15

Computational Paninian Grammar Framework

Akshar Bharati and Rajeev Sangal

15.1 Introduction

This chapter presents the Computational Paninian Grammar (CPG) framework and its comparision with Tree Adjoining Grammar and Supertagging. Paninian framework is explicated through its application to modern Indian languages. These languages have free word order and a rich system of case endings and postpositions.

In the Paninian framework, a sentence is analyzed in terms of dependency relations, more specifically modifier-modified relations.[1] A verb is related to its arguments through karaka relations (pronounced “kaaraka”), which are syntactico-semantic relations. Other relations are verb-verb relations indicating simultaneity, precedence, and the like; and noun-noun relations; and so forth.

Paninian grammar framework is an elegant account of modern Indian languages, and a computational account as well. As a result, the theory has elements that allow a system to be built for derivation and representation of meaning.

Section 15.2 goes into details of CPG for modern Indian languages. Section 15.3 presents some surprising similarities between CPG and Tree Adjoining Grammar (TAG). Relation with supertagging also follows naturally. Section 15.4 gives details of a constraint parser for CPG.

15.2 CPG for Indian languages

The Paninian framework considers *information* as central to the study of language. When a writer (or a speaker) uses language to convey some information to the reader (or the hearer), he codes the information in the language string.[2] Similarly, when a reader (or a hearer) receives a language string, he extracts the information coded in it. The Computational Paninian Grammar framework (CPG) is primarily concerned with how the information is coded and how it can be extracted.

Two levels of representation can be readily seen in language use: One, the actual language string (or sentence), two, what the speaker has in his mind. The latter can also be called as the meaning. Paninian framework has two other important levels: *karaka* and *vibhakti* (figure 15.1).

The surface level is the uttered or the written sentence. The vibhakti level is the

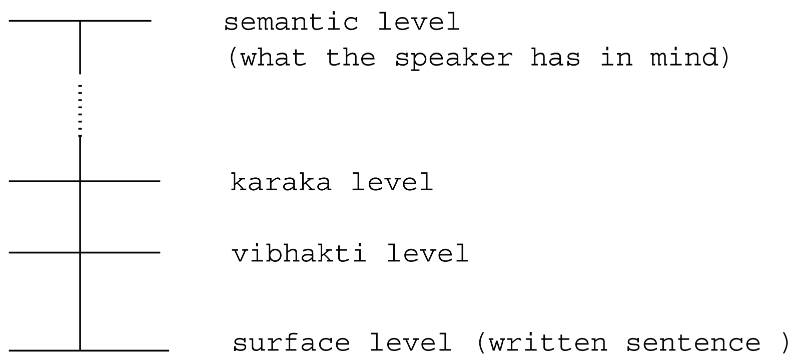

semantic level
(what the speaker has in mind)
karaka level
vibhakti level
surface level (written sentence)

FIGURE 15.1 Levels of representation/analysis in the Paninian model

level at which there are local word groups together with case endings, prepositions or postposition markers.[3]

At the karaka level, we have karaka relations, verb-verb relations, and so on. Karaka relations are syntactico-semantic (or semantico-syntactic) relations between the verbs and other related constituents (typically nouns) in a sentence. They capture a certain level of semantics that is close to thematic relations but different from it. But this is the level of semantics that is important syntactically and that is reflected in the surface form of the sentence(s).

The vibhakti level abstracts away from many minor (including orthographic and idiosyncratic) differences among languages. The topmost level relates to what the speaker has in mind. This may be considered to be the ultimate meaning level. Between this level and vibhakti level is the karaka level. It includes karaka relations and a few additional relations such as taadaarthya (purpose). One can imagine several levels between the karaka and the ultimate level, each containing more semantic information. Thus, karaka is one in a series of levels, but one that has relationship to semantics on one hand and syntax on the other.

As mentioned earlier, vibhakti for verbs can be defined similar to that for the nouns. A head verb may be followed by auxiliary verbs (which may remain as separate words or may combine with the head verb). Such information, consisting of the verb ending, the auxiliary verbs, and the like, is collectively called vibhakti for the verb. The vibhakti for a verb gives information about tense, aspect, and modality (TAM) and is therefore also called the TAM label. TAM labels are purely syntactic, determined from the verb form and the auxiliary verbs.

The grammar gives the mapping between the levels. For example, CPG specifies a mapping between the karaka level and the vibhakti level, and the vibhakti level and the surface form.

It has been shown earlier (Bharati et al., 1995) that the Paninian grammar is particularly suited to free-word-order languages. It gives a mapping between karaka relations and vibhakti, and uses position information only secondarily. As the Indian languages have (relatively) free word order and vibhakti, they are eminently suited to be described by Paninian grammar.

15.2.1 Karaka-vibhakti mapping

The most important insight regarding the karaka-vibhakti mapping is that it depends on the verb and its tense aspect modality (TAM) label. The mapping is represented by two structures: verb karaka frame (earlier called karaka chart) and karaka frame transformation. The basic karaka frame for a verb or a class of verbs specifies the vibhakti permitted for each of the nouns, which are its karaka relations.[4] In other words, when the verb in a sentence has the basic TAM label, then vibhakti is specified by the karaka frame for each of its related nouns in the sentence.

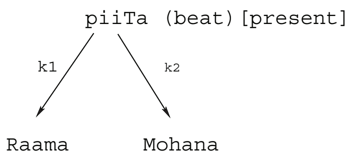
The basic karaka frame for three of the karakas is given in figure 15.2. This explains

Karaka	*Vibhakti*	*Presence*
Karta	ϕ	mandatory
Karma	ko or ϕ	mandatory
Karana	se or dvaaraa	optional

FIGURE 15.2 Default karaka frame

the vibhaktis in sentences (1)a and (1)b.

Take for example, the following modifier-modified structure: Using the karaka frame,



we get the following vibhaktis:

raama [ϕ], mohana [ko], piiTa [taa_hei]

which yield the following sentences:

(1)	a.	*raama*	*mohana*	*ko*	*piiTataa*	*hei.*
		Ram	Mohan	-ko	beat	is

Ram beats Mohan.

b.	*mohana*	*ko*	*raama*	*piiTataa*	*hei.*
	Mohan	-ko	Ram	beat	is

Ram beats Mohan.

Note that no order is specified by the grammar among the nouns.

The basic TAM label chosen for Hindi is *taa_hei* and roughly corresponds to present indefinite tense. For TAM labels other than basic, there are karaka frame transformation rules that need to be used first. For a given verb in a sentence, appropriate karaka-vibhakti mapping can be obtained in a two step procedure: First, using the basic karaka frame for the verbal root, and applying the transformation rule depending on its TAM label, a transformed karaka frame is obtained for the verb. Second, the transformed karaka frame determines the karakas of the verb using the vibhakti information. (Note that any other TAM label could have been chosen as basic without any problem. Moreover, the TAM labels are purely syntactic in nature and can be determined by looking at the verb form and the associated auxiliary verbs.)

It is important to reemphasize that the transformation depends on the TAM label, which is purely syntactic, and not on tense/time, aspect, and modality which are semantic. The TAM label can be determined for Hindi and other Indian languages by syntactic forms of the verb and its auxiliaries without the need to refer to any semantic aspects. The specification for obtaining TAM labels can be given by a finite state machine (Bharati et al., 1995, Chapter 4).

Paninian theory (i.e., karaka frames and karaka frame transformations) can be used to generate (or analyze) sentences such we have seen. However, there are additional constraints that would disallow the following sentences to be generated, for example[5]:

(2) a. **ladake ne raama ne laDakii ko kitaaba dii.*
boy -ne Ram -ne girl -ko book gave

*The Ram the boy gave a book to the girl.

b. **laDake ne laDakii ko kitaaba phoola dii.*
boy -ne girl -ko book flower gave

*The boy gave a book a flower to the girl.

The constraints are

1. Each mandatory karaka in the karaka frame for each verb group, is expressed *exactly once*. (In other words, a given mandatory karaka generates only one noun group with the specified vibhakti in its karaka frame unlike in (2)a.)
2. Each optional karaka in the karaka frame for each verb group, is expressed *at most once*.
3. Each source word group satisfies some karaka relation with some verb (or some other relation). In other words, there should no unconnected source word group in a sentence, otherwise, the sentence becomes bad as in (2)b.

Karaka frames are based on the idea of aakaankshaa and yogyataa. A karaka frame for a verb expresses its aakaankshaas, or demands, and specifies the vibhaktis that must be used (i.e., yogyataa) with word groups that satisfy the demands. The same ideas can be used to handle noun-adjectives, noun-noun relations, verb-verb relations, and so on each of which can be viewed as a demand-satisfaction pair.

15.2.2 Complex sentences

Let us now consider the generation of sentences with more than one verb group. We begin with an example. Suppose the speaker (or writer) wants to express the fact that

(3) a. *raama ne mohana ko phala khaakara bulaaya*
Ram erg. Mohan dat fruit having-eaten called

Ram called Mohan having eaten the fruit.

This can be expressed by the structure at the karaka level shown in figure 15.3.

One way of realizing it in a sentence is to choose *kara* TAM label for *khaa* (eat). This label specifies the temporal precedence relation with its parent node *bulaa* (call).[6]

The vibhaktis for the nouns can now be generated using the karaka frame (i.e., the basic karaka frame together with karaka frame transformation rule for *kara* shown in figure 15.4). In this example, the transformed karaka frame shows that the karta of *khaa* is not expressed. The vibhaktis for the nouns in figure 15.3 are as follows:

raama [ϕ], mohana [ko], phala [ϕ], khaa [kara], bulaa [taa_hei]

The generated sentence reads as:

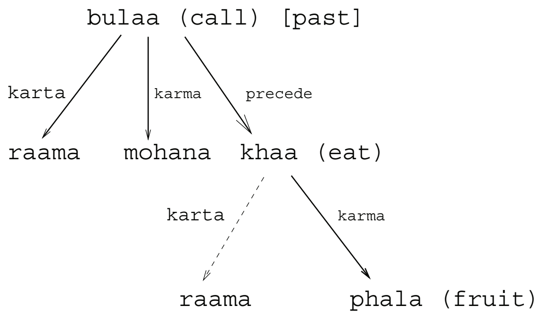


FIGURE 15.3 Modifier-modified relations for a complex sentence (shared karakas shown by dotted lines)

raama mohana ko phala khaakara bulaataa hei.

The karaka frame transformation rules are given in figure 15.4.

TAM label	*Transformation*
kara	Karta must not be expressed.
	Karma is optional.
naa	Karta and karma are optional.
taa_huaa	Karta and karma are optional.

FIGURE 15.4 Transformation rules for complex sentences

The karaka-sharing rule is given in Rule S1.

Rule S1: The karta of a verb with TAM label *kara* is the same as the karta of the verb modified by the verb.

15.2.3 Some additional constraints

Three additional constraints are included to handle some more grammatical constructions (Bharati et al., 2002). They are introduced along with example constructions in this subsection.

Sentential Argument of Verb

Some verbs take a sentence as their argument. For example,

(4) a.

raama	*ne*	*kahaa*	*ki*	*vaha*	*ghara*	*jaaegaa*	
Ram	erg.	said	that	he	home	will	go.

Ram said that he will go home.

Here *vaha ghara jaaegaa* (he will go home) is the karma or k2 argument of *kahaa* (say). In other words, the argument of the verb is a verb. In the preceding sentence, that argument appears to the right of the verb rather than to the left in case of normal arguments.

Consequently, two additional information can be specified in the karaka frame:

- ltype: type of lexical category (noun, verb, etc.)
- posn : Position of source word (l for left, r for right)

Here, is an example karaka frame for *kahaa* (or say) where *sampradaana* stands for beneficiary. When the values of ltype and posn are left blank, they stand for "noun" and *l* (or left) respectively by default.

Karaka	*Vibh*	*Pres*	*ltype*	*posn*
Karta	ϕ	m		
Karma	ki or ϕ	m	*v*	*r*
Sampradana	se or dvaaraa	opt		

FIGURE 15.5 Example karaka frame for 'kahaa'(say)

Adjectives

In case of adjectives, there is a demand frame associated with them, and they require a noun. However, the noun is the parent of the adjective. Therefore, an additional information to indicate the directionality of relation can also be given. For example, for adjectives, the demand frame is given in figure 15.6, where "p" says that the noun

Karaka	*Vibh*	*Pres*	*ltype*	*posn*	*reln*
nmod		*m*	n		p

FIGURE 15.6 Example demand frame for adjective

is the parent of the adjective. Spaces left blank are filled by defaults. (The default for vibhakti is any vibhakti without any constraint; other defaults were given earlier.)

15.3 Lexicalized Representation: Correspondence with TAG

The karaka frames of a verb are elementary lexicalized grammatical structures in CPG, much like supertags. There is a direct correspondence between derivation trees on one hand and modifier-modified trees on the other. As an example, the TAG derivation tree for the sentence.

The boy saw a girl.

is shown in figure 15.7.

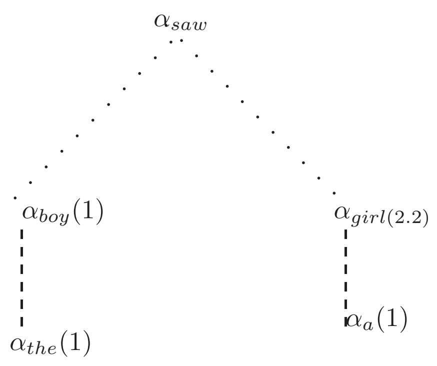


FIGURE 15.7 A TAG derivation tree.

Similarly, the modifier-modified tree in CPG for the following sentence in Hindi:

(5) a.

usa	*ladake*	*ne*	*eka*	*ladakii*	*ko*	*dekhaa*
that	boy	ergative	a	girl	accusative	saw

That boy saw a girl.

is shown in figure 15.8

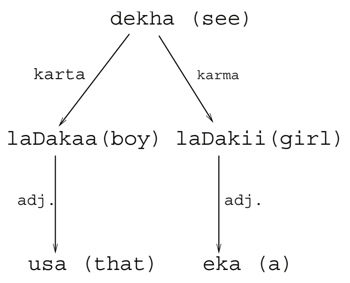


FIGURE 15.8 A CPG modifier-modified tree

The similarities are too obvious to be missed. Note also the correspondence between tree addresses in elementary trees of TAG (e.g., address (1) and (2.2) in α_{saw}) and the karaka label of demand word in CPG (e.g., karta and karma of *dekhaa* (saw)).

Addresses where substitution takes place in an elementary tree indicate its arguments; similarly, the karakas indicate the "arguments" of a verb or a demand word. For example, address 1 of α_{saw}, a formal object at this level of analysis, can be mapped to an appropriate theta role such as agent at the next level of analysis. In the case of CPG, the label karta, again a formal relation at this level of analysis, can be mapped to agent theta role at the next level of analysis.

If one compares the elementary trees (i.e., initial trees and auxiliary trees) in TAG with karaka frames in CPG, one again finds a similarity. An initial tree for a verb anchor (or a demand word) is like a karaka frame (or a demand frame). The initial tree for a verb specifies (along with initial trees for noun) how the nouns "fit" in with the verb anchor. This "fitting" is primarily in terms of word order. The karaka frame in CPG, on the other hand, specifies what vibhaktis must occur with the nouns, for them to "fit" in with the verb. This fitting is in terms of constraints on vibhaktis. All this is perfectly reasonable: Position works for positional languages, and vibhakti works for free-word-order languages. (See Bharati et al. 1997, where CPG has also been applied to English.)

Choosing a karaka frame out of the possible frames for a verb is like supertagging. Just as in TAG, supertagging helps select the elementary trees; similarly, in CPG, selection of the karaka frame would be helped using supertagging. As the grammar for Hindi in CPG framework becomes more extensive with several karaka frames for each verb, choosing the right karaka frame would help speed up parsing, much like how supertagging helps in parsing using TAG. Experimental studies will be carried out after a more extensive grammar is available.

15.4 Constraint-Based Parsing

The Paninian theory outlined herein can be used for building a parser. The first stage of the parser takes care of morphology. For each word in the input sentence, a dictionary or a lexicon is looked up, and the associated grammatical information is retrieved. In the next stage, local word grouping takes place, in which, based on local information, certain words are grouped together, yielding noun groups and verb groups. These are the word groups at the vibhakti level (i.e., typically each word group is a noun or verb with its vibhakti, TAM label, etc.). These involve grouping postpositional markers with nouns, auxiliary verbs with main verbs, and so forth. Rules for local word grouping are given by finite state machines. Finally, the karaka relations among the elements are identified in the last stage, called the *core parser* (Bharati et al., 1995, Chapter 6).

The task of the core parser is to identify karaka relations. It requires karaka frames and transformation rules. For a given sentence after the word groups have been formed, each of the noun groups is tested against each row (called *karaka restriction*) in each karaka frame for each of the verb groups (provided the noun group is to the left of the verb group whose karaka frame is being tested). When testing a noun group against a karaka restriction of a verb group, vibhakti information is checked, and if found satisfactory, the noun group becomes a candidate for the karaka of the verb group.

This can be shown in the form of a constraint graph. Nodes of the graph are the word groups, and there is an arc labeled by a karaka from a verb group to a noun group, if the noun group satisfies the karaka restriction in the karaka frame of the verb group. (There is an arc from one verb group to another, if the karaka frame of the former shows that it takes a sentential or verbal karaka.) The verb groups are called demand groups, since they make demands about their karakas, and the noun groups are called source groups because they satisfy demands.

As an example, consider a sentence containing the verb *khaa* (eat):

(6) a. *baccaa haatha se kelaa khaataa hi*
child hand -se banana eats

The child eats the banana with his hand.

Its word groups are marked, and *khaa* (eat) has the same karaka frame as in figure 15.2. Its constraint graph is shown in figure 15.9.

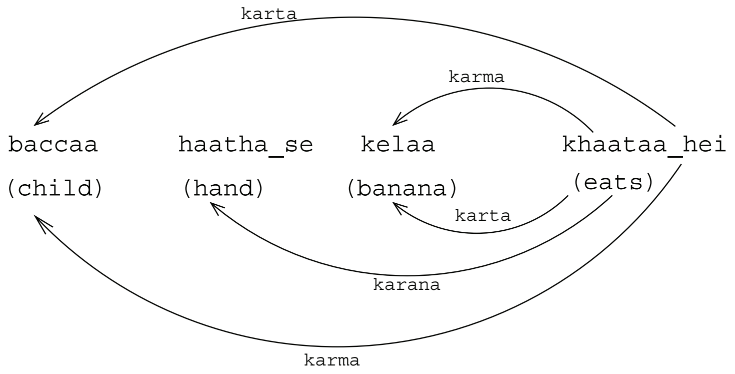


FIGURE 15.9 Constraint graph for sentence (6)

A parse is a sub-graph of the constraint graph containing all the nodes of the constraint graph and satisfying the following conditions:

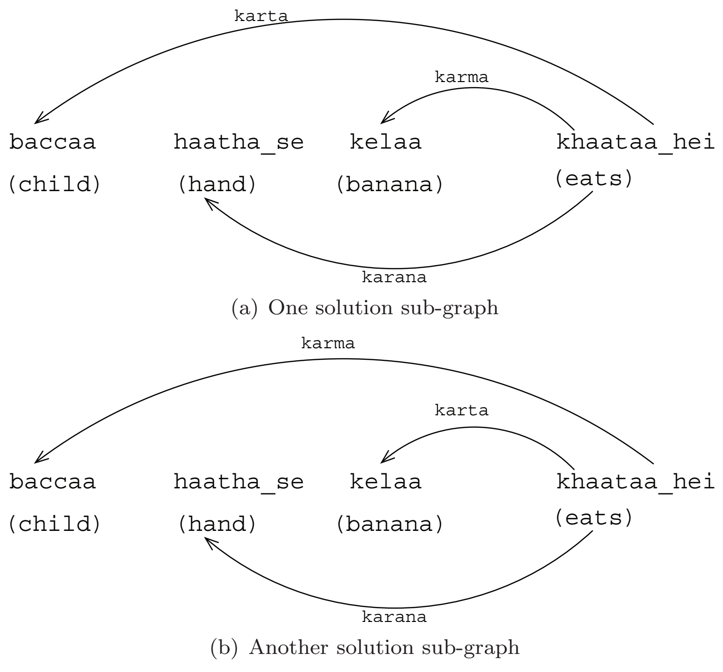
C1. For each of the mandatory karakas in a karaka frame for each demand group, there should be *exactly one* outgoing edge labeled by the karaka from the demand group.

C2. For each of the desirable or optional karakas in a karaka frame for each demand group, there should be *at most one* outgoing edge labeled by the karaka from the demand group.

C3. There should be *exactly one* incoming arc into each source group.

If several subgraphs of a constraint graph satisfy these conditions, it means that there are multiple parses and the sentence is ambiguous. If no subgraph satisfies the above constraints, the sentence does not have a parse and is probably illformed.

For the example sentence (6), and its constraint graph in figure 15.9, its subgraphs which are solution parses, are given in figure 15.10. Note that only one karta and one karma relations occur in each of the two parses (to satisfy constraint C1), and all the other constraints are also satisfied.



(a) One solution sub-graph

(b) Another solution sub-graph

FIGURE 15.10 Two parses for sentence (6)

15.4.1 Integer programming constraints

A parse can be obtained from the constraint graph using integer programming. A constraint graph is converted into an integer programming problem by introducing a variable x for an arc from node i to j labeled by karaka k in the constraint graph such that for every arc there is a variable. The variables take their values as 0 or 1. A parse is an assignment of 1 to those variables whose corresponding arcs are in the parse subgraph, and 0 to those that are not. Equality and inequality constraints in integer programming problem can be obtained from the conditions (C1, C2, and C3) listed earlier, as follows respectively:

1. For each demand group i, for each of its mandatory karakas k, the following

equalities must hold:

$$M_{i,k} : \sum_j x_{i,k,j} = 1$$

Note that $M_{i,k}$ stands for the equation formed, given a demand word i and karaka k (given that the karaka frame has already been selected like in supertagging). Thus, there will be as many equations as combinations of i and k.

2. For each demand group i, for each of its optional or desirable karakas k, the following inequalities must hold:

$$O_{i,k} : \sum_j x_{i,k,j} \leq 1$$

3. For each of the source groups j, the following equalities must hold:

$$S_j : \sum_{i,k} x_{i,k,j} = 1$$

Thus, there will be as many equations as there are source words.

The cost function to be minimized is the sum of all the variables.

15.4.2 Constraint parser using matching and assignment

With the constraints (C1, C2, and C3) specified earlier, the parsing problem reduces to bipartite graph matching[7] and assignment problems (Bharati et al., 1995, chapter 6). These have efficient solutions even in the worst case.

To perform the reduction of the problem of finding a solution graph to finding a matching, first a bipartite graph may be constructed.

A *bipartite graph* G(V,U,E) is defined as:

U: set of nodes $u_1, u_2, ..., u_n$
V: set of nodes $v_1, v_2, ..., v_m$
E: edges between U and V, and $U \bigcap V = \phi$.

The bipartite graph is constructed in three stages:

1. For every source node *s* in the constraint graph, form a node *s* in U.
2. For every demand node *d* in the constraint graph and every mandatory karaka *k* in the karaka chart for *d*, form a node *v* in V. (Thus, for every pair (*d*,*k*) there is a node in V.)
3. For every edge (d,s) labeled by karaka *k* in the constraint graph, create an edge between node (*d*,*k*) in V to *s* in U.

For example, for the constraint graph in figure 15.9 (but assuming that the optional karana karaka is mandatory) we have the bipartite graph in figure 15.11.

A *matching* M of a bipartite graph $G = (U, V, E)$ is a subset of edges with the property that no edges of M share the same node. The matching problem is to find a *maximal* matching of G, that is, a matching with the largest number of edges. A maximal matching is called a *complete* matching, if every node in U and V has an edge.

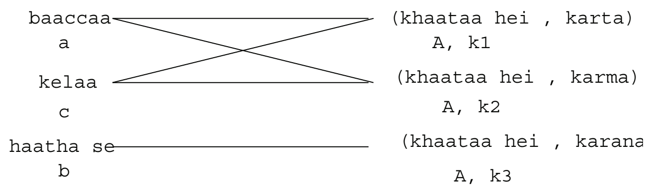
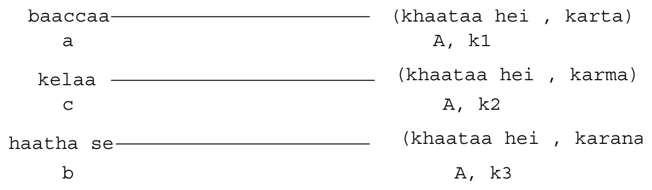
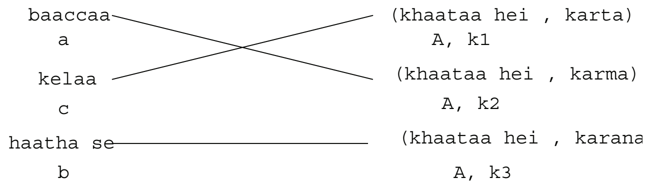


FIGURE 15.11 Bipartite graph for constraint graph in figure 15.9

There are two maximal complete matchings of the graph in figure 15.11. They are shown in figure 15.12. They correspond to the parses shown in figure 15.10.



(a) Solution corresponds to parse in figure 15.10(a)



(b) Solution corresponds to parse in figure 15.10(b)

FIGURE 15.12 Maximal (complete) matchings of bipartite graph of figure 15.11

Now we show that finding a maximal matching in a bipartite graph is the same as finding a parse in a constraint graph. Let M be a maximal matching of a bipartite graph G. If M is complete, it represents a parse. The proof is easy. For an edge between a node (d,k) in V and a node S in U, it represents the demand for karaka k of the demand node d being satisfied by the source node s. Since all the nodes in V have exactly one edge in M, the original constraint C1 is satisfied. Since all the nodes in U have exactly one edge in M, constraint C3 is satisfied. (And since there are no optional karakas, C2 is satisfied trivially.)

If M is not complete, that is, it does not have an edge on at least one node in U or V, then G does not have a parse. If a node in V does not have an edge, it is a violation of constraint C1, otherwise, it is a violation of constraint C3. Therefore, M is not a parse. Let the cardinality of M be m. Clearly $|U| > m$ or $|V| > m$. Since M is a maximal matching, there is no matching on G with a larger number of edges. Therefore, any other maximal matching (which might choose a different set of edges) will cover exactly m nodes in U and m nodes in V. Consequently, some node in U or V would again not have an edge. Thus, any other maximal matching would also not give us a parse.

The converse is left as an exercise.

To find a maximal matching of a bipartite graph, there is the well-known augmenting path algorithm. (See Papadimitrou and Steiglitz 1982 and Ahuja et al. 1993 for a description.)[8]

This works when all the karakas are mandatory. To handle optional karakas as well, the problem can be reduced to assignment problem, which has a known solution by the Hungarian method with the time complexity $O(n^3)$. See Bharati et al. (1995, section. 6.4.2) to see the reduction.

15.5 Conclusions

In this chapter, we have introduced Computational Paninian Grammar (CPG) formalism, which gives an elegant account for Indian languages. It uses demand frames based on root, transformation of frames based on TAM (tense-aspect-modality), and so on. There are also well developed karaka sharing rules. CPG has also been applied to English elsewhere (Bharati et al., 1997).

There are remarkable similarities between the tree adjoining grammar (TAG) and the Computational Paninian grammar (CPG). Derivation trees in TAG and modifier-modified trees in CPG have a one-to-one correspondence. It should be noted that the existing TAG and CPG are both lexicalized and have good locality. CPG has optional arguments and sentential arguments of verbs as a part of its karaka frame.

It should be further mentioned here that a dependency treebank for Hindi (based on the Paninian framework) is being created. The framework covers all the relations given here, but it also includes participial and relative clause constructions. Once such a treebank is available, it will be used for conducting experiments pertaining to supertagging as well as full parsing.

15.6 Acknowledgments

The development of the Computational Paninian Grammar (CPG) framework has major inputs from Vineet Chaitanya, who really should be the principal author. The work also owes a debt of gratitude to K.V. Ramakrishnamacharyulu of Rashtriya Sanskrit Vidyapeetha, Tirupati, who helped with many discussions and clarifications. Amba Kulkarni did work on the development of ideas as well as on implementation of the Paninian framework. The authors were supported by Satyam Computers from 1998 to 2002, which is gratefully acknowledged.

This chapter combines material available in different sources. It includes material from Bharati and colleagues (1995, 2002) on Paninian grammar and parsing.

Notes

1. Paninian theory was formulated by Panini 2,500 years ago for Sanskrit. It evolved with the contributions of grammarians that followed.

2. By string we mean any word, phrase, sentence, paragraph, and the like.

3. For positional languages such as English, it would also include position or word order information.

4. What karaka relations are applicable for a verb obviously depends on the particular verb. Not all verbs will take all possible karaka relations. For example, akarmaka (roughly, intransitive) verbs do not take karma karaka.

5. A '*' before a sentence indicates that it is not a good sentence.

6. However, it comes packaged, so to say, with the karaka sharing constraint that karta of *khaa* must be the same as that of its parent. Since this constraint is satisfied, the choice of TAM as *kara* is acceptable. More on this later.

7. We are indebted to Somnath Biswas for suggesting the reduction.

8. The fastest known algorithm has asymptotic time complexity of $O(|V|^{1/2}.|E|)$ and is based on max flow problem (Hopcroft and Karp, 1973). The reduction itself can be carried out in linear time on number of nodes and edges.

References

Ahuja, R., Magnanti, T.L., and Orlin, J. (1993). Network Flows:Theory, Algorithms, and Applications Prentice Hall.

Bendapudi, P. (1992). *Algorithmic Aspects of Natural Language Parsing Using Paninian Framework*. M.Tech. thesis, Dept. of CSE, IIT Kanpur.

Bharati, A., Bhatia, M., Chaitanya, V., and Sangal, R. (1997). Paninian Grammar Framework Applied to English. South Asian Language Review.

Bharati, A., Chaitanya, V., and Sangal, R. (1994). Anusaraka or Language Accessor: A Short Introduction. In *Automatic Translation*, Thiruvananthpuram, Int. School of Dravidian Linguistics.

Bharati, A., Chaitanya, V., and Sangal, R. (1995). Natural Language Processing: A Paninian Perspective. Prentice-Hall.

Bharati, A., and Sangal, R., (1993). Parsing Free Word Order Languages in the Paninian Framework. In *Proceedings of Annual Meeting of Association for Computational Linguistics*, pp. 105–111.

Bharati, A., Sangal, R., and Reddy, T.P. (2002). A Constraint Based Parser Using Integer Programming. In *Recent Advances in NLP, Proc. of International Conference on NLP-2002*.

Bhatia, M. (1995). Paninian Theory Applied to English. Dept. of CSE, IIT Kanpur, 1995. Bachelor's thesis.

Bhatt, R. (1993). Paninian Theory for English. Dept. of CSE, IIT Kanpur, 1993. Bachelor's thesis.

Papadimitrou, C., and Steiglitz, K. (1982). Combinatorial Optimization: Algorithms and Complexity. Prentice Hall.