Features and Formulae

Mark Johnson*
Brown University

Feature structures are a representational device used in several current linguistic theories. This paper shows how these structures can be axiomatized in a decidable class of first-order logic, which can also be used to express constraints on these structures. Desirable properties, such as compactness and decidability, follow directly. Moreover, additional types of feature values, such as "set-valued" features, can be incorporated into the system simply by axiomatizing their properties.

1. Introduction

Many modern linguistic theories, such as Lexical-Functional Grammar (Bresnan 1982), Functional Unification Grammar (Kay 1985), Generalized Phrase Structure Grammar (Gazdar et al. 1985), Unification Categorial Grammar (Haddock et al. 1987), (Uszkoreit 1986), and Head-Driven Phrase Structure Grammar (Pollard and Sag 1987), replace the atomic categories of a context-free grammar with a "feature structure" that represents the syntactic and semantic properties of the phrase. These feature structures are specified indirectly in terms of constraints that they must satisfy. Lexical entries constrain the feature structures that can be associated with terminal nodes of the syntactic tree, and phrase structure rules simultaneously constrain the feature structures that can be associated with a parent node and its immediate descendants.

That is, lexical entries and syntactic rules used to construct a syntactic phrase structure tree all contribute constraints on the feature structures that appear as the labels on nodes in the syntactic tree. The tree is well formed if and only if all of these constraints are simultaneously satisfiable. Thus for the purposes of recognition a method for determining the *satisfiability* of such constraints is required; the precise nature of the satisfying feature structures (of which there may be infinitely many) is of secondary importance.[1]

A variety of different types of feature structures have been proposed in the literature, but most work on unification-based grammar has centered on a certain type of feature structure known as an *attribute-value structure*. The elements in an attribute-value structure come in two kinds: *constant elements* and *complex elements*. Constant elements are atomic entities with no internal structure: i.e. they have no attributes. Complex elements have zero or more attributes, whose values may be any other element in the structure, including a complex element. An element can be the value of zero, one or several attributes. Attributes are *partial*: it need not be the case that every attribute is defined for every complex element.

* Department of Cognitive and Linguistic Sciences, Providence, RI 02912 USA

1 The *validity problem* is also of interest, since it provides a way of "extracting information" about all of the satisfying feature structures. In the framework developed below, if ϕ is a formula representing a system of constraints and $\phi \rightarrow \theta$ is valid, then θ is a true description of every feature structure satisfying ϕ.

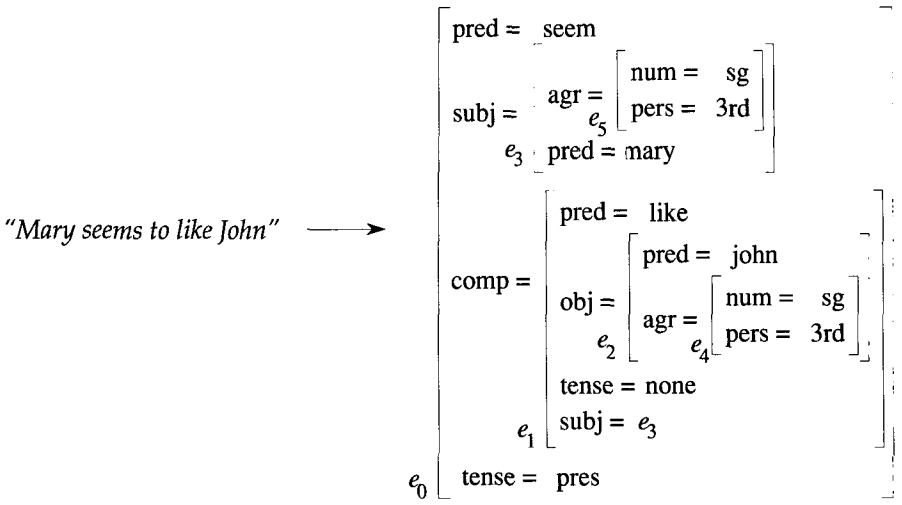

Figure 1
An attribute-value structure for *Mary seems to like John*

Example 1
Figure 1 depicts an attribute-value structure. The attribute-value element labeled e_0 in Figure 1 might be associated with the sentence *Mary seems to like John*.

The attribute-value structure depicted in Figure 1 contains six complex elements $e_0, \ldots, e_5$ and eight constant elements *seem*, *like*, *john*, *sg*, *3rd*, *mary*, *none*, and *pres*. The element e_0 is a complex attribute-value element with four attributes: *pred*, *subj*, *comp*, and *tense*: the order in which the attributes appear in the diagram is irrelevant. The value of its *pred* attribute is the constant *seem* (which abbreviates the relation denoted by the verb *seem*), and the value of its *tense* attribute is the constant element *pres* (which indicates that the clause is in the present tense). The values of the *subj* and *comp* attributes are the complex elements e_3 and e_1 (which represent the subject and the complement of the verb *seem*, respectively). The element e_3 also appears as the subject of e_1, indicating that *Mary* is also the (understood) subject of the verb *likes* as well.

The element e_1 is a complex attribute-value element with four attributes *pred*, *obj*, *subj*, and *tense*. The value of its *pred* attribute is the constant element *like* (which abbreviates the relation denoted by the verb *like*) and the value of its *tense* attribute is the constant element *none* (which indicates that the clause is untensed). The values of the attributes *obj* and *subj* of e_1 are the complex elements e_2 and e_3, respectively (which represent the subject and object of the clause). Both e_4 and e_5 have the same attributes *num* and *pers*, and the values of these attributes of e_4 are identical to the corresponding values of these attributes of e_5. Nevertheless, e_4 and e_5 are distinct elements.

An operation called *unification* plays an important role in most accounts of feature structures (Kay 1985; Shieber 1986). The unification operation "combines" or "merges" two elements into a single element that agrees with both of the original elements on the values of all of their defined sequences of attributes, so the unification of two complex elements requires the unification of the values of any attributes they have in common. The unification operation *fails* if it requires the unification of distinct

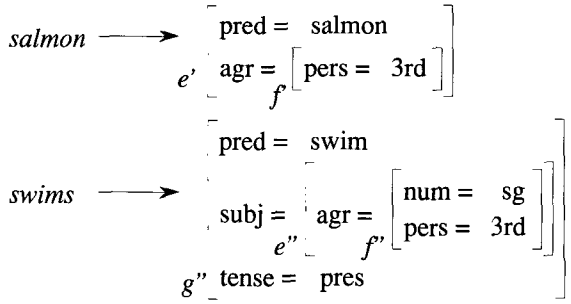

Figure 2
Lexical entries for *salmon* and *swim*

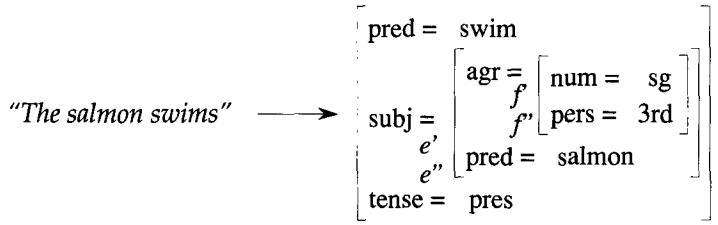

Figure 3
An example of attribute-value unification

constant elements (a *constant–constant clash*) or the unification of a constant element and a complex element (a *constant–complex clash*).

Example 2
A grammar might assign the attribute-value structures in Figure 2 to the NP *the salmon* and the VP *swims*, respectively. Note that e' does not have a *num* attribute, since *the salmon* can be either singular or plural, and that e'' does not have a *pred* attribute.

The attribute-value structure for the sentence (*the*) *salmon swims* is obtained by unifying e' and e'', which corresponds to identifying *salmon* as the subject of *swims*. The resulting element inherits the value of the *pred* attribute from e' and the value of the *num* attribute from e''. The unification of e' and e'' requires the unification of f' and f'' as well.

Although it might not be obvious from this simple example, a large number of syntactic constructions from a variety of natural languages can be described in such a unification-based framework (many of the analyses presented in Bresnan 1982 can be expressed in such a "pure" unification-based framework). Nevertheless, it is often convenient and sometimes necessary to extend the basic unification framework to include a wider variety of feature structures.

For example, "negative values" and "disjunctive values" allow grammars and lexical entries to be written much more succinctly, as the following examples show (based on Karttunen 1984).

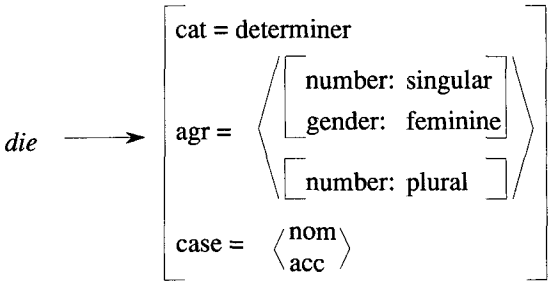

Figure 4
Disjunction in the lexical entry for *die*

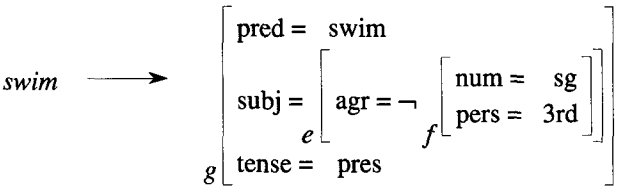

Figure 5
Negation in the lexical entry for *swim*

Example 3
In German the determiner *die* must have accusative or nominative case, and agrees with either feminine singular nouns or plural nouns of any gender. In a framework with disjunctive values only one lexical entry for *die* is required.[2]

Example 4
In the basic unification framework described above the tensed verb *swim* would require multiple lexical entries, since it agrees with first person, second person, and plural third person subjects; i.e., a subject with any agreement features other than third person singular. In a framework with "negative values" it requires only the single lexical entry in Figure 5, where "¬" identifies a "negative value."

As mentioned earlier, other kinds of feature structures besides attribute-value structures have been proposed in the literature. Johnson and Klein (1986) and Johnson and Kay (1990) show how "set-valued" features can be used to express Discourse Representation Theory (Kamp 1981) in a complex-feature based grammar formalism. The highly simplified example below is meant solely to show one way in which set-valued features can be used—no claims are made for its linguistic correctness.

2 "Disjunctive" features are depicted using angle brackets, since curly brackets are used in this paper to depict "set-valued" features. Below we reinterpret the "disjunctive" and "negative" features depicted in this example and the next as disjunctions and negations of constraints.

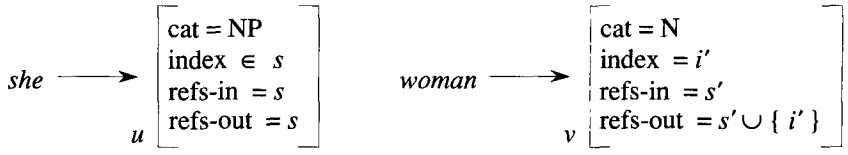

Figure 6
Set-values in the lexical entries for *she* and *woman*

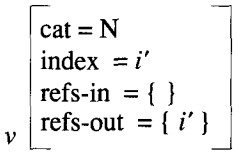

Figure 7
The result of unifying s' in Figure 6 with the empty set

Example 5
A naive theory of anaphoric dependencies between indefinite NPs and anaphoric pronouns can be constructed as follows. Each NP has an *index* attribute whose value is a "reference marker," and two NPs are coreferential iff they share the same reference marker.[3] Every feature structure associated with a node in the syntactic tree has attributes *refs-in* and *refs-out*, whose values are the sets of discourse entities available preceding and following this node, respectively. The grammar constrains the value of the *refs-out* attribute of an indefinite NP to be the union of its *refs-in* attribute and the singleton set containing the value of the NP's *index* attribute; this adds the NP's index to the set of available indices. Similarly, the grammar requires the values of a pronoun's *refs-in* and *refs-out* attributes to be identical, and that its *index* attribute be a member of the value of its *refs-in* attribute. This requires that the pronoun refer to an entity previously introduced into the discourse. In a framework with set values the lexical entries for (*a*) *woman* and *she* could be as seen in Figure 6.

Unifying the value s' of the *refs-in* attribute of the lexical entry for *woman* with the empty set (which corresponds to the empty discourse context) produces the feature structure depicted in Figure 7.

Further, the unification of the value of the *refs-out* attribute in Figure 7 with the value of the *refs-in* attribute of u in Figure 6 (the lexical entry for *she*), which corresponds to interpreting the pronoun as an anaphor within the context established by the single NP *a woman* produces the feature structure depicted in Figure 8.

Extending the possible feature structures beyond the basic attribute-value features complicates the basic unification operation, however. For example, Moshier and Rounds (1987) and Pereira (1987) point out that it is not obvious how to extend unification

3 Reference markers in DRT correspond approximately to the referential *indices* associated with NPs in GB theory.

$$u\begin{bmatrix} \text{cat} = \text{NP} \\ \text{index} = i' \\ \text{refs-in} = \{ i' \} \\ \text{refs-out} = \{ i' \} \end{bmatrix}$$

Figure 8
The result of unifying the value of the *refs-out* attribute of Figure 7 with *s* in Figure 6

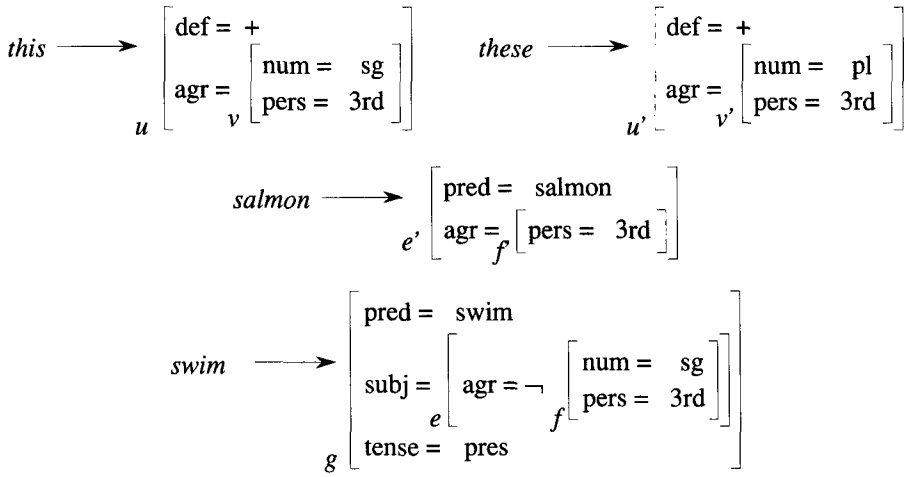

Figure 9
Feature structures demonstrating interaction of negative values and unification

to negative feature values; specifically, some apparently plausible extensions lose the associativity property of unification.

Example 6
Consider the feature structures in Figure 9, which might be assigned to the singular determiner *this*, the plural determiner *these*, the noun *salmon*, and the verb *swim* (the latter two structures are the same as those depicted in Figures 2 and 5). These can be used to analyze utterances such as *these salmon swim* and (the ill-formed utterance) *this salmon swim*, which involve the unification of u, e', and e or u', e', and e, respectively.

Suppose a negative value is interpreted as a *constraint* that a feature structure either satisfies or does not satisfy, and suppose further that in Figure 9 the negative feature constraint f is satisfied by the value f'. Then e' and e in Figure 9 unify, and moreover further unification of e' with either u' or u succeeds, undesirably in the latter case. (Reinterpreting the negative constraint f so that f' fails to satisfy it does not help, since the unification of e, e', and u' should succeed). On the other hand, we obtain the results we desire if e' is unified with u or u' before being unified with e. If e' is first unified with u, then f' is unified with v, and further unification of e' with e fails, since v does not satisfy f. If e' is first unified with u' then f' is unified with v' and further unification of e' with e succeeds, since v' does satisfy f. Thus under this interpretation

of negation and unification, the success or failure of a sequence of unifications depends on the order in which they are performed.[4]

2. Feature Structures and Function-free Formulae

These problems have generated a considerable body of work on the mathematical properties of feature structures and the constraints and operations that apply to them. Following Kaplan and Bresnan (1982), Pereira and Shieber (1984), Kasper and Rounds (1986, 1990), and Johnson (1988, 1990a) the constraints that determine the feature structures are regarded as formulae from a language for describing feature structures, rather than as feature structures themselves.

Disjunction and negation appear only in expressions from the description language, rather than as components of the feature structures that these expressions describe. Thus the lexical entries in the examples above will be interpreted as formulae that constrain the feature structures that can be associated with these lexical items in a syntactic tree, rather than the feature structures themselves. For example, the feature matrices depicted in Figures 2, 4–6, and 9 will be interpreted as graphical depictions of formulae expressing constraints on linguistic objects, rather than the linguistic objects that satisfy these constraints. This avoids any need to refer to "negative" or "disjunctive" objects as entities appearing in a feature structure.

As explained below, the familiar attribute-value "unification algorithm" can be interpreted as computing the atomic consequences of a purely conjunctive formula (where the graphs it operates on are data structures efficiently representing such formulae), and unification failure corresponds to the unsatisfiability of that conjunction (Kasper and Rounds 1990; Johnson 1988, 1990a; Pereira 1987).

The most widely known model of feature structures and constraint language is the one developed to explain disjunctive feature values by Kasper and Rounds (1986, 1990) and Kasper (1986, 1987). The Kasper–Rounds treatment resolves the difficulties in interpreting disjunctive values by developing a specialized language for expressing these constraints. Various proposals to extend the Kasper–Rounds approach to deal with negative feature values are described by Moshier and Rounds (1987), Moshier (1988), Kasper (1988), Dawar and Vijayashanker (1989, 1990), Langholm (1989); other extensions to this framework are discussed by Dörre and Rounds (1989), Smolka (1988, 1989), and Nebel and Smolka (1989); and Shieber (1989) discusses the integration of such feature systems into a variety of parsing algorithms.

One difficulty with this approach is that the constraint language is "custom built," so important properties, such as compactness and decidability, must be investigated from scratch. Moreover, it is often unclear if the treatment can be extended to handle other types of feature structures as well. Rounds (1988) proposes a model for set-valued features, but he does not provide a language for expressing constraints on such set-valued entities, or investigate the computational complexity of systems of such constraints.

This paper follows an alternative strategy suggested in Johnson (1990a): *axiomatize* the relevant properties of feature structures in some well-understood language (here first-order logic) and *translate* constraints on these structures into the same language.

4 It is possible to avoid these problems by augmenting feature structures with "inequality arcs," as was first proposed (to my knowledge) by Karttunen (1984) and discussed in Johnson (1990a), Johnson (in press) and pages 67–72 of Johnson (1988). However, it is hard to justify the existence of such arcs if feature structures are supposed to be *linguistic* objects (rather than data structures that represent formulae manipulated during the parsing process).

Thus the satisfiability problem for a set of constraints on feature structures is reduced to the satisfiability problem for the axioms conjoined with the translation of these constraints in the target language. Importantly, techniques used to determine satisfiability in the target language can be used to determine the satisfiability of the feature constraints as well. In this paper the properties of attribute-value structures and constraints on them are expressed in a *decidable class* of first-order formulae: this means that the satisfiability problem for such formulae, and hence the feature constraints that they express, is always decidable.

Of course, some linguistic analyses make use of feature structure constraint systems that can encode undecidable problems. For example, *subsumption* constraints, which are useful in the description of agreement phenomena in coordination constructions (Shieber 1989) can be used to encode undecidable problems, as Dörre and Rounds (1989) have shown. Clearly such constraints cannot be expressed in a decidable class, but often they can be axiomatized in other standard logics. Johnson (1991) shows how (positively occurring) subsumption constraints can be axiomatized in first-order logic, and sketches treatments of sort constraints and nonmonotonic devices such as ANY values (Kay 1985) and 'constraint equations' (Kaplan and Bresnan 1982) can be formalized in second-order logic using circumscription.

2.1 Axiomatizing Feature Structures with Function-Free Formulae

This section shows how the important properties of feature structures can be axiomatized using formulae from the *Schönfinkel-Bernays class*, which is the class of first-order formulae of the form

$$\exists x_1 \ldots x_n \forall y_1 \ldots y_n \varphi$$

where φ contains no function symbols or quantifiers. (Thus no existential quantifier can appear in the scope of a universal quantifier.) This class of formulae was chosen because it is both decidable (see e.g. Lewis and Papadimitriou 1981) and can express the quantification needed to describe the particular set operations proposed here, as well as a variety of other interesting types of feature structures and constraints. (For a general discussion of decidable classes see Gurevich 1976 and Dreben and Goldfarb 1979.) The next section shows how the various kinds of constraints on feature structures described above can be translated into this class of formulae, so any system of such feature constraints is decidable as well.

The elements of a feature structure, both complex and constant, constitute the domain of individuals in the intended interpretation. The attributes are binary relations over this domain.[5] We proceed by axiomatizing the conditions under which an interpretation corresponds to a well-formed feature structure, formulating them in essentially the same way as Smolka (1988, 1989) does.

The axiomatization begins by describing the properties of the constant elements of attribute-value structures. The attribute-value constants are the denotation of certain constant symbols of the language of first-order logic, but not all constants (of the first-order language) will denote attribute-value constants since it is convenient to have constants that denote other entities as well. The following axiom schemata express the requirement that attribute-value constants have no attributes and that all attribute-

5 This differs from earlier work (Johnson 1988) in which values and attributes were both conceptualized as individuals. In fact, research in progress indicates that it is advantageous to conceptualize of attributes as individuals and attribute relations in terms of a 3-place relation *arc*, where $arc(x, a, y)$ is true iff the value of x's attribute a is y. This permits the quantification over attributes needed to define both simple and parameterized *sorts* to be expressed.

value constants are distinct; i.e., that distinct attribute-value constants denote different entities.

1. For all attribute-value constants c and attributes a, $\forall\, x\ \neg a(c, x)$.
2. For all distinct pairs of attribute-value constants c_1, c_2, $c_1 \neq c_2$.

The next axiom schema requires attributes to be single-valued.

3. For all attributes a, $\forall\, xyz\ a(x, y) \wedge a(x, z) \rightarrow y = z$.

This completes the axiomatization of attribute-value feature structures.[6] The claim is that *any* interpretation that satisfies these axioms is an attribute-value structure, i.e. 1–3 constitute a *definition* of attribute-value structures. Such interpretations can be viewed as (possibly infinite and disconnected) directed graphs, where the individuals constitute the graph's nodes and the attribute relations the arcs between those nodes.

Thus these axioms admit a much wider class of models than do most other treatments of feature structures (e.g., Kasper and Rounds (1990) require feature structures to be a certain type of finite automata). In fact it is easy to add axioms requiring attribute-value structures to have additional properties such as acyclicity. But since the axioms that define attribute-value structures are in effect *assumptions* that *stipulate* the nature of linguistic entities, we obtain a more general theory the weaker these axioms are. Thus 1–3 are intended to stipulate only the properties of attribute-value structures that are required by linguistic analyses.

Note that the *partiality* of attributes is of crucial importance: if attributes were required to be total rather than partial functions, we could not axiomatize them with formulae from the Schönfinkel-Bernays class. (An axiom schema requiring attributes to be total functions would have instances of the form $\forall x\, \exists y\ a(x, y)$, which do not belong to the Schönfinkel-Bernays class).

Example 7
The interpretation corresponding to the attribute-value structure depicted in Figure 1 has as its domain the set $D = \{seem, like, john, sg, 3rd, mary, pres, none\} \cup \{e_0, \ldots, e_5\}$. The attributes denote relations on $D \times D$. For example, *pred* denotes the relation $\{\langle e_0, seem\rangle, \langle e_1, like\rangle, \langle e_2, john\rangle, \langle e_3, mary\rangle\}$. It is straightforward to check that all of the axioms hold in this interpretation.

Instead of providing entities in the interpretation that serve as the denotation for "disjunctive" or "negative" features, we follow Kasper and Rounds (1986, 1990), Moshier and Rounds (1987), and Johnson (1988, 1990) in permitting disjunction and negation only in the constraint language. Since the classical semantics of disjunction and negation for first-order languages is consistent and monotonic, a consistent, monotonic semantics for negative and disjunctive feature constraints follows directly. (An example is presented below; see Johnson (1990) and especially Section 2.10 of Johnson (1988) for further discussion).

We turn now to the set-valued features. The most straightforward way of introducing set-valued features would be to combine some standard axiomatization of set theory with the axiomatization of attribute-value structures just presented. Unfortunately,

6 This axiomatization is finite iff the sets of attribute symbols and constant symbols are finite. In the intended computational and linguistic applications this is always the case.

all of the formulations of set-theory I am aware of, such as Zermelo–Fraenkel set-theory, are expressed in languages whose satisfiability problem is undecidable. While this does not imply that the satisfiability problem for set-valued feature-structure constraints is also undecidable (since the feature constraint language may have restricted expressiveness), it does mean that its decidability cannot be shown by noting that a translation into a decidable class of formulae exists.

Also, as an anonymous reviewer points out, since the intended linguistic applications only require finite sets and operations such as union and intersection, standard theories of sets (such as Zermelo–Fraenkel set-theory) are much more powerful than needed.

Instead, we axiomatize just those properties of set-valued features that our feature constraints require using formulae from the Schönfinkel-Bernays class. We interpret the two-place relation *in* as the membership relation; $in(x, y)$ is true in a model iff x is a member of y. We place no restrictions on this relation, but in other formulations axioms of foundation and extensionality could be used to ensure that the *in* relation can be interpreted as the set-membership relation of Zermelo–Fraenkel set theory. Thus this axiomatization presented here will admit models in which the values of set-valued features do not have these properties.[7] These additional properties of the set-membership relation don't seem to be needed in linguistic analyses, so such stipulations are not made here.

The axiom of foundation requires that all sets are well founded; i.e., that the transitive closure of the set-membership relation is irreflexive, or more informally, that no set directly or indirectly contains itself as a member. Versions of set-theory that relax this restriction have been proposed by, e.g., Aczel (1988), and Rounds (1988) argues that non–well founded sets may be appropriate models of set values in feature structures. The paradoxes associated with non–well founded set theories are avoided here because the axiom of comprehension that asserts the existence of paradoxical sets is not included in this axiomatization; i.e., the only way of defining a set is either by explicitly listing its members or by means of union and intersection operations.

The axiom of extensionality requires that if sets S_1 and S_2 contain exactly the same members then $S_1 = S_2$; without extensionality it is possible for two different sets to contain exactly the same members. Admittedly the primary reason for omitting an extensionality axiom is that it does not appear to be axiomatizable using Schönfinkel-Bernays' formulae, but three other reasons motivate this decision.

First, as noted in Shieber (1986) and in Example 1 above, feature structures in general are not extensional (e.g., two distinct attribute-value elements can have exactly the same attributes and values), and it seems reasonable to treat set-values in a nonextensional fashion as well.

Second, extensionality could produce undesirable interactions with the attribute-value component of feature structures. Since set-valued features can also have attributes (for example, in LFG (Kaplan and Bresnan 1982)) a conjunction is associated with a set-value that also has attributes), extensionality would prohibit there being

7 In fact there are Schönfinkel-Bernays axioms that require the *in* relation to be acyclic. Define a new relation, say in^+, by the axioms

$$\forall\, e\, s\ in(e, s) \rightarrow in^+(e, s)$$

$$\forall\, e\, s\, s'\ in^+(e, s) \wedge in^+(s, s') \rightarrow in^+(e, s').$$

Then in any model in^+ denotes a superset of the transitive closure of the *in* relation. The following axiom requires that this transitive closure is irreflexive, i.e. that no set is contained in itself.

$$\forall\, s\, \neg in^+(s, s).$$

two set-valued features that contain exactly the same elements but that differ on the value of some attribute, something a linguistic analysis might reasonably require.

Third, as far as I am aware, no linguistic analysis requires sets to be extensional. Appealing to the same general considerations that were used to justify the attribute-value axioms, since the assumption that sets are extensional is not required, the stipulation is not made here.

It is necessary to define some predicates that describe set-values. We begin by presenting a general first-order axiomatization of these predicates, and then approximate these with formulae from the Schönfinkel-Bernays class.

Most of the definitions are straightforward, and are given without explanation.[8] The unary predicate *null* is true of an element iff that element has no members.

4. $\forall x\ null(x) \leftrightarrow \neg\exists y\ in(y,x)$

The ternary relation $union(x,y,z)$ is true only if every element in z is in x or y.

5. $\forall xyz\ union(x,y,z) \leftrightarrow \forall u\ in(u,z) \leftrightarrow in(u,x) \vee in(u,y)$

The ternary relation $intersection(x,y,z)$ is true only if every element in z is in x and in y.

6. $\forall xyz\ intersection(x,y,z) \leftrightarrow \forall u\ in(u,z) \leftrightarrow in(u,x) \wedge in(u,y)$

The binary relation $singleton(u,x)$ is true if and only if u is the only member of x.

7. $\forall ux\ singleton(u,x) \leftrightarrow in(u,x) \wedge \forall v\ in(v,x) \rightarrow u = v$

Unfortunately the axioms 4–7 do not belong to the Schönfinkel-Bernays class, so we cannot guarantee the decidability of systems of constraints defined using them simply by noting a translation into this class exists. However, in all of the linguistic applications I am aware of these predicates always appear positively, and in this case these axioms can be replaced by the corresponding "one-sided" axioms given below. (The predicate *null* is an exception, since some HPSG analyses (Pollard and Sag 1987) require the set of unsaturated arguments of some phrases to be non-null. However, it is possible to require that a set s is nonempty by introducing a new constant u and require that $in(u,s)$.)

4′. $\forall xy\ null(x) \rightarrow \neg in(y,x)$

5′. $\forall uxyz\ union(x,y,z) \rightarrow (in(u,z) \leftrightarrow in(u,x) \vee in(u,y))$

6′. $\forall uxyz\ intersection(x,y,z) \rightarrow (in(u,z) \leftrightarrow in(u,x) \wedge in(u,y))$

7′. $\forall uxv\ singleton(u,x) \rightarrow (in(u,x) \wedge in(v,x) \rightarrow u = v)$

These one-sided definitions are incorrect when these predicates appear negatively (i.e., in the scope of an odd number of negation symbols after all other proposition connectives have been expressed in terms of $\wedge$, $\vee$, and $\neg$). For example, an interpretation

8 In the following axioms all of the connectives are to be interpreted as right-associative.

with an empty *in* relation can satisfy $\neg null(x)$. As Johan van Benthem and the anonymous reviewer independently pointed out to me, it is possible to prove that so long as the relations *null*, *union*, *intersection*, and *singleton* appear only positively in linguistic constraints, any model satisfying 4′–7′ differs from a model satisfying 4–7 at most in the denotation of these relations; other relations, in particular the attribute relations or even the *in* relation, are not affected by the one-sided approximation. The following proposition expresses this.

Proposition
Let x be a tuple of variables, A be any relation symbol, $\Psi(A)$ be any formula in which A appears only positively, and $\varphi(x)$ be a formula in which A does not appear. Then

(i) $\mathcal{M} \models \Psi(A) \wedge \forall x\, A(x) \leftrightarrow \varphi(x)$

if and only if there is a model $\mathcal{M}'$ differing from $\mathcal{M}$ only on the denotation it assigns to A such that

(ii) $\mathcal{M}' \models \Psi(A) \wedge \forall x\, A(x) \rightarrow \varphi(x)$.

Proof
The left to right direction is obvious. The proposition follows from right to left as follows. Let $\mathcal{M}'$ be any model that satisfies (ii). A model $\mathcal{M}$ that satisfies (i) can be constructed as follows. Let $\mathcal{M}$ be the model that agrees with $\mathcal{M}'$ except possibly on A, where $[\![A]\!]_{\mathcal{M}} = [\![A]\!]_{\mathcal{M}'} \cup [\![\lambda x \varphi(x)]\!]_{\mathcal{M}'}$. Now we check that $\mathcal{M}$ satisfies (i). Since $[\![A]\!]_{\mathcal{M}} \supseteq [\![A]\!]_{\mathcal{M}'}$ and A appears only positively in $\Psi(A)$, $\mathcal{M} \models \Psi(A)$. Further $\mathcal{M} \models \forall x A(x) \leftarrow \varphi(x)$ by construction. Since A does not appear in $\varphi(x)$, $[\![\lambda x \varphi(x)]\!]_{\mathcal{M}} = [\![\lambda x \varphi(x)]\!]_{\mathcal{M}'}$, and since $[\![A]\!]_{\mathcal{M}} \supseteq [\![A]\!]_{\mathcal{M}'}$, $\mathcal{M} \models \forall x A(x) \rightarrow \varphi(x)$ as well. Thus $\mathcal{M}$ satisfies (i) as required. In fact we have shown something stronger; the denotation of A in $\mathcal{M}'$ is a subset of the denotation of A in $\mathcal{M}$. ■

2.2 Expressing Constraints

A feature structure is specified implicitly by means of the constraints that it must satisfy. This section shows how such constraints can be translated into quantifies-free and function-free prenex formulae. There is a plethora of different notations for expressing these constraints: the constraint requiring that the value of attribute a of (the entity denoted by) x is (the entity denoted by) y is written as $\langle x\ a \rangle = y$ in PATR-II (Shieber 1986), as $(x\ a) = y$ in LFG (Kaplan and Bresnan 1982), and as $x(a) \approx y$ in Johnson (1988), for example. Here we express attribute-value constraints using the attribute relations a, so this constraint would be expressed as $a(x, y)$. Set-valued constraints are expressed using the relations *in*, *null*, *union*, and *singleton* defined in the previous section. The propositional connectives are used to express negative and disjunctive feature constraints. This section shows how constraints on feature bundles can be specified using equality, the attribute relations, and the set predicates axiomatized in the last section. (In fact as far as the theoretical results of this paper are concerned all that is important is that the constraints are taken to *mean* the same thing as these formulae, irrespective of the *notation* in which they are expressed.)

Example 2 (continued)
The lexical entries for *salmon* and *swims* in Figure 2 are the following formulae, where e', e'', f', f'' and g'' are constants of the first-order language that are not attribute-value constants.[9]

8a. $pred(e', salmon) \wedge agr(e', f') \wedge pers(f', 3rd)$

8b. $pred(g'', swim) \wedge tense(g'', pres) \wedge subj(g'', e'') \wedge agr(e'', f'') \wedge num(f'', sg) \wedge pers(f'', 3rd).$

Example 3 (continued)
The lexical entry for the determiner *die* of Figure 4 is the following formula, where x is a (non–attribute-value) constant that denotes the feature structure of the determiner, and y and z are constants that are not attribute-value constants.

9. $cat(x, determiner) \wedge agr(x, y) \wedge (case(x, nom) \vee case(x, acc)) \wedge (number(y, plural) \vee (number(y, singular) \wedge gender(y, feminine)))$

Example 4 (continued)
The lexical entry for the verb *swim* of Figure 5 is the following formula, where g is a constant that denotes the feature structure of the verb, and e and f are constants that are not attribute-value constants.[10]

10. $pred(g, swim) \wedge tense(g, pres) \wedge subj(g, e) \wedge agr(e, f) \wedge \neg(num(f, sg) \wedge pers(f, 3rd))$

The lexical entries for the determiners *this* and *these* of Figure 9 are the following formulae, where u, v, u' and v' are constants that are not attribute-value constants, and u denotes the feature structure of *this* and u' denotes the feature structure of *these*.

11. $def(u, +) \wedge agr(u, v) \wedge num(v, sg) \wedge pers(v, 3rd)$

12. $def(u', +) \wedge agr(u', v') \wedge num(v', pl) \wedge pers(v', 3rd)$

---

9 Instead of *naming* all of the nonroot attribute-value elements with constants as is done here, it is possible to merely assert their existence using an existential quantification. For example, the lexical entry for *salmon* could be the formula

$$\exists f' pred(e', salmon) \wedge agr(e', f') \wedge pers(f', 3rd)$$

where f' is an existentially quantified variable. This formulation has the advantage that no 'renaming' is needed when determining *subsumption* of systems of attribute-value constraints. (The subsumption relation between systems of constraints is used in certain types of 'unification based' parsers (Shieber 1989).) That is, a system of constraints represented by a formula φ subsumes another system of constraints represented by θ iff $A \models \theta \rightarrow \varphi$, where A is the conjunction of the axioms defining the relevant types of feature structures.

10 The formulation (10) of the negative constraint depicted in Figure 5 does not imply that f has either a *num* or *pers* attribute. Conceivably, one might want to interpret such a negative constraint as requiring f to have both *num* and *pers* attributes with values differing from either *sg* or *3rd*, respectively. The formula below expresses this interpretation.

$$pred(g, swim) \wedge tense(g, pres) \wedge subj(g, e) \wedge agr(e, f) \wedge num(f, u) \wedge pers(f, v) \wedge \neg(u = sg \wedge v = 3rd)$$

Example 5 (continued)
The lexical entries for *she* and *woman* of Figure 6 are the formulae (13) and (14), where u denotes the feature structure of the pronoun, v denotes the feature structure of the noun, and w, s, s', s'', i, and i' are constants that are not attribute-value constants.

13. $cat(u, np) \wedge refs\text{-}in(u, s) \wedge refs\text{-}out(u, s) \wedge index(u, i) \wedge in(i, s)$
14. $cat(v, n) \wedge index(v, i') \wedge refs\text{-}in(v, s') \wedge refs\text{-}out(v, s'') \wedge singleton(i', w) \wedge union(s', w, s'')$

In general then, a system of feature structure constraints can be viewed as a function-free and quantifier-free formula. These constraints are satisfiable if and only if there is an interpretation that simultaneously satisfies the corresponding formula and the axioms presented in the previous section, or equivalently, the conjunction of this formula and the relevant axioms from the axiomatization. This conjunction is itself a formula from the Schönfinkel-Bernays class, and so the satisfiability problem for systems of feature structure constraints is decidable.

Further, we can apply results on the computational complexity of the satisfiability problem for the Schönfinkel-Bernays class to determine the computational complexity of the satisfiability problem for systems of such feature constraints. Since (universal) quantifiers appear only in the axiomatization of feature structures and not in the feature constraints themselves, the number of quantifiers appearing in the conjunction of the feature constraints and the axiomatization is a constant, and does not vary with the size of the system of feature constraints. By Proposition 3.2 of Lewis (1980), the satisfiability problem for a formula F with u universal quantifiers in the Schönfinkel-Bernays class requires nondeterministic time polynomial in $|F|^u$, so the problem is in NP. The reductions presented in Kasper and Rounds (1986) and Johnson (1988) can be used to show that the problem is NP-hard, so the satisfiability problem for feature constraints with set-values (as defined above) is NP-complete.

2.3 Unification and Satisfaction

This section discusses the relationship between unification and the axiomatization presented above.

Unification identifies or merges exactly the elements that the axiomatization implies are equal. The unification of two complex elements e and e' causes the unification of the values of all attributes a that are defined on both e and e'. Similarly, the conjunction of the formulae $e = e'$, $a(e, f)$, $a(e', f')$ and the axioms given above implies that $f = f'$, since axiom schema (3) requires that attributes are single valued.

Similarly, the unification of two attribute-value structures fails either when two distinct constant elements are unified (a constant-constant clash) or when a constant and a complex element are unified. The formula $x = x'$ is unsatisfiable under exactly the same circumstances in the theory axiomatized above. The formula $x = x'$ conjoined with $x = c$ and $x' = c'$ for distinct attribute-value constants c, c' is unsatisfiable, since $c \neq c'$ by axiom schema (2). Also, $x = x'$ is unsatisfiable when conjoined with $a(x, y)$ for any y and $x' = c$, since $\neg a(c, y)$ by axiom schema (1).

If attention is restricted to purely conjunctive attribute-value systems, the corresponding formulae can be represented as a directed graph, where nodes represent (first-order) constants, and an arc labeled a from x to y encodes the atom $a(x, y)$. Then the standard attribute-value 'unification algorithm' can be used as a specialized inference procedure that takes as input such a graph encoding of a conjunction of

attribute-value relations and returns (the graph encoding of) the conjunction of all of their atomic consequences.

As Kasper (1986, 1987) noted in a different setting, the steps of the attribute-value unification algorithm are just applications of the axioms 1–3. It 'forward chains' using axiom schema (3) (for which the graph representation provides efficient indexing), and checks at each step that 1 and 2 are not falsified; if they are falsified the unification algorithm halts and reports a unification failure. Atomic equalities $x = y$ are represented by a 'forwarding pointer' from x to y (as in the UNION-FIND algorithm (Gallier 1986; Nelson and Oppen 1980; Johnson in press)).

Example 2 (continued)
The unification of e' and e'' in Figures 2 and 3 corresponds to conjoining the formula $e' = e''$ to the conjunction of 8a and 8b, resulting in the formula 15a.

15a. $e' = e'' \wedge pred(e', salmon) \wedge agr(e', f') \wedge pers(f', 3rd) \wedge pred(g'', swim) \wedge tense(g'', pres) \wedge subj(g'', e'') \wedge agr(e'', f'') \wedge num(f'', sg) \wedge pers(f'', 3rd).$

This formula can be simplified by substituting e' for e'' to yield 15b (this substitution corresponds exactly to the first step of the unification algorithm, *viz.* redirecting e'' to e'). The affected subformulae are in boldface below.

15b. $e' = e'' \wedge pred(e', salmon) \wedge agr(e', f') \wedge pers(f', 3rd) \wedge pred(g'', swim) \wedge tense(g'', pres) \wedge \mathbf{subj(g'', e')} \wedge \mathbf{agr(e', f'')} \wedge num(f'', sg) \wedge pers(f'', 3rd).$

Since 15b contains the conjunction of $agr(e', f')$ and $agr(e', f'')$, axiom schema (3) requires that $f' = f''$, so 15b can be further simplified by substituting f' for f'' to yield 15c.

15c. $e' = e'' \wedge f' = f'' \wedge pred(e', salmon) \wedge agr(e', f') \wedge pers(f', 3rd) \wedge pred(g'', swim) \wedge tense(g'', pres) \wedge subj(g'', e') \wedge \mathbf{agr(e', f')} \wedge \mathbf{num(f', sg)} \wedge \mathbf{pers(f', 3rd)}.$

The duplicate occurrences of $agr(e', f')$ and $pers(f', 3rd)$ can be deleted, yielding 15d (these last two steps correspond exactly to the unification of f' and f'' in Figure 3).

15d. $e' = e'' \wedge f' = f'' \wedge pred(e', salmon) \wedge agr(e', f') \wedge pers(f', 3rd) \wedge pred(g'', swim) \wedge tense(g'', pres) \wedge subj(g'', e') \wedge num(f', sg).$

No further simplifications are possible, and 15d is satisfiable. In fact 15d describes the structure depicted in Figure 3, as expected.

The standard unification algorithm is unable to handle negative constraints correctly, as noted above. However, because negation is interpreted declaratively (in fact, classically) in the first-order language used to express constraints here, its treatment is straightforward and unproblematic, and suggests ways of extending the unification algorithm to cover these cases (Johnson 1990b, to appear).

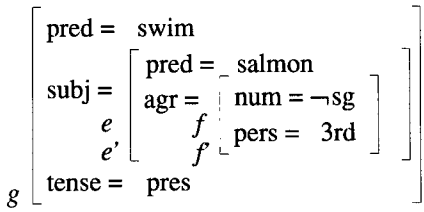

Figure 10
A graphical depiction of the formula 16b

Example 4 (continued)
The unification of e' and e (i.e. the lexical entries for *salmon* and *swim*) of Figure 9 corresponds to the conjunction of the formula $e = e'$ to the conjunction of 8a and 10, which is the formula 16a.

16a. $e = e' \wedge pred(e', salmon) \wedge agr(e', f') \wedge pers(f', 3rd) \wedge pred(g, swim) \wedge tense(g, pres) \wedge subj(g, e) \wedge agr(e, f) \wedge \neg(num(f, sg) \wedge pers(f, 3rd))$

This can be simplified by straightforward applications of axiom schema (3), equality substitution, and propositional equivalences to obtain 16b.

16b. $e = e' \wedge f = f' \wedge pred(e, salmon) \wedge pers(f, 3rd) \wedge pred(g, swim) \wedge tense(g, pres) \wedge subj(g, e) \wedge agr(e, f) \wedge \neg num(f, sg)$.

This formula could be depicted as in Figure 10, where such matrices are now to be understood as graphical depictions of formulae. The further unification of e' with u, the lexical entry for *this*, corresponds to the conjunction of $e' = u$ to the conjunction of the formulae 16b and 11, which is the formula 16c.

16c. $e = e' \wedge f = f' \wedge e' = u \wedge pred(e, salmon) \wedge pers(f, 3rd) \wedge pred(g, swim) \wedge tense(g, pres) \wedge subj(g, e) \wedge agr(e, f) \wedge \neg num(f, sg) \wedge def(u, +) \wedge agr(u, v) \wedge num(v, sg) \wedge pers(v, 3rd)$.

By substituting e for both e' and u in 16c, we obtain 16d.

16d. $e = e' \wedge f = f' \wedge e = u \wedge pred(e, salmon) \wedge pers(f, 3rd) \wedge pred(g, swim) \wedge tense(g, pres) \wedge subj(g, e) \wedge agr(e, f) \wedge \neg num(f, sg) \wedge \mathbf{def(e, +)} \wedge \mathbf{agr(e, v)} \wedge num(v, sg) \wedge pers(v, 3rd)$.

Again, since 16d contains the conjunction of $agr(e, f)$ and $agr(e, v)$, axiom schema (3) requires that $f = v$, so 16d can be further simplified by substituting f for v, yielding 16e.

16e. $e = e' \wedge f = f' \wedge e = u \wedge f = v \wedge pred(e, salmon) \wedge pers(f, 3rd) \wedge pred(g, swim) \wedge tense(g, pres) \wedge subj(g, e) \wedge agr(e, f) \wedge \neg num(f, sg) \wedge def(e, +) \wedge \mathbf{agr(e, f)} \wedge \mathbf{num(f, sg)} \wedge \mathbf{pers(f, 3rd)}$.

$$
g\begin{bmatrix}
\text{pred} = \text{swim} \\
\text{subj} = \begin{bmatrix}
\text{pred} = \text{salmon} \\
\text{def} = + \\
\text{agr} = \begin{matrix} e \\ e' \\ u' \end{matrix} \begin{bmatrix} f \\ f' \\ v' \end{bmatrix} \begin{bmatrix} \text{num} = \text{pl} \\ \text{pers} = \text{3rd} \end{bmatrix}
\end{bmatrix} \\
\text{tense} = \text{pres}
\end{bmatrix}
$$

Figure 11
A graphical depiction of the formula 16f′

The formula 16e is unsatisfiable, since it contains conjunction of both $num(f,sg)$ and its negation $\neg num(f,sg)$. This is the desired result, since the utterance *this salmon swim* is ill formed.

On the other hand, the unification e' in 16b (c.f. Figure 10) is with u', the lexical entry for *these*, corresponds to the conjunction of $e' = u'$, 16b and 12, which is the formula 16c′.

16c′. $e = e' \wedge f = f' \wedge e' = u' \wedge pred(e,salmon) \wedge pers(f,3rd) \wedge pred(g,swim) \wedge tense(g,pres) \wedge subj(g,e) \wedge agr(e,f) \wedge \neg num(f,sg) \wedge def(u',+) \wedge agr(u',v') \wedge num(v',pl) \wedge pers(v',3rd)$.

By following the same steps as were used to simplify 16c to 16e, 16c′ can be simplified to 16e′.

16e′. $e = e' \wedge f = f' \wedge e = u' \wedge f = v' \wedge pred(e,salmon) \wedge pers(f,3rd) \wedge pred(g,swim) \wedge tense(g,pres) \wedge subj(g,e) \wedge agr(e,f) \wedge \neg num(f,sg) \wedge def(e,+) \wedge agr(e,f) \wedge num(f,pl) \wedge pers(f,3rd)$.

One of duplicate conjuncts $agr(e,f)$ can be deleted, and since $num(f,pl)$ implies $\neg num(f,sg)$ (by instances $sg \neq pl$ of (2) and $\forall xyz\ num(x,y) \wedge num(x,z) \rightarrow y = z$ of (3)), 16e′ can be further simplified to 16f′, where $\neg num(f,sg)$ has also been deleted.

16f′. $e = e' \wedge f = f' \wedge e = u' \wedge f = v' \wedge pred(e,salmon) \wedge pers(f,3rd) \wedge pred(g,swim) \wedge tense(g,pres) \wedge subj(g,e) \wedge agr(e,f) \wedge def(e,+) \wedge num(f,pl) \wedge pers(f,3rd)$.

This formula is satisfiable, as desired, since the utterance *these salmon swim* is well formed. This formula could be depicted as in Figure 11, where again the matrix is to be understood as a graphical depiction of the formula 16f′.

The set-valued examples are somewhat more complicated because they involve quantification.

Example 5 (continued)
The unification of s' with the empty set in Figure 6 corresponds to the conjunction of 14 with the formula $null(s')$, as given in 17a.

17a. $cat(v,n) \wedge index(v,i') \wedge refs\text{-}in(v,s') \wedge refs\text{-}out(v,s'') \wedge singleton(i',w) \wedge union(s',w,s'') \wedge null(s')$.

Now $singleton(i',w) \wedge union(s',w,s'')$ implies by axioms (5) and (7) that $\forall u\ in(u,s'') \leftrightarrow u = i' \vee in(u,s')$. Further, since $null(s')$ implies by axiom (4) that $\forall u \neg in(u,s')$, it follows that 17a is equivalent to 17b.

17b. $cat(v,n) \wedge index(v,i') \wedge refs\text{-}in(v,s') \wedge refs\text{-}out(v,s'') \wedge singleton(i',w) \wedge union(s',w,s'') \wedge null(s') \wedge \forall u\ (in(u,s'') \leftrightarrow u = i')$.

Unifying the value of the *refs-out* attribute of Figure 7 with the value of the refs-in attribute of u in Figure 6 corresponds to conjoining $s = s''$ with the conjunction of 17b and 13, yielding 17c.

17c. $s = s'' \wedge cat(u,np) \wedge refs\text{-}in(u,s) \wedge refs\text{-}out(u,s) \wedge index(u,i) \wedge in(i,s) \wedge cat(v,n) \wedge index(v,i') \wedge refs\text{-}in(v,s') \wedge refs\text{-}out(v,s'') \wedge singleton(i',w) \wedge union(s',w,s'') \wedge null(s') \wedge \forall u\ (in(u,s'') \leftrightarrow u = i')$.

This can be simplified by substituting s for s'' and noting that $\forall u\ (in(u,s) \leftrightarrow u = i')$ and $in(i,s)$ implies that $i = i'$, as required.

17d. $s = s'' \wedge cat(u,np) \wedge refs\text{-}in(u,s) \wedge refs\text{-}out(u,s) \wedge index(u,i) \wedge in(i,s) \wedge cat(v,n) \wedge index(v,i) \wedge refs\text{-}in(v,s') \wedge refs\text{-}out(v,s) \wedge singleton(i,w) \wedge union(s',w,s) \wedge null(s') \wedge \forall u\ (in(u,s) \leftrightarrow u = i)$.

3. Conclusion

The general approach adopted here of separating the feature structures and the constraints that they must satisfy is used in most accounts of feature structures. The novel aspect of this work is that feature structures are axiomatized in and the constraints on feature structures are expressed in a decidable class of first-order logic, so important results such as decidability and compactness follow directly. The Schönfinkel-Bernays class of formulae used in this paper are sufficiently expressive so that "set-valued" features can be axiomatized quite directly.

We conclude with some tentative remarks about the implementation of the system described here. Although a general-purpose first-order logic theorem prover could be used to determine the satisfiability of Schönfinkel-Bernays formulae, it should be possible to take advantage of the syntactic restrictions these formulae satisfy to obtain a more efficient implementation. One way in which this might be done is as follows.

First, the axioms should be expressed in *clausal form*, i.e. in the form

$$\exists x_1 \ldots x_n \forall\ y_1 \ldots y_n\ A_1 \wedge \ldots \wedge A_m \rightarrow B_1 \vee \ldots \vee B_n$$

where the A_i and B_j are atoms. These can be used in a 'forward chaining' inference procedure using 'semi-naive evaluation' (see Genesereth and Nilsson (1987) for details).

For example, the clausal form expansion of axiom (5′) for *union* is

18a. $\forall\ xyzu\ \ union(x,y,z) \wedge in(u,z) \rightarrow in(u,x) \vee in(u,y)$

18b. $\forall\ xyzu\ \ union(x,y,z) \wedge in(u,x) \rightarrow in(u,z)$

18c. $\forall\ xyzu\ \ union(x,y,z) \wedge in(u,y) \rightarrow in(u,z).$

Second, if efficiency comparable to the standard (purely conjunctive) unification algorithm is to be achieved, it is necessary to efficiently index atoms on their arguments (both from the original constraints and those produced as consequences during the inference process just described). If we were dealing with only purely conjunctive formulae we could use a graph-based representation similar to the one used in the standard attribute-value unification algorithm, but since axioms such as 18a have disjunctive consequents we need a data structure that can represent nonconjunctive formulae, even if all of the linguistic constraints associated with lexical entries and syntactic rules are purely conjunctive. This problem is an instance of the general problem of disjunction, and it seems that some of the techniques proposed in the feature-structure literature to deal with disjunction (e.g. Eisele and Dörre 1988; Dörre and Eisele 1990; Maxwell and Kaplan 1989a, 1989b) can be applied here too.

Acknowledgments

I would like to thank Nick Asher, Jochen Dörre, Andreas Eisele, Martin Emele, Martin Kay, Ron Kaplan, Lauri Karttunen, Harry Lewis, John Maxwell, Bill Rounds, Gert Smolka, Stuart Shieber, Rich Thomason, Johan van Benthem, and Jürgen Wedekind as well as an anonymous reviewer and audiences at the CLIN Dag in Utrecht and at the DFKI in Saarbrücken for their important contributions to the material presented in this paper. The idea of translating feature constraints into a specialized language with desirable computational properties arose in conversation with Jürgen Wedekind. Harry Lewis noted that attribute-value structures could be axiomatized using formulae from the Schönfinkel-Bernays class, and guided me to the relevant results. Naturally, all errors remain my own. The final revision of this paper was completed at the Institut für maschinelle Sprachverarbeitung, Universität Stuttgart, which I thank for providing a congenial research environment.

References

Aczel, P. (1988). Non-Well-Founded Sets. CSLI Lecture Notes Series, University of Chicago Press, Chicago.

Bresnan, J. (1982). *The Mental Representation of Grammatical Relations*. The MIT Press, Cambridge, Mass.

Dawar, A. and Vijayashanker, K. (1990). *Three-Valued Interpretation of Negation in Feature Structure Descriptions*. University of Delaware Technical Report 90-03.

Dawar, A. and Vijayashanker, K. (1989). "A Three-Valued Interpretation of Negation in Feature Structures," in *The 27th Annual Meeting of the Association of Computational Linguistics*, Vancouver.

Dörre, J. and Eisele, A. (1990). "Feature Logic with Disjunctive Unification," in *The Proceedings of COLING—1990*, Helsinki, Finland.

Dörre, J. and Rounds, W. (1989). *On Subsumption and Semiunification in Feature Algebras*. IWBS Report, IBM Germany.

Dreben, B. and Goldfarb, W. D. (1979). *The Decision Problem: Solvable Classes of Quantificational Formulas*. Addison-Wesley, Reading, Mass.

Eisele, A. and Dörre, J. (1988). "Unification of Disjunctive Feature Descriptions," in *The Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, Buffalo, New York, 286–294.

Gallier, J. H. (1986). *Logic for Computer Science*. Harper and Row, New York.

Gazdar, G.; Klein, E.; Pullum, G.; and Sag, I. (1985). *Generalized Phrase Structure Grammar*. Blackwell, Oxford, England.

Genesereth, M. and Nilsson, N. (1987). *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, Los Altos.

Gurevich, Y. (1976). "The Decision Problem for Standard Classes," *The Journal of Symbolic Logic*. 41.2, 460-464.

Haddock, N. J.; Klein, E.; and Morrill, G. (1987). *Categorial Grammar, Unification*

Grammar. University of Edinburgh Edinburgh Working Papers in Cognitive Science 1.

Johnson, M. (1991). "Logic and Feature Structures," in *Proceedings of IJCAI 1991*, Sydney.

Johnson, M. (1990a). "Expressing Disjunctive and Negative Feature Constraints with Classical First-order Logic," in *The Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, Pittsburgh, PA.

Johnson, M. (1990b). "Features, Frames and Quantifier-Free Formulae," in P. Saint-Dizier and S. Szpakowicz, eds., *Logic and Logic Grammars for Language Processing*, Ellis Horwood, New York.

Johnson, M. (1988). *Attribute-Value Logic and the Theory of Grammar*. CSLI Lecture Notes Series, University of Chicago Press, Chicago.

Johnson, M. (in press). "Attribute-Value Logic and Natural Language Processing," in J. Wedekind, ed., *Studies in Unification Grammar*, The MIT Press, Cambridge, Mass. (originally presented at the Titisee Conference, Germany, 1988).

Johnson, M. and Kay, M. (1990). "Semantic Operators and Anaphora," in *The Proceedings of COLING 1990*, Helsinki, Finland.

Johnson, M. and Klein, E. (1986). "Discourse, Parsing and Anaphora," in *The Proceedings of COLING 1986*, Bonn, West Germany.

Kamp, H. (1981). "A Theory of Truth and Semantic Representation," in J. A. G. Groenendijk, T. M. V. Janssem and M. B. J. Stokhof, eds., *Formal Methods in the Study of Language*, Mathematical Centre Tracts, Amsterdam.

Kaplan, R. and Bresnan, J. (1982). "Lexical-functional grammar, a formal system for grammatical representation," in J. Bresnan, ed., *The Mental Representation of Grammatical Relations*, The MIT Press, Cambridge, Mass.

Karttunen, L. (1984). "Features and Values," in *COLING-1984*, The Association for Computational Linguistics, Stanford University.

Kasper, R. T. (1988). "Conditional Descriptions in Functional Unification Grammar," in *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, Buffalo, New York.

Kasper, R. T. (1987). "A Unification Method for Disjunctive Feature Structures," in *The Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford University.

Kasper, R. T. (1986). *Feature Structures: A Logical Theory with Application to Language Analysis*, Ph.D. Thesis, University of Michigan.

Kasper, R. T. and Rounds, W. C. (1990). "The Logic of Unification in Grammar," *Linguistics and Philosophy*. 13.1, 35–58.

Kasper, R. T. and Rounds, W. C. (1986). "A Logical Semantics for Feature Structures," in *The Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, Columbia University, New York.

Kay, M. (1985). "Unification in Grammar," in V. Dahl and P. Saint-Dizier, eds., *Natural Language Understanding and Logic Programming*, North Holland, Amsterdam, The Netherlands.

Langholm, T. (1989). *How to Say No with Feature Structures*. Department of Mathematics, University of Oslo, COSMOS Report No. 13.

Lewis, H. (1980). "Complexity Results for Classes of Quantificational Formulae," *Journal of Computer and System Sciences*. 21, 317–353.

Lewis, H. and Papadimitriou, C. (1981). *Elements of the Theory of Computation*. Prentice Hall, New Jersey.

Maxwell, J. and Kaplan, R. (1989a). *A Method for Disjunctive Constraint Satisfaction*. Xerox PARC ms.

Maxwell, J. T., III and Kaplan, R. (1989b). "An Overview of Disjunctive Constraint Satisfaction," in *International Workshop on Parsing Technologies*, Carnegie Mellon, Pittsburgh, PA.

Moshier, M. (1988). *Extensions to Unification Grammar for the Description of Programming Languages*, Ph.D. Thesis, University of Michigan.

Moshier, M. D. and Rounds, W. C. (1987). "A Logic for Partially Specified Data Structures," in *The ACM Symposium on the Principles of Programming Languages*, Association for Computing Machinery, Munich, Germany.

Nebel, B. and Smolka, G. (1989). *Representation and Reasoning with Attributive Descriptions*. IBM Stuttgart IWBS Report 81.

Nelson, G. and Oppen, D. C. (1980). "Fast Decision Procedures Based on Congruence Closure," *Journal of the Association for Computing Machinery*. 27.2, 245–257.

Pereira, F. C. N. (1987). "Grammars and Logics of Partial Information," in *The Proceedings of the International Conference on Logic Programming*, Melbourne,

Australia.
Pereira, F. C. N. and Shieber, S. M. (1984). "The Semantics of Grammar Formalisms Seen as Computer Languages," in *COLING-84*, The Association for Computational Linguistics, Stanford University.
Pollard, C. and Sag, I. (1987). *Information-Based Syntax and Semantics, Volume 1*. CSLI Lecture Notes, Chicago University Press, Chicago.
Rounds, W. (1988). *Set Values for Unification-Based Grammar Formalisms and Logic Programming*. Center for the Study of Language and Information CSLI Report 129.
Shieber, S. M. (1989). *Parsing and Type Inference for Natural and Computer Languages*. SRI International Technical Note 460.
Shieber, S. M. (1986). *An Introduction to Unification-Based Approaches to Grammar*. CSLI Lecture Notes Series, University of Chicago Press, Chicago.
Smolka, G. (1989a). *Attributive Concept Descriptions with Unions and Complements*. IBM Stuttgart IWBS Report 68.
Smolka, G. (1989b). *Feature Constraint Logics for Unification Grammars*. IBM Deutschland Wissenschaftliches Zentrum IWBS Report No. 93.
Smolka, G. (1988). *A Feature Logic with Subsorts*. IBM Deutschland GmbH. Lilog Report No. 33.
Uszkoreit, H. (1986). "Categorial Unification Grammar," in *COLING-86*, 187–194.