Incremental Analysis of Japanese Dependency Relations with a Generalized Discrimination Network

Manabu Okumura*, Tomoyosi Akiba**, Hozumi Tanaka**
* School of Information Science,
Japan Advanced Institute of Science and Technology, Hokuriku
Tatsunokuchi, Ishikawa, 923-12, Japan
E-mail: oku@jaist-east.ac.jp
** Department of Computer Science,
Tokyo Institute of Technology
2-12-1, O-okayama, Meguro-ku, Tokyo, 152, Japan
E-mail: {akiba,tanaka}@cs.titech.ac.jp

Abstract

In this paper we present a semantics-driven incremental disambiguation approach for Japanese structural ambiguity. Structural ambiguity occurs in Japanese analysis when we determine dependency relations in which one phrase modifies another in a sentence. We show that by using a generalized discrimination network(GDN) as a representation of a set of case frames, we can resolve structural ambiguity earlier than by ordinary integrated-and-incremental approaches. In our approach, semantic checking can be executed not only after both are analyzed, but also as soon as two modifiers which might modify the same modifyee are analyzed. Further integrated with lower level analyses, such as morphological analysis, our truly incremental disambiguation approach is considered useful in that our earlier decision making based on syntactic and semantic knowledge contributes to the further suppression of ambiguities at lower levels.

1 Introduction

In natural language analysis, one of the most important tasks is to resolve ambiguities in a sentence, because if they are not sufficiently resolved, many anomalous and undesirable results are obtained. Structural ambiguity occurring in syntactic analysis is one such ambiguity. It is known to cause a Catalan number of ambiguous parse trees[5], which increase at an exponential rate if syntactic analysis is performed separately from other analyses, such as semantic analysis, that might follow it. In such a case, all the parse trees have to be checked, one by one, for their semantic appropriateness by semantic analysis.

One solution of this problem of so-called two-stage approach is to avoid generating a combinatorial number of individual parse trees explicitly, and to produce a compact representation of ambiguity that leaves all local ambiguities packed. Representations such as the syntactic graph[22], the constraint network[15], the modifying relation table[23] and the right-branching tree[7], and notions such as ambiguity procrastination[21] and least commitment parse[20] are considered to belong to this category.

The other solution is to integrate syntactic analysis with other analyses, such as semantic analysis, and to resolve ambiguity incrementally during one analytical process[16, 2, 12, 9]. In this approach, semantic knowledge, such as case frames, is used to check the applicability of grammar rules in syntactic analysis. It contributes to the suppression of anomalous partial parse trees generated during the analytical process, and thus the derivation of a combinatorial number of parse trees from them can be avoided.

In this paper we present a semantics-driven incremental disambiguation approach for Japanese structural ambiguity. Structural ambiguity occurs in Japanese analysis when we determine dependency relations in which one phrase modifies another in a sentence(this is described in more detail in section three). However, syntactic knowledge, such as grammar rules, is not very useful for structural disambiguation in Japanese because of the language's relatively free word order and widespread ellipsis. Therefore, in our approach semantic knowledge(case frames) is used directly to resolve structural ambiguity incrementally during the analytical process.

We show that by using a generalized discrimination network (GDN)[19] as a representation of a set of case frames, we can resolve structural ambiguity earlier than by ordinary integrated-and-incremental approaches. In ordinary approaches, semantic checking(in which semantic appropriateness of a potential dependency relation is checked in terms of case frames) is executed after both the modifying phrase(modifier) and the modified phrase(modifyee) are analyzed. In our approach, however, semantic checking can be executed not only after both are analyzed, but also as soon as two modifiers which might modify the same modifyee are analyzed. In Japanese, a modifyee(usually a verb) usually appears after all of its modifiers in a sentence, and so semantic checking begins sooner in our approach than in ordinary ones. GDN enables to resolve word sense and structural ambiguities truly incrementally whenever a phrase appears in a sentence. Further integrated with lower level analyses, such as morphological analysis, our truly incremental disambiguation approach is considered useful in that our earlier decision making based on syntactic and semantic knowledge contributes to the further suppression of ambiguities at lower levels. At first we proposed GDN for incremental word sense disambiguation,but in this paper we show that it is also useful for incremental structural disambiguation.

In section two, we briefly outline GDN. In section three, we describe how GDN drives incremental analysis of Japanese dependency relations.

2 Outline of Generalized Discrimination Networks

In this section, we outline GDN's characteristics and principles to make comprehensible the explanation of incremental structural disambiguation with GDN, which is described in the following section. For details, please refer to [19].

2.1 Characteristics of Generalized Discrimination Networks

A discrimination network is a generalization of a decision tree[3] and has been used for various problem solving systems[4], especially in representing multiple word meanings compactly in natural language processing [10, 17, 13, 1]. A discrimination network is considered a directed acyclic graph with one root node and many leaf nodes.

Figure 1 is a portion of the discrimination network that represents word senses(and case frames) of Japanese verbs. Each branch of the network is labeled as one of two types of the constraint. The first type is a selectional restriction on surface cases(postpositions 'ga,' 'wo,' and so on) with which the verbs co-occur. The other type is on the entry word of the verbs(for example, 'verb/kakeru'), which indicates the entry word that a word sense(a case frame)

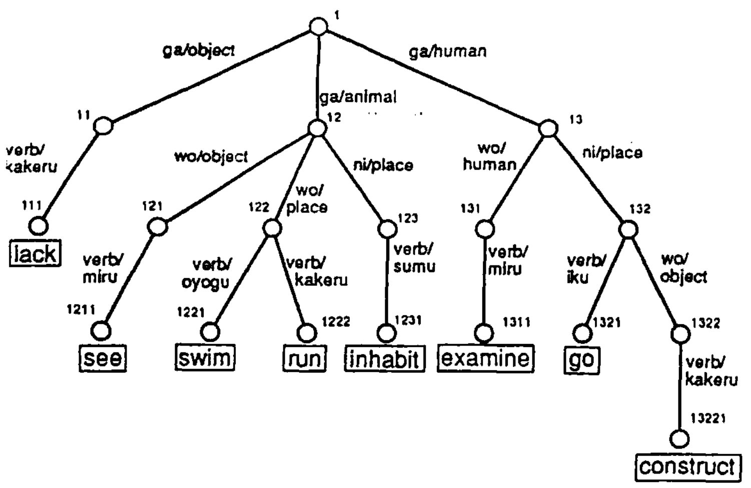

Figure 1: A portion of the discrimination network of word senses of Japanese verbs

belongs to. Each leaf node of the network points to a word sense of the verbs, which is represented by the boxed label. A set of constraints in the path from the root node to a leaf node represents the case frame for the word sense corresponding to the leaf node. Other nodes represent ambiguous word meanings that include all word senses corresponding to the leaf nodes below them, because from these the further traversal along branches to multiple nodes is possible. The root node corresponds to the most ambiguous meaning[1].

The semantic analysis(word sense disambiguation) process using a discrimination network is a step by step downward traversal of the network from the root node to a leaf node guided by branches, the constraints of which are satisfied by the information of the analyzed phrases. In this process, semantically inappropriate alternatives are rejected, and appropriate word senses are selected by virtue of information about other co-occurring phrases in the sentence.

A discrimination network has the following advantages:

- In the discrimination network, a unique node can represent multiple word senses that correspond to the leaf nodes below it. Therefore, the downward traversal of the network corresponds to the continuous refinement of an ambiguous word meaning into a more specific one. This is what the constraint programming paradigm[6] will achieve;

- The discrimination network's search algorithm is more efficient than a linear search, because the downward traversal is guided by constraints which are labels of branches, and the search space can be gradually narrowed down;

- Because the common selectional restrictions(constraints) can be merged into only one branch in the network, we can compact a set of case frames. Therefore, we need only check once for one constraint to see if the constraint is satisfied, thus avoiding wasteful and repetitious computations.

Despite the discrimination network's impressive characteristics mentioned above, it does have some critical problems. Because the network is traversed downward from the root node, constraints must be entered one by one from constraints that are labels of branches connected to the root node. However, because incrementally obtained constraints might deviate from an a priori-fixed order and because some constraints that are necessary for traversing the network downward might not be obtained, it is not always possible to traverse the network downward during the analytical process.

[1]We explain identifiers attached to the nodes in the next subsection.

GDN solves these problems of the discrimination network that prevent the incremental word sense disambiguation approach, and enables the downward traversal of the network, which uses the incrementally obtained constraints during the analytical process. We think incremental disambiguation[16] is also a better strategy for word sense disambiguation, because a combinatorial explosion of the number of total ambiguities might occur if word sense ambiguity is not incrementally resolved as early as possible whenever constraints are obtained during the analytical process of a sentence.

As surface variations such as relative clauses and passive forms in English demonstrate, problematic situations often develop for the discrimination network. In Japanese, the situation is even worse. Because Japanese has greater word order flexibility, the degree of deviation from the a priori-fixed order can be considerable. In addition, in Japanese, phrases that can be easily understood from contextual information are often omitted. Therefore, we think GDN is essential to the analysis of such a language as Japanese.

In the next subsection, we briefly describe how GDN executes semantic analysis.

2.2 Incremental Semantic Analysis with Generalized Discrimination Networks

Consider the analytical process of the sentence 'hasi wo kawa ni kakeru(someone constructs a bridge over the river)'[2], using the network shown in Figure 1. For the traversal of GDN, we provide correspondences between a constraint and a 'conditional identifier.' A conditional identifier consists of an identifier of a node followed by an 'if-clause' that represents a list of unsatisfied constraints. This correspondence between constraint and identifier means that if a constraint of a branch is satisfied, the nodes of the corresponding identifiers can be reached in the network conditionally(if the constraints in the if-clause are satisfied). An identifier with no if-clause means that a node of the identifier can be reached unconditionally. In Figure 1, identifiers of the nodes are given, and unsatisfied constraints in the if-clause are computed as a list of the constraints, other than the satisfied one, in the path from the root node to the reached node. For example, if the constraint *wo/human* is satisfied, the network can be traversed downward to the node of the corresponding identifier 131 if the constraint *ga/human* is satisfied.

Now we informally describe the analytical process of the above sentence, in which a necessary constraint *ga/human* is not obtained(ellipsis of the postpositional phrase corresponding to 'someone') and the order of the obtained constraints is irregular(order flexibility of postpositional phrases). Figure 2 shows a 'state' transition which represents the discrimination process. A state is represented in the form of a conditional identifier. The initial state(in which no constraints are obtained) is 1(the identifier of the root node). After the phrase 'hasi wo(a bridge)' is analyzed, the state is computed as follows, with the current state(the initial state) and a set of conditional identifiers {121 *if ga/animal*, 1322 *if ga/human* & *ni/place*} corresponding to the constraint satisfied by the phrase:

> Both 121 and 1322 include 1 as a prefix-numerical string, so the longer strings 121 and 1322 are returned[3]. Because the current state has no if-clause, the if-clause of the next state becomes the same as the if-clause of the conditional identifiers corresponding to the obtained constraint. Therefore, the next state becomes {121 *if ga/animal*, 1322 *if ga/human* & *ni/place*}.

Next, the phrase 'kawa ni(over the river)' satisfies the constraint *ni/place*, which corresponds to a set of the conditional identifiers

[2]The postpositional phrase corresponding to 'someone' is omitted in the original Japanese sentence.

[3]As shown in Figure 1, identifiers of mutually reachable nodes in the network are in a prefix-numerical string relation with each other. If one node is reachable from the other, the identifier of the subordinate one is returned. This operation corresponds to a downward traversal of the network by the obtained constraints.

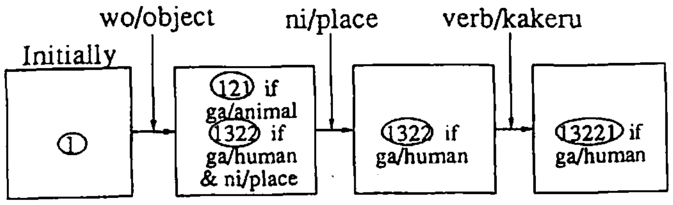

Figure 2: The discrimination process using GDN

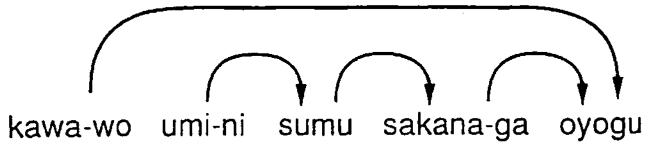

Figure 3: Dependency structure of the sentence 'kawa wo umi ni sumu sakana ga oyogu'

{123 *if ga/animal*, 132 *if ga/human*}. Any combinations of the identifiers but 132 and 1322 are not in a prefix relation, and so the analyses fail. Therefore, the resultant identifier is 1322 from identifiers 132 and 1322. The resultant if-clause is *if ga/human* because the constraint *ni/place* in the if-clause of the current state is obtained and removed from it.

Similarly, after the phrase 'kakeru(construct)' is analyzed, the final result of the traversal for the above case becomes 13221 *if ga/human*, which means that node 13221 is reachable if the constraint *ga/human* is obtained.

Note that the word sense of the verb 'kakeru,' which appears in the last of the sentence, is uniquely determined to be 'construct' among three word senses(corresponding to identifiers 111, 1222 and 13221) before the verb is analyzed. Further, the place of the missing phrase is detected as the unsatisfied constraint *ga/human*.

Compared with the ordinary approach for semantic analysis using case frames, where semantic checking(used to check whether a postpositional phrase('kawa ni') satisfies a selectional restriction of a verb) is not executed until a verb('kakeru') is analyzed, our approach executes earlier semantic checking before a verb is analyzed later in the sentence. Our approach executes semantic checking(word sense disambiguation) truly incrementally whenever a phrase is analyzed.

3 Incremental Dependency Analysis with Generalized Discrimination Networks

Unlike other languages which are based on phrase structure, Japanese sentences have a dependency structure among their phrases. A Japanese sentence can be segmented into a sequence of phrases called 'bunsetu.' In this paper the term 'phrase' is used to mean Japanese 'bunsetu.' A phrase is regarded as a minimal semantic element in a sentence. The internal structure of a phrase has been well studied in Japanese linguistics, so the details are not given here. We assume the results of past 'inner-phrase' analyses. The dependency relation is one in which a phrase modifies another in a sentence. The dependency structure for the sentence 'kawa wo umi ni sumu sakana ga oyogu(Fishes that inhabit the sea swim in the river)' is shown in Figure 3. In Figure 3, arrows from modifying phrases(modifiers) to modified ones(modifyees) represent dependency relations. For example, the postpositional phrase 'kawa wo(in the river)' modifies the verb 'oyogu(swim).'

In this section, we describe how these dependency relations among phrases are analyzed incrementally during the analytical

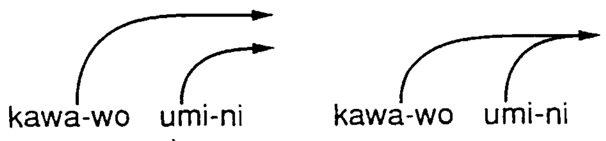

Figure 4: The intermediate state of incremental dependency analysis at the phrase 'umi ni'

process of a sentence, based on GDN-represented case frames.

3.1 The Process of Incremental Dependency Analysis

Incremental dependency analysis is to determine dependency relations between two phrases incrementally as a sentence is analyzed from left to right. The intermediate state of incremental dependency analysis (after a phrase is analyzed) is represented as a set of stacks[4] of arrows which pass over or start from the phrase and do not end. The stack indicates a set of potential dependency relations in which modifyees are not determined[5]. Multiple stacks might exist because of the ambiguity. For example, the state at the time after the phrase 'umi ni' is analyzed in the sentence 'kawa wo umi ni sumu sakana ga oyogu' is shown in Figure 4. The left stack is for the structure in which two phrases('kawa wo' and 'umi ni') modify a different modifyee, and the right stack is for that in which two phrases modify the same modifyee[6].

Dependency analysis operations act on each stack of the current state whenever a phrase is analyzed. There are two types of operations; those for treating the phrase as a modifyee and those for treating the phrase as a modifier. We call them 'In phase' and 'Out phase' operations, respectively. We provide the following In and Out phase operations, which are illustrated in Figure 5:

- In

  pass no phrases modify the phrase. The current stack is handed to the Out phase unchanged.

  pop phrases(an arrow) on the stack top of the current stack modify the phrase. The element of the stack top is popped and the stack is handed to the Out phase.

- Out

  merge an arrow from the phrase is merged with the arrow of the stack top, regarding the phrase as modifying the same modifyee as the set of phrases indicated by the arrow.

  push a new arrow is generated from the phrase and is pushed on the stack top.

As is clear from the above descriptions, both operations in the phases can be non-deterministically executed, unless knowledge useful for disambiguation is taken into account. Therefore, the ambiguity of dependency relations occurs in the analytical process and causes multiple stacks, as shown in Figure 4. Although our approach resolves the ambiguity as early and as much as possible using syntactic and semantic knowledge described in subsequent subsections, we also pack a set of stacks if possible. As in Tomita's algorithm[24, 18], we make a set of stacks structured in the tree form in cases where stack tops are common. This tree-structured stack avoids redundant computation as much as possible. For example, two stacks

[4]We use a stack representation to naturally take into account the 'no crossing principle' in Japanese, as described in the next subsection.

[5]Fixed dependency relations are stored differently as a part of intermediate semantic representation for a sentence.

[6]This is the meaning of the merged arrow, as described below.

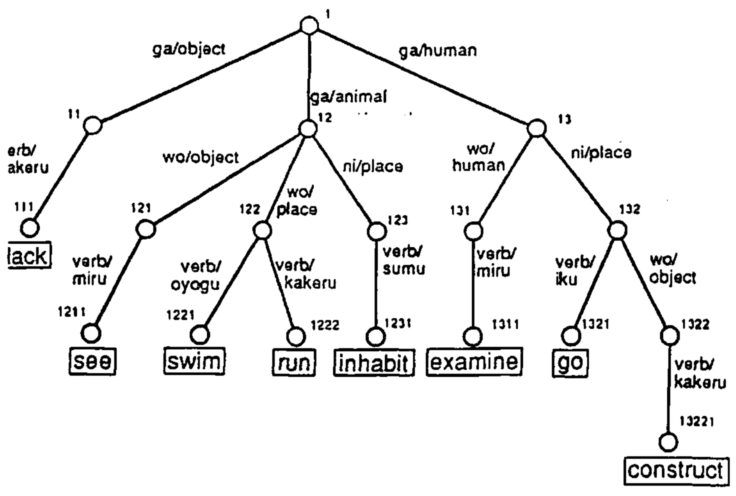

Figure 1: A portion of the discrimination network of word senses of Japanese verbs

belongs to. Each leaf node of the network points to a word sense of the verbs, which is represented by the boxed label. A set of constraints in the path from the root node to a leaf node represents the case frame for the word sense corresponding to the leaf node. Other nodes represent ambiguous word meanings that include all word senses corresponding to the leaf nodes below them, because from these the further traversal along branches to multiple nodes is possible. The root node corresponds to the most ambiguous meaning[1].

The semantic analysis(word sense disambiguation) process using a discrimination network is a step by step downward traversal of the network from the root node to a leaf node guided by branches, the constraints of which are satisfied by the information of the analyzed phrases. In this process, semantically inappropriate alternatives are rejected, and appropriate word senses are selected by virtue of information about other co-occurring phrases in the sentence.

A discrimination network has the following advantages:

- In the discrimination network, a unique node can represent multiple word senses that correspond to the leaf nodes below it. Therefore, the downward traversal of the network corresponds to the continuous refinement of an ambiguous word meaning into a more specific one. This is what the constraint programming paradigm[6] will achieve;

- The discrimination network's search algorithm is more efficient than a linear search, because the downward traversal is guided by constraints which are labels of branches, and the search space can be gradually narrowed down;

- Because the common selectional restrictions(constraints) can be merged into only one branch in the network, we can compact a set of case frames. Therefore, we need only check once for one constraint to see if the constraint is satisfied, thus avoiding wasteful and repetitious computations.

Despite the discrimination network's impressive characteristics mentioned above, it does have some critical problems. Because the network is traversed downward from the root node, constraints must be entered one by one from constraints that are labels of branches connected to the root node. However, because incrementally obtained constraints might deviate from an a priori-fixed order and because some constraints that are necessary for traversing the network downward might not be obtained, it is not always possible to traverse the network downward during the analytical process.

[1]We explain identifiers attached to the nodes in the next subsection.

GDN solves these problems of the discrimination network that prevent the incremental word sense disambiguation approach, and enables the downward traversal of the network, which uses the incrementally obtained constraints during the analytical process. We think incremental disambiguation[16] is also a better strategy for word sense disambiguation, because a combinatorial explosion of the number of total ambiguities might occur if word sense ambiguity is not incrementally resolved as early as possible whenever constraints are obtained during the analytical process of a sentence.

As surface variations such as relative clauses and passive forms in English demonstrate, problematic situations often develop for the discrimination network. In Japanese, the situation is even worse. Because Japanese has greater word order flexibility, the degree of deviation from the a priori-fixed order can be considerable. In addition, in Japanese, phrases that can be easily understood from contextual information are often omitted. Therefore, we think GDN is essential to the analysis of such a language as Japanese.

In the next subsection, we briefly describe how GDN executes semantic analysis.

2.2 Incremental Semantic Analysis with Generalized Discrimination Networks

Consider the analytical process of the sentence 'hasi wo kawa ni kakeru(someone constructs a bridge over the river)'[2], using the network shown in Figure 1. For the traversal of GDN, we provide correspondences between a constraint and a 'conditional identifier.' A conditional identifier consists of an identifier of a node followed by an 'if-clause' that represents a list of unsatisfied constraints. This correspondence between constraint and identifier means that if a constraint of a branch is satisfied, the nodes of the corresponding identifiers can be reached in the network conditionally(if the constraints in the if-clause are satisfied). An identifier with no if-clause means that a node of the identifier can be reached unconditionally. In Figure 1, identifiers of the nodes are given, and unsatisfied constraints in the if-clause are computed as a list of the constraints, other than the satisfied one, in the path from the root node to the reached node. For example, if the constraint *wo/human* is satisfied, the network can be traversed downward to the node of the corresponding identifier 131 if the constraint *ga/human* is satisfied.

Now we informally describe the analytical process of the above sentence, in which a necessary constraint *ga/human* is not obtained(ellipsis of the postpositional phrase corresponding to 'someone') and the order of the obtained constraints is irregular(order flexibility of postpositional phrases). Figure 2 shows a 'state' transition which represents the discrimination process. A state is represented in the form of a conditional identifier. The initial state(in which no constraints are obtained) is 1(the identifier of the root node). After the phrase 'hasi wo(a bridge)' is analyzed, the state is computed as follows, with the current state(the initial state) and a set of conditional identifiers {121 *if ga/animal*, 1322 *if ga/human* & *ni/place*} corresponding to the constraint satisfied by the phrase:

> Both 121 and 1322 include 1 as a prefix-numerical string, so the longer strings 121 and 1322 are returned[3]. Because the current state has no if-clause, the if-clause of the next state becomes the same as the if-clause of the conditional identifiers corresponding to the obtained constraint. Therefore, the next state becomes {121 *if ga/animal*, 1322 *if ga/human* & *ni/place*}.

Next, the phrase 'kawa ni(over the river)' satisfies the constraint *ni/place*, which corresponds to a set of the conditional identifiers

[2]The postpositional phrase corresponding to 'someone' is omitted in the original Japanese sentence.

[3]As shown in Figure 1, identifiers of mutually reachable nodes in the network are in a prefix-numerical string relation with each other. If one node is reachable from the other, the identifier of the subordinate one is returned. This operation corresponds to a downward traversal of the network by the obtained constraints.

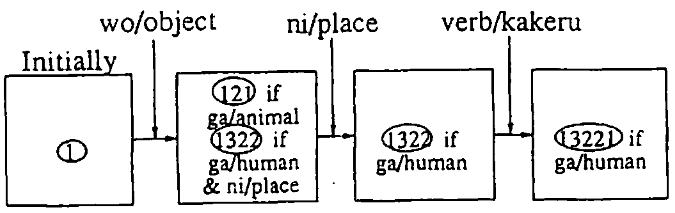

Figure 2: The discrimination process using GDN

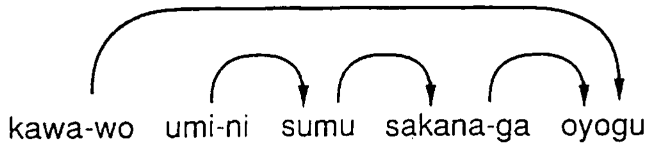

Figure 3: Dependency structure of the sentence 'kawa wo umi ni sumu sakana ga oyogu'

{123 *if ga/animal*, 132 *if ga/human*}. Any combinations of the identifiers but 132 and 1322 are not in a prefix relation, and so the analyses fail. Therefore, the resultant identifier is 1322 from identifiers 132 and 1322. The resultant if-clause is *if ga/human* because the constraint *ni/place* in the if-clause of the current state is obtained and removed from it.

Similarly, after the phrase 'kakeru(construct)' is analyzed, the final result of the traversal for the above case becomes 13221 *if ga/human*, which means that node 13221 is reachable if the constraint *ga/human* is obtained.

Note that the word sense of the verb 'kakeru,' which appears in the last of the sentence, is uniquely determined to be 'construct' among three word senses(corresponding to identifiers 111, 1222 and 13221) before the verb is analyzed. Further, the place of the missing phrase is detected as the unsatisfied constraint *ga/human*.

Compared with the ordinary approach for semantic analysis using case frames, where semantic checking(used to check whether a postpositional phrase('kawa ni') satisfies a selectional restriction of a verb) is not executed until a verb('kakeru') is analyzed, our approach executes earlier semantic checking before a verb is analyzed later in the sentence. Our approach executes semantic checking(word sense disambiguation) truly incrementally whenever a phrase is analyzed.

3 Incremental Dependency Analysis with Generalized Discrimination Networks

Unlike other languages which are based on phrase structure, Japanese sentences have a dependency structure among their phrases. A Japanese sentence can be segmented into a sequence of phrases called 'bunsetu.' In this paper the term 'phrase' is used to mean Japanese 'bunsetu.' A phrase is regarded as a minimal semantic element in a sentence. The internal structure of a phrase has been well studied in Japanese linguistics, so the details are not given here. We assume the results of past 'inner-phrase' analyses. The dependency relation is one in which a phrase modifies another in a sentence. The dependency structure for the sentence 'kawa wo umi ni sumu sakana ga oyogu(Fishes that inhabit the sea swim in the river)' is shown in Figure 3. In Figure 3, arrows from modifying phrases(modifiers) to modified ones(modifyees) represent dependency relations. For example, the postpositional phrase 'kawa wo(in the river)' modifies the verb 'oyogu(swim).'

In this section, we describe how these dependency relations among phrases are analyzed incrementally during the analytical

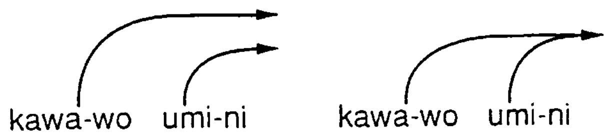

Figure 4: The intermediate state of incremental dependency analysis at the phrase 'umi ni'

process of a sentence, based on GDN-represented case frames.

3.1 The Process of Incremental Dependency Analysis

Incremental dependency analysis is to determine dependency relations between two phrases incrementally as a sentence is analyzed from left to right. The intermediate state of incremental dependency analysis (after a phrase is analyzed) is represented as a set of stacks[4] of arrows which pass over or start from the phrase and do not end. The stack indicates a set of potential dependency relations in which modifyees are not determined[5]. Multiple stacks might exist because of the ambiguity. For example, the state at the time after the phrase 'umi ni' is analyzed in the sentence 'kawa wo umi ni sumu sakana ga oyogu' is shown in Figure 4. The left stack is for the structure in which two phrases('kawa wo' and 'umi ni') modify a different modifyee, and the right stack is for that in which two phrases modify the same modifyee[6].

Dependency analysis operations act on each stack of the current state whenever a phrase is analyzed. There are two types of operations; those for treating the phrase as a modifyee and those for treating the phrase as a modifier. We call them 'In phase' and 'Out phase' operations, respectively. We provide the following In and Out phase operations, which are illustrated in Figure 5:

- In

  pass no phrases modify the phrase. The current stack is handed to the Out phase unchanged.

  pop phrases(an arrow) on the stack top of the current stack modify the phrase. The element of the stack top is popped and the stack is handed to the Out phase.

- Out

  merge an arrow from the phrase is merged with the arrow of the stack top, regarding the phrase as modifying the same modifyee as the set of phrases indicated by the arrow.

  push a new arrow is generated from the phrase and is pushed on the stack top.

As is clear from the above descriptions, both operations in the phases can be non-deterministically executed, unless knowledge useful for disambiguation is taken into account. Therefore, the ambiguity of dependency relations occurs in the analytical process and causes multiple stacks, as shown in Figure 4. Although our approach resolves the ambiguity as early and as much as possible using syntactic and semantic knowledge described in subsequent subsections, we also pack a set of stacks if possible. As in Tomita's algorithm[24, 18], we make a set of stacks structured in the tree form in cases where stack tops are common. This tree-structured stack avoids redundant computation as much as possible. For example, two stacks

[4]We use a stack representation to naturally take into account the 'no crossing principle' in Japanese, as described in the next subsection.

[5]Fixed dependency relations are stored differently as a part of intermediate semantic representation for a sentence.

[6]This is the meaning of the merged arrow, as described below.

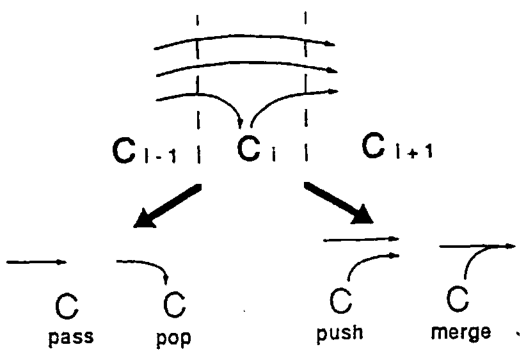

Figure 5: Dependency Analysis Operations

(Mod, Type)	An Example	Usage
(n,n)	Hanako no(Hanako's)	Hanako no hon(book) =Hanako's book
(n,v)	hon wo(book)	hon wo katta(bought) =bought a book
(v,n)	katta(bought)	katta hon(book) =the book which someone bought
(v,v)	katte(bought)	katte kaetta(came home) =bought something and came home

Figure 6: Classification of phrases

```
[e,d,c,b,a]
[e,b,a]
```

are made into a tree-structured stack

```
[e,[d,c,b,a],
   [b,a]]
```

because both stacks' tops are 'e.' In this way, tree-structured stack avoids redundant computation for two stack tops. In addition, because of this tree-structuring the push operation need be executed only once for a set of stacks, because the pushed element makes the stack top common for all stacks and they can be tree-structured into one stack.

3.2 Syntactic Knowledge for Structural Disambiguation

In this paper we use only syntactic and semantic knowledge for incremental structural disambiguation. Most of this knowledge is used to check the applicability of the above operations. Syntactic constraints for Japanese dependency relations are as follows:

1. Each phrase, except the last one in a sentence, must modify one succeeding phrase.
2. No dependency relations may cross one another[7].
3. Each phrase has two types for dependency relations: its own type and a type of the phrase that it can modify. The types are n(for noun/postpositional phrases) and v(for verb phrases). Therefore, phrases are divided into four classes, based on their type(*Type*) and their possible modifyee's type(*Mod*), as shown in Figure 6. For a phrase to modify the other one, the possible modifyee's type of the modifier must be equivalent to the type of the modifyee.

Constraint 1 cannot be fully checked until the end of a sentence. Constraint 2 can be naturally incorporated by representing the state of dependency analysis as a stack, because if we use only operations on the stack top no crossing occurs. Constraint 3 can be used to check whether the stack top can be popped in the pop operation. Further, arrows from phrases of the same class can only be merged in the Out phase. In this paper, for the sake of expository clarity, and because we present only a set of case frames as other knowledge useful for structural disambiguation, we treat only phrases of class (n,v) and (v,n), that is, dependency relations between noun(postpositional) phrases and verb phrases. Thus, classes of phrases are not explicitly shown, because the meaning of phrases expressed in English will reveal them. We think other classes of phrases can be treated similarly in our framework if other useful knowledge is incorporated into it.

We do not use grammar rules for dependency analysis of Japanese syntax for the reason that we do not think it is so useful in resolving the ambiguity of Japanese dependency relations, because the relatively free word order and widespread ellipsis in Japanese might cause more ambiguity in grammar rule application during grammar-based dependency analysis. Instead, we use mainly semantic knowledge(case frames) for structural disambiguation, but also some syntactic knowledge. Next, we describe how GDN-represented case frames resolve dependency ambiguity incrementally during the analytical process.

3.3 Earlier Semantic Checking with Generalized Discrimination Networks

In analyzing dependency relations incrementally, semantic checking needs to be executed during the analytical process, in which semantic appropriateness of a potential dependency relation(that is, whether a modifier satisfies a selectional restriction of a modifyee) is checked and structural ambiguity is resolved. In the ordinary integrated approach, semantic checking between a modifier and a modifyee is not executed until both are analyzed. That is, it is executed only in pop operation. So this approach determines that the phrases 'kawa wo(in the river)' and 'sakana ga(fishes)' modify the verb 'oyogu(swim)' in Figure 3 at the time when the verb is analyzed.

As described in the last section, however, GDN enables earlier semantic checking between a postpositional phrase and a verb whenever a postpositional phrase appears in a sentence. GDN also enables earlier semantic checking of a dependency relation between a modifier and a modifyee(before the modifyee appears), and rejection of semantically inappropriate dependency relations as soon as two modifiers that might modify the same modifyee are analyzed.

To perform this, we introduced the 'merge' operation, which combines arrows from multiple modifiers(as shown in Figure 5), regarding the multiple modifiers as modifying the same modifyee. For example, in Figure 4, arrows from the phrases 'kawa wo' and 'umi ni' are merged when the phrase 'umi ni' is analyzed, regarding the two phrases as modifying the same modifyee.

The hypothesis that two modifiers might modify the same modifyee is made as soon as they are analyzed. They are checked for semantic appropriateness in terms of case frames, and the semantically inappropriate one is rejected at this time. Using GDN for representing a set of case frames, semantic checking of the hypothesis is naturally realized by the further traversal of GDN from the current state(the result of the traversal using the former modifier) using the latter one, as described below.

As described in the last section, incremental semantic analysis using GDN corresponds to a downward traversal of the network from the current state(the result of the traversal using the previously analyzed phrases) using information of the newly analyzed phrase. This traversal hypothesizes that the phrase modifies the same verb as the previously analyzed ones because the further traversal from the current state using the phrase means that the phrase is checked for semantical appropriateness using the same case frame as before.

Therefore, the hypothesis that two modifiers modify the same modifyee naturally corresponds to the trial of the further traversal from the current state(corresponding to the previous modifier) using the newly analyzed modifier. If the traversal succeeds, the

[7]Exceptional sentences are known to exist for this constraint, but we now observe it strictly.

hypothesis is proved semantically appropriate. Failure means that the hypothesis is semantically inappropriate, and the merge operation fails.

As a result, the discrimination state of GDN makes the element of the stack. In the merge operation, the result of the further traversal from the state of the stack top using the newly analyzed phrase makes the new stack top. In the push operation, the result of the traversal from the root node makes the element of the stack top.

For example, in the sentence 'kawa wo umi ni sumu sakana ga oyogu,' when the phrase 'umi ni' is analyzed, the stack top is state 122 *if ga/animal*, which corresponds to the analysis result of the phrase 'kawa wo.' The further traversal from the current state using the constraint 'ni/place,' which the phrase 'umi ni' satisfies, fails because the identifier of the current state is not in a prefix relation with the identifiers 123 and 132 corresponding to the constraint. Thus, the merge operation fails, and the potential dependency relation in which the phrases 'kawa wo' and 'umi ni' modify the same phrase can be rejected. As a result of the disambiguation, the stack only remains for the structure in which the phrases 'kawa wo' and 'umi ni' modify a different phrase, such as the left one in Figure 4.

Similarly, in the pop operation, the appropriateness of dependency relations is checked in terms of the traversal of GDN. In this case, the traversal from the current node(the analysis result of all modifiers) using information of the modifyee completes the checking of the appropriateness of all dependency relations between modifiers and the modifyee.

3.4 Heuristics for the Depth of Embedding

As described in the preceding subsections, syntactic and semantic constraints can be applied to merge and pop operations, and inappropriate candidates of dependency relations can be rejected. No constraints, however, are applied to pass and push operations, and so the depth of stacks might become deeper[8] and the number of stacks might become bigger, no matter how the constraints earlier rejected inappropriate candidates. Stack depth is considered the depth of center embedding of a sentence, and so we think we may decide the maximum allowable depth of embedding and restrict the stack depth to that limit, pruning stacks whose depth exceeds the limit. The maximal number of allowable center embedding has been estimated in psycholinguistic researches[8, 11]. Moreover, where dependency relations between noun(postpositional) and verb phrases are concerned, we think the maximal number could be estimated based on the average number of verb appearances in a sentence.

Further, as described in section 3.1, a set of stacks is packed in tree-structured form, and the push operation is executed only once when a phrase is analyzed.

3.5 An Example

In this subsection, we illustrate the process of incremental dependency analysis with GDN using the sentence 'kawa wo umi ni sumu sakana ga oyogu(Fishes that inhabit the sea swim in the river).' We use GDN in Figure 1.

First, the phrase 'kawa wo(in the river)' is analyzed. The initial stack is empty, so the pop operation fails and only the pass operation is executed in the In phase. In the Out phase, because the stack is empty, the merge operation fails and only the push operation is executed. The result of the traversal from the root node using information from the phrase 'kawa wo,' that is, state 122 *if ga/animal*, is pushed on the stack top.

[122({kawa-wo})]

The intermediate state of the process is represented by a set of stacks. A stack is represented in the list form, where the stack top is the leftmost element. Each element of a stack consists of the state in GDN and the semantic representation for a fragment of a sentence in which fixed dependency relations are stored. For brevity, however, the if-clause in the state and the semantic representation are omitted here. The element of a stack is also accompanied by a set of phrases, which contributes to the state or the semantic representation of the element, where a fixed dependency relation is indicated by parentheses.

When the phrase 'umi ni(the sea)' is analyzed, the stack top already contains information about the phrase 'kawa wo.' In the In phase, the pop operation fails because of syntactic constraint 3 in section 3.2, that is, the type of possible modifier of the phrase 'umi ni' is not equivalent to the type of the stack top. And the stack is passed to the Out phase unchanged. In the Out phase, the merge operation fails between the analyzed phrase and the stack top because the further traversal from the current state(the state of the stack top) is impossible using information from the phrase 'umi ni,' though the syntactic constraint(that the class of the stack top is equivalent to that of the analyzed phrase) is satisfied. At this time, the phrases 'kawa wo' and 'umi ni' are judged to modify a different phrase. As a result, only the push operation is executed and the stack becomes as follows:

[{123,132}({umi-ni}), 122({kawa-wo})]

At the time of the phrase 'sumu(inhabit),' the pop operation succeeds in regards to the stack top(the phrase 'umi ni') because syntactic constraint 3 is satisfied and GDN can be further traversed from the current state(the state of the stack top) using information from the phrase 'sumu.' Therefore, the dependency relation between the phrases 'umi ni' and 'sumu' is fixed[9], and the stack top is popped. The pass operation is also executed, and so two stacks are handed to the Out phase. In the Out phase, the merge operation fails in regards to both stacks because of the class mismatch between the stack top(either the phrase 'umi ni' or 'kawa wo') and the phrase 'sumu.' Only the push operation is executed, and the set of stacks becomes as follows:

[1231, [({(umi-ni,sumu)}), 122({kawa-wo})],
[({sumu}), {123,132}({umi-ni}), 122({kawa-wo})]]

Two stacks that correspond to the choice of the pop or pass operation are tree-structured. However, the pushed element is not exactly common in regards to two stacks, because the if-clause in the state and the semantic representation of the pushed element depends on whether or not the pop operation is executed before the push operation. Here the difference is implicitly indicated by a set of phrases which accompanies a stack. The pushed element differs as follows: when the analyzed phrase(the modifyee) is a verb phrase, the pushed element differs in the if-clause in the state and the semantic representation, because the start node of the traversal is different between the state of the stack top and the root node, and then the dependency relation is fixed in the pop operation. When the analyzed phrase is a noun phrase, the state in the pushed element is common but the semantic representation for the modifier exists when the pop operation is executed, as shown below. Nonetheless, the identifier of the pushed element can be shared, and so whether the traversal from the pushed element is possible is checked only once, and redundant computation can be avoided. We think such imperfect tree-structuring is still satisfactory.

Hereafter we focus mainly on the explanation of a stack that leads to the semantically appropriate dependency structure(the meaningful interpretation of a sentence).

Next, at the time of the phrase 'sakana ga(fishes),' the pop operation succeeds because the further traversal from the state of the stack top using information from the phrase 'sakana ga' [10] succeeds, and the dependency relation is fixed(the case marker can

[8] The maximal depth might be the number of phrases in a sentence.

[9] We do not describe here the way to obtain the semantic representation.

[10] When a verb phrase modifies a noun phrase, a case marker of the modifyee('sakana') to the modifier('sumu') is omitted, and any case marker is supplemented and the traversal is attempted.

be guessed to be 'ga' as a by-product of the traversal). The pass operation is also executed. In the Out phase, in regards to the stacks after the pop operation, both the push and merge operations succeed. The merge operation succeeds between the phrases 'kawa wo' and 'sakana ga' because both syntactic and semantic constraints(the traversal of GDN) are satisfied. The following set of stacks is obtained(the 'meaningful' stack is second from the bottom):

```
[[ 12, [({sakana-ga}), 1231, [({(umi-ni,sumu)}), 122({kawa-wo})],
                             [({sumu}), {123,132}({umi-ni}),
                                        122({kawa-wo})]],
    [({((umi-ni,sumu),sakana-ga)}), 122({kawa-wo})],
    [({(sumu,sakana-ga)}), {123,132}({umi-ni}), 122({kawa-wo})]],
 [ 122({kawa-wo,((umi-ni,sumu),sakana-ga)})],
 [ 123({umi-ni,(sumu,sakana-ga)}), 122({kawa-wo})]]
```

Note that at this time we have six stacks but only three stack tops by tree-structuring. Moreover, assuming the maximum depth of the stack to be three, one of the stacks can be rejected.

At the last phrase 'oyogu(swim),' the pop operation succeeds for all three stack tops. However, five stacks which are not empty after the pop operation are rejected because syntactic constraint 1 in section 3.2 requires a stack, which indicates a set of modifiers whose modifyees are not determined, to be empty at the end of a sentence. We now obtain the dependency structure shown in Figure 3.

4 Conclusion

We proposed an incremental disambiguation approach for Japanese structural ambiguity. In our approach, semantic knowledge(case frames) is used directly to resolve structural ambiguity incrementally during the analytical process. We also showed that by using a generalized discrimination network(GDN) as a representation of a set of case frames, we can resolve ambiguity earlier than by ordinary integrated-and-incremental approaches. GDN enables to resolve word sense and structural ambiguities truly incrementally whenever a phrase appears in a sentence.

We first proposed GDN for incremental word sense disambiguation and in this paper we showed that it is also useful for incremental structural disambiguation. We think our GDN-based framework is well suited for incremental Japanese analysis in that

- GDN copes well with the Japanese language's flexible word order and widespread ellipsis, as described in section 2.2;
- GDN enables earlier structural and word sense disambiguation as a representation of case frames.

Moreover, our truly incremental disambiguation approach is considered useful not only for real-time natural language processing, but also for resolving ambiguities at lower levels when this syntactic and semantic analysis module is integrated with lower level analysis modules, such as morphological analysis or speech recognition, because our earlier decision making based on syntactic and semantic knowledge contributes to the further suppression of ambiguities at lower levels which might be caused if the time of disambiguation is delayed as in the ordinary incremental approach.

Lytinen[14] proposed the semantics-first approach for integrated analysis. In this approach, semantic analysis is executed first; semantic relationships between constituents are proposed, and then grammar rules are searched. Note that the semantics-first approach never considers meaningless parse trees. This is because semantics does not notice any syntactic relationships between two constituents which are semantically anomalous. In the syntax-first approach, however, all possible parse trees must be constructed and some of them are immediately rejected as semantically anomalous. We think [14] supports our approach, as it provides circumstantial evidence that the average case complexity is empirically suggested worse for the syntax-first approach than for the semantics-first one.

We described our model for incremental dependency analysis in quite simplified form, and we have to consider more factors to implement a practical Japanese dependency analyzer, such as follows:

- The difference between obligatory and optional cases. GDN represents a set of case frames that consist of a set of obligatory case slots, and we must consider how to treat optional cases in our framework;
- The treatment of missing case markers. Case markers are sometimes missed in postpositional phrases such as the one which includes the topic marker 'ha.' The missing case markers might cause ambiguity during the traversal of GDN because multiple constraints might be satisfied by such phrases. We should treat such postpositional phrases in a different way.

We will also have to take contextual knowledge into account in the future.

References

[1] G. Adriaens and S.L. Small. Word expert parsing revisited in a cognitive science perspective. In S.L. Small, G.W. Cottrell, and M.K. Tanenhaus, editors, *Lexical Ambiguity Resolution : Perspectives from Psycholinguitics, Neuropsychology, and Artificial Intelligence*, pages 13–43. Morgan Kaufmann Publishers, 1988.

[2] R.J. Bobrow and B.L. Webber. Knowledge representation for syntactic/semantic processing. In *Proc. of the 1st National Conference on Artificial Intelligence*, pages 316–323, 1980.

[3] G. Brassard and P. Bratley. *Algorithmics*. Prentice Hall, 1988.

[4] E. Charniak, C.K. Riesbeck, and D.V. McDermott. *Artificial Intelligence Programming*. Lawrence Erlbaum Associates, 1980.

[5] K. Church and R. Patil. Coping with syntactic ambiguity or how to put the block in the box on the table. *American Journal of Computational Linguistics*, 8(3-4):139–149, 1982.

[6] M. Dincbas. Constraints, logic programming and deductive databases. In *Proc. of the France-Japan Artificial Intelligence and Computer Science Symposium 86*, pages 1–27, 1986.

[7] T. Gunji. *Japanese Phrase Structure Grammar*. Reidel, 1987.

[8] K. Hasida. A constraint-based approach to linguistic performance. In *Proc. of the 13th International Conference on Computational Linguistics*, volume 3, pages 149–154, 1990.

[9] G. Hirst. Semantic interpretation and ambiguity. *Artificial Intelligence*, 34(2):131–177, 1988.

[10] P.S. Jacobs. Concretion: Assumption-based understanding. In *Proc. of the 12th International Conference on Computational Linguistics*, pages 270–274, 1988.

[11] P.N. Johnson-Laird. *Mental Models*. Cambridge University Press, 1983.

[12] S.L. Lytinen. Dynamically combining syntax and semantics in natural language processing. In *Proc. of the 5th National Conference on Artificial Intelligence*, pages 574–578, 1986.

[13] S.L. Lytinen. Are vague words ambiguous? In S.L. Small, G.W. Cottrell, and M.K. Tanenhaus, editors, *Lexical Ambiguity Resolution : Perspectives from Psycholinguitics, Neuropsychology, and Artificial Intelligence*, pages 109–128. Morgan Kaufmann Publishers, 1988.

[14] S.L. Lytinen. Semantics-first natural language processing. In *Proc. of the 9th National Conference on Artificial Intelligence*, pages 111–116, 1991.