. J=John, M=Mary, s=salanghanta.

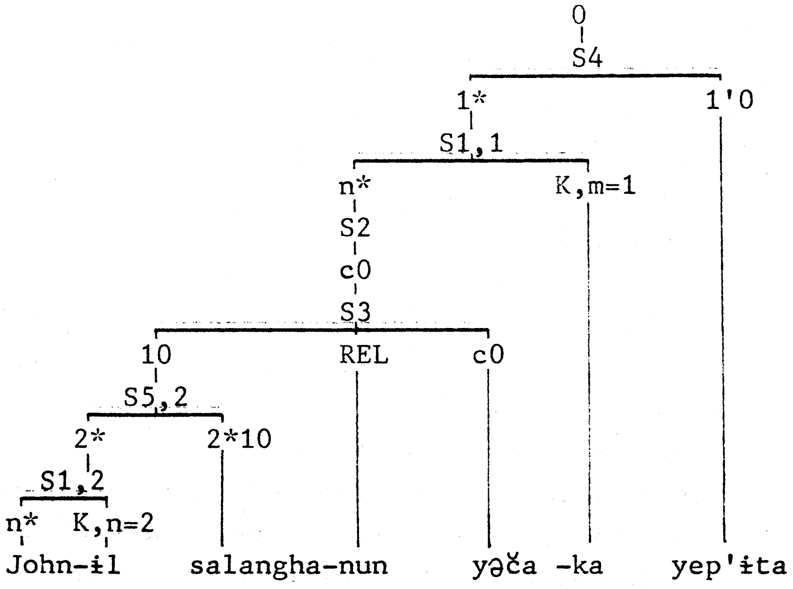


Figure 3.4 Parsing tree for (3-15).

4. Determinism Hypothesis and SPN

4.1 Introduction to Determinism Hypothesis

The **Determinism Hypothesis** is defined as follows (Marcus 1980): "the syntax of any natural language can be parsed by a mechanism which operates strictly deterministically in that it does not simulate a nondeterministic machine."

Taking the Determinism Hypothesis as a given, Marcus proposes three properties of any deterministic parser: (1) it must be partially data driven; (2) it must be able to reflect expectations that follow from general grammatical properties of the partial structures built up during the parsing process; and (3) it must have some sort of look-ahead facility, even if it is basically left-to-right.

4.2 Korean and Determinism Hypothesis

Figure 4.1 shows examples for explaining three properties of the deterministic parser in Korean.

The parser must:

Be partially data driven

(1a) [[Mary]$_{n*}$[ka]$_{n=1}$]$_{1*}$ (nominative case)

(1b) [[Mary]$_{n*}$[lɨl]$_{n=2}$]$_{2*}$ (accusative case)

Reflect expectations

(2a) Mary-ka [nonta]$_{10}$ 'Mary plays'

(2b) Mary-ka note-lɨl [John-eke]$_{3*}$ [čunta]$_{3*2*10}$
'Mary gives John a note'

Have some sort of look-ahead

(3a) [Mary-ka John-ɨl salanghanun sasil-ɨl]$_{2*}$
nae-ka anta. 'I know the fact that Mary loves John'

(3b) Mary-ka [John-ɨl salanghanun namča-lɨl]$_{2*}$
salanghanta. 'Mary loves a man who loves John'

Figure 4.1 Some examples which motivate the structure of a parser in Korean.

First, before the parser recognizes the case particle, it cannot determine the case for 'Mary' in (1a) and (1b) of Figure 4.1. The deterministic parser must determine the case when it recognizes the postpositioned case particle after 'Mary'. In a hypothesis-driven parser, it may first assume that the first term phrase has a nominative case particle among other alternatives. If that rule is applied to (1b), then the parser becomes to find that it is the wrong hypothesis, and must make a backtracking in order to apply another alternative rule. Hence, any determinisc parser must be partially data driven.

Second, in (2a) and (2b) of Figure 4.1, 'Mary-ka' belongs to the 1* category. In the SPN graph of Figure 4.1, an input place m* of the transitions S4 and S5 becomes 1*. 1* place is one of input places of transitions S4 or S5. Having that information, we can expect that the next input word belongs to (1,0) or k*10 category, and S4 or S5 may be the next firing rule. On the other hand, in (2b), after recognizing 'John-eke', we can expect that its main verb belongs to one of categories (3,0), 3*10, 3*m*10, or m*3*10.

But because only 3*2*10 category exists in Korean, a deterministic parser must be able to apply the rules which have 3*2*10 place as one of its input places. This property says that a deterministic parser can not be entirely bottom-up. It must reflect expectations based on the information which follows from the partial structures built during the parsing process.

Finally, if a deterministic parser is to correctly analyze such a pair of sentences as (3a) and (3b), it must have a sufficient look-ahead facility. After we see only 'Mary-ka', we cannot determine whether it is the subject of the verb 'salangha' or not. The syntactic structures can not be determined until we see a word after 'salanghanun'. Thus a deterministic parser must have a large enough window on the clause to see sufficient input data.

4.3 SPN and Determinism Hypothesis

First, the parser based on SPN analyzes a word and the corresponding lexical place gets the token. Hence the SPN parser is partially data-driven.

Second, if a place p is filled with a token, every transition t_i of $p \in I(t_i)$ has the possibility that they are enabled and fire. Those transitions are just expected rules which follow from the partial structures built during the parsing process. Thus the SPN parser can reflect expectations.

Finally, in order to apply a transition, all of its input places must have tokens. Such input places of SPN play the role of the look-ahead facility.

5. Conclusion

The rules and categories of RPTQ have the one-to-one correspondence to the transitions and places of SPN; hence SPN naturally represents the structure of syntactic rules of RPTQ. Next, since SPN satisfies three parsing principles for the Determinism Hypothesis, SPN also represents the structure of a deterministic parser. In these sense, SPN bridges gaps between the structure of a linguistic framework RPTQ and the structure of a deterministic parser. From the above mentioned facts, we can say that these two structures are preserved by SPN, and it connects the linguistic aspect with the computational aspect.

The semantic part should be considered in order to complete the parsing. Since each syntactic rule of RPTQ has its corresponding translation rule, the semantics may be easily included in SPN.

References

Bień, J.S. & S. Szpakowicz (1982): "Toward a Parsing Method for Free Word Order Languages", COLING82 Abstracts, Charles University, Prague, Czechoslovakia.

Dowty, D.R., R.E. Wall & S. Peters (1981): Introduction to Montague Semantics, D.Reidel Publishing Company, Holland.

Lee, K. (1981a): "Case Assignment in Korean", Language Education, Language Institute, Jeonnam University, Kwangjoo, Jeonnam, Korea, Vol.12, 269-285.

Lee, K. (1981b): "On Free Word Order and Case in Korean: A Montague Grammar Approach", MAL: The Journal of Korean Language Institute, Yonsei University, Seoul, Korea, Vol.6, 51-87 /in Korean/.

Lee, K. (1982): "On Case Shifting", Linguistic Journal of Korea, The Linguistic Society of Korea, Vol.7, No.2, 497-517.

Marcus, M.P. (1980): A Theory of Syntactic Recognition for Natural Language, MIT Press, Massachusetts.

Montague, R. (1974): "The Proper Treatment of Quantification in Ordinary English", In R.Thomason (ed.), Formal Philosophy: Selected Papers of Richard Montague, 247-270, New Haven: Yale University Press.

Peterson, J.L. (1981): Petri Net Theory and the Modeling of Systems, Prentice-Hall Inc., Englewood Cliffs.

Robinson, J.J. (1981): "Perspectives on Parsing Issues", Proceedings: 19th Annual Meeting of the Association for Computational Linguistics, 95, Stanford University.