IJCLA VOL. 4, NO. 1, JAN-JUN 2013, PP. 55–77
RECEIVED 26/11/12 ACCEPTED 11/01/13 FINAL 29/03/13

Structural Underspecification and Resolution within a Processing-oriented Grammar Formalism

TOHRU SERAKU

University of Oxford, UK

ABSTRACT

A challenge to modeling incrementality in language processing is posed by complex NPs in some verb-final languages, where a parser does not see whether a clause that a parser currently processes is part of a complex NP and how deeply it is embedded. These indeterminacies are handled by structural underspecification and resolution within Dynamic Syntax. This article points out that the previous implementation of the mechanism faces a formal problem of introducing indistinguishable nodes into the tree, and proposes a solution by letting a parser determine node-addresses flexibly. Concrete analyses are given to Japanese relatives as a case of complex NPs in verb-final languages.

KEYWORDS**:** *Dynamic Syntax, incrementality, Japanese, relative clauses*

1 Introduction

A central issue in recent processing studies is whether the incremental parsing thesis holds of verb-final languages. Despite initial negative suggestions [14], there has been a growing body of research pointing to a conclusion in which the answer is positive [6]. From a parser's point of view, particularly challenging are complex NPs (e.g. NP with a relative clause, NP with an appositive clause) in some verb-final

languages such as Japanese and Korean: a complex NP in these languages consists of a clause ending with a verb and a head noun following the clause. So, in processing a clause, a parser does not see in advance (a) whether the current clause is a main clause or part of a complex NP and, if it is part of a complex NP, (b) how deeply it is embedded.

These two indeterminacies are illustrated by the Japanese strings (1, 2, 3). First, as shown in (1), argument NPs in Japanese may be dropped when they are identifiable contextually. The parentheses in (1) indicate that *Mary-ga* and *hon-o* may be dropped.

(1) (*Mary-ga*) (*hon-o*) *ka-tta.*
(Mary-NOM) (book-ACC) buy-PAST
'Mary bought a book.'

In Japanese, a relative clause precedes a head noun. Thus, the relative clause *Mary-ga ka-tta* in (2) is identical to the string (1) if *hon-o* is dropped in (1).

(2) [[*Mary-ga* *ka-tta*] *hon*]-*wa* *omoshiroi.*
[[Mary-NOM buy-PAST] book]-TOP interesting
'A book which Mary bought is interesting.'

Note that the string (2) contains no morpheme that marks a relative clause.[1] Thus, a parser, which processes *Mary*, cannot see whether *Mary* belongs to a relative clause as in (2) or a matrix clause as in (1). Further, as demonstrated in (3), a parser, which has processed the complex NP string *Nai-ta otoko*, is still unable to see whether this complex NP belongs to a matrix clause or, as in (3), it is part of a larger complex NP.

(3) [[*Nai-ta* *otoko*]-*o* *nagusame-ta* *hito*]-*ga* *nige-ta*
[[cry-PAST man]-ACC comfort-PAST person]-NOM run.away-PAST
'A person who comforted a man who cried ran away.'

[1] It is reported that a verb in a relative clause in Japanese has a special intonation [13]. This intonational cue, however, is not available until a parser processes the verb *kau* (= 'buy') in (2). In Korean, the verbal suffix -*u*(*n*) indicates a relative clause [17], but, once again, this morphological cue is not available until a parser processes a verb.

An appropriate parser for Japanese must be flexible enough to accommodate these two indeterminacies.

A reasonable method of handling such indeterminacies is to introduce structural indeterminacies to trees. This idea is implemented within Dynamic Syntax (DS) [2, 8, 10] as structural underspecification and resolution. This is intuitively plausible, but, as will be pointed out, the previous analysis [2, 9, 13] ends up inducing indistinguishable nodes into the tree. This constitutes a rather serious problem because it overturns a principal basis for explaining diverse linguistic data (Greek clitics [3], Japanese clefts [16]) and it prevents the DS modeling of English dialogue [15] from being applied to Japanese dialogue. In short, complex NPs in verb-final languages such as Japanese offer a good test case for evaluating the DS formalism.

The aim of this article is to point out a formal problem that the extant DS treatment of complex NPs suffers from and to propose a solution by letting a parser determine node-addresses flexibly. The refined DS parser, it is argued, provides a more realistic model of language understanding in that a "look ahead" mechanism may be avoided and that intonational cues are more effectively utilized. To illustrate this point, the article examines Japanese relatives as a case of complex NPs in verb-final languages.

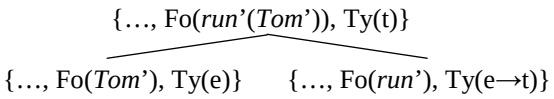
2 Dynamic Syntax

Dynamic Syntax (DS) is a grammar formalism that models knowledge of language; thus, DS is a theory of competence and regarded as generative grammar in the sense explicated by Noam Chomsky [4]. Unlike mainstream generative grammar, however, knowledge of language, or competence, is defined as a set of constraints on language performance, more specifically, the building-up of interpretation in context [2, 8, 10]. With such constraints, a parser processes a string of words left-to-right, and builds up semantic representation incrementally, without a separate level of syntactic structure: "syntax" within DS is no more than a set of constraints on how to build up a semantic tree progressively in context.

2.1 *Trees and Tree Descriptions*

The aim of a parser is to construct a semantic tree that represents an interpretation of a string in context on the basis of word-by-word processing. Trees in DS are binary, an argument node being on the left and a functor node being on the right. Each node is decorated with a declarative unit, consisting of a formula and labels.[2] A formula is semantic content at a node, and labels indicate various properties of the content; one example of labels is a logical type, which indicates the combinatorial property of the content. A formula is represented with the predicate Fo, whose argument comes from D_{Fo} = {*Tom*', *run*', …}. Content of some lexical items is not an element in D_{Fo}; for instance, the content of *she* is a place-holding variable U, called "meta-variable", whose value is supplied contextually. A logical type is represented with the predicate Ty, whose argument comes from D_{Ty} = {e, t, e→t, …}. D_{Ty} is a finite set (for instance, it does not include a type for five-place predicates), and no operations are stipulated to generate types, such as type-lifting and composition of functors. For example, the parse of *Tom runs* gives rise to the semantic tree (4); for the sake of simplicity, tense is ignored throughout this article.

(4)



The notation "…" in each declarative unit indicates additional labels which are not explicitly shown here. Another example of labels is a decoration in LOFT (Logic Of Finite Trees [1]). This is a language to talk about trees, which enables a parser to describe the other nodes in the tree from the perspective of a current node. LOFT-operators are defined as follows. First, there are operators to model an immediate dominance relation: $\langle\downarrow_0\rangle$ is for argument daughters and $\langle\downarrow_1\rangle$ for functor daughters. For instance, $\langle\downarrow_0\rangle$Ty(e) indicates that the argument daughter is of type-e; this label holds at the top node in (4). The inverses, $\langle\uparrow_0\rangle$ and $\langle\uparrow_1\rangle$, describe a mother node from the perspective of an argument node and from the perspective of a functor node, respectively. Second, operators with the Kleene star * model a dominance relation. $\langle\downarrow_*\rangle$ describes a node somewhere below the current node, together with its inverse, $\langle\uparrow_*\rangle$. These operators may describe a node at an arbitrary distance, but not across a "LINK" relation. Third, the "down" operator <D> and the "up" operator <U> model the weakest relation and may describe a node across a "LINK" relation. Finally, <L> and its inverse $\langle L^{-1}\rangle$ describe a node within

[2] Formally, DS structure is represented by a set of declarative units, where their relations are governed by LOFT (Logic Of Finite Tree) [1].

another structure that is LINKed from/to a current node. (For LINK relations, see Section 2.4.)

Another type of label is a node identifier, Tn(a), where Tn is a tree-node predicate. If a node is annotated with Tn(a), Tn(a0) indicates its argument daughter, and Tn(a1) indicates its functor daughter. A root node is marked by Tn(0), its argument daughter being by Tn(00) and its functor daughter being by Tn(01). Thus, the declarative unit at the root node in (4) is more precisely as in (5).

(5) {…, Tn(0), $<\downarrow_0>$Tn(00), $<\downarrow_1>$Tn(01), Fo(*run*'(*Tom*')), Ty(t), ◇}

This declarative unit contains a pointer ◇. In a DS tree, there always exists a single node that is under development. Such an active node is marked by a pointer ◇.

In non-final states, a tree is a "partial" structure in the sense that there exists a node decorated with a set of "requirements". A tree is said to be well-formed iff there are no outstanding requirements, and a string is said to be grammatical iff there exists a tree update that leads to a well-formed tree. A requirement is notated as the label ?α at a node, which requires that α will hold at the node. For instance, ?Ty(e) requires that the node will be decorated with Ty(e). Every node is introduced with requirements and every single tree-update is driven by some form of requirements. A parser runs a set of actions in order to satisfy requirements, as we shall see in the next sub-section.

2.2 *Actions for Tree Updates*

Trees grow progressively on the basis of left-to-right processing of a string in context without postulating an independent level of syntactic structure. The starting point of tree update is determined by the AXIOM, which introduces an initial node with the following declarative unit:

(6) {?Ty(t), ◇}

?Ty(t) requires that this node will be of type-t. This requirement corresponds to the parser's goal to build up an interpretation of a string: in this sense, tree growth is goal-directed. As a string is processed word-by-word, the initial node becomes increasingly richer: it is updated gradually and monotonically by a combination of general, lexical, and pragmatic actions.[3]

[3] In earlier works [10], the initial node is also annotated with Tn(a), an arbitrary node-address. Tn(a) is not articulated in recent works [2, 8], the

First, general actions are a set of actions that are stored in the DS system and that are not lexicalized. Each general action is formulated as a program, or a sequence of instructions to update a tree. Instructions are in the conditional format (7).

(7)
	IF	… ("…" is a condition to be met by a node highlighted by ◇)
	THEN	… ("…" is an action to be run if the condition is met)
	ELSE	… ("…" is an action to be run if the condition is not met)

The application of general actions is optional: a parser may run general actions at any time as long as the IF block is met by an active node. Examples of general actions will be presented in the next sub-section.

Second, lexical actions are a set of actions that are stored in the DS system and that are lexicalized. Lexical items also encode a sequence of instructions to update a tree, but lexical actions differ from general actions in terms of optionality: a package of actions encoded in a lexical item α must be run every time α is parsed. For instance, *inu* (= 'dog') encodes the macro of actions (8), where `put`(α) is a primitive action to decorate a node with α.

(8)
	IF	?Ty(e)
	THEN	`put`(Fo(ε, x, *inu*'(x)), Ty(e))
	ELSE	ABORT

Thus, (8) declares that if a current node is decorated with ?Ty(e), a parser annotates the node with Fo(ε, x, *inu*'(x)) and Ty(e). ABORT in the ELSE block ensures that this action cannot be executed unless the IF block is met. In (8), (ε, x, *inu*'(x)) is a type-e term that denotes a dog, expressed in Epsilon Calculus.[4] As shown in (1), argument NPs in

---

assumption being that the node introduced by the AXIOM is a root node of the whole tree. In Section 4, I shall modify the AXIOM so that it introduces a node that is underspecified for a node-address.

[4] Epsilon Calculus is a formal study of arbitrary names in natural deduction in Predicate Logic, proposed by David Hilbert. Every quantified NP is mapped onto an epsilon term, a type-e term defined as a triple: an operator, a variable, and a restrictor. In the case of (ε, x, *inu*'(x)), the existential operator ε binds the variable x that is restricted by the predicate *inu*'. This term stands for an arbitrary witness of the Predicate Logic formula ∃x.*inu*'(x). Since quantified NPs are uniformly analyzed as type-e terms, a quantified NP at an object position is handled without assuming type-shifting or quantifier movement [5]. A scope relation is expressed in a scope statement, where each term is in a dependency relation to others. This statement is constructed gradually as quantified NPs are parsed. Once a complete statement arises,

Japanese may be dropped. Thus, verbs encode a macro of actions to build up a propositional skeleton with argument slots. If NPs are dropped, such slots are contextually assigned content; if NPs have been processed, such slots collapse with the nodes that have been created by the parse of these NPs (cf. Section 3).

Third, pragmatic actions are a set of actions whose schematic rule-structures are stored in the DS system but whose execution involves pragmatic inference. A case of pragmatic actions pertinent to the present article is SUBSTITUTION, which saturates a meta-variable. For instance, the parse of *he* puts a meta-variable Fo(U_{MALE}) at a node, with a requirement that the node will be annotated with a formula denoting a male. This requirement drives SUBSTITUTION, replacing the variable with a content denoting a male with reference to contextual factors. SUBSTITUTION resolves underspecification in content. This is a quite familiar process in linguistics, but DS assumes another, less familiar form of underspecification: underspecification of structural relation.

2.3 *Structural Underspecification and Resolution*

Within DS, a node may be initially unfixed and resolved later. There are three types of general actions to induce unfixed relations with different locality restrictions:

(9)
a. LOCAL *ADJUNCTION: to induce a node that is "locally" unfixed
b. *ADJUNCTION: to induce a node that is "non-locally" unfixed
c. GENERALIZED ADJUNCTION: to induce a node that is "globally" unfixed

These general actions may be run only if a pointer ◇ is at a type-t-requiring node; so, unfixed nodes are always hung from a type-t-requiring node.

First, LOCAL *ADJUNCTION induces an unfixed node that must be fixed within a local proposition. This node is decorated with $<\uparrow_0><\uparrow_{1*}>$?Ty(t). This means that if a pointer ◇ moves up from an argument node (and possibly keeps going through functor nodes), then a parser finds a type-t-requiring node. For instance, $<\uparrow_0><\uparrow_{1*}>$?Ty(t) may be $<\uparrow_0>$?Ty(t), $<\uparrow_0><\uparrow_1>$?Ty(t), $<\uparrow_0><\uparrow_1><\uparrow_1>$?Ty(t), and so on.

every term in a proposition is "evaluated": it reflects the full scope relation into the restrictor of that term. Since this evaluation process is not pertinent, it is disregarded in this article.

Given this restricted dominance relation, the node is fixed under the closest type-t-requiring node. If a pointer crosses a type-t-requiring node, the relation includes more than one $\langle\uparrow_0\rangle$, as in $\langle\uparrow_0\rangle\langle\uparrow_1\rangle\langle\uparrow_1\rangle\langle\uparrow_0\rangle$?Ty(t), which contradicts $\langle\uparrow_0\rangle\langle\uparrow_{1*}\rangle$?Ty(t). This unfixed relation is resolved by a case particle. For instance, the lexical action encoded in the nominative-case particle *ga* puts the label $\langle\uparrow_0\rangle$?Ty(t) at an unfixed node, fixing it as a subject node under the closest type-t-requiring node.

Second, *ADJUNCTION induces an unfixed node that may be resolved at any node as long as the unfixed relation does not cross a LINK relation. Such nodes are marked by $\langle\uparrow_*\rangle$?Ty(t), which ensures that a pointer may cross a type-t-requiring node. This non-local unfixed relation cannot be resolved lexically. For instance, the accusative-case particle *o* narrows down possible fixed positions to a set of object nodes, each under some type-t-requiring node, but it does not specify a unique position. However, this unfixed relation may be resolved by the general action UNIFICATION: ?Ty(α)-unfixed node unifies with a Ty(α)-fixed node, as a result of which the fixed node is annotated with the union of the two declarative units.

Third, GENERALIZED ADJUNCTION induces a node that is wholly unfixed (i.e. may be across a LINK boundary). This globally unfixed relation is modeled by decorating the unfixed node with <U>?Ty(t), where the "up" operator <U> models a dominance relation across a LINK relation, allowing a pointer ◊ to move up and to cross a LINK boundary (cf. Section 2.1). An unfixed node induced by GENERALIZED ADJUNCTION may not be resolved by the parse of case particles for the same reason as stated in the last paragraph.

2.4 *LINK Relations*

Within DS, two structures may be built up in tandem, one of which is LINKed to the other. LINK is a relation between two structures that share a formula, and it is used for modeling, among other things, relatives in the following manner: a parser builds up an adjunct structure and LINKs the top node of the adjunct structure to a fresh node in an emergent main structure; a parser enriches this fresh node with the content of the adjunct structure. In this course of LINK transitions there are two crucial steps.

First, the general action LINK INTRODUCTION induces a LINK relation between a top node in an adjunct structure and a new type-e-requiring node in an emergent main structure. From the perspective of a node in a main structure, the top node of an adjunct structure may be described by the operator <L> (cf. Section 2.1). So, the label <L>α at a node in a main structure declares that if a parser looks at a LINKed node in an adjunct structure, the LINKed node is annotated with α.

Given the inverse operator $<L^{-1}>$, the following relation holds: $<L^{-1}>Tn(a) \Leftrightarrow Tn(aL)$.

(10) LINK INTRODUCTION

IF Ty(t), <D>(Fo(α))
THEN `make`(<L^{-1}>); `go`(<L^{-1}>); `put`(?∃x.Fo(x[α]), ?Ty(e))
ELSE ABORT

In (10), `make` and `go` are primitive actions concerning a node creation and a pointer movement, respectively. The IF block requires that a current node be of type-t and that a node somewhere below this node be decorated with Fo(α), where α is an arbitrary type-e term.[5] The THEN block requires that, if the IF block is satisfied, a parser initiate an inverse LINK relation from the current node to a fresh node in an unfolding main structure, and decorate the node with the requirements: ?∃x.Fo(x[α]) and ?Ty(e). ?∃x.Fo(x[α]) requires that this node will be decorated with a term that contains α as a sub-term; this ensures that the two LINKed structures share a term α.

Second, the fresh node in an emergent main structure is decorated by a head noun, and enriched with the content of the adjunct structure. This enrichment process is formulated as the general action LINK EVALUATION.

(11) LINK EVALUATION

IF Ty(e), Fo(ε, y, φ(y))
THEN IF <L>(Fo(ψ[(ε, x, P(x))]))
THEN `put`(Fo(ε, y, φ(y)&ψ[y/(ε, x, P(x))]))
ELSE ABORT
ELSE ABORT

(ε, y, φ(y)) is the content of a head noun, and ψ is the content of a relative clause, where (ε, x, P(x)) is the content of a gap in the relative clause. A parser reflects ψ into the term (ε, y, φ(y)) as an additional restrictor by re-binding (ε, x, P(x)) in ψ with the variable y, as in (ε, y, φ(y)&ψ[y/(ε, x, P(x))]). As a consequence, this composite term denotes an entity that satisfies not only the description of the head noun but also the description of the relative clause.

[5] In the previous work [9], the operator with the Kleene-star $\downarrow_*$ (instead of the "down" operator <D>) was used. This article presents LINK INTRODUCTION by replacing $\downarrow_*$ with <D>. This is because Japanese relatives are not sensitive to islands, as will be pointed out in Section 4.4. The next section shows that, even if this modification is made, the present version of LINK INTRODUCTION is not adequate.

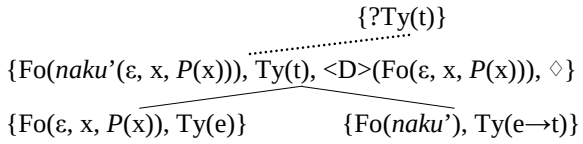
3 The Problem

Let us outline the previous DS account of Japanese relatives [2, 9, 13]. Consider (12), where the head noun *otoko* (= 'man') is preceded by the relative clause *Nai-ta*.

(12) [*Nai-ta* *otoko*]-*ga* *nige-ta.*
[cry-PAST man]-NOM run.away-PAST
'A man who cried ran away.'

In this earlier view, the AXIOM induced the initial node (6). Since *naku* (= 'cry') may belong to an embedded structure of an arbitrary depth, a parser introduced a globally unfixed type-t-requiring node by running GENERALIZED ADJUNCTION. This unfixed relation is shown by the dotted line in (13). Under this node, a parser ran the lexical actions encoded in *naku*, constructing a propositional template with a subject slot. Since no argument NPs had been parsed, a parser annotated this subject slot with the term (ε, x, *P*(x)), where *P* is an abstract predicate.[6]

(13) Parsing *Nai-ta*[7]

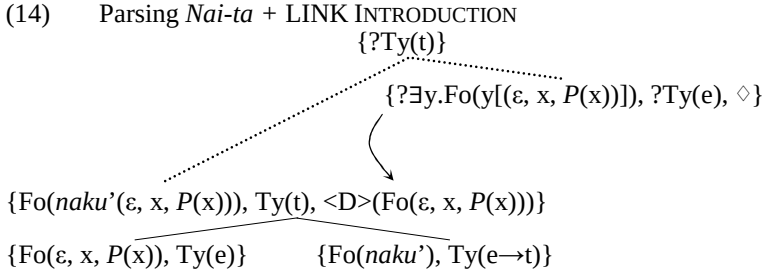


Then, in order to parse the head noun *otoko*, a parser executed LINK INTRODUCTION, initiating an inverse LINK relation from the type-t node to a new type-e-requiring node in an unfolding main structure, as shown by the curved arrow in (14). This node was also globally unfixed with respect to the root node since it might turn out to be part of a larger structure.

[6] In some previous accounts [2, 13], the node for a gap is notated as a variable. But this article follows a more recent account [9] in decorating the node with a term involving an abstract predicate P. However, this is just for expository purposes, and the analysis to be proposed in Section 4 may be recast in line with the previous accounts [2, 13].

[7] In this and subsequent trees, only relevant labels are expressed in declarative units.

(14) Parsing *Nai-ta* + LINK INTRODUCTION



The current node in (14) was then decorated by the parse of the head noun *otoko*, and enriched by LINK EVALUATION. The resulting declarative unit is shown in (15).

(15) {Fo(ε, y, *otoko'*(y)&*naku'*(y)), Ty(e), ◊}

This type-e node was fixed as a subject by the parse of the nominative-case particle *ga*. The parse of *nigeru* (= 'run away') then created a main structure, where the type-e node decorated with the declarative unit (15) was identified as a subject node.

Notice that the previous DS account ends up with two unfixed nodes of the same type hung from the same node, as shown by the two dotted lines in (14). That is, the AXIOM set out an initial node as the root node of the whole tree, and with respect to this root node two unfixed nodes were introduced for the relative clause and for the head noun. But multiplication of unfixed relations is not licit: in Logic Of Finite Trees [1], each node must be uniquely identifiable with respect to the other nodes in a tree; but if two unfixed nodes with the same locality restriction were hung from the same node, they would be indistinguishable and cannot be uniquely defined in the tree.[8]

More than one unfixed node, however, may be hung from the same node if they are of different sorts. Recall that there are three types of locality restrictions on unfixed relations and that they differ in terms of where an unfixed node may be resolved (cf. Section 2.3). This means that if two unfixed nodes have different locality restrictions, they are distinguishable and may be introduced from the same node. In (14), however, the two unfixed relations are both globally unfixed and cannot be distinguished. Thus, the tree (14) is formally illegitimate, and

[8] "Structural underspecification and resolution" is formally similar to "functional uncertainty" within LFG [7], but it seems there is no LFG analogue of the unique-unfixed-node constraint; a functional uncertainty for FOCUS may have more than one solution if the value is a set, each member of the set being associated with different values [11] (Mary Dalrymple p.c.).

it is concluded that the previous DS account of Japanese relatives [2, 9, 13] is inadequate.

The problem of multiplying unfixed relations occurs generally in the DS treatment of complex NPs in verb-final languages such as Japanese and Korean. This is because a modifier (e.g. relative clause) in these languages precedes a head noun, and the head noun could be part of a larger complex NP. The challenge is how a parser processes complex NPs incrementally in these languages without multiplying unfixed relations with the same locality restriction.

4 Solution

This section proposes a solution to the problem raised in the last section. The heart of the proposal is to let a parser determine node-addresses flexibly. To this end, I shall drop the assumption that the AXIOM introduces a root node of the whole tree and that a head noun is processed with respect to this root node.

Firstly, the AXIOM is modified so that it introduces a node decorated with not only the type requirement ?Ty(t) but also the node-address requirement ?∃x.Tn(x), together with a place-holding variable for a node-address, as in Tn(U).

(16) AXIOM (modified)

{Tn(U), ?∃x.Tn(x), ?Ty(t), ◇}

The meta-variable U may be substituted with 0, in which case the node is identified as a root node. Alternatively, it may be substituted with an arbitrary constant "a", whose actual manifestation will be determined at a later step (cf. Section 4.1).[9]

Second, LINK INTRODUCTION is modified as in (17), where the essential point is that a node for a head noun is structurally underspecified with respect to a new type-t-requiring node. In plain English, (17) declares the following: if a node is of type-t and decorated with a proposition involving a term α, a parser initiates an inverse LINK relation from this propositional node to a type-e-requiring node; this type-e-requiring node is annotated with the requirement that this node will be annotated with a term containing α as a sub-term; a parser

[9] The use of meta-variables in modeling an underspecification of node-address is inspired by Ronnie Cann, and the use of arbitrary constants to saturate such meta-variables is suggested by Ruth Kempson. I am grateful for insightful discussions I have had with them.

structurally underspecifies this type-e-requiring node with respect to a new type-t-requiring node.[10]

(17) LINK INTRODUCTION (modified)

IF Ty(t), <D>(Fo(α))
THEN make(<L^{-1}>); go(<L^{-1}>); put(?∃x.Fo(x[α]), ?Ty(e));
make(<↑*>); go(<↑*>);
put(Tn(U), ?∃x.Tn(x), ?Ty(t)); go(<↓*>)
ELSE ABORT

4.1 *Illustration One: Simple Cases of Relatives*

For illustration, let us consider the simple case of relatives (12), repeated here as (18).

(18) [*Nai-ta* *otoko*]-*ga* *nige-ta.*
[cry-PAST man]-NOM run.away-PAST
'A man who cried ran away.'

An initial node is set out by the modified AXIOM. Unlike the previous DS account [2, 9, 13], a parser may process the relative clause *Nai-ta* directly under this initial node.

(19) Parsing *Nai-ta*

{Tn(U), ?∃x.Tn(x), Fo(*naku'*(ε, x, *P*(x))), Ty(t), <D>(Fo(ε, x, *P*(x))), ◇}

{Fo(ε, x, *P*(x)), Ty(e)} {Fo(*naku'*), Ty(e→t)}

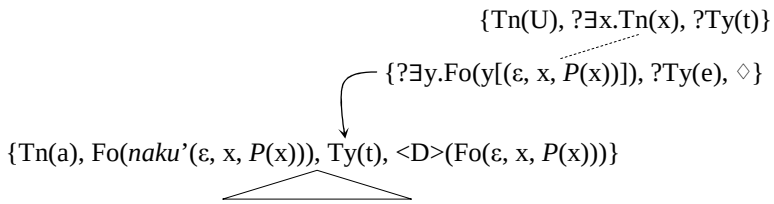
If the string ended here, a parser would identify the top node as a root node of the tree by saturating Tn(U) as Tn(0). In (18), however, *Nai-ta* is a relative clause.[11] Further, it is unknown at this point how deeply

[10] The locality restriction on this type-e-requiring unfixed node is the same as that imposed by *ADJUNCTION. This is because a head noun may be long-distance scrambled; for the detail, see a DS account of long-distance scrambling [2].

[11] When the verb *naku* appears in a relative clause, it has a special intonation [13] (cf. Section 1). This intonational cue cannot be made use of in the previous analysis [2, 9, 13], where GENERALIZED ADJUNCTION had to fire before the parse of relative clauses. By contrast, in my analysis, a parser does not run GENERALIZED ADJUNCTION, and may process a relative clause

the relative clause is embedded. Thus, a parser substitutes Tn(0) with Tn(a), where "a" is an arbitrary constant whose manifestation is worked out at a later step. A parser then runs LINK INTRODUCTION (17) in order to create a node for the head noun *otoko* (= 'man'). At this stage, the tree has been updated as in (20), where a triangle schematizes the internal structure.

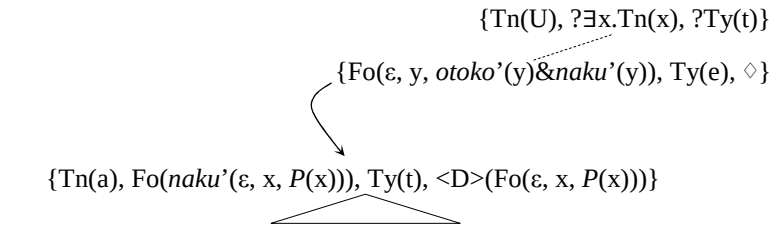
(20) Parsing *Nai-ta* + LINK INTRODUCTION



In (20), the current node is non-locally unfixed with respect to a new type-t-requiring node (cf. footnote 10). This non-local unfixed relation is shown by the dashed line.

Now that a type-e-requiring node is present, a parser may process the head noun *otoko* (= 'man'), decorating the node with content and type, and LINK EVALUATION then incorporates the content of the relative clause into the node.

(21) Parsing *Nai-ta otoko* + LINK EVALUATION

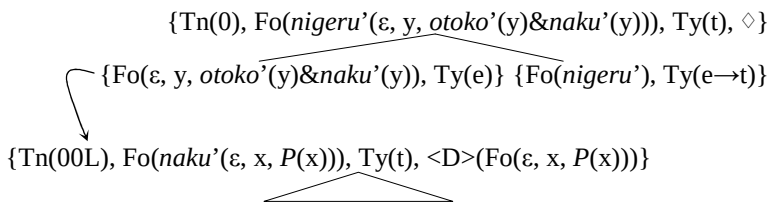


The rest of the process is as usual: the nominative-case particle *ga* marks the current node in (21) as a subject node, and the matrix verb *nigeru* (= 'run away') fleshes out a main propositional structure, where the subject node is identified as the node for the head noun. Finally, a parser saturates Tn(U) at the top node as Tn(0), ensuring that this is a root node. Once this node-address is specified, the actual manifestation

directly under an initial node set out by the AXIOM. The intonational cue then helps the parser to saturate Tn(U) at the initial node as Tn(a).

of the arbitrary constant "a" in Tn(a) is automatically explicated as Tn(00L).

(22) Parsing [*Nai-ta otoko*]-*ga nige-ta*



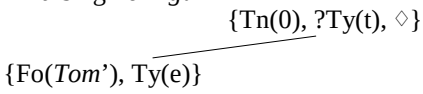
Notice that in the tree update above, no multiple unfixed nodes have been induced. This is because a node for a head noun is structurally underspecified with respect to a new type-t-requiring node that may be distinct from the root node of the whole tree.

The account is also applicable to (23), where, unlike (18), part of the matrix clause (i.e. *Tom-ga*) is processed before the relative clause *nai-ta*.

(23) *Tom-ga* [*nai-ta* *otoko*]-*o* *nagusame-ta.*
Tom-NOM [cry-PAST man]-ACC comfort-PAST
'Tom comforted a man who cried.'

Again, an initial node is set out by the AXIOM (16), and after LOCAL *ADJUNCTION creates a type-e-requiring unfixed node, *Tom* decorates the node with content and type and *ga* fixes it as a subject node. Since *Tom-ga* is part of a matrix clause, Tn(U) may be saturated as Tn(0), a node-address for a root node of the whole tree.

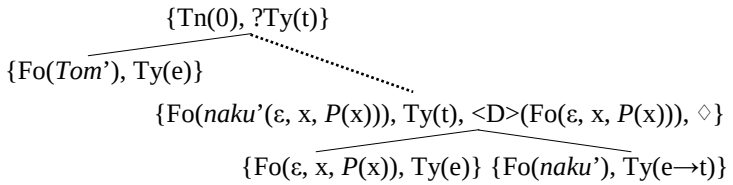
(24) Parsing *Tom-ga*



What comes next is *naku* (= 'cry'). A parser would develop the current propositional structure if *naku* were a matrix verb. In (23), an intonational break between *Tom-ga* and *nai-ta* signals that *naku* is an embedded verb, and a parser runs GENERALIZED ADJUNCTION to induce

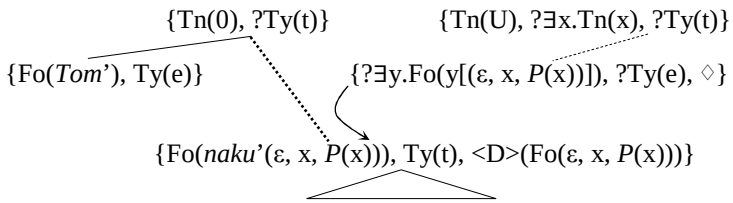
a globally unfixed type-t-requiring node.[12] The lexical actions encoded in *naku* flesh out this type-t-requiring node, providing a propositional template where a subject slot is decorated with the term (ε, x, *P*(x)), as usual.

(25) Parsing *Tom-ga nai-ta* + GENERALIZED ADJUNCTION



A parser runs LINK INTRODUCTION, initiating an inverse LINK relation to a type-e-requiring node that is unfixed with respect to a fresh type-t-requiring node.

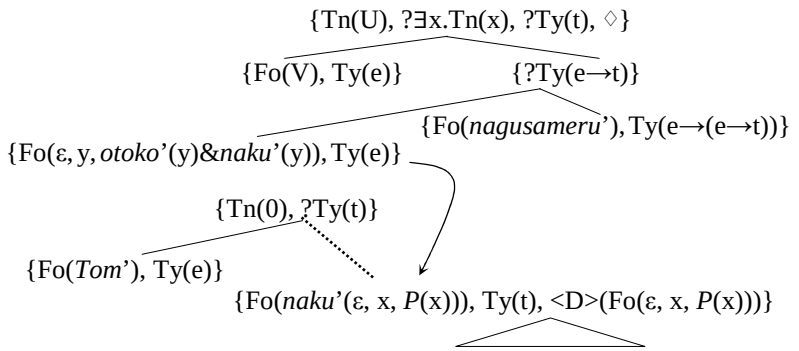
(26) Parsing *Tom-ga nai-ta* + LINK INTRODUCTION



The rest of the process is as usual: (a) the head noun *otoko* decorates the current node with content and type; (b) LINK EVALUATION incorporates the content of the relative clause into the node for the head noun; (c) the accusative-case particle *o* marks this node as an object under the type-t-requiring node; (d) the matrix verb *nagusameru* (= 'comfort') develops this type-t-requiring node by providing a propositional schema, where the object slot collapses with the node for the head noun and the subject slot is decorated with a meta-variable as in Fo(V).

[12] Here, *ADJUNCTION cannot fire because this general action requires that a current node not have any dominated node. In the present case, the current node has a dominated node (i.e. the node decorated with Fo(*Tom*')).

(27) Parsing *Tom-ga* [*nai-ta otoko*]-*o nagusame*



Now, a parser may saturate Tn(U) at the current node as Tn(0). As a result, this node is identified with the node set out by the AXIOM. Concomitantly, the node decorated with Fo(V) collapses with the node decorated with Fo(*Tom*'). (Recall that the dotted line indicates a globally unfixed relation, which may cross a LINK boundary.) For reasons of space, only the declarative unit at the root node is provided here as (28), which correctly represents the truth-conditional content of the string (23).

(28) {Tn(0), Fo(*nagusameru*'(ε, y, *otoko*'(y)&*naku*'(y))(*Tom*')), Ty(t), ◊}

4.2 *Illustration Two: Relative Clause Nesting*

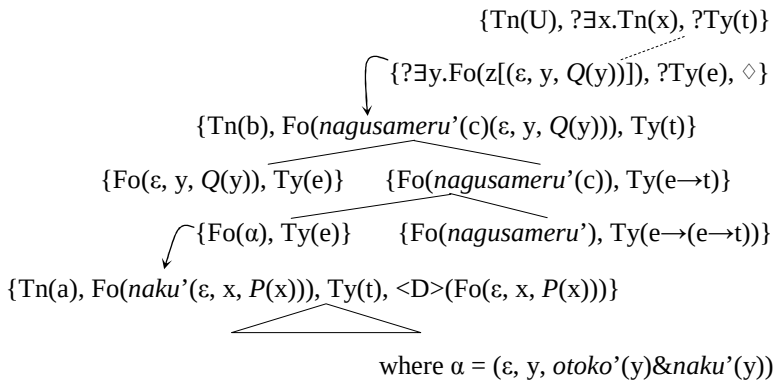
In the present account, a node for a head noun is structurally underspecified within a new propositional structure, and once this propositional structure is fully developed a parser may run LINK INTRODUCTION to induce another inverse LINK relation. Thus, the account naturally models successive relative clause embedding without failing to capture the left-to-right processing of the sequence. To illustrate, consider the case of relative clause nesting as in (29), where the complex NP *Nai-ta otoko* (= 'a man who cried') is part of the relative clause that modifies the head noun *hito* (= 'person').

(29) [[*Nai-ta otoko*]-*o nagusame-ta hito*]-*ga nige-ta*

[[cry-PAST man]-ACC comfort-PAST person]-NOM run.away-PAST
'A person who comforted a man who cried ran away.'

The parse of this string up to *otoko* (= 'man') gives rise to the same tree as (21). The unfixed node for *otoko* is marked as an object node by the accusative-case particle *o*. Then, *nagusameru* (= 'comfort') provides a propositional template; an object slot collapses with the node for *otoko*, and a subject slot is decorated with Fo(ε, y, Q(y)). At this stage, a parser may run LINK INTRODUCTION once again in order to parse the head noun *hito*, initiating another inverse LINK relation from the propositional node decorated with Fo(*nagusameru*'(c)(ε, y, *Q*(y))) to a type-e-requiring node.

(30) Parsing [*Nai-ta otoko*]-*o nagusame-ta* + LINK INTRODUCTION



where α = (ε, y, *otoko*'(y)&*naku*'(y))

The rest of the process is as usual: (a) the current node is decorated by the head noun *hito*; (b) LINK EVALUATION reflects the content of the relative clause into the node for *hito*; (c) the node for *hito* is marked as a subject by the nominative-case particle *ga* under a new type-t-requiring node; (d) this type-t-requiring node is fleshed out by *nigeru* (= 'run away'), where the subject slot collapses with the node for *hito*; (e) finally, Tn(U) at the top node is saturated as Tn(0), a node-address for a root node of the whole tree. The declarative unit at the root node is shown in (31).

(31) {Fo(*nigeru'*(ε, z, *hito*'(z)&*nagusameru*'(ε, y, *otoko*'(y)&*naku*'(y))(z))), Tn(0), Ty(t), ◇}

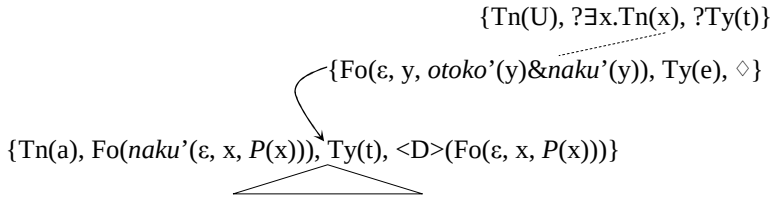
4.3 *Illustration Three: Scrambling of Complex NPs*

Japanese allows the permutation of arguments, so-called "scrambling". Thus, a head noun modified by a relative clause may be fronted: compare (23) with (32).

(32) [*Nai-ta* *otoko*]-*o* *Tom-ga* *nagusame-ta.*
[cry-PAST man]-ACC Tom-NOM comfort-PAST
'Tom comforted a man who cried.'

Scrambling is also dealt with by the present account. In (32), the parse of the relative clause *Nai-ta* provides a propositional template, where a subject slot is decorated with Fo(ε, x, *P*(x)), and LINK INTRODUCTION initiates an inverse LINK relation from this type-t node to a type-e-requiring unfixed node. This unfixed node is decorated by the head noun *otoko* (= 'man') and enriched by LINK EVALUATION.

(33) Parsing *Nai-ta otoko* + LINK EVALUATION



The current node is marked as an object node by the accusative-case particle *o*. Then, a pointer ◇ goes up to the type-t-requiring node, where *Tom-ga* induces a subject node and *nagusameru* (= 'comfort') creates a propositional template, where a subject slot collapses with the node for *Tom*. Finally, Tn(U) is saturated as Tn(0). The root node is decorated with (34); this declarative unit is exactly the same as the one in (28), which predicts that the string (32) is truth-conditionally equivalent to the string (23).

(34) {Tn(0), Fo(*nagusameru*'(ε, y, *otoko*'(y)&*naku*'(y))(*Tom*')), Ty(t), ◇}

4.4 *Illustration Four: Unbounded-Dependency and Island-Insensitivity*

Japanese relatives exhibit "unbounded-dependency": a head noun may be associated with a gap in a relative clause across a clause boundary.

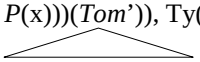
Thus, in (35), the head noun *otoko* (= 'man') is associated with the subject gap of *naku* (= 'cry') across the clause boundary *Tom-ga … i-tta*.

(35) [[*Tom-ga* [*nai-ta* *to*] *i-tta*] *otoko*]-*ga* *nige-ta.*
[[Tom-NOM [cry-PAST COMP] say-PAST] man]-NOM run.away-PAST
'A man who Tom said cried ran away.'

Prior to the head noun *otoko*, the parse of (35) leads to the semantic tree (36).

(36) Parsing *Tom-ga nai-ta to i-tta*

{Tn(a), Fo(*iu*'(*naku*'(ε, x, *P*(x)))(*Tom*')), Ty(t), <D>(Fo(ε, x, *P*(x))), ◊}



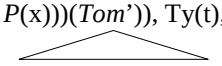
<D>(Fo(ε, x, *P*(x))) declares that the term (ε, x, *P*(x)) is found somewhere below the current node (possibly, across a LINK boundary; cf. Section 2.4.) Thus, the IF block of LINK INTRODUCTION is met and a parser initiates an inverse LINK relation to a type-e-requiring node, imposing a requirement that this node will be annotated with a term containing (ε, x, *P*(x)) as a sub-term. This type-e-requiring node is decorated by the head noun *otoko* (= 'man') and enriched by LINK EVALUATION.

(37) Parsing *Tom-ga nai-ta to i-tta otoko* + LINK EVALUATION

{Tn(U), ?∃x.Tn(x), ?Ty(t)}

{Fo(ε, y, *otoko*'(y)&*iu*'(*naku*'(y))(*Tom*')), Ty(e), ◊}

{Tn(a), Fo(*iu*'(*naku*'(ε, x, *P*(x)))(*Tom*')), Ty(t), <D>(Fo(ε, x, *P*(x)))}



The current node is marked as a subject by the nominative-case particle *ga*, and the matrix verb *nigeru* (= 'run away') creates a propositional schema where a subject slot collapses with the node for *otoko*. The root node in the final state is decorated with the declarative unit in (38).

(38) {Tn(0), Fo(*nigeru*'(ε, y, *otoko*'(y)&*iu*'(*naku*'(y))(*Tom*'))), Ty(t), ◊}

Given the lack of restrictions on where the term to be shared in the pair of LINKed structures is to be detected, it is predicted that Japanese relatives are not sensitive to "islands": that is, a head noun may be associated with a gap across an island boundary [12]. Thus, as shown in (39), the head noun *hito* (= 'man') may be associated with the subject gap of *kau* (= 'buy') even though this association crosses a complex NP island boundary that is formed by *Ka-tta tokei*.

(39) [[*Ka-tta* *tokei*]-*ga* *nisemonoda-tta* *hito*]-*ga* *nai-ta.*
[[buy-PAST watch]-NOM fake-PAST man]-NOM cry-PAST
'A man such that a watch he bought was a fake cried.'

One may wonder whether the use of the operator <D> in LINK INTRODUCTION is a stipulation, but there is a rationale. As has been assumed, verbs in Japanese provide a propositional skeleton where argument slots are decorated with meta-variables, and saturation of meta-variables is not structurally constrained. So, LINK INTRODUCTION is defined with the operator <D>, which models the weakest dominance relation, so that the label <D>(Fo(α)) and the primitive action put(?∃x.Fo(x[α]), ?Ty(e)) ensure that a term which will inhabit a node for a head noun may be found "deep inside" the relative clause structure (i.e. across a LINK relation).

5 Conclusion

This article has pointed out that the extant DS account of complex NPs in verb-final languages, especially Japanese relatives, is not adequate in that it multiplies unfixed relations with the same locality restriction. This formal problem disappears if node-addresses are specified flexibly. To this end, the AXIOM and LINK INTRODUCTION are modified and tested against a range of data posed by Japanese relatives.

In closing, it should be noted that the refined DS parser is more realistic than the past DS parser [2, 9, 13]. In the previous account, some sort of "look ahead" device needs to be assumed: that is, a parser must foresee that an incoming string has an embedded clause and run GENERALIZED ADJUNCTION before it starts to process the string. Although it was suggested that intonational cues were available to the parser, such cues would not obtain until a verb within a relative clause is parsed. By contrast, the parser proposed in this article may start to process a string without executing GENERALIZED ADJUNCTION in advance because an initial node set out by the AXIOM does not have to

be a root node of the whole tree and it may be developed by the parse of an embedded clause. This account makes use of intonational cues more effectively in order to saturate Tn(U), an underspecified node-address.[13]

ACKNOWLEDGMENTS. I have benefitted from constructive and encouraging exchanges with Ronnie Cann, David Cram, Mary Dalrymple, Ruth Kempson, and Jieun Kiaer. My deepest gratitude goes to Ruth Kempson, who provided me with a number of valuable comments on this work. Needless to say, the author alone is responsible for any inadequacies in the present article. This research was supported by the Clarendon Fund Scholarship, the Oxford-Kobe Scholarship, and the Sasakawa Fund Scholarship.

References

1. Blackburn, P., Meyer-Viol, W.: Linguistics, Logic, and Finite Trees. Bulletin of Interest Group of Pure and Applied Logics 2, 2-39 (1994)
2. Cann, R., Kempson, R., Marten, L.: The Dynamics of Language. Elsevier, Oxford (2005)
3. Chatzikyriakidis, S., Kempson, R.: Standard Modern and Pontic Greek person restrictions. Journal of Greek Linguistics 11, 127-166 (2011)
4. Chomsky, N.: Aspects of the Theory of Syntax. MIT Press, MA, Cambridge (1965)
5. Heim, I., Kratzer, A.: Semantics in Generative Grammar. Blackwell, Oxford (1998)
6. Kamide, Y.: Incrementality in Japanese sentence processing. In: Nakayama, M. et al. (eds.) The Handbook of East Asian Psycholinguistics, Vol. 2, Japanese. Cambridge University Press, Cambridge (2006)
7. Kaplan, R. M., Zaenen, A.: Long-distance dependencies, constituent structure, and functional uncertainty. In: Baltin, M., Kroch, A. (eds.)

[13] The mechanism proposed in this article has an implication for a cross-linguistic modeling of relatives in verb-final languages. The AXIOM remains invariant across languages, but it is possible that LINK INTRODUCTION is realized in a different form in different languages. For instance, in Korean, where the suffix -(*u*)*n* serves as a marker for relative clauses [17], the macro of actions in LINK INTRODUCTION may be lexically encoded in the relative clause marker. Further, the essential idea of the modified LINK INTRODUCTION is to underspecify a node for a head noun within a new propositional structure. This insight is quite general and may be made use of when we define general actions for other types of complex NPs.

Alternative Conceptions of Phrase Structure. University of Chicago Press, Chicago (1989)
8. Kempson, R., Gregoromichelaki, E., Howes, C.: The Dynamics of Lexical Interfaces. CSLI, Stanford (2011)
9. Kempson, R., Kurosawa, A.: At the syntax-pragmatics interface. In: Hoshi, H. (ed.) The Dynamics of Language Faculty. Kuroshio, Tokyo (2009)
10. Kempson, R., Meyer-Viol, W., Gabbay, D.: Dynamic Syntax. Blackwell, Oxford (2001)
11. King, T. H.: Focus domains and information structure. In: Butt, M., King, T. H. (eds.) The Proceedings of the LFG 97 Conference. CSLI, Stanford (1997)
12. Kuno, S.: The Structure of the Japanese Language. MIT Press, MA, Cambridge (1973)
13. Kurosawa, A.: On the Interaction of Syntax and Pragmatics. Ph.D. thesis, King's College London (2003)
14. Pritchett, B. L.: Grammatical Competence and Parsing Performance. University of Chicago Press, Chicago (1992)
15. Purver, M., Cann, R., Kempson, R.: Grammars as parsers. Research on Language and Computation 4, 289-326 (2006)
16. Seraku, T.: Multiple foci in Japanese clefts and the growth of semantic representation. In: Aloni, M. et al. (eds.) Lecture Notes in Computer Science 7218. Springer, Berlin (2012)
17. Sohn, H.: The Korean Language. Cambridge University Press, Cambridge (1999)

Tohru Seraku
St. Catherine's College,
University of Oxford,
Manor Road, Oxford, OX1 3UJ, UK.
E-mail: <tohru.seraku@stcatz.ox.ac.uk>