PETRI NET GRAMMARS FOR NATURAL LANGUAGE ANALYSIS

Key-Sun Choi

An approach is described to parsing and logical translation that was inspired by Lee, K.'s (1983) work on AMG (Augmented Montague Grammar) for case languages, and execution rules of Petri nets. Each grammar rule consists of a syntactic part that specifies an acceptable fragment of a parse tree, and a semantic part that specifies how the logical forms correspoding to the constituents of the fragment are to be combined to yield the logical form for the fragment in the intensional logic.

The proposed model, SPNG (Semantic Petri Net Grammar) is built based on Petri nets and its parsing mechanism resembles execution rules of its underlying Petri nets. But SPNG parsing mechanism includes several constraints which block out wrong applications of rules. Also, an isomorphism exists between SPNG and AMG.

SPNG has the same analytic power with Turing machine, since it is an extended Petri net with inhibitor arcs (Peterson 1981). Hence, natural languages can be parsed by SPNG sufficiently.

1. Introduction

This paper represents a formalism of natural language grammars with accompanying parser. The grammars are called Semantic Petri Net Grammars (SPNG) because their parsing algorithms resemble the *execution* of Petri nets (Peterson 1981), and an application of each rewriting rule corresponds to a firing of transition in Petri nets. The parser works partially bottom-up and partially top-down as parsing states change.

The underlying linguistic formalism of SPNG is Augmented Montague Grammar (AMG) of Lee (1981a,b, 1982a,b, 1983a,b,c) which is a revised version of Montague grammar (Montague 1974). Since AMG has principally investigated phenomena of partially free word-order languages such as Korean and Japanese, examples for SPNG are restricted to such languages. AMG, however, can also handle other languages (Lee 1983c), and SPNG can too.

Rewriting rules are represented by *rule schema,* and each of them includes a syntactic rule and its translation rule of AMG. Comparing rule schema with Gazdar's (1981) rule, consider his top-level rule of declarative sentence structure and meaning:

<4,[(S) (NP) (VP)], (VP' ^NP')>.

The first element of this triple supplies the rule number (which we have set to 4 for consistency with the sample grammar of section 4), the second the syntactic rule and the third the semantic rule. The semantic rule states that the intensional logic translation of the S-constituent is compounded of the VP-translation (as functor) and the NP-translation (as operand). On the other hand, the corresponding rule

schemata has the following form:

R4: (<NP,NP'>, <VP,VP'>)<S, VP' ^NP'>.

The left-hand side represents *input,* and the right-hand side is *output* of a transition R4 in SPNG.

2. Introduction to Petri Nets

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 transition t_j to a collection of places $O(t_j)$.The definition of Petri net is as follows (Peterson 1981):

Definition 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,* $m\geq 0$. $T=\{t_1, t_2,\ldots,t_m\}$ is a finite set of *transitions,* $m\geq 0$, The set of places and the set of transitions are disjoint, $P\cap T=\phi$. I:T->P∞ is the *input* function, a mapping from transitions to bags of places. O:T→P∞ 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 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.

$P=\{p_1, p_2,p_3\}$ $\quad$ $T=\{t_1,t_2\}$

$I(t_1)=\{p_1\}$ $\quad$ $O(t_1)=\{p_2,p_2\}$

$I(t_2)=\{p_2,p_3\}$ $\quad$ $O(t_2)=\{p_1,p_3\}$.

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 $\mu_i\in N$ where N is the set of natural numbers and $i=1,2,\ldots,n$. 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 are 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* O 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 2. In the Petri net structure of example 1, if each of place p_1 and p_3 has one taken, its marking is $\mu=(1,0,1)$. In this case, t_1 is enabled and fires; then one token is removed from p_1. Since the output bag is $O(t_1)=\{p_2,p_2\}$, two tokens are deposited in p_2; that is, $\mu=(0,2,1)$, whose marked Petri net structure is represented by the Petri net graph of Figure 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 $\mu=(1,1,1)$.

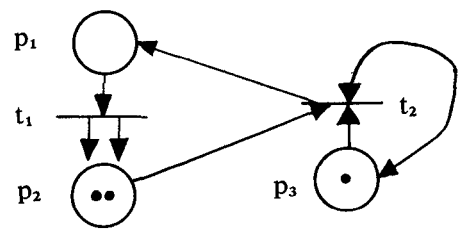


Figure 1. A marked Petri net graph for $\mu=(0,2,1)$.

3. Adaptation of Montague Grammar to Petri Nets

In Montague grammars, a syntactic rule forming a complex expression contains three sorts of information (Dowty, et al. 1981): (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.

Now, consider an analogy between the rules of AMG and transitions of Petri net. Each *category* of AMG can be considered to be a *place* of Petri net, and each *rule* of AMG 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 AMG. 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 the *output category* and the *output place. Tokens* of Petri net correspond to *expressions* of AMG. An expression is either a word (or lexical item) or a constituent (or phrase). While tokens in ordinary Petri nets are simply one kind of marker (i.e., dots), each expression belongs to one of the categories in AMG or SPNG.

4. Semantic Petri Net Grammars

Semantic Petri Net Grammars (SPNG) are an extended version of Petri nets. The graphical representaton of SPNG will be called the *SPNG Graph.* The basic notations of SPNG graphs follow those of Petri net graphs as they are.

Definition 2. A Semantic Petri Net Grammar is a 8-tuple $G=((P,R,I,O,B,Cv, Cn,\sigma)$ where

1) $P=(p_1,p_2,\ldots,p_n)$ is a finite set of *places,* $n\geqslant 0$.
2) $R=(R_1,R_2,\ldots,R_m)$ is a finite set of transition *rules,* $m\geqslant 0$.
3) $I:R\rightarrow P\infty$ is the *input* function, mapping from rules to bags of places.
4) $O:R\rightarrow P$ is the *output* function, mapping from rules to places.
5) B is a set of *basic expressions.*
6) Cv is the set of variable categories.
7) Cn is the set of numbered categories, and $Cv\cap Cn=\{0\}$.
8) $\sigma:P\rightarrow Cv$ is the *place-labelling* function, mapping from places to variable categories in the one-to-one manner.

Variable categories Cv mean (1) variablized categories of AMG rules which are not assigned natural numbers, and (2) names of morphemes which are not given categories in AMG. *Numbered categories* (1,0), (2,0) and (3,0) belong to the same variable category (m,0). REL belongs to Cv, where REL stands for relativizers such as 'nin'. But, since the category 0 is not variablized in AMG, it belongs to sets of both variable categories and numbered categories. A token with a numbered category resides in a place with a variable category which is matched with the numbered category.

The codomain of the function O is one place, since the value of each AMG rule is one category. P and I have the same definitions as the ordinary Petri nets. But, P has many other meanings. Each place p_i has its unique label $\sigma(p_i)$ which is a variable category of Cv. For the convenience of notation, the name of a place p_i will use its labelled category name $\sigma(p_i)$ of Cv. There are two kinds of places, the *categorial place* and the *functional place.* Labels of categorial places are variablized categories of AMG. Functional places represent morphemes or terminal strings to which categories in AMG are not assigned. The graphical notation of categorial places is the circle O, and that of functional places is the squre □. In another view, there are two kinds of place: the *basic places* and the *derived places.* Basic places include functional places and represent basic expressions. Derived places are labelled with derived categories of complex expressions. But because some categories stand for basic expressions and complex expressions according to circumstances, the sets of basic places and derived places are not mutually disjoint. The place of (m,0) is such a place. Figure 2 shows the above classification.

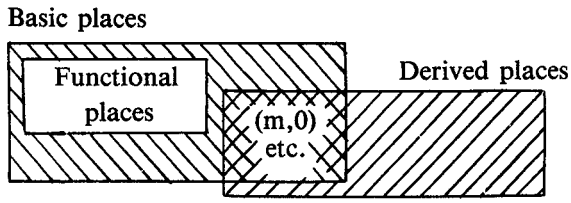


Figure 2. All the places except functional places are categorial places.

The notation for the rules will use R instead of T which represents a finite set of transitions in the Petri nets. But the notion of rules is just that of transitions. From now on, T will be used for indicating *translation rules* of Montague grammar. Each rule Ri of SPNG corresponds to the syntactic rule Si explicitly and to the translation rule Ti implicitly in AMG. Almost all of syntactic rules are realized sufficiently by their corresponding transitions, since the result of a syntactic rule is the concatenation of input strings. But, some syntactic rules of AMG reguire special conditions and special actions. Hence, each rule Ri is defined as follows:

Definition 3. Each rule transition Ri has attributes of a 3-tuple (T_i, $[C_i]$, $[A_i]$) where

1) Ti is the *translation rule,* mapping from syntactic expressions to logical forms in the intensional logic,
2) Ci is the optional *special conditions,*
3) Ai is the optional *special actions.*

A set of basic expressions B is a kind of lexicon. B may be called a *lexicon.* Lexical items in B include basic expressions and functional words to which no category is assigned such as case markers, complementizer, and so on. In parsing, each terminal string is searched through entries of B, and if found, a token is made for each string by copying the information of the corresponding entry. That token consists of the terminal string, its categories, logical forms and features. Details for tokens will be explained in the next section. The definition of B is as follows:

Definition 4. Each *entry* of a set of basic expressions B is an ordered 4-tuple (<entry>, <categories>, <logical forms>, [<features>]) where

1) <entry> is a terminal string.
2) <categories> is the list of <entry>'s categories.
3) <logical forms> is the list of <entry>'s logical forms in the intensional logic.
4) <features> is the optional list of features attributed to the <entry>.

For example, the entry of a proper name 'Mary' is (Mary,n*,m). For a determiner 'modin', its entry is (modin,(c0,n*), $\lambda P[\lambda Q \forall x[P\{x\} \rightarrow Q\{x\}]]$).

The case markers such as 'ka', 'lil', and 'eke' do not belong to any categories in AMG. But, these functional words play an important role in assigning cases to

noun phrases. In the rule S1 of AMG, if a string of noun phrase n*∈Cv and a case marker are concatenated, n* has a value according to its case marker. Although they are not assigned to any category, the case information is positioned in <categories> in B, in order to give a number to the variable category n*. Its format is 'case = m' where *m* is a natural number. For example, the nominative case marker 'ka' has its entry such as '(ka,case = 1).' If 'ka' is postpositioned after a noun phrase n*, then n* becomes 1*.

4.1 Fixed Order Between Places

Since Petri Nets are used for synchronizing asynchronous inputs, the ordering between input places is unnecessary. But, in natural languages, there are many cases such that the order between constituents in a sentence must be fixed.

As an illustration, consider the sentence-formation rule S4: $F_4([\alpha]_{n*}, [\beta]_{n0}) = [\alpha\beta'']_0$ of AMG. The rule R4 in SPNG corresponds to this rule S4. The input arguments of a rule Ri are represented by I(Ri), and the output is O(Ri). Hence, S4 becomes.

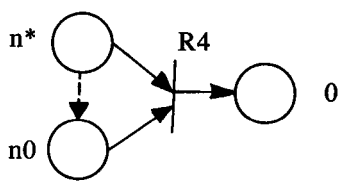
(1) $I(R4) = \{n*, n0\}$ $O(R4) = \{0\}$

Since Korean is a verb-final language, the order of input strings should be a term phrase n* and then its corresponding verb phrase n0. However, this fixed-order relation cannot be described by a bag; {n*,n0} and {n0,n*} make no differences. In order to describe the fixed word-orderness between places in a bag, the concept of the *ordered bag* is used. The ordered bag (n*,n0) in parentheses means that a string of category n* precedes a string of n0. Hence, (1) is changed to (1)'

(1)' $I(R4) = (n*, n0)$ $O(R4) = \{0\}$

In SPNG graph, the order of two input places is described by a *dotted directed arc* (— →) between them. The next figure shows the SPN graph of (1')

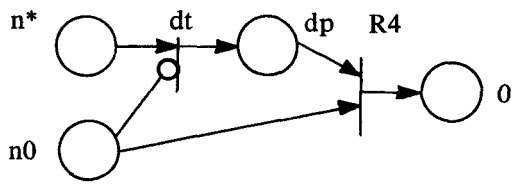
(2)



The ordered bag can be described in extended Petri nets with inhibitor arcs (Peterson 1981). An inhibitor arc has a small circle O rather than an arrowhead at the transition. The firing rule is changed as follows: A transition is enabled when tokens are in all of its (normal) inputs and all places of inhibitor arcs have no token. (2')

represents (2) in Petri nets with inhibitor arcs. A dummy transition dt and a dummy place dp are used.

(2′)

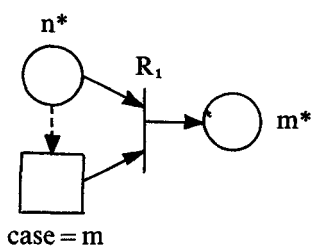


4.2 Rules for Parsing

Although R4 in SPNG uses S4 in AMG as it is, many other rules of AMG should be changed for parsing. While rules of AMG are made in order to *generate* surface strings, rules of SPNG *parse* surface strings. For example, consider the case marking rule $S1{:}F1_{,m}$ $([\alpha]_{n*}) = [\alpha\text{-}K]_{m*}$, where n is a variable, and m is a natural number. This rule generates a case marked term phrase m* from a term phrase n*. For parsing, input arguments of R1 should be two places of a term phrase n* and a case marker′ case = m′. Its output argument is the place m*. Hence, SPNG and its graph for R1 are (3) and (3′) respectively.

(3) $I(R1) = (n*, case = m)$ $O(R1) = \{m*\}$

(3′)



4.3 Optional Places

Consider the following rule S2 for the NP-derivation from a common noun c0:

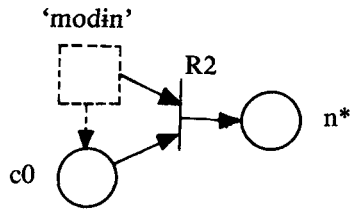
(4) S2: $F_2([\alpha]_{C0}) = [\alpha]_{n*}$ or
$[\text{modin } \alpha]_{n*}$
where 'modin' is a universal determiner.

In Korean, a common noun without a determiner can become a noun. 'modin' is optional in S2. An optional place is represented by a bracket [ ] in SPNG and

by dotted lines in SPNG graph. Since 'modin' is an *optional functional word,* then it is drawn by a *dotted square.* (5) is the SPNG representation of (4) and its graph (6):

(5) I(R2) = (['modin'], c0)
O(R2) = {n*}

(6)



5. Parsing with Semantic Petri Net Grammars

The parsing with SPNG will be simply called the *SPNG Parsing.* The SPNG parsing is based on the execution rules of Petri nets. The execution is carried out on marked Petri nets as shown in section 2. In marked Petri nets, some places have tokens at any state. If every place of a transition contains sufficient tokens, then that transition is enabled. These tokens are described simply as dots in the Petri net graph; that is, there are no differences between tokens although they are located at different places. However, tokens in the SPNG parsing represent some expressions. Since places are used to store tokens of expressions, each token has information for handling the expression. This information is used for the condition of rule firing, and its logical form is used for translation. When a place has more than one token in marked Petri nets, any token in the place is removed at the time of rule firing. But, SPNG parser must determine which token is selected, because different tokens represent different expressions. This problem is solved by giving a proper data structure to places. Since the parsing is processed according to the operation of rules, its formal definition will be introduced.

5.1 Tokens

After a token is generated by copying its information in B, it resides at a basic place whose variable category is matched with the numbered category of the terminal string in that token. Tokens for complex expressions are the results of firing some sequence of rules. The former type of tokens are called *basic tokens,* and the latter types are *complex tokens.* For a detailed illustration, tokens are formally defined:

Definition 5′ A token is an ordered 4-tuple
(<expression>, <category>, <logical form>, [<features>]) where

1) <expression> is either a terminal or a concatenated string of terminal,
2) <category> is the <expression>'s categories,
3) <logical form> is the <expression>'s logical forms in the intensional logic,
4) <features> is the optional list of features.

The configuration of a token is similar to that of an entry of lexicon B. When <expression> is a terminal, all the components are just the copy of its entry of B. This case occurs when the SPNG parser reads an input string and that string of <expression> is found in an <entry> of B. For example, a token for the determiner 'modin-*every*' is just the copy of the entry 'modin' in B. In the conventional notion of *derivation tree,* a *terminal node* corresponds to a basic place which has a basic token, and a *nonterminal node* is a derived place which has a complex token. If a token reaches the place of 0 (sentence), then the parsing is terminated. The place 0 is just the *root node* of derivation trees. The conventional concept of *derivation* corresponds to a firing sequence of rules.

5.2 Places

A place may have more than one token. For example, consider a small SPNG parser using the rules S1, S4, and S5 of AMG (Lee 1982b). For an input sentence "Mary-ka John-il salaŋhanta", the next label-bracketing form is derived:

(7) $[[\text{Mary-ka}]_1^*\ [[\text{John-il}]_2^*\ [\text{salaŋhanta}]_{2*10}]_0$

In (7), there are two term phrase "Mary-ka" (1*) and "John-il" (2*). The verb "salaŋhanta" (2*10) is combined with its neighboring term phrase "John-il" before combining with "Mary-ka". The SPNG parser recognizes two case-marked term phrases "Mary-ka" and "John-il" before seeing the verb "salaŋhanta". But because labels of categories are variables, there is no place whose category label is 1* or 2*. There exists only the place m*. Hence both tokens must be located in the place m*. After the parser recognizes the verb "salaŋhanta", one of two tokens in m* must be selected. In AMG, locally neighboring strings are first combined, that is, the most recently recognized string is first selected. By investigating this *last-in last-out* mechanism, *stacks* are necessarily used for the data structure of places. If a place p_i receives first a token t_1 and next t_2, then its notation will be $<t_2,t_1>$. The first element in the angle brackets is the top component of the stack, and the last element is the bottom component.

In order to represent a place, the following notation will be used.

Notation 1° A place p_i is represented in the form of
$\sigma p_i<t_{i_1},t_{i_2},\ldots,t_{i_m}>$ where
1) σp_i is the category labelled to p_i,
2) t_{ij}'s are tokens in p_i, for $j=1,2,\ldots,m$,
3) angle bracket means a stack.

If the place m* receives two tokens $[\text{Mary-ka}]_{1*}$ and then $[\text{John-il}]_{2*}$, then its notation is

m*<(John-il,2*, John′), (Mary-ka,1*,Mary′)>

where **Mary′** and **John′** are the logical forms of **Mary** and **John** respectively.

5.3 Operation of Rules

Every rule has its *condition* part and *action* part. There are two kinds of conditions: common conditions and special conditions. *Common conditions* are checked before any rule is fired. Hence, SPNG parser always checks common conditions before firing any rule. But, *special conditions* depend on rules. While some rules have no special conditions, other rules have particular and specific conditions.

Similarly, there are two kinds of actions: common actions and special actions. *Common actions* consist of the next three parts: (1) concatenation of <expression>'s of tokens in input places, (2) derivation of <category> of the output token from those of input tokens, and (3) translation from <logical form>'s of input tokens to that of the output token and the λ-reduction. *Special actions* depend on rules in the same way as special conditions.

From now on, several conditions will be investigated, and then the formal definition of rule schema will be introduced in order to reflect the above roles of rules.

Case-Matchability Condition

Case-matchability conditions are a kind of common condition for firing rules. It is to check whether case indices between input tokens are matched or not. In ordinary Petri nets, more than one transitions are enabled simultaneously, and one of them is chosen randomly and fires. The sufficient condition for a transition Rj to be enabled is that every input place of Rj has tokens. However, in SPNG, besides the above condition for enabling a rule Rj, another condition is needed to check the matchability between categories of tokens in input places I(Rj).

For example, consider the rule R4. I(R4) = (n*,n0). Assume that $n^*\langle t_{2*}\rangle$ and $n0\langle t_{20}\rangle$, where <category>'s of tokens t_{2*} and t_{20} are 2* and 20 respectively. Since the case index 2 is matched, and then the rule R4 is enabled. If $n^*\langle t_{3*}\rangle$, the matchability condition fails.

Priority Condition

There are two types of priority. One is the functional-place priority, and the other is the special-condition priority.

(i) Functional-Place Priority

There are two kinds of rules: functional rules and non-functional rules. *Functional rules* concern functional place. In a functional rule, at least one of input places is a functional place. Each functional place receives one of the basic tokens which represent case markers, complementizers, relativizers and so on. Case markers give

a syntactic role to every term phrase. Complementizers play the role of catalyzers which transform original verbs to complex verbs by the case shifting rules. Relativizers changes a verb phrase to a adjectival phrase. In these ways, a functional place of a basic rule makes or changes the syntactic role of categorial places which are concatenated with the functional place. Input places of *non-functional rules* contain no functional place. Hence, the functional-place priority is built as follows:

"When bags of input places of a functional rule and a non-functional rule have common places and at least one of them has a token, the functional rule has the higher priority over the non-functional rule."

For the illustration, assume that $I(R_n) = (p_1, p_2)$ and $I(R_f) = (p_2, p_f)$, where R_n is a non-functional rule, R_f is a functional rule, and p_f is a functional place. p_2 is a common place of R_n and R_f. In this Petri net, although R_n is enabled, it does not fire and wait-and-sees the next input (Figure 3). If the next input is on p_f, then R_f is also enabled and fires; R_n is automatically disabled.

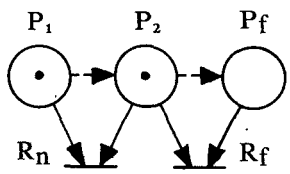


Figure 3. The enabled rule R_n must wait-and-see whether p_f has a token or not.

(ii) Special-Condition Priority

Consider a case in which two rules are enabled simultaneously. If one of those rules has special conditions, then that rule has higher priority over the other rule. It is one kind of *conflict-resolution* (Peterson 1981). If both of the two rules either have no special conditions or have special conditions, one of them is chosen randomly. However, in AMG, there is no such case. An example can be seen in the next section, which is a conflict between the sentence formation rule R4 and the reflexivization rule R20.

For example, consider the next sentence (Lee 1982a):

(8) current position (cp)

[Mary-ka]$_1$* [horaŋi-eke]$_3$* [[tol-il]$_2$* [mək]$_{2*10}$]$_{10}$-ke hayəssta.

(1) by S5

(2) violated to the stack operation

The *current position pointer* (cp) of SPNG parser indicates the end of "mək". In this current state, the m* place is a stack with tokens < t_2*,t_3*,t_1* > and the place m*n0 has a token <t_{2*10}> where

t_1* = (Mary-ka,1*,..),
t_2* = (tol-il,2*,..),
t_3* = (horaŋi-eke,3*,..), and
t_{2*10} = (mək,2*10,..).

By the rule R5, t_2* and t_{2*10} are combined. Then, t_2* is removed from the palce m*. The stack place m* becomes <t_3*,t_1*>, and m0 has a token <t_{10}>. But in this state, the matching fails, despite the fact that the case index 1 is matched between 1* and 10. One reason is that they violate the stack operation, since t_1* is at the bottom of the stack place m*. The other reason is that t_3* has no matching case with the token t_{10} or tokens in other places. "hayəssta" doesn't have a case index of 3*. But, by the *complementation* rule S50 and the *causative* or *factitive construction* rule S60 in AMG (Lee 1982a), we can derive the following well-formed structure of (9).

(9) [[Mary-ka]$_1$* [[horaŋi-eke]$_3$* [[tol-il]$_2$*
[[mək-ke]$_{\langle 2*10\rangle}$' [hayəssta]$_{3*2*10}$]$_{3*10}$]$_{10}$].

Consider the state in which the current position pointer indicates "mək" such as Figure 4.

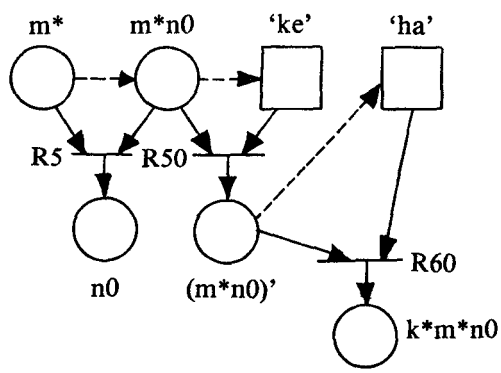


Figure 4. R5 does not fire, and wait-and-sees 'ke.'

In order to give the parsing result of (9), SPNG parser tests whether the case 2*10 of "mək" is matched with the case 2* "tol-il" of the place m* by the case-matchability condition; but, although that condition is satisfied, it does not fire.

Because m*n0 place is also an input place of a functional rule R50, SPNG parser must wait-and-see the next input.

5.4 Rule Schema

A rule schema is a kind of *pattern-action* rule. Generally, the left-hand side of a pattern-action rule represents conditions, and its right-hand side represents actions. The left-hand side of a rule schema consists of input places and special conditions for firing its corresponding transition rule. Its right-hand side contains its output place and special actions which are applied to the generating output token. Each place is represented by *Notation 1,* and each token of a place is also represented by *Definition 6.* In a rule schema, since the representation of a token is a prototype pattern of strings, <logical form> of a token in an output place is just the same as its translation rule. In other words, a rule schema Ri′ describes the information for conditions and actions for firing the transition rule Ri.

Definition 6. A *Rule Schema* Ri′ corresponding to a rule Ri has the following form:

Ri′: (I(Ri), [Ci] → (O(Ri), [Ai] where

1) I(Ri) is the (ordered) bag of input places,
2) Ci is the optional special conditions for rule firing,
3) O(Ri) is the output place,
4) Ai is the optional special actions.

For example, consider the syntactic rule S4 and its translation rule T4 in (10). Its rule schema R4′ is shown in (11).

(10) S4: $F_4([\alpha]_{m^*}, [\delta]m0) = [\alpha\delta]0$
T4: $\alpha'(\hat{}\,\delta')$

(11) R4′: $(p_1, p_2) \rightarrow p_3$ where
$p_1 = m^*(\alpha, m^*, \alpha')$
$p_2 = m^*(\delta, m^\circ, \delta')$
$p_3 = 0(\alpha\delta, 0, \alpha'(\hat{}\,\delta'))$.

Parsing mechanisms are applied based on the rule schema. Rule schemata are grouped dynamically whenever a new token is generated. At every state of parsing, if the first component pi_1 is a basic place in ordered input places $(p_{i1}, p_{i2}, \ldots, p_{in})$ for a rule Ri, then the rule schema Ri′ becomes active. Furthermore,if a place p_j receives a token t_k, it activates every rule schema Ri′ whose first input place of ordered bag I(Ri) is p_j. Such activated rules made a *rule-packet.* Since this rule-packet is dynamically made whenever its first place in the bag of input place receives a token, we call this mechanism the *dynamic rule-packeting.* SPNG parser expects that the next input will be matched with one of next remnant input places of active

rules in the dynamic rule-packet. If a rule Ri is fired and its first input place has no token, that rule is deactivated and removed from the rule-packet.

If all conditions of the left-hand side of Ri′ are satisfied, then all patterns of the left-hand side are copied onto those of the right-hand side; hence, its translation rule is automatically applied. For example, consider an input sentence "Mary-ka gətninta" whose abbreviation is "Mk g". For parsing this sentence, one more rule schema R1′ besides R4′ of (11) is required as follows:

(12) R1′: $(p_4, p_5) \rightarrow p_1$
where $p_4 = n^*(\alpha, n^*, \alpha')$
$p_5 = CM(K, case = m)$
$p_1 = m^*(\alpha\text{-}K, m^*, \alpha')$.

place / Str	n*	CM	m*	m0	0	cand. rules	fired rule
Mk g							
k g	(M,n*,M′)					R1	
g	(M,n*,M′)	(k,case = 1)				R1	R1
g			(Mk,1*,M′)			R4 R5	
			(Mk,1*,M′)	(g,10,g′)		R4 R5	R4
					(Mk g,0, M′(^g′))		

Figure 5. Traces of parsing for Mary-ka gətninta.

Traces of parsing are shown in Figure 5 (**Str** means the current remaining input strings, **CM** is case marker, and **cand. rules** is candidate rules which can be enabled).

6. Parsing Algorithms

In this section, SPNG parsing algorithms are presented based on the previous sections.

Algorithm 1. (SPNG parser)

Assume that the lexicon B contains all the input strings, and there is no homonym which has differenct categories.

1. Read an input string s.

If s is the period (.), *then go to* 9.
2. Search s through B.
3. Generate a token t and
   Copy the entry of s in B into t.
4. Put t into the place p whose variable category p is matched with the <category> of t.
5. *If* there is no enabled rule,
   then go to 1.
   else perform the next step.
6. *If* there are more than one enabled rules,
   then choose a rule by special-condition priority.
7. Check conditions such that
   if functional-place priority is checked,
   then go to 1.
   if case-matchability condition is violated,
   then go to 5.
   if special conditions are not satisfied,
   then the enabled rule is ignored, and *go to 5.*
8. Fire the rule, that is, remove the input tokens and generate the output token such that
   Concatenate terminal strings,
   Perform special actions,
   Translate into the logical form.
   go to 1.
9. *If* every place has no token,
   then one input sentence is successfully parsed,
   else the input sentence is ill-formed.

In step 5, how can the parser know which rules are enabled? That is, how can the control structure of SPNG be described in an algorithm instead of a graphical representation. The most important problem is the maintenance of ordered bags.

Definition 7. The *hierarchy* function **h** maps from a pair of a rule Ri and a place to a value such that

$$h: R \times I(R) \rightarrow N$$
$$h: R \times O(R) \rightarrow 0$$
$$h: R \times (P\text{-}(I(R) \cup O(R))) \rightarrow \text{nil}$$

where the function h assigns to a place in I(R) its position number in I(R), and to places in an ordinary bag the same number.

Example 3. Consider the next rule Ri and the application of **h.** Let #(Ri) represent the number of input place in I(Ri). In this case, #(Ri) = 5 since the number of input places is 5.

I(Ri) =	$(p_1,p_2,$		$\{p_3,p_4\},p_5).$			O(Ri) =	p_6
h (Ri,pj)=	1	2	3	4	5		0

Definition 8. The *next-expected hierarchy* function H: R→N maps from a rule Ri into the place which can be expected to receive a token.

Given a token t, the following algorithm shows a maintenance method of ordered bags and enabled rules:

Algorithm 2. (Maintenance of ordered bags and rules)

1. Get a token t which belongs to a place P_j.
2. *If* places of I(Ri) have no token,
   then H(Ri) = 1,
   else if the first place of I(Ri) has at least one token,
   then H(Ri) = 2.
3. *Repeat for* every Ri such that $p_j \in$I (Ri)
   if the rule Ri satisfies h(Ri,P_j) <H(Ri),
   then begin
   insert Ri to a set of candiates of enabled rules,
   increase H(Ri) by 1,
   if H(Ri) > #(Ri)
   then begin
   Ri is enabled,
   if all conditions are satisfied.
   then begin
   fire Ri to the output place P_k for
   h (Ri,P_k) = 0 such that
   remove all tokens from I(Ri),
   generate a token t in P_k,
   go to 1
   end
   end
   end
   else remove Ri from the set of candidates of enabled rules.
   end of repeat
4. Terminate the algorithm.

In Figure 6, tables are made for illustrating the Algorithm 2. Consider an input sentence as

(13) $[[[\text{Mary}]_n*[\text{ka}]_{\text{case}=1}]_1*\ [\text{gətninta}]_{10}]_0$

Figure 6. Traces of Algorithm 2 for (13).

place	tokens	hierarchy of places in a rule					
		R1	R2	R3	R4	R5(1)	R5(2)
n*		1					
m*		0			1	1	1
CM		2					
m0					2	0	0
m*n0						2	2
0					0		
H		1	1	1	1	1	1
#		2			2	2	2
active rule?							

(1) input: Mary-ka gətn̈inta.

place	tokens	hierarchy of places in a rule					
		R1	R2	R3	R4	R5(1)	R5(2)
n*	Mary	1					
m*		0			1	1	1
CM		2					
m0					2	0	0
m*n0						2	2
0					0		
H		2	1	1	1	1	1
#		2			2	2	2
active rule?		y					

(2) input: -ka gətn̈inta.

place	tokens	hierarchy of places in a rule					
		R1	R2	R3	R4	R5(1)	R5(2)
n*	Mary	1					
m*		0			1	1	1
CM	-ka	2					
m0					2	0	0
m*n0						2	2
0					0		
H		3	1	1	1	1	1
#		2			2	2	2
active rule?		y,f					

(3) input: gətn~~i~~nta.

place	tokens	hierarchy of places in a rule					
		R1	R2	R3	R4	R5(1)	R5(2)
n*		1					
m*	Mary-ka	0			1	1	1
CM		2					
m0					2	0	0
m*n0						2	2
0					0		
H		1	1	1	2	2	2
#		2			2	2	2
active rule?					y	y	y

(4) input: gətn~~i~~nta.

place	tokens	hierarchy of places in a rule					
		R1	R2	R3	R4	R5(1)	R5(2)
n*		1					
m*	Mary-ka	0			1	1	1
CM		2					
m0	gətninta.				2	0	0
m*n0						2	2
0					0		
H		1	1	1	3	2	2
#		2			2	2	2
active rule?					y,f	y	y

(5) input:

place	tokens	hierarchy of places in a rule					
		R1	R2	R3	R4	R5(1)	R5(2)
n*		1					
m*		0			1	1	1
CM		2					
m0					2	0	0
m*n0						2	2
0	Mary-ka gətninta.				0		
H		1	1	1	1	1	1
#		2			2	2	2
active rule?							

(6) input:

7. Conclusion

SPNG represents AMG framework as it is from the point of view of theoretical and computational linguistics. The AMG approach offers profound advantages over traditional approaches: it dispenses with transformations without loss of insight, offers large linguistic coverage, and couples simple, semantically well-motivated rules of translation to the syntactic rules.

The preceding pages have discussed many different topics, including specific phenomena in the grammar of Korean, a partially free word order language. This discussion results in a conclusion such that every AMG rule is well represented in SPNG. SPNG explicitly reflects AMG framework. Each rule of AMG corresponds to a transition rule in SPNG. Arguments and values of each modified AMG rule for parsing are (ordered) bag of input places and output place of the corresponding transition rule. Translation rules are included in the attributes of transition rules of SPNG. Also, those attributes include special conditions and special actions which are coherent to syntactic rules of AMG. Variable categories labelled input places describe the categories of arguments in syntactic rules of AMG. Since every token has its numbered category, a token can be received by a place whose variable category is matched with the numbered category of the token.

SPNG has the control structure for parsing. Parsing mechanism with SPNG resembles execution rules of its underlying Petri net. But, the SPNG parsing mechanism includes several conditions to block out wrong applications of rules.

For comparison, ATN (Woods 1970) has its control structure, which is based on its underlying RTN (recursive transition network). But, this mechanism induces frequent backtrackings for free word order languages (Castelfranchi, et al. 1982, Ferrari 1982).

SPNG has a high flexibility to experiment many complex rules. Each SPNG rule is represented by a *primitive* of Petri net. Each primitive consists of a transition, its (ordered) bag of input places and an output place. Overall rules of SPNG are combined by merging places of primitive rules such that places with the same label of a variable category are unified into one place. For example, the resultant structure (Figure 7) of concatenating two rules Ri and Rj shows the relation between two rules:

$$I(Ri) = (p_1,p_2) \qquad O(Ri) = \{p_4\}$$
$$I(Rj) = (p_3,p_4) \qquad O(Rj) = \{p_1\}.$$

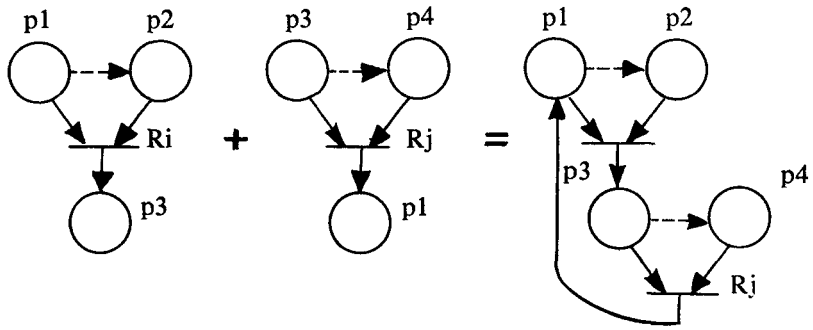


Figure 7. '+' operation means the 'concatenation'.

When rules are concatenated into the overall framework of SPNG, each rule preserves its characteristics in a *primitive*, without affecting other rules. Hence, the modification, addition and deletion of rules are highly flexible.

SPNG has the same analytic power with Turing machine, since it is an extended Petri net with inhibitor arcs (Peterson 1981). Hence, every natural language can be parsed by SPNG. On the other hand, SPNG can be used as a generation model such that it generates surface strings from their logical forms in the intensional logic. If rule transitions are performed conversely, SPNG can be easily transformed to a generation model.

SPNG framework is perspicuous. It has no hidden structure and no hidden information. Because it can reflect the linguistic framework AMG as it is, experimention with parsing is relatively easy.

REFERENCES

Castelfranchi, C., D. Parisi and O. Stock (1982) '"Free" Order Language: An Experimental Lexicon Based Parser,' in Hajičova, ed., *COLING82 Abstracts,* Univerzita Karlova, Praha, 65-73.

Choi, K.-S. and G.C. Kim (1983a) 'Parsing Korean Based on Revised PTQ,' *Proc. of the '83 Workshop on Formal Grammar,* February (19-20), Logico-Linguistic society of Japan, Kyoto.

_____ (1983b) 'Parsing Korean: A Free Word-order Language,' to be appeared in *Proc. of International Conf. of the Assoc. for Literary and Linguistic Computing,* April 6-9, Marine's Memorial Club, San Francisco.

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

Ferrari, G. and I. Prodanof (1982) 'Revising an ATN Parser,' in Hajičova, ed., *COLING82 Abstracts,* 101-105.

Gazdar, G. (1981) 'Phrase Structure Grammar,' in P. Jacobson and G.K. Pullum, eds., *The Nature of Syntatic Representation,* D. Reidel, Dortrecht.

Lee, K. (1981a) 'Case Assignment in Korean,' *Eohakgyoyuk: Language Education* 12, Language Institute, Chonnam National University, Gwangju, Korea, 269-285.

_____ (1981b) 'On Free Word Order and Case in Korean: A Montague Grammar Approach,' *MAL: The Journal of Korean Language Institute* 6, Yonsei University, Seoul, 51-87 (in Korean).

_____ (1982a) 'On Case Shifting,' *The Linguistic Journal of Korea* 7.2, 497-517. The Linguistic Society of Korea.

_____ (1982b) A Montague Grammar for Case Languages,' *Proc. of XIIIth International Congress of Linguistics,* Tokyo.

_____ (1983a) 'Case Assignment and Complex Verbs in Japanese: a Categorial Approach,' in *Festschrift for Masatake Muraki,* International Christian University, Tokyo, 100-124.

_____ (1983b) 'AMG: Case Theory in Montague Grammar,' *Proc. of '83 Kyoto Workshop on Formal grammar,* Feb. 19-20, Kyoto.

_____ (1983c) 'Relativization in an Augmented Montague Grammar,' presented at *'83 Seoul Workshop on Formal Grammar,* Dec. 19-21, Seoul, Ewha Woman's Univ.

Montague, R. (1974) *Formal Philosophy: Selected Papers of Richard Montague,* edited and with an introduction by Richmond Thomason, New Haven, CT, Yale University Press.

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

Woods, W.A. (1970) 'Transition Network Grammars for Natural Language Analysis,' *Comm. ACM* 13 (Oct.) 591-606.

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