An Algorithm for Functional Uncertainty

Ronald M. KAPLAN and John T. MAXWELL III

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304 USA

Abstract: The formal device of functional uncertainty has been introduced into linguistic theory as a means of characterizing long-distance dependencies alternative to conventional phrase-structure based approaches. In this paper we briefly outline the uncertainty concept, and then present an algorithm for determining the satisfiability of acyclic grammatical descriptions containing uncertainty expressions and for synthesizing the grammatically relevant solutions to those descriptions.

1. Long-distance Dependencies and Functional Uncertainty

In most linguistic theories long-distance dependencies such as are found in topicalization and relative clause constructions are characterized in terms of categories and configurations of phrase-structure nodes. Kaplan and Zaenen (in press) have compared this kind of an analysis with one based on the functional organization of sentences, and suggest that the relevant generalizations are instead best stated in functional or predicate-argument terms. They defined and investigated a new formal device, called "functional uncertainty" that permits a functional statement of constraints on unbounded dependencies. In this paper, after reviewing their formal specification of functional uncertainty, we present an algorithm for determining the satisfiability of grammatical descriptions that incorporate uncertainty specifications and for synthesizing the smallest solutions to such descriptions.

/Kaplan and Zaenen (in press)/ started from an idea that /Kaplan and Bresnan 1982/ briefly considered but quickly rejected on mathematical and (/Kaplan and Zaenen/ suggest, mistaken) linguistic grounds. They observed that each of the possible underlying positions of an initial phrase could be specified in a simple equation locally associated with that phrase. In the topicalized sentence *Mary John telephoned yesterday*, the equation (in LFG notation) (↑ TOPIC)=(↑ OBJ) specifies that *Mary* is to be interpreted as the object of the predicate *telephoned*. In *Mary John claimed that Bill telephoned yesterday*, the appropriate equation is (↑ TOPIC)=(↑ COMP OBJ), indicating that *Mary* is still the object of *telephoned*, which because of subsequent words in the string is itself the complement (indicated by the function name COMP) of the top-level predicate *claim*. The sentence can obviously be extended by introducing additional complement predicates (*Mary John claimed that Bill said that .... that Henry telephoned yesterday*), for each of which some equation of the general form (↑ TOPIC)=(↑ COMP COMP .... OBJ) would be appropriate. The problem, of course, is that this is an infinite family of equations, and hence impossible to enumerate in a finite disjunction appearing on a particular rule of grammar. For this technical reason, Kaplan and Bresnan abandoned the possibility of specifying unbounded uncertainty directly in functional terms.

Kaplan and Zaenen reconsidered the general strategy that Kaplan and Bresnan began to explore. Instead of formulating uncertainty by an explicit disjunctive enumeration, however, they provided a formal specification, repeated here, that characterizes the family of equations as a whole. A characterization of a family of equations may be finitely represented in a grammar even though the family itself has an infinite number of members. They developed this notion from the elementary descriptive device in LFG, the functional-application expression. This has the following interpretation:

(1) $(f\ s)=v$ holds if and only if f is an f-structure, s is a symbol, and the pair $<s, v> \in f$.

An f-structure is a hierarchical finite function from symbols to either symbols, semantic forms, f-structures, or sets of f-structures, and a parenthetic expression thus denotes the value that a function takes for a particular symbol. This notation is straightforwardly extended to allow for strings of symbols, as illustrated in expressions such as (↑ COMP OBJ) above. If $x=sy$ is a string composed of an initial symbol s followed by a (possibly empty) suffix string y, then

(2) $(f\ x)\equiv((f\ s)\ y)$

$(f\ \varepsilon)\equiv f$, where ε is the empty string.

The crucial extension to handle unbounded uncertainty is to allow the argument position in these expressions to denote a set of strings. Suppose α is a (possibly infinite) set of symbol strings. Then Kaplan and Zaenen say that

(3) $(f\ \alpha)=v$ holds if and only if $((f\ s)\ \mathrm{Suff}(s,\alpha))=v$ for some symbol s, where $\mathrm{Suff}(s,\alpha)$ is the set of suffix strings y such that $sy \in \alpha$.

Thus, an equation with a string-set argument holds if it would hold for a string in the set that results from a sequence of left-to-right symbol choices. This kind of equation is trivially unsatisfiable if α denotes the empty set. If α is a finite set, this formulation is equivalent to a finite disjunction of equations over the strings in α. Passing from finite disjunction to existential quantification enables us to capture the intuition of unbounded uncertainty as an underspecification of exactly which choice of strings in α will be compatible with the functional information carried by the surrounding surface environment.

Kaplan and Zaenen of course imposed the further requirement that the membership of α be characterized in finite specifications. Specifically, for linguistic, mathematical, and computational reasons they required that α in fact be drawn from the class of *regular languages*. The characterization of uncertainty in a particular grammatical equation can then be stated as a regular expression over the vocabulary of grammatical function names. The infinite uncertainty for the topicalization example above, for example, can be specified by the equation (↑ TOPIC)=(↑ COMP* OBJ), involving the Kleene closure operator. A specification for a broader class of topicalization sentences might be (↑ TOPIC)=(↑ COMP* GF), where GF denotes the set of primitive grammatical functions {SUBJ, OBJ, OBJ2, XCOMP, ...}. Various restrictions on the domain over which these dependencies can operate--the equivalent of the so-called island constraints--can be easily formulated by constraining the uncertainty language in different ways. For example, the restriction for English and Icelandic that adjunct clauses are islands (Kaplan & Zaenen, in press) might be expressed with the equation (↑ TOPIC)=(↑ (GF-ADJ)* GF). One noteworthy consequence of this functional approach is that appropriate predicate-argument relations can be defined without relying on empty nodes or traces in constituent structure.

In the present paper we study the mathematical and computational properties of regular uncertainty. Specifically, we show that two important problems are decidable and present algorithms for computing their solutions. In LFG the f-structures assigned to a string are characterized by a *functional description* ('f-description'), a Boolean combination of equalities and set-membership assertions that acceptable f-structures must satisfy. We show first that the verification problem is decidable for any functional description that contains regular uncertainties. We then prove that the satisfiability problem is decidable for a linguistic interesting subset of descriptions, namely, those that characterize acyclic structures.

2. Verification

The verification problem is the problem of determining whether or not a given f-structure F satisfies a particular functional description for some assignment of elements of F to the variables in the description. This question is important in lexical-functional theory because the proper evaluation of LFG's constraint equations depends on it. It is easy to show that the verification problem for an f-description including an uncertainty such as $(f\,\alpha)=v$ is decidable if F is a noncyclic f-structure. If F is noncyclic, it contains only a finite number of function-application sequences and thus only a finite number of strings that might satisfy the uncertainty equation. The membership problem for the regular sets is decidable and each of those strings can therefore be tested to see whether it belongs to the uncertainty language, and if so, whether the uncertainty equation holds when the uncertainty is instantiated to that string. Alternatively, the set of application strings can be treated as a (finite) regular language that can be intersected with the uncertainty language to determine the set of strings (if any) for which the equation must be evaluated.

This alternative approach easily generalizes to the more complex situation in which the given f-structure contains cycles of applications. A cyclic F contains at least one element g that satisfies an equation of the form $(g\,y)=g$ for some string y. It thus involves an infinite number of function-application sequences and hence an infinite number of strings any of which might satisfy an uncertainty. But a finite-state machine can be constructed that accepts exactly the strings of attributes in these application sequences, for example, by using the Kasper/Rounds automaton model for f-structures (Kasper and Rounds, 1986). These strings thus form a regular language whose intersection with the uncertainty language is a regular set I containing all the strings for which the equation must be evaluated. If I is empty, the uncertainty is unsatisfiable. Otherwise, the set may be infinite, but if F satisfies the uncertainty equation for any string at all, we can show the equation will be satisfied when the uncertainty is instantiated to one of a finite number of short strings in I. Let n be the number of states in a minimum-state deterministic finite-state acceptor for I and suppose that the uncertainty equation holds for a string w in I whose length $|w|$ is greater than n. From the Pumping Lemma for regular sets we know there are strings x, y, and z such that $w=xyz$, $|y|\geq 1$, and for all $m\geq 0$ the string xy^mz is in I. But these latter strings can be application-sequences in F only if y picks out a cyclic path, so that $((f\,x)\,y)=(f\,x)$. Thus we have

$(f\,w)=v$ iff
$(f\,xyz)=v$ iff
$(((f\,x)\,y)\,z)=v$ iff
$((f\,x)\,z)=v$ iff
$(f\,xz)=v$

with xz shorter than w but still in I and hence in the uncertainty language α. If $|xz|$ is greater then n, this argument can be reapplied to find yet a shorter string that satisfies the uncertainty. Since w was a finite string to begin with, this process will eventually terminate with a satisfying string whose length is less than or equal to n. We can therefore determine whether or not the uncertainty holds by examining only a finite number of strings, namely, the strings in I whose length is bounded by n.

This argument can be translated to an efficient, practical solution to the verification problem by interleaving the intersection and testing steps. We enumerate common paths from the start-state of a minimum-state acceptor for α and from the f-structure denoted by f in F. In this traversal we keep track of the pairs of states and subsidiary f-structures we have encountered and avoid retraversing paths from a state/f-structure pair we have already visited. We then test the uncertainty condition against the f-structure values we reach along with final states in the α acceptor.

3. Satisfiability

It is more difficult to show that the satisfiability problem is decidable. Given a functional description, can it be determined that a structure satisfying all its conditions does in fact exist? For trivial descriptions consisting of a single uncertainty equation, the question is easy to answer. If the equation has an empty uncertainty language, containing no strings whatsoever, the description is unsatisfiable. Otherwise, it is satisfied by the f-structure that meets the requirements of any string freely chosen from the language, for instance, one of the shortest ones. For example, the description containing only $(f$ TOPIC$)=(f$ COMP* GF$)$ is obviously satisfiable because $(f$TOPIC$)=(f$SUBJ$)$ clearly has a model. There is a large class of nontrivial descriptions where the question is easy to answer for essentially the same reason. If we know that the satisfiability of the description is the same no matter which strings we choose from the (nonempty) uncertainty languages, we can instantiate the uncertainties with freely chosen strings and evaluate the resulting description with any satisfiability procedure (for example, ordinary attribute-value unification) that works on descriptions without uncertainties. The important point is that for descriptions in this class we only need to look at a single string from each uncertainty language, not all the strings it contains, to determine the satisfiability of the whole system. Particular models that satisfy the description will depend on the strings that instantiate the uncertainties, of course, but whether or not such models exist is independent of the strings we choose.

Not all descriptions have this desirable free-choice characteristic. If the description includes a conjunction of an uncertainty equation with another equation that defines a property of the same variable, the description may be satisfiable for some instantiations of the uncertainty but not for others. Suppose that the equation $(f$ TOPIC$)=(f$ COMP* GF$)$ is conjoined with the equations $(f$ COMP SUBJ NUM$)=$SG and $(f$ TOPIC NUM$)=$PL. This description is satisfiable on the string COMP COMP SUBJ but not on the shorter string COMP SUBJ because of the SG/PL inconsistency that arises. More generally, if two equations $(f\,\alpha)=v_\alpha$ and $(f\,\beta)=v_\beta$ are conjoined in a description and there are strings in α that share a common prefix with strings in β, then the description as a whole may be satisfiable for some strings but not for others. The choice of x from α and xy from β, for example, implies a further constraint on the values v_α and v_β: $(f\,x)=v_\alpha$ and $(f\,xy)=((f\,x)\,y)=v_\beta$ can hold only if $(v_\alpha\,y)=v_\beta$, and this may or may not be consistent with other equations for v_α.

We can formulate more precisely the conditions under which the uncertainties in a description may be freely instantiated without affecting satisfiability. For simplicity, in the analysis below we consider a particular string of one or more symbols in a non-uncertain application expression to be the trivial uncertainty language containing just that string. Also, although our satisfiability procedure is actually implemented within the general framework of a directed graph unification algorithm (the congruence closure method outlined by /Kaplan and Bresnan 1982/), we present it here as a formula rewriting system in the style of /Johnson 1987/. This enables us to abstract away from specific details of data and control structure which are irrelevant to the general line of argument. We begin with a few definitions. We say that

(5) A description is in *canonical form* if and only if
  (a) It is in disjunctive normal form,
  (b) Application expressions appear only as the left-sides of equations,
  (c) None of its uncertainty languages is the empty string ε, and
  (d) For any equation $f=g$ between two distinct variables, one of the variables appears in no other conjoined equation.

There is a simple algorithm for converting any description to a logically equivalent canonical form. First, every statement containing an application expression $(g\,\beta)$ not to the left of an equality is replaced by the conjunction of an equation $(g\,\beta)=h$, for h a new variable, with the statement formed by substituting h for $(g\,\beta)$ in the original statement. This step is iterated until no offending application expressions remain. The equation $(f\,\alpha)=(g\,\beta)$, for example, is replaced by the conjunction of equations $(f\,\alpha)=h\ \wedge\ (g\,\beta)=h$, and the membership statement $(g\,\beta)\in f$ becomes $h\in f\wedge(g\,\beta)=h$. Next, every equation of the form $(f\ \varepsilon)=v$ is replaced by the equation $f=v$ in accordance with the identity (2) above. The description is then

transformed to disjunctive normal form. Finally, for every equation of the form $f=g$ between two distinct variables both of which appear in other conjoined equations, all occurrences of g in those other equations are replaced by f. Each of these transformations preserves logical equivalence and the algorithm terminates after introducing only a finite number of new equations and variables and performing a finite number of substitutions.

Now let Σ be the alphabet of attributes in a description and define the set of first attributes in a language α as follows:

(5) $\text{First}(\alpha) \equiv \{s \text{ in } \Sigma \mid sz \text{ is in } \alpha \text{ for some string } z \text{ in } \Sigma^*\}$

Then we say that

(6) (a) Two application expressions $(f\ \alpha)$ and $(g\ \beta)$ are *free* if and only if
(i) f and g are distinct, or (ii) $\text{First}(\alpha)\cap\text{First}(\beta)=\emptyset$ and ε is in neither α nor β.

(b) Two equations are *free* if and only if their application expressions are pairwise free.

(c) A functional description is *free* if and only if it is in canonical form and all its conjoined equations are pairwise free.

If all the attribute strings on the same variable in a canonical description differ on their first element, there can be no shared prefixes. The free descriptions are thus exactly those whose satisfiability is not affected by different uncertainty instantiations.

3.1 Removing interactions

We attack the satisfiability problem by providing a procedure for transforming a functional description D to a logically equivalent but free description D' any of whose instantiations can be tested for satisfiability by traditional algorithms. We show that this procedure terminates for the descriptions that usually appear in linguistic grammars, namely, the descriptions whose minimal models are all acyclic. Although the procedure can detect that a description may have a cyclic minimal model, we cannot yet show that the procedure will always terminate with a correct answer if a cyclic specification interacts with an infinite uncertainty language.

The key ingredient of this procedure is a transformation that converts a conjunction of two equations that are not free into an equivalent finite disjunction of conjoined equations that are pairwise free. Consider the conjoined equations $(f\ \alpha)=v_\alpha$ and $(f\ \beta)=v_\beta$ for some value expressions v_α and v_β, where $(f\ \alpha)$ and $(f\ \beta)$ are not free. Strings x and y arbitrarily chosen from α and β, respectively, might be related in any of three significant ways: Either (a) x is a prefix of y (y is xy' for some string y'), (b) y is a prefix of x (x is yx'), or (c) x and y are identical up to some point and then diverge (x is zs_xx' and y is zs_yy' with symbol s_x distinct from s_y). Note that the possibility that x and y are identical strings is covered by both (a) and (b) with either y' or x' being empty, and that noninteracting strings fall into case (c) with z being empty. In each of these cases there is a logically equivalent reformulation involving either distinct variables or strings that share no first symbols:

(7) (a) x is a prefix of y:
$(fx)=v_\alpha \wedge (fxy')=v_\beta$ iff
$(fx)=v_\alpha \wedge ((fx)\ y')=v_\beta$ iff
$(fx)=v_\alpha \wedge (v_\alpha\ y')=v_\beta$ (by substituting v_α for (fx)

(b) y is a prefix of x:
$(fyx')=v_\alpha \wedge (fy)=v_\beta$ iff
$(v_\beta\ x)=v_\alpha \wedge (fy')=v_\beta$

(c) x and y have a (possibly empty) common prefix and then diverge:
$(fzs_xx')=v_\alpha \wedge (fzs_yy')=v_\beta$ iff
$(fz)=g \wedge (g\ s_xx')=v_\alpha \wedge (g\ s_yy')=v_\beta$
for g a new variable and symbols $s_x \neq s_y$

All ways in which the chosen strings can interact are covered by the disjunction of these reformulations. We observe that if these specific attribute strings are considered as trivial uncertainties and if v_α and v_β are distinct from f, the resulting equations in each case are pairwise free.

In this analysis we transfer the dependencies among chosen strings into different branches of a disjunction. Although we have reasoned so far only about specific strings, an analogous line of argument can be provided for families of strings in infinite uncertainty languages. The strings in these languages fall into a finite set of classes to which a similar case analysis applies. Let $<Q_\alpha, \delta_\alpha, q_\alpha, F_\alpha, \Sigma>$ be the states, transition function, start state, final states, and alphabet of a (perhaps nondeterministic) finite-state machine that accepts α and let $<Q_\beta, \delta_\beta, q_\beta, F_\beta, \Sigma>$ be an acceptor for β. Let δ^* be the usual extension of δ to strings in Σ^* and define

(8) $\text{Prefix}(\alpha,q) \equiv \{x \mid q \in \delta^*_\alpha(q_\alpha,x)\}$
(the prefixes of strings in α that lead to state q)

$$\text{Suffix}(\alpha,q) \equiv \begin{cases} \{x \mid \delta^*(q,x)\cap F_\alpha \neq \emptyset\} & \text{if } q\in Q_\alpha \\ \bigcup_{p\in q} \text{Suffix}(\alpha,p) & \text{if } q \subseteq Q_\alpha \end{cases}$$

(the suffixes of strings in α whose prefixes lead to states q)

and note that $\text{Prefix}(\alpha,q)$ and $\text{Suffix}(\alpha,q)$ are regular sets for all q in Q_α (since finite-state acceptors for them can easily be constructed from the acceptor for α). Further, every string in α belongs to the concatenation of $\text{Prefix}(\alpha,q)$ and $\text{Suffix}(\alpha,q)$ for some state q in Q_α. The prefixes of all strings in α thus belong to a finite number of languages $\text{Prefix}(\alpha,q)$, and every prefix that is shared between a string in α and a string in β also belongs to a finite number of classes formed by intersecting two of regular sets of this type. The common prefix languages fill the role of the prefix strings in the three-way analysis above. All interactions of the strings in α and β that lead through states q and r, respectively, are covered by the following possibilities:

(9) (a) Strings from α are prefixes of strings from β:
$(f\ \alpha\cap\text{Prefix}(\beta,r))=v_\alpha \wedge (v_\alpha\ \text{Suffix}(\beta,r))=v_\beta$

(b) Strings from β are prefixes of strings from α:
$(f\ \beta\cap\text{Prefix}(\alpha,q))=v_\beta \wedge (v_\beta\ \text{Suffix}(\alpha,q))=v_\alpha$

(c) Strings have a common prefix and then diverge on some s_α and s_β in Σ:
$(f\ \text{Prefix}(\alpha,q)\cap\text{Prefix}(\beta,r))=g_{q,r} \wedge$
$[(g_{q,r}\ s_\alpha\text{Suffix}(\alpha,\delta_\alpha(q,s_\alpha)))=v_\alpha \wedge$
$(g_{q,r}\ s_\beta\text{Suffix}(\beta,\delta_\beta(r,s_\beta)))=v_\beta]$

where the $g_{q,r}$ in (9c) is a new variable and $s_\alpha \neq s_\beta$. Taking the disjunction of these cases over the cross-product of states in Q_α and Q_β and pairs of distinct symbols in Σ, we define the following operator:

(10) $\text{Free}((f\ \alpha)=v_\alpha, (f\ \beta)=v_\beta) \equiv$

$$\bigvee_{\substack{q\in Q_\alpha \\ r\in Q_\beta}} \left[\begin{array}{ll} [(f\ \alpha\cap\text{Prefix}(\beta,r))=v_\alpha \wedge (v_\alpha\ \text{Suffix}(\beta,r))=v_\beta] & \text{(a)} \\ \vee\ [(f\ \beta\cap\text{Prefix}(\alpha,q))=v_\beta \wedge (v_\beta\ \textbf{Suffix}(\alpha,q))=v_\alpha] & \text{(b)} \\ \vee\ [(f\ \text{Prefix}(\alpha,q)\cap\text{Prefix}(\beta,r))=g_{q,r} \wedge & \\ \quad \bigvee_{\substack{s_\alpha,s_\beta\in\Sigma \\ s_\alpha\neq s_\beta}} [(g_{q,r}\ s_\alpha\text{Suffix}(\alpha,\delta_\alpha(q,s_\alpha)))=v_\alpha \wedge & \text{(c)} \\ \qquad (g_{q,r}\ s_\beta\text{Suffix}(\beta,\delta_\beta(r,s_\beta)))=v_\beta]\ ] & \end{array}\right]$$

This operator is the central component of our satisfiability procedure. It is easy to show that Free is truth-preserving in the sense that $\text{Free}((f\ \alpha)=v_\alpha, (f\ \beta)=v_\beta)$ is logically equivalent to the conjunction $(f\ \alpha)=v_\alpha \wedge (f\ \beta)=v_\beta$. Any strings x and y that satisfy the uncertainties in the conjunction must fall into one of the cases in (7). If $y=xy'$ applies (case 7a), we have $(fx)=v_\alpha \wedge (v_\alpha\ y')=v_\beta$. But x leads to some state r_x in Q_β and therefore belongs to $\text{Prefix}(\beta,r_x)$ while y' belongs to $\text{Suffix}(\beta,r_x)$. Thus, x satisfies $(f\ \alpha\cap\text{Prefix}(\beta,r_x))=v_\alpha$ and y' satisfies $(v_\alpha\ \text{Suffix}(\beta,r_x))=v_\beta$, and (10a) is satisfied for one of the r_x disjunctions. A symmetric argument goes through if case (7b) obtains.

Now suppose the strings diverge to s_xx' and s_yy' for distinct s_x and s_y after a common prefix z (case 7c) and that z leads to q in Q_α and r in Q_β. Then z belongs to $\text{Prefix}(\alpha,q)\cap\text{Prefix}(\beta,r)$ and satisfies the uncertainty $(f\ \ \text{Prefix}(\alpha,q)\cap\text{Prefix}(\beta,r))=g_{q,r}$. Since x' belongs to

Suffix(α,$\delta_\alpha(q,s_x)$)) and y' belongs to Suffix(β,$\delta_\beta(r,s_y)$)), the $g_{q,r}$ equations in the s_α,s_β disjunction also hold. Thus, if both original equations are satisfied, one of the disjunctions in (10) will also be satisfied. Conversely, if one of the disjunctions in (10) holds for some particular strings, then we can find other strings that satisfy both original equations. If $(f\ \alpha \cap \text{Prefix}(\beta,r)) = v_\alpha$ holds for some string x in α leading to state r in β's acceptor and $(v_\alpha\ \text{Suffix}(\beta,r)) = v_\beta$ holds for some string y' in Suffix(β,r), then $(f\ \alpha) = v_\alpha$ holds because x is in α and $(f\ \beta) = v_\beta$ holds because $((f\ x)\ y') = v_\beta = (f\ xy')$ and xy' is in β. The arguments for the other cases in (10) are similarly easy to construct. Thus, logical equivalence is established by reasoning back and forth between strings and languages and between strings and their prefixes and suffixes.

If the operands to Free are from a description in canonical form, then the canonical form of the result is a free description—all its conjoined equations are pairwise free. This is true whether or not the original equations were free, provided that the value expressions v_α and v_β are distinct from f (if either value was f, the original equations would have only cyclic models, a point we will return to below). In the first two cases in (10), the resulting equations are free because they have distinct variables (if neither v_α nor v_β is f). In the third case, the f equation is free of the other two because $g_{q,r}$ is a new variable, and the two $g_{q,r}$ equations are free because the first symbols of their uncertainties are distinct. In sum, the Free operator transforms a conjunction of two non-free equations into a logically equivalent formula whose canonical form is free.

The procedure for converting a description D to free form is now straightforward. The procedure has four simple steps:

(11) (a) Place D in canonical form.

(b) If all conjoined equations in D are pairwise free, stop. D is free.

(c) Pick a conjunction C in D with a pair of non-free equations $(f\ \alpha) = v_\alpha$ and $(f\ \beta) = v_\beta$, and replace C in D with the canonical form of its other equations conjoined with $\text{Free}(((f\ \alpha) = v_\alpha, (f\ \beta) = v_\beta)$

(d) Go to step (a).

3.2 Termination

If D has only acylic models, this procedure will terminate after a finite number of iterations. We argue that there are a certain number of ways in which the equations in each conjunction in D's canonical form can interact. Initially, for a conjunction C of N equations, the maximal number of non-free pairs is $N(N-1)/2$, on the worst-case assumption that every equation may potentially interact with every other equation. Suppose step (11c) is applied to two interacting equations in C. The result will be a disjunction of conjunctions each of which includes the remaining equations from C and new equations introduced by one of the cases in (10). In cases (10a) and (10b) the interaction is removed from the common variable of the two equations (f) and transferred to a new variable (either v_α or v_β). In case (10c), the interaction is actually removed from the system as a new variable is introduced. Since new variables are introduced only when an interaction is removed, the number of new variables is bounded. Thus each interaction is processed only a bounded number of times before it is either removed (10c) or transferred to a variable that it was previously associated with (10a, b). However, it can only transfer to a previous variable if the description has cyclic models. Suppose that f is reached again through a series of (10a,b) steps. Then there is a conjoined sequence of equations $(f\ \alpha) = v_\alpha$, $(v_\alpha\ \alpha_1) = v_{\alpha_1}$, ..., $(v_{\alpha_n}\ \alpha_{n+1}) = f$. But these can only be satisfied if there is some string x in $\alpha\alpha_1 \ldots \alpha_{n+1}$ such that $(f\ x) = f$ and this holds only of cyclic models. Since the number of variables introduced is bounded by the original number of possible interactions, all actual interactions in the system must eventually disappear either through the application of (10c) or by being transferred to a variable whose other equations it does not interact with.

As we argued above, the satisfiability of a free description can be determined by arbitrarily instantiating the residual uncertainties to particular strings and then applying any traditional satisfiability algorithm to the result. Given the Free operator and the procedure in (11), the satisfiability of an arbitrary acyclic description is thus decidable.

The possibility of nontermination with cyclic descriptions may or may not be a problem in linguistic practice. Although the formal system makes it easy to write descriptions of this sort, very few linguistic analyses have made use of them. The only example we are aware of involves modification structures (such as relative clauses) that both belong to the element they modify (the head) and also contain that element internally as an attribute value. But our procedure will in fact terminate in these sorts of cases. The difficulty with cycles comes from their interaction with infinite uncertainties. That is, the description may have cyclic models, but the cyclic specifications will not always lead to repeating variable transfers and nontermination. For example, if the cycle is required by an uncertainty that interacts with no other *infinite* uncertainty, the procedure will eventually terminate with a free description. This is what happens in the modification case, because the cycle involves a grammatical function (say RELCLAUSE or MOD) which belongs to no infinite uncertainty.

For cycles that are not of this type, there is a straightforward modification to the procedure in (11) that at least enables them to be detected. We maintain with each uncertainty a record of all the variables that it or any of its ancestors have been associated with, and recognize a potentially nonterminating cycle when the a transfer to a variable already in the set is attempted. If we terminate the procedure when this happens, assuming in effect that all subsequent disjunctions are unsatisfiable, we cannot be sure that all possible solutions will be accounted for and thus cannot guarantee the completeness of our procedure in the cyclic case. We can refine this strategy by recording and avoiding iteration over combinations of variables and uncertainty languages. We thus safely explore more of the solution possibilities but perhaps still not all of them. It is an open question whether or not there is a satisfiability procedure different from the one we have presented that terminates correctly in all cases. On the other hand, it is also not clear that potential solutions that might be lost through early termination are linguistically significant. Perhaps they should be excluded by definition, much as /Kaplan and Bresnan 1982/ excluded c-structure derivations with nonbranching dominance chains because of their linguistically uninteresting redundancies.

4. The Smallest Models

The satisfiability of a description in free form is independent of the choice of strings from its uncertainty languages, but of course different string choices result in different satisfying models for the description. An infinite number of strings can be chosen from even a very simple functional uncertainty such as $(f\ \text{COMP}^*\ \text{SUBJ}) = v$, and thus there are an infinite number of distinct possible models. This is reminiscent of the infinite number of models for descriptions with no uncertainties at all (just $(f\ \text{SUBJ}) = v$), but in this case the models are systematically related in the natural subsumption ordering on the f-structure lattice. There is one smallest structure; the others include the information it contains and thus satisfy the description. But they also include arbitrary amounts of additional information that the description does not call for. This is discussed by /Kaplan and Bresnan 1982/, where the subsumption-minimal structure is defined to be the grammatically relevant one.

The models corresponding to the choice of different strings from an infinite uncertainty are also systematically related to each other but on an metric that is orthogonal to the subsumption ordering. Again appealing to the Pumping Lemma for regular sets, strings that are longer than the number of states in an uncertainty's minimal-state finite-state acceptor include a substring that is accepted by some repeating sequence of transitions. Replicating this substring arbitrarily still yields a string in the uncertainty, so in a certain sense these replications contribute no new grammatically interesting

information. Since all the information is essentially contained in the shorter string that has no occurrence of this particular substring, we define this to be the grammatically relevant representative for the whole class. Thus a description with uncertainties has only a finite number of linguistically significant models, those that result from the finite disjunctions that are introduced in converting the description to free form and from choosing among the finite number of short strings in the residual uncertainties.

5. Performance Considerations

We have outlined a general, abstract procedure for solving uncertainty descriptions, making the smallest number of assumptions about the details of its operation. The efficiency of any implementation will depend in large measure in just how details of data structure and explicit computational control are fixed.

There are a number of obvious optimizations that can be made. First, although not required by the abstract procedure, performance will clearly be better if deterministic, minimal-state finite-state machines are used to represent the uncertainties. This reduces the size of the state cross-products, which is the leading term in the number of disjunctions that must be processed. Second, the cases in the Free operator are not mutually distinct: if identical strings belong to the two uncertainty languages, those would fall into both cases (a) and (b) and hence be processed twice with exactly equivalent results. The solution to this redundancy is to restrict one of the cases (say (a)) so that it only handles proper prefixes, consigning the identical strings to the other case. Third, when pairs of symbols are enumerated in the (c) case, there is obviously no point in even considering symbols that are in the alphabet but are not First symbols of the suffix uncertainties. This optimization is applied automatically if only the transitions leaving the start-states are enumerated and the finite-state machines are represented with partial transition functions pruned of transitions to failure states.

Fourth, a derivative uncertainty produced by the Free operator will sometimes be empty. Since equations with empty uncertainties are unsatisfiable by definition, this case should be detected and that disjunctive branch immediately discarded. Fifth, the same derivative suffix and prefix languages of a particular state may appear in pursuing different branches of the disjunction or processing different combinations of equations. Some computational advantage may be gained by saving the derivative finite-state machines in a cache associated with the states they are based on. Finally, successive iterations of the Free procedure may lead to transparent inconsistencies (an assertion of equality between two distinct symbols or equating a symbol to a variable that is also used as a function). It is important to detect these inconsistencies when they first appear and again discard the corresponding disjunctive branch. In fact, if this is done systematically, iterated application of the Free operator by itself simulates the effect of traditional unification algorithms, with variables corresponding to f-structures or nodes of a directed graph.

There are also some less obvious but also quite important performance considerations. What we have described is an equational rewriting system that is quite different from the usual recursive unification algorithm that operates on directed graph representations. Directed graph data structures index the information in the equations so that related structures are quickly accessible through the recursive control structure. Since our procedure does not depend for its correctness on the order in which interacting equations are chosen for processing, it ought to be easy to embed Free as a simple extension of a traditional algorithm. However, traditional unification algorithms do not deal with disjunction gracefully. In particular, they typically do not expect new disjunctive branches to arise during the course of a recursive invocation; this would require inserting a fork in the recursive control structure or saving a complete copy of the current computational context for each new disjunction. We avoid this awkwardness by postponing the processing of the functional uncertainty until all simple unifications are complete. Before performing a simple unification step, we remove from the data structures all uncertainties that need to be resolved and store them with a pointer to their containing structures on a queue or agenda of pending unifications. Uncertainty processing can be resumed at a later, more convenient time, after the simpler unifications have been completed. (Indeed, if one of the simpler unifications fails, the uncertainty may never be processed at all.) Waiting until simpler unifications are done means that no computational state has to be preserved; only data structures have to be copied to insure the independence of the various disjunctive paths.

We also note that as long as the machinery for postponing functional uncertainty for some amount of time is needed, it is often advantageous to postpone it even longer than is absolutely necessary. In particular, we found that if uncertainties are postponed until predicates (semantic form values for PRED attributes) are assigned to the f-structures they belong to, the number of cases that must be explored is dramatically reduced. This is because of the coherence condition that LFG imposes on f-structures with predicates: an f-structure with a predicate can only contain those governable functions that are explicitly mentioned by the predicate. Any other governable functions are considered unacceptable. Thus, if we wait until the predicate is identified, we need only consider the small number of governable attributes that any particular predicate allows, even though the initial attributes in an uncertainty may include the entire set of governable functions (SUBJ, OBJ, and various kinds of obliques and complements), and this may be quite large. The effect is to make the processing of long-distance dependencies sensitive to the subcategorization frame of the predicate; we have observed enormous overall performance improvements from applying this delay strategy. Note that in a left-to-right parsing model, the processing load therefore increases in relative clauses just after the predicate is seen, and this might have a variety of interesting psycholinguistic implications.

Finally, we observe that there is a specialization of the Free operator that applies when an uncertainty interacts with several non-uncertainty equations (equations whose attribute expressions have singleton First sets). Instead of separating one interaction from the uncertainty with each application of Free, the uncertainty is divided in a single step into a minimum number of disjunctive possibilities each of which interacts with just one of the other equations. The disjunction contains one branch for each symbol in the uncertainty's First set that is an initial attribute in one of the other equations, plus a single branch for all of the residual initial symbols:

$$(12)\quad (f\,a)=v \text{ iff } (f\,s_1\text{Suffix}(a,\delta(q_a,s_1)))=v \ldots \vee (f\,s_n\text{Suffix}(a,\delta(q_a,s_n)))=v \vee (f\,a\text{-}\{s_1,\ldots s_n\}\Sigma^*)=v$$

The statement of the generic Free algorithm (10) is simplified by considering specific attributes as trivial regular languages, but this suggests that complex finite-state machinery would be required to process them. This alternative works in the opposite direction: it reduces leading terms in an uncertainty to simple attributes before pursuing their interactions, so that efficient attribute matching routines of a normal unification procedure can be applied. This alternative has a second computational advantage. The generic algorithm unwinds the uncertainty one attribute at a time, constructing a residual regular set at each step, which is then processed against the other non-uncertain equations. The alternative processes them all at once, avoiding the construction of these intermediate residual languages. This is a very important optimization, since we found it to be the most common case when we embedded uncertainty resolution in our recursive unification algorithm.

Uncertainty specifications are a compact way of expressing a large number of disjunctive possibilities that are uncovered one by one as our procedure operates. It might seem that this is an extremely expensive descriptive device, one which should be avoided in favor of apparently simpler mechanisms. But the disjunctions that emerge from processing uncertainties are real: they represent independent grammatical possibilities that would require additional computational resources no matter how they were expressed. In theories in which long-distance dependencies are based on empty phase-structure nodes and implemented, for example, by gap-threading machinery, ATN HOLD lists, and the like, the exact location of these empty nodes is not signaled by any information directly visible in the sentence. This

increases the number of phrase-structure rules that can be applied. What we see as the computational cost of functional uncertainty shows up in these systems as additional resources needed for phrase-structure analysis and for functional evaluation of the larger number of trees that the phrase-structure component produces. Unlike phrasally-based specifications, functional uncertainties in LFG are defined on the same level of representation as the subcategorization restrictions that constrain how they can be resolved, which our coherence-delay strategy easily takes advantage of. But the fact remains that functional uncertainties do generate disjunctions, and thus strongly highlight the already perceived need for efficient disjunction-processing techniques if acceptable performance is to be achieved with LFG and related grammatical formalisms. Recent disjunction proposals by /Kasper 1987/ and /Eisele and Dörre 1988/ are important steps in the development of the necessary computational technology.

6. Conclusion

The notion of regular functional uncertainty thus has very nice mathematical properties. Our state-decomposition algorithm provides a very attractive method for resolving functional uncertainties as other phrasal and functional constraints are computed during the parse of a sentence. This algorithm expands the uncertainties incrementally, introducing at each point only as much disjunction as is necessary to avoid interactions with other functional information that has already been taken into account. We have recently added this algorithm and the functional uncertainty notation to our LFG Grammar Writer's Workbench, and we can now rigorously but easily test a wide range of linguistic hypotheses. We have also begun to investigate a number of other computational heuristics for the efficient, controlled expansion of uncertainty.

Kaplan and Zaenen (in press) first proposed the idea of functional uncertainty as sketched in this paper to account for the properties of long-distance dependencies within the LFG framework. In this framework, it has already shed new light on long-standing problems like island constraints (see, e.g., /Saiki 1985/ for an application to Japanese). But the notion is potentially of much wider use: first, it can be adapted to other unification grammar formalisms to handle facts of a similar nature; and second, it can be used to handle phenomena that are traditionally not thought of as falling into the same class as long-distance dependencies but that nevertheless seem to involve nonlocal uncertainty. A discussion of its application in the LFG framework to infinitival complements can be found in /Johnson 1986/ for Dutch and /Netter 1986/ for German; /Karttunen (In press)/ discusses how similar extensions to Categorial Unification Grammar (CUG) can account in a simple way for related facts in Finnish that would otherwise require type-raising. Halvorsen has suggested that scope ambiguities in semantic structures might also be characterized by this device.

Acknowledgements

Our understanding of the linguistic applications of functional uncertainty developed over a long period of time in discussions with Joan Bresnan, Kris Halvorsen and Annie Zaenen. Discussions with Mark Johnson helped us in the early formulations of the satisfiability procedure, and Bill Rounds assisted us in understanding the difficulties of the cyclic case. We are grateful for the invaluable assistance these colleagues have provided.

References

Eisele, A. and Dörre, J. 1988. Unification of disjunctive feature descriptions. *Proceeedings of the 26th Annual Meeting of the Association for Computational Linguistics.*

Johnson, M. 1986. *An LFG description of the double infinitive construction in Dutch and German,* CSLI report.

Johnson, M. 1987. *Attribute-value logic and the theory of grammar.* Unpublished doctoral dissertation, Stanford University.

Kaplan, R. M. and Bresnan, J. 1982. Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan (ed.), *The mental representation of grammatical relations.* Cambridge: MIT Press.

Kaplan, R. M. and Zaenen, A. In press. Long-distance dependencies, constituent structure, and functional uncertainty. In M. Baltin and A. Kroch (eds.), *Alternative Conceptions of Phrase Structure.* Chicago: Chicago University Press.

Karttunen, L. In press. Radical Lexicalism. In M. Baltin and A. Kroch (eds.), *Alternative Conceptions of Phrase Structure.* Chicago: Chicago University Press.

Kasper, R. 1987. A unification method for disjunctive feature descriptions. *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics.*

Kasper, R. and Rounds. W. 1986. A logical semantics for feature structures. *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics.*

Netter, K. 1986. Getting Things out of Order. *COLING 11.*

Saiki, M. 1985. On the coordination of gapped constituents in Japanese. *CLS* 21.