Computing with Features as Formulae

Mark Johnson*
Brown University

This paper extends the approach to feature structures developed in Johnson (1991a), which uses Schönfinkel-Bernays' formulae to express feature structure constraints. These are shown to be a disjunctive generalization of Datalog clauses, as used in database theory. This paper provides a fixed-point characterization of the minimal models of these formulae that serves as the theoretical foundation of a forward-chaining algorithm for determining their satisfiability. This algorithm, which generalizes the standard attribute-value unification algorithm, is also recognizable as a nondeterministic variant of the semi-naive bottom-up algorithm for evaluating Datalog programs, further strengthening the connection between the theory of feature structures and databases.

1. Introduction

Despite their simplicity, a surprisingly wide range of linguistic phenomena can be described in terms of simple equality constraints on values in attribute-value structures, which are a particularly simple kind of feature structure (see Shieber 1986; Johnson 1988; Uszkoreit 1986; and Bresnan 1982 for examples of some of these analyses). But some phenomena do not seem to be able to be described in such a pure 'unification' framework. For example, the analysis of conjunctions in LFG (Kaplan and Maxwell 1988b) and the formalizations of Discourse Representation Theory (Kamp 1981) presented in Johnson and Klein (1986) and Johnson and Kay (1990) require additional mechanisms for representing and manipulating aggregates or sets of values in ways that are beyond the capability of such "pure" attribute-value systems. Further, sortal constraints (which also cannot be expressed as simple equality constraints) can be used to formulate simpler and more comprehensible grammars (Carpenter 1992; Carpenter and Pollard 1991; Pollard and Sag 1987, 1992).

Versions of both of these kinds of constraint, as well as the familiar attribute-value constraints, can be expressed as *Schönfinkel-Bernays' formulae* (as demonstrated in Johnson 1991a, 1991b), so that the problem of determining the satisfiability of a system of such constraints is reduced to the satisfiability problem for the corresponding formula. This class of formulae (defined in Section 3.1) seems to be expressive enough for most linguistic purposes when used with an external phrase-structure backbone. That is, these formulae are used as *annotations* on phrase structure rules in the manner described in, e.g., Kaplan and Bresnan (1982), Shieber (1986), and Johnson (1988). This paper extends the author's previous paper on the topic (Johnson 1991a) by sketching several other linguistic applications of Schönfinkel-Bernays' formulae (including a version of D-theory [Marcus, Hindle, and Fleck 1983; Vijay-Shanker 1992]), and presenting a least-fixed-point theorem that serves as the theoretical basis for a "forward-chaining" algorithm for determining satisfiability of Schönfinkel-Bernays' formulae. Interestingly, this algorithm can be viewed both as a straightforward generalization

* Cognitive and Linguistic Sciences, Box 1978, Brown University, Providence, RI. E-mail: Mark_Johnson@brown.edu

of the standard attribute-value unification algorithm and also as a nondeterministic variant of the semi-naive evaluation method for Datalog clauses.

Several extended "unification-based" constraint formalisms have been developed. In this paper, the term "feature structure" denotes any kind of structured entity used as a component of a category label. An attribute-value structure is a particularly simple kind of feature structure of the kind used in "pure" unification-based frameworks (Shieber 1986). Some extensions to the basic attribute-value framework are rather weak, e.g., allowing disjunctive and negative constraints and preserving decidability.[1] Such systems require an "off-line" phrase structure backbone to which these constraints are attached. It seems that most of the constraints that can be expressed in these formalisms can be expressed as Schönfinkel-Bernays' formulae, the constraint formalism described below.

A second class of extended constraint formalisms has been devised to be capable of expressing the entire grammar as systems of constraints and as far as I know, for all of these systems the problem of determining the satisfiability of an arbitrary system of constraints that they can express is undecidable.[2] This is because the recognition problem for an arbitrary "unification-based" grammar is undecidable unless the size of the phrase structure tree is constrained somehow, e.g., by the offline parsability constraint (Johnson 1988; Kaplan and Bresnan 1982; Pereira 1982; Shieber 1992), but there seems to be no natural way to impose such constraints in these systems because the encoding of the phrase structure tree in the feature structure is not distinguished from other features.[3] Thus in order to maintain decidability the system described here is *not* designed to be capable of expressing phrase structure constraints directly, and must be used with an external phrase-structure component, as in LFG (Bresnan 1982). (However, Bob Carpenter [p.c.] points out that one can impose a bound on the size of the feature structure that can serve as an analysis [say, some polynomial of the length of the input], and so ensure decidability.) Interestingly, a first-order logic-based approach similar to the one presented in this paper can also be developed for extended constraint formalisms capable of expressing the entire grammar, but this is not discussed further here; see Johnson (in press b) for details.

In the approach developed here Schönfinkel-Bernays' formulae are used to express a variety of feature structure constraints. Previous work has shown that these formulae are expressive enough to define arbitrary disjunctions and negations of constraints (Johnson 1990a, 1990b), a kind of 'set-valued' entity (Johnson 1991a), and they can be used to impose useful sort constraints (Johnson 1991b). The expression of D-theory constraints on nodes in trees is discussed in this paper.

This paper extends the ideas in these earlier papers with theoretical results that suggest a forward-chaining algorithm for determining the satisfiability of an arbitrary Schönfinkel-Bernays' formula. This generalizes the standard feature-graph unification algorithm and is closely related to the semi-naive bottom-up algorithm used in database theory.

1 For examples of this approach see Dawar and Vijay-Shanker (1990), Dörre and Eisele (1990), Johnson (1988, 1990a, 1990b, 1991a, 1991b, in press a), Karttunen (1984), Kasper (1987a, 1987b, 1988), Kasper and Rounds (1986, 1990), Langholm (1989), Pereira (1987), and Smolka (1992).

2 Examples of this approach are Carpenter, Pollard, and Franz (1991), Dörre (1991), Dörre and Eisele (1991), Johnson (in press b), Kay (1979, 1985a, 1985b), Pollard and Sag (1987), Rounds and Manaster-Ramer (1987), Smolka (1988), and Zajac (1992).

3 While it may well be that the universal recognition and parsing problems for natural language are undecidable (Chomsky [1986, 1988] points out that there is no contrary evidence), I know of no evidence that this is actually the case. It seems reasonable then to also investigate formalisms that can only express *decidable* systems of constraints (and for which there exist satisfiability-testing algorithms) if linguistically adequate systems can be found.

Specifically, it is shown that the satisfying Herbrand models of an arbitrary Schönfinkel-Bernays' formula are the fix points of certain functions, and that the least fixed points of these functions are all of the models of the formula that are "minimal" in a certain sense. This leads to a forward-chaining algorithm for computing all of the atomic consequences of a Schönfinkel-Bernays' formula; the fixed-point theorem shows that this suffices to determine the satisfiability of an arbitrary Schönfinkel-Bernays' formula.

2. Constraints, Partial Information, and Feature Structures

This approach exploits the fact that constraints on well-formed linguistic structures (e.g., well-formedness constraints imposed by the grammar) do not need to be isomorphic to the structures that satisfy them. Although the distinction between constraints and structures that satisfy them might seem too obvious to warrant comment, it is not made in most work on feature structures.

A common view holds that feature structures are inherently "partially specified" entities, which "unify" or merge with other feature structures to yield more instantiated feature structures in an "information-preserving" way (Shieber 1986). If two feature structures contain "contradictory information," then it is impossible to merge them to produce a consistent object; unification is then said to fail. The feature structure for an utterance is the result (if one exists) of unifying all of the feature structures for the lexical entries and syntactic rules in appropriate ways. Thus in this view feature structures play *two* roles; not only do they serve as linguistic structures, but they are also used to encode constraints that the linguistic structures must satisfy (see Section 2.10 of Johnson (1988) for an extended discussion).

That is, under this view feature structures serve not only as linguistic structures that may or may not satisfy a constraint, but are also interpreted as 'representing' or 'describing' all of the feature structures that they subsume. Given this dual role for feature structures, it is important in this approach that if a feature structure S satisfies a constraint α, then every feature structure subsumed by S should also satisfy α (Pereira 1987). If this "upward closure" property holds, then the set of feature structures satisfying any constraint can be represented by the set of its "minimal models." Unfortunately, many useful constraints do not have this property. For example, under a classical interpretation, the set of feature structures satisfying negated feature structure constraints are not upward-closed (Moshier and Rounds 1987).

The work described in this paper pursues a different approach. Following Kaplan and Bresnan (1982), feature structures are only (components of) linguistic structures, and not partial descriptions of (other) linguistic structures. As such, a feature structure either does or does not satisfy any particular set of constraints. An utterance is well-formed just in case there is some linguistic structure that *satisfies* all of the constraints imposed by the grammar and has the phonological form of that utterance as its phonological form (which itself is just another constraint that the structure must satisfy). Since the relationship between a feature structure and a constraint that it satisfies is essentially the same as the relationship between an interpretation and a formula that is true under that interpretation, it seems natural to conceive of a constraint as a kind of formula (in a format that allows efficient computational manipulation) that has feature structures as its intended interpretations.

This approach is more general in that it does not rely on the upward-closure property, and it allows constraints on feature structures to have a structure quite different from the feature structures that they constrain. The subsumption relation on

feature structures plays no special role in this approach; specifically, it is not required that the set of structures that satisfy a constraint be upward-closed.

In general, a linguistic structure S must satisfy several constraints, say $\alpha_1, \ldots, \alpha_n$, in order to be well formed, so in order to solve the recognition and parsing problems, all we need do is determine if there are any S that satisfy $\alpha_1, \ldots, \alpha_n$, and if so, describe them somehow.

It is convenient to devise a language for expressing constraints, so that the α_i are well-formed formulae of this language, and its satisfaction relation is exactly the satisfaction relation mentioned above. Viewed from this perspective, the problem of determining if there is a structure S that satisfies the constraints $\alpha_1, \ldots, \alpha_n$ is the same as the problem of determining if the formula α is satisfiable, where α is $\alpha_1 \wedge \cdots \wedge \alpha_n$ and conjunction is given the standard interpretation. Algorithms for deciding the satisfiability of arbitrary formulae in this language (if they exist) can therefore be used to determine the satisfiability of the linguistic constraints. Moreover, if $\alpha \models \alpha'$ then α' is a true description of every model of α, i.e., the logical consequences of α are descriptions of every well-formed linguistic structure that satisfies the constraints. Thus the logic of the constraint language provides in principle all the necessary tools for determining if a set of constraints are satisfiable, and if they are, providing descriptions of the satisfying structures.

From this perspective, an "information state" is a kind of formula, and "unifying" two such information states is accomplished by conjoining them and simplifying the resulting formula, not by some manipulation of their models. Partial information states are those that are satisfied by more than one interpretation. The consequence relation corresponds to the subsumption relation of traditional unification grammar (a formula α "contains more information" than formula α' iff $\alpha \models \alpha'$), and unsatisfiability corresponds to unification failure.

3. Languages for Expressing Feature Structure Constraints

There are many different possible constraint languages. Specialized languages can be constructed specifically for the task of expressing feature structure constraints (such as Kasper and Rounds's FDL [Kasper and Rounds 1990] and Johnson's attribute-value languages [Johnson 1988]). Alternatively, the constraints may be able to be expressed in some standard language, so that the satisfiability problem for linguistic constraints is reduced to the satisfiability problem for that language, as is done here.[4]

Johnson (1990a), following a suggestion first made in Kaplan and Bresnan (1982), showed how attribute-value constraints could be formalized in the quantifier-free subset of first-order logic, while later work (Johnson 1991a, 1991b) proposed a different formalization in the Schönfinkel-Bernays' subset of first-order formulae.[5]

Roughly speaking, there is a trade-off between the expressive power of a language and its computational tractability. For example, the satisfiability problem for the language consisting of conjunctions of equalities and inequalities of first-order terms can

4 A third approach, developed by Smolka (1992), is to define a specialized language tailored for expressing attribute-value constraints and note its translation into some standard language, in this case, also the Schönfinkel-Bernays' class.

5 Of course, there is no a priori reason for these subsets of first-order logic to be optimally suited for expressing feature structure constraints. Kasper and Rounds (1990) and more recently Blackburn (1991) and Blackburn and Spaan (1992) have suggested that it may be useful to express feature structure constraints in a special kind of modal logic. Johnson (1991b) also discusses the application of general first-order logic and nonmonotic logics to the specification of more complex constraints on feature structures.

be decided in quasi-linear time using the congruence-closure algorithm, but this language can only express conjunctions of feature-value equalities and inequalities. If this language is extended to allow disjunctions (so that disjunctive feature-value constraints can be expressed), the satisfiability problem becomes NP-complete (Gallier 1986; Kasper and Rounds 1990; Nelson and Oppen 1980).

Since disjunctive constraints seem to be a practical necessity for describing natural languages (Barton, Berwick, and Ristad 1987; Karttunen 1984), most practical feature structure systems will probably have NP-hard satisfiability problems. Given that we have to solve an NP-hard problem anyway, it seems reasonable to investigate the most expressive feature structure constraint language that has an NP-complete satisfiability problem. The Schönfinkel-Bernays' class, used in the manner described here, appears to be the most expressive language for feature structure constraints proposed in the literature so far whose satisfiability problem is no harder than NP.

3.1 The Schönfinkel-Bernays' Class

The Schönfinkel-Bernays' class (hereafter SB) is the class of first-order closed prenex formulae without function symbols in which no existential quantifier occurs in the scope of any universal quantifier. That is, a formula is in SB iff it has no free variables and is of the form

$$\exists v_1 \ldots \exists v_m \; \forall x_1 \ldots \; \forall x_n \alpha,$$

where α contains no quantifier symbols or function symbols. SB formulae are a proper subset of first-order formulae, and they are interpreted in exactly the same way as first-order formulae. The body α may contain boolean connectives (including negation), which can be used to express arbitrary boolean combinations of constraints.

Unlike the satisfiability problem for full first-order logic, which is undecidable (co-recursively enumerable), the satisfiability problem for SB is decidable; in fact it is PSPACE-complete (Lewis and Papadimitriou 1981). Further, if SB_n is the class of SB formulae with n or fewer universal quantifiers, then for any fixed n the satisfiability problem for SB_n is NP-complete (Lewis 1980). In the applications described here, the number of universal quantifiers is fixed (i.e., it does not vary with the utterance or even with the grammar), so the corresponding satisfiability problems are all NP-complete.

The class of SB formulae is interesting for other reasons besides its ability to express a wide range of linguistic constraints. As shown below, the class of SB formulae in clausal form constitute an extension of Datalog that allows disjunctive consequents.

3.2 Formalizing Attribute-Value Structures Using SB

SB is both simple and expressive enough that grammar designers might choose to state linguistic constraints directly in SB, rather than in terms of attributes and values. Nevertheless, it is important to understand how the properties of attribute-value structures can be stated in SB, since many of the techniques used to formalize them can be applied to other linguistically interesting structures as well.

In fact there are several ways of formalizing attribute-value structures in SB, all of which seem to be linguistically equivalent. What follows is a formalization in SB that allows values to be used as attributes and allows attributes to be quantified over (this is handy for stating "sort constraints"), but no special claims are made for it over and above any other SB formalization.

Following Johnson (1991b), attribute-value feature structures can be specified in SB in the following way. We can conceptualize of attribute-value arcs as instances of

a three-place relation *arc*, where $arc(x, a, y)$ means that there is an arc leaving node x labeled a pointing to node y.[6]

Of course, not all interpretations qualify as attribute-value structures; e.g., those which satisfy both $arc(x, a, y)$ and $arc(x, a, z)$ for some $y \neq z$ violate the requirement that there is at most one arc with any given label leaving any node. We can express this requirement as an SB formula that is true in the intended interpretations (namely attribute-value feature structures).

$$\forall x\ \forall a\ \forall y\ \forall z\ \ arc(x, a, y) \wedge arc(x, a, z) \rightarrow y = z. \tag{1}$$

Similarly, we can express the properties of the "attribute-value constants" with SB formulae. Let *con* be a property (i.e., a one-place relation) true of the "attribute-value constant" elements. These elements are required to have no arcs leaving them. The following formula expresses this requirement.

$$\forall x\ \forall a\ \forall y\ \ \sim (con(x) \wedge arc(x, a, y)). \tag{2}$$

Note that the word "constant" in the name "attribute-value constant" is misleading here, since in this framework not all SB constant symbols will denote attribute-value "constants." More precisely, being an 'attribute-value constant' is a property of an individual in an interpretation (i.e., an element of a feature structure), whereas being a constant is a property of a symbol in a formula. Constants can be used to denote complex attribute-value entities as well as attribute-value constants.

Finally, we require that the names of attribute-value constants denote distinct attribute-value constants. We reserve a finite subset N of the constants of our language for use as the names of attribute-value constants, and require that they satisfy the following schemata.[7]

$$\text{For each } c \text{ in } N,\ con(c). \tag{3}$$

$$\text{For each distinct pair } c_1, c_2 \text{ in } N,\ c_1 \neq c_2. \tag{4}$$

Schema (3) requires each symbol in N to denote an attribute-value constant, and schema (4) enforces distinctness in essentially the same manner as that used in the specification systems of algebraic data-type theory (Kapur and Musser 1987).

Formulas (1) and (2) and the instances of schemata (3) and (4) can be regarded as *defining* attribute-value feature structures. These axioms are quite permissive: in

6 Johnson (1991a) and Smolka (1992) propose that an attribute-value arc labeled a from x to y be conceptualized as an instance of a two-place relation $a(x, y)$. For most applications there is little substantive difference between these two approaches; the approach taken here allows attributes to be quantified over, e.g., to state sortal constraints, and permits values to be used as attributes, as in e.g., LFG (Kaplan and Bresnan 1982); for discussion and linguistic applications see also Johnson (1988).

7 As Patrick Blackburn (p.c.) points out, one consequence of this is that every model of these constraints will contain individuals corresponding to each attribute-value constant (since each constant symbol will be assigned a denotation). Whether this is desirable or problematic is debatable, but as he pointed out, it is easy to devise a conceptualization in which each attribute-value constant c_i is conceptualized as a one-place predicate $c_i(\cdot)$ that is true of at most one element. Under such a conceptualization (which can be formalized in SB as shown below) attribute-value constants would be the unique members of one-element sorts.

(i)	For each c in N, $\forall x\ \forall y\ c(x) \wedge c(y) \rightarrow x = y$.	(*Uniqueness*)
(ii)	For each c in N, $\forall x\ c(x) \rightarrow con(x)$.	(*Constant property*)
(iii)	For each distinct pair c_1, c_2 in N, $\forall x \sim (c_1(x) \wedge c_2(x))$.	(*Disjointness*)

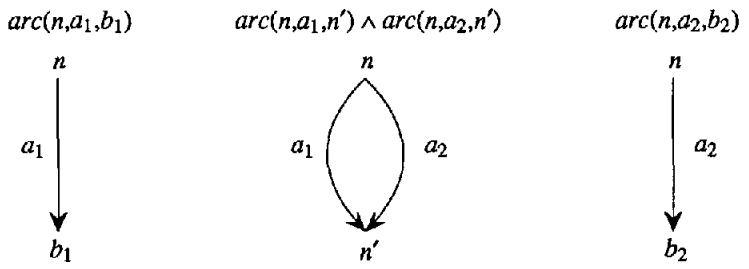

Figure 1
Three constraints expressed as formulae and also depicted graphically.

addition to the usual finite acyclic feature structures, they allow infinite structures, cyclic structures, structures in which complex values serve as attributes, etc. While ruled out by fiat in standard treatments, admitting these additional structures causes no linguistic difficulties that I am aware of (in fact, some analyses crucially depend on their existence, as described in section 2.1.3 of Johnson [1988]), so in the interests of parsimony additional constraints that forbid them are not stipulated.

In fact, because SB formulae possess the finite model property (i.e., if an SB formula has a model, then it has a finite model), restricting attention to finite models does not change the set of satisfiable SB formulae. Therefore it could have no effect on the set of well-formed utterances. Cyclic feature structures can be prohibited with a constraint formalizable in SB, as described in Johnson (1991b), and one can express a constraint in SB that requires that all attributes are "attribute-value constants" (even though there appears to be no linguistic motivation for such a constraint, and indeed, some analyses crucially depend on this not being the case, as pointed out in Johnson [1988]).

To summarize, the simplest SB axioms defining attribute-value structures are quite permissive, allowing a wider range of structures to count as attribute-value structures than many other formalizations. However, all of the major restrictions on attribute-value structures discussed in the literature either have no effect whatsoever in this framework, or else can be directly stated as additional SB constraints.

3.3 Expressing Feature Structure Constraints with SB

In this approach, simple attribute-value constraints are represented by quantifier-free atomic formulae. For example, a constraint that the value of n's a_1 arc is b_1 would be represented by the atom $arc(n, a_1, b_1)$, a constraint that the value of n's a_2 arc is b_2 is represented by $arc(n, a_2, b_2)$, and a constraint that the value of n's a_1 arc is the same as the value of its a_2 arc is represented by the conjunction $arc(n, a_1, n') \wedge arc(n, a_2, n')$ (n' is the single value of both arcs). These three constraints are depicted graphically in Figure 1. Note that the graphs in this figure are (depictions of) formulae, not attribute-value feature structures.

Attribute-value "unification" is the conjunction and simplification of the formulae expressing the constraints to be unified. If all three constraints in the example of Figure 1 are conjoined together with axioms (1–3) above, then by (1) it follows that $b_1 = n' = b_2$. Further, if b_1 and b_2 are distinct constant symbols in N (thus they name attribute-value constants), then $b_1 \neq b_2$ is an instance of (4), and the conjunction is therefore unsatisfiable. For further examples and a discussion of how the disjunction and negation of attribute-value constraints are transparently representable as SB formulae, see Johnson (1991a, 1991b).

A major motivation for using SB is that a wide variety of constraints, in addition to standard attribute-value constraints, can be expressed using it. This allows a grammar developer to introduce a wide variety of "designer features" with possibly idiosyncratic, customized properties, while guaranteeing that the composite system is decidable (usually in NP-time, as noted above).

For example, suppose we want to impose *sort* restrictions of the following kind. To abbreviate the lexical entries of verbs we might introduce the one-place predicate *3rd-sg*, where $3rd\text{-}sg(x)$ indicates that the value of x's *person* attribute is *3rd* and x's *number* attribute is *singular*. This constraint can be expressed using the following SB formula.

$$\forall x\ 3rd\text{-}sg(x) \leftrightarrow arc(x, person, 3rd) \wedge arc(x, number, singular). \tag{5}$$

Similarly, constraints that restrict the possible values of certain attributes can be imposed. For example, one might want to require that the value of every arc labeled *number* is either singular or plural. This constraint can be expressed as the following SB formula.

$$\forall x\ \forall y\ arc(x, number, y) \rightarrow y = singular \vee y = plural. \tag{6}$$

These examples demonstrate only a small fraction of the variety of the feature structure constraints that can be expressed in SB. Even though all of these examples are based on attribute-value features, other sorts of features can be described in SB as well. For example, Johnson (1991a) shows how to formulate a variety of constraints on 'set-valued' features in SB.

3.4 Expressing Tree Structure Constraints with SB Formulae

Inspired by the work on description theory or 'D-theory' (Marcus, Hindle, and Fleck 1983; Vijay-Shanker 1992), this section shows how some elementary constraints on precedence and dominance in a tree can be expressed as SB formulae. It differs from that work in that different kinds of constraints are expressible (Vijay-Shanker was concerned with the formalization of a different kind of grammar), and that all of the constraints expressible in the system described below are decidable (this follows from the fact that they are defined and expressed using Schönfinkel-Bernays' formulae). These constraints are intended to appear as annotations on phrase structure rules (in the same way that attribute-value constraints do) and could be used to enforce a variety of "long-distance" relationships, such as the co- and contra-indexing constraints of binding theory (i.e., equality and inequality constraints on the values of *index* attributes).

The axiomatization begins by defining the primitive tree structure relations *precedes* and *dominates*. Once these primitive tree structure relations are defined, they can be used to approximate more complex relationships such as *c-commands*, as described below. All of these axioms are in the Schönfinkel-Bernays' class, so the satisfiability of arbitrary boolean combinations of such constraints is decidable.

First, note that the standard definition of trees in terms of the binary relations $<$ (linear precedence) and D (domination) can be expressed directly as Schönfinkel-Bernays' formulae. The axioms presented below are just the definitions of trees given in Partee, ter Meulen, and Wall (1990) and Wall (1972) using the syntax of first-order logic. Axioms (7a-c) require that $<$ is a strict partial order, and axioms (8a-c) require

that D is a weak partial order over the nodes in a tree. In what follows, $N(x)$ is interpreted as meaning that x is a tree node.

$$\forall x\ \neg x < x \qquad \textit{(irreflexivity)} \qquad (7a)$$
$$\forall x\ \forall y\ x < y \rightarrow \neg y < x \qquad \textit{(asymmetry)} \qquad (7b)$$
$$\forall x\ \forall y\ \forall z\ x < y \wedge y < z \rightarrow x < z \qquad \textit{(transitive closure)} \qquad (7c)$$
$$\forall x\ D(x,x) \leftrightarrow N(x) \qquad \textit{(reflexivity)} \qquad (8a)$$
$$\forall x\ \forall y\ D(x,y) \wedge D(y,x) \rightarrow x = y \qquad \textit{(antisymmetry)} \qquad (8b)$$
$$\forall x\ \forall y\ \forall z\ D(x,y) \wedge D(y,z) \rightarrow D(x,z) \qquad \textit{(transitive closure)} \qquad (8c)$$

Axiom (9) requires that there is a node that dominates all other nodes, and axiom (10) requires that for each pair of nodes either one precedes the other or one dominates the other. Axiom (11) enforces the "no tangling" constraint.

$$\exists\ x\ N(x) \wedge \forall y\ N(y) \rightarrow D(x,y) \qquad \textit{(single root condition)} \qquad (9)$$
$$\forall x\ \forall y\ N(x) \wedge N(y) \rightarrow ((x < y \vee y < x) \leftrightarrow \neg(D(x,y) \vee D(y,x))) \qquad \textit{(exclusivity)} \qquad (10)$$
$$\forall w\ \forall x\ \forall y\ \forall z\ w < x \wedge D(w,y) \wedge D(x,z) \rightarrow y < z \qquad \textit{(nontangling condition)} \qquad (11)$$

Finally, the following axioms (implicit in the standard treatments cited above) require the precedence and dominance relations to range over tree nodes.

$$\forall x\ \forall y\ x < y \rightarrow N(x) \wedge N(y) \qquad (12a)$$

$$\forall x\ \forall y\ D(x,y) \rightarrow N(x) \wedge N(y) \qquad (12b)$$

This concludes the specification of linear precedence and dominance relations over nodes. We now turn to the specification of other relations in terms of these. The *proper dominance* relation P can be defined in terms of dominance as follows.

$$\forall x\ \forall y\ P(x,y) \leftrightarrow x \neq y \wedge D(x,y). \qquad (13)$$

However, many interesting linguistic relations cannot be defined by Schönfinkel-Bernays' axioms. For example, the *c-commands* relation C is defined by the following formula (which says that x c-commands y iff x does not dominate y, and every node z that properly dominates x also properly dominates y).

$$\forall x\ \forall y\ C(x,y) \leftrightarrow \neg D(x,y) \wedge \forall z\ P(z,x) \rightarrow P(z,y). \qquad (14)$$

It is easy to see that this definition is not equivalent to a Schönfinkel-Bernays' formula by expanding the equivalence into two implications and moving the embedded quantifier out.

$$\forall x\ \forall y\ \forall z\ C(x,y) \rightarrow (\neg D(x,y) \wedge (P(z,x) \rightarrow P(z,y))). \qquad (14a)$$

$$\forall x\ \forall y\ \exists z\ (\neg D(x,y) \wedge P(z,x) \rightarrow P(z,y)) \rightarrow C(x,y). \qquad (14b)$$

Formula (14b) is not in SB because it contains an existential quantifier inside the scope of a universal quantifier. There are a number of ways to respond to this problem.

First, we can abandon the attempt to work within the Schönfinkel-Bernays' class, and work with some other language. Rounds (1988) describes such a language called LFP, whose decidability follows from the fact that the domain of quantification is

restricted (just as in SB). However, it seems to be difficult to devise a decidable system capable of simultaneously expressing both tree structure and the variety of feature structure constraints that the SB approach described here can. Blackburn, Gardent, and Meyer-viol (1993) introduce a modal language L^T for describing trees decorated with feature structures, whose satisfiability problem is undecidable. In the long run, such specialized "designer logics" may provide the most satisfying integration of tree structure and feature structure constraints.

Second, the 'one-sided' approximation (14a) can be used in place of the correct axiom (14). The effect of using such one-sided approximations was investigated in Johnson (1991a). It was shown there that if ξ is a formula such as the one in (14) and ξ' is the one-sided approximation (14a), then for any formula $\varphi^+(C)$ in which C only appears positively, $\xi \wedge \varphi^+(C)$ is satisfiable iff $\xi' \wedge \varphi^+(C)$ is satisfiable. That is, if we are concerned only with positively occurring constraints, we can simplify (14) to (14a), i.e., ignore (14b), without affecting constraint satisfiability.

Third, we can regard formulae such as (14) as the "macro" (15), used to expand constraints at the interface between the syntactic rules and the constraint solver. This "macro expansion" rewrites *c-commands* constraints into boolean combinations of constraints that the constraint solver can handle.

$$C(x,y) \Rightarrow \neg D(x,y) \wedge \forall z\ P(z,x) \rightarrow P(z,y). \tag{15a}$$

$$\neg C(x,y) \Rightarrow D(x,y) \vee \exists z\ (P(z,x) \wedge \neg P(z,y)). \tag{15b}$$

The second and the third approaches differ in important ways. In the second approach, *c-commands* is a relation that is "understood" by the constraint solver (albeit only in its one-sided form), so it can be used to define other relations. In the third approach, *c-commands* constraints are not primitive constraints, so relations defined in terms of *c-commands* must also be expressible in terms of "macro expansion." In the second approach, constraints are quantifier-free formulae (quantifiers appear only in the axioms), so the satisfiability problem is in NP. But in the third approach, macro expansion produces formulae that contain additional quantifiers, so the satisfiability problem may be PSPACE-complete.

3.5 Limitations on Constraints Expressible with SB Formulae

Of course, SB is not as expressive as full first-order logic. It is incapable of expressing *functional* relationships, since these require an existential quantifier inside the scope of a universal quantifier. This means, among other things, that it is impossible to state a constraint in SB requiring that a certain node must exist (as was noted in the discussion of *c-command* in the previous section) or that all nodes possess certain attributes. Thus for example, the following constraint, which requires that every *tensed* entity possess *number* and *person* attributes, is a first-order formula that is not in SB, since it requires a functional relationship between entities with *tense* attributes and the values of their *number* and *person* attributes.

$$\forall x\ \forall y\ arc(x, tense, y) \rightarrow (\exists z\ arc(x, number, z)) \wedge (\exists z\ arc(x, person, z)) \tag{16}$$

Similarly, a number of other extensions to the basic attribute-value framework discussed in the literature cannot be formalized in SB. *Subsumption* constraints, used in the treatment of (natural language) conjunction, are not expressible as SB formulae because the satisfiability problem for conjunctions of subsumption and attribute-value constraints is undecidable (Dörre and Rounds 1992). Positively occurring *functional*

(E1) $\forall x\ x = x.$

(E2) $\forall x\ \forall y\ x = y \rightarrow y = x.$

(E3) $\forall x_0 \ldots \forall x_n\ x_j = x_0 \wedge P(x_1, \ldots, x_j, \ldots, x_n) \rightarrow P(x_1, \ldots, x_0, \ldots, x_n)$
for $j = 1, \ldots, n$, for every predicate symbol P appearing in φ.

(E4) $\forall x_0 \ldots \forall x_n\ x_j = x_0 \rightarrow f(x_1, \ldots, x_j, \ldots, x_n) = f(x_1, \ldots, x_0, \ldots, x_n)$
for $j = 1, \ldots, n$, for every function symbol f appearing in φ.

Figure 2
Equality axiom schemata for a first-order formula φ.

uncertainty constraints, used in the LFG treatment of long-distance dependencies (Kaplan and Zaenen 1989) appear to have a decidable satisfiability problem (Kaplan and Maxwell 1988a), but the satisfiability problem for arbitrary boolean combinations of functional uncertainty constraints is undecidable (Keller 1991), so these cannot be expressed using SB formulae either (since the quantifier-free subclass of SB is closed under boolean operations).

3.6 The Equality Relation

In this paper the intended interpretation of the equality relation is identity; i.e., $a = b$ if and only if a and b denote the same individual. However, for some purposes (e.g., in the least-fixed-point characterization of minimal models given below) this "special" interpretation of the equality complicates matters, and it is more convenient to treat the equality relation as a "normal" relation that is defined by a set of axioms E.

The idea is that E has the property that a formula φ is satisfiable under the identity interpretation of equality if and only if $\{\varphi\} \cup E$ is satisfiable in an interpretation in which equality is not given any special treatment. In effect, the axioms E require that the equality relation denotes an equivalence relation, and permit the substitution of equals for equals. Together these imply that no predicate can distinguish equal individuals. This means that in terms of satisfiability and the consequence relation, exactly the same results are obtained irrespective of whether equality is treated as identity or defined by the axioms E.

Such treatments of equality in first-order logic are well known and described in standard texts. For example, Chang and Lee (1973) give the axiom schemata in Figure 2, which generates syntactic equality axioms E for a first-order formula φ, and prove that E has the properties just described.[8]

What is important here is that for an SB formula φ the instances of the axiom schemata are all SB formulae, and there are only finitely many instances of these schemata.

This means that for an arbitrary SB formula φ there is another SB formula ξ such that φ is satisfiable with respect to an identity interpretation of equality if and only if $\varphi \wedge \xi$ is satisfiable with respect to an interpretation in which equality is treated like any other relation. Thus a method for determining the satisfiability of SB formulae without equality can be used to determine satisfiability of SB formulae in which equality is interpreted as identity.

8 Of course, (E4) has no instances if φ is an SB formula, since SB formulae do not contain function symbols.

(17) $\forall x\ x = x.$

(18) $\forall x\ \forall y\ x = y \rightarrow y = x.$

(19) $\forall x\ \forall a\ \forall y\ \forall x_1\ x = x_1 \wedge arc(x,a,y) \rightarrow arc(x_1,a,y).$

(20) $\forall x\ \forall a\ \forall y\ \forall a_1\ a = a_1 \wedge arc(x,a,y) \rightarrow arc(x,a_1,y).$

(21) $\forall x\ \forall a\ \forall y\ \forall y_1\ y = y_1 \wedge arc(x,a,y) \rightarrow arc(x,a,y_1).$

(22) $\forall c_1\ \forall c_2\ c = c_1 \wedge con(c) \rightarrow con(c_1).$

(23) $\forall x\ \forall y\ x = y \wedge 3rd\text{-}sg(x) \rightarrow 3rd\text{-}sg(y).$

Figure 3
The equality axioms for *arc*, *con*, and *3rd-sg* predicates.

For example, consider the SB formulae in (1–4) and (5–16). These contain the three-place relation symbol *arc* and the one-place relation symbols *con* and *3rd-sg*. The equality axioms obtained from schemata (E1–E4) for any system of constraints that mention just these relations are given in Figure 3.

4. Clausal Form and Disjunctive Datalog

It is technically easier to work with a syntactically restricted class of SB formulae where the body of each formula has a particular syntactic form known as *clausal form* or *Skolem standard form*.

Definition
A **clause** is a formula of the form $\sim \alpha_1 \vee \cdots \vee \sim \alpha_m \vee \beta_1 \vee \cdots \vee \beta_n$, where each α_i and β_j is an atomic formula (i.e., is of the form $p(t_1,\ldots,t_n)$), and $m,n \geq 0$. A formula φ is in clausal form iff it is a conjunction of clauses.

The $\sim a_i$ are called **negative literals** and the β_j are called **positive literals**. A clause for which $m = 0$ (i.e., one that consists solely of positive literals) is called a **positive clause**, and one for which $n = 0$ (i.e., one that consists solely of negative literals) is called a **negative clause**. A clause for which $n = 1$ is called a **definite clause**. A **Horn clause** is a clause for which $n \leq 1$, i.e., either a negative clause or a definite clause.

Abusing notation somewhat, a formula φ in clausal form will sometimes also be treated as the set of the clauses that make up the conjunction φ. Similarly, because clauses are used as rewriting rules below, the clause $\sim \alpha_1 \vee \cdots \vee \sim \alpha_m \vee \beta_1 \vee \cdots \vee \beta_n$ will sometimes be written as the equivalent implication $\alpha_1 \wedge \cdots \wedge \alpha_m \rightarrow \beta_1 \vee \cdots \vee \beta_n$.

A formula φ in clausal form does not contain any quantifier symbols. As is standard, all variables in φ are treated as implicitly universally quantified at the clausal level. Existentially quantified variables in SB formulae are inessential, in that they can always be directly replaced by Skolem constants.

Restricting attention to SB formulae in clausal form imposes no real restriction on the class of constraints expressible. Standard procedures for transforming first-order formulae into clausal form, such as the ones described in Chang and Lee (1973), Duffy

(1991), or Genesereth and Nilsson (1987), transform SB formulae into SB formulae in clausal form.

Interestingly, clausal form SB formulae correspond one-to-one with an extension to Datalog (Ullman 1988) that allows disjunctive "heads" or consequences. In notation borrowed from disjunctive logic programming (Kowalski 1979; Lobo, Minker, and Rajasekar 1992; Loveland 1987), the clauses above would be written as listed in the appendix. While the connection between logic programming and feature structures is well known (Aït-Kaci 1984; Aït-Kaci and Podelski 1993; Carpenter 1991, 1992; Höhfeld and Smolka 1988; Pereira 1987; Shieber 1992; Smolka 1992), this shows that the theory of feature structure constraints is also related to database theory as well.

Negative clauses correspond to Datalog integrity constraints, and clauses with a single positive literal are definite clauses. Simple assertions, e.g., about the existence of arcs, consisting of exactly one positive literal are Datalog atomic clauses. Clauses with two or more positive literals cannot be expressed in Datalog itself, but require the disjunctive extension of Datalog. The appendix displays all of the SB formulae mentioned in this paper so far in clausal form in Datalog notation; (6′), (10′) and (7″) are expressed in the disjunctive extension to Datalog. In fact, the axioms defining attribute-value structures (1–4) and syntactic equality (E1–E3) are all Horn Datalog clauses; i.e., the disjunctive extension is not needed for defining attribute-value feature structures.

5. Determining the Satisfiability of SB Formulae

This section describes a forward-chaining algorithm for determining the satisfiability of SB formulae in clausal form. This algorithm is a nondeterministic variant of the semi-naive evaluation method for Datalog clauses in which the union-find algorithm is used to efficiently maintain equivalence classes of equal terms. It is also recognizable as a generalization of the standard unification algorithm for feature structures to arbitrary Horn SB constraints.[9] The treatment is informal because the goal of the section is to point out several important standard implementation techniques rather than to advance a totally new algorithm.

The key intuition behind the algorithm is this. To demonstrate the satisfiability of a set S of clauses, it is sufficient to exhibit a set A of ground atoms drawn from the Herbrand base of S such that the following conditions hold (the next section proves this assertion).

(a) For no ground instance $\sim \alpha'_1 \vee \cdots \vee \sim \alpha'_m$ of any negative clause in S are all of $\alpha'_1, \ldots, \alpha'_m$ in A. (If they were, then that clause would be falsified by A.)

(b) For each ground instance $\beta'_1 \vee \cdots \vee \beta'_n$ of a positive clause in S, at least one of the β'_i is in A.

(c) For each ground instance $\alpha'_1 \wedge \cdots \wedge \alpha'_m \rightarrow \beta'_1 \vee \cdots \vee \beta'_n$ of an implication in S, if all of the $\alpha'_1, \ldots, \alpha'_m$ are in A then so is at least one of the $\beta'_1, \ldots, \beta'_n$. (In fact, the other two conditions are just special cases of this condition.)

9 It is a generalization of the algorithm described in Hegner (1991), which treats Horn combinations of attribute-value constraints.

5.1 Naive Evaluation

One could attempt to find such a set A in the following manner. First, one nondeterministically selects a β'_i from each of the ground instances of the positive clauses in S and adds these to A. Then one attempts to close A with respect to condition (c); if all of the antecedents $\alpha'_1, \ldots, \alpha'_m$ of some (ground instance of an) implication are in A, then one of the consequents $\beta'_1, \ldots, \beta'_n$ is nondeterministically selected and added to A, unless at least one of them is already present. (Of course, all such nondeterministic paths might have to be investigated.) Periodically, condition (a) is checked; if it fails to hold, then this nondeterministic path on the search for A must be abandoned. Nondeterminism arises solely from the presence of disjunction in consequents of clauses; if S is a set of Horn clauses then the fixed-point calculation proceeds deterministically.

Ignoring the checking of condition (a), the method is essentially computing a fixed-point of the nonnegative clauses in S via a kind of iterative approximation known as *naive evaluation*. Naive evaluation is unnecessarily computationally inefficient. Once the set A is large enough to require an atom α to be added to A, naive evaluation "rediscovers" this requirement on all subsequent passes.

5.2 Semi-Naive Evaluation

Semi-naive evaluation avoids rediscovering the same fact in the same way by insisting that each time a clause is applied at least one of the antecedents was just discovered on the previous round (Ullman 1988, 1989). This is done by maintaining two sets of atoms, A and ΔA, where A is the set of atoms discovered one or more iterations ago, and ΔA is the set of atoms discovered at the last iteration. The nondeterministic semi-naive algorithm for computing a set A (if it exists) is sketched in Figure 4. In that algorithm *choose* is a "function" that nondeterministically picks one member from its set argument; it can be implemented using, e.g., backtracking. Ullman describes methods of matching clauses in S against the sets A and ΔA that avoids calculating all of the ground instances of the clauses in S.

The semi-naive algorithm can be used directly with the syntactic equality axioms given in Section 3.4 as a decision procedure for SB formulae, and hence for systems of feature structure constraints. However, the resulting system is inefficient because the equality axioms, specifically the instances of schemata (E2) and (E3), cause the "copying" of any atom containing an argument that appears in an equality atom to all members of the equivalence class containing that argument.

For example, if $p(a), q(b)$ and $a = b$ are atoms in A, then instances of (E2) and (E3) ensure that $p(b), q(a)$ and $b = a$ will be added to ΔA and thence to A. In general, if it is discovered that n constants $a_1, \ldots a_n$ are equal, then A will ultimately contain the n^2 equalities $a_i = a_j$, $1 \leq i \leq n$, $1 \leq j \leq n$, as well as at least n "copies" of any predicate containing any a_i.

5.3 Union-Find and Equality

As noted above, the equality axioms ensure that the relation that the equality symbol denotes is an equivalence relation and the substitutivity of equals for equals. In general, the *union-find* algorithm (Corman, Leiserson, and Rivest 1990; Gallier 1986; Nelson and Oppen 1980) maintains the equivalence classes of the equality relation far more efficiently than an approach that uses the syntactic equality axioms.

The equivalence classes are encoded by associating each constant with a pointer that is either null or points to another constant, where a points to b only if $a = b$. These pointer correspond exactly to the "invisible pointers" used in standard implementations of the attribute-value unification algorithm.

Input: A set of SB clauses S.

Output: A set of ground clauses A iff S is satisfiable.

$A := \emptyset$,

$\Delta A := \{choose(\{\beta'_1, \ldots, \beta'_n\}) : \beta'_1 \vee \cdots \vee \beta'_n$ is a ground instance of a positive clause in $S\}$,

until $\Delta A := \emptyset$ *do*

$A := A \cup \Delta A$,

if $\{\alpha'_1, \ldots, \alpha'_m\} \subseteq A$, where $\sim \alpha'_1 \vee \cdots \vee \sim \alpha'_m$ is a ground instance of a negative clause in S and at least one of the α'_i is in ΔA,

then fail,

$\Delta A := \{choose(\{\beta'_1, \ldots, \beta'_n\}) : \alpha'_1 \wedge \cdots \wedge \alpha'_m \rightarrow \beta'_1 \vee \cdots \vee \beta'_n$ is a ground instance of an implication in S such that $\{\alpha'_1, \ldots, \alpha'_m\} \subseteq A$, at least one of the α'_i is in ΔA and no β'_j is in $A\}$,

return A.

Figure 4
The semi-naive algorithm for computing A.

The *find* operation dereferences its argument, i.e., it follows these pointers until it reaches a constant with a null pointer, which is the equivalence classes' representative. Just as in the standard attribute-value unification algorithm, all arguments are always dereferenced before they are used.

The *union* operation, called whenever an atom $a = b$ is added to the set A, merges their equivalence classes by redirecting the pointer associated with the representative of one of them to point to the representative of the other.[10]

In this approach, only atoms that contain the redirected constant need to be added to ΔA and thence to A. For example, if $p(a)$ and $q(b)$ are atoms in A and the equality $a = b$ is discovered, causing a to be redirected to b, then only $p(b)$ is added to ΔA, and thence to A. Further, the "original" atom $p(a)$ is no longer required; indeed, the new atom $p(b)$ is exactly an argument-dereferenced variant of the old atom, so it is not necessary to copy the atom at all. In general, equalities between n items are represented by $n - 1$ nonnull pointers, and copying of atoms can be avoided by argument dereferencing.

5.4 An Example

This section presents a very simple example that demonstrates the semi-naive algorithm and the union-find techniques. The clauses used are the attribute-value axiom schemata (1–4) and the axioms defining the sort *3rd-sg* (5), as well as the additional

10 The union-find algorithm achieves quasi-linear running time when it incorporates path compression and union by rank (Corman, Leiserson, and Rivest 1990).

$$S = \left\{ \begin{array}{ll} 3rd\text{-}sg(u), arc(u, number, v), v \neq sg, & \left. \begin{array}{l} \\ \\ \\ \\ \\ \\ \end{array} \right\} \textit{constraints} \\ \forall x\ \forall a\ \forall y\ \forall z\ arc(x,a,y) \wedge arc(x,a,z) \rightarrow y = z, & \textit{AV axioms} \\ \forall x\ \forall a\ \forall y\ \sim con(x) \vee \sim arc(x,a,y), & \\ con(sg), con(pl), con(3rd), & \\ sg \neq pl, sg \neq 3rd, pl \neq 3rd, & \\ \forall x\ 3rd\text{-}sg(x) \rightarrow arc(x, person, 3rd), & \\ \forall x\ 3rd\text{-}sg(x) \rightarrow arc(x, number, sg) & \textit{sort defn.} \end{array} \right.$$

Figure 5
Input clauses.

constraints $3rd\text{-}sg(u)$, $arc(u, number, v)$ and $v \neq sg$. These latter constraints assert the existence of an entity (denoted by u) with the property *3rd-sg* and with a *number* arc whose value v is something other than *pl*. The complete set of clauses is given in Figure 5. In practice the attribute-value axioms would probably not be explicitly enumerated but "built in," so that appropriate instances are generated only when needed.

Now we proceed to iteratively calculate the sets ΔA and A using the algorithm in Figure 4. We calculate the first initial set of new atoms, ΔA_0. $A_0 = \emptyset$, of course.

$$\Delta A_0 = \{3rd\text{-}sg(u), arc(u, number, v), con(sg), con(pl), con(3rd)\}$$

On the first iteration we note that the antecedents of both clauses defining the sort *3rd-sg* are satisfied (with x bound to u), so ΔA_1 is given as follows.

$$\begin{array}{rcl} A_1 & = & \{3rd\text{-}sg(u), arc(u, number, v), con(sg), con(pl), con(3rd)\} \\ \Delta A_1 & = & \{arc(u, person, 3rd), arc(u, number, sg)\} \end{array}$$

In the second iteration the antecedents of the first attribute-value axiom are satisfied, so ΔA_2 contains an equality atom.

$$\begin{array}{rcl} A_2 & = & \left\{ \begin{array}{l} 3rd\text{-}sg(u), arc(u, number, v), con(sg), con(pl), con(3rd), \\ arc(u, person, 3rd), arc(u, number, sg) \end{array} \right\} \\ \Delta A_2 & = & \{v = sg\} \end{array}$$

The equality atom causes v to be redirected to sg, and at this stage the inconsistency of the derived atom $v = sg$ in ΔA_2 with the input constraint $v \neq sg$ in S is detected. (It may be helpful to think of the constraint $v \neq sg$ as the equivalent clause $v = sg \rightarrow false$). The algorithm therefore returns with failure, indicating that the set S is unsatisfiable. At the point at which the inconsistency is detected, the set A contains the following atoms, where $v \Rightarrow sg$ indicates that v is redirected to sg.

$$A_3 = \left\{ \begin{array}{l} 3rd\text{-}sg(u), arc(u, number, v), con(sg), con(pl), con(3rd), \\ arc(u, person, 3rd), arc(u, number, sg), \\ v \Rightarrow sg \end{array} \right\}$$

The correspondence of this procedure to the standard attribute-value unification algorithm is quite strong. In this procedure, the attribute-value axiom (1) detects situations in which some node has two arcs with the same label pointing to, say, y and z. If such

a situation arises, the equality $y = z$ is inferred, which results in y being redirected to z and causes all of the arcs leaving y to be added to ΔA, where they will be compared with the arcs leaving z. The other attribute-value axiom schemata (2–4) detect constant–constant and constant–complex clashes, causing failure if one is found.

Efficient processing demands that the atoms in A be indexed by their arguments to speed up the matching atoms with the antecedents of clauses. One way of doing this is to store on each constant a list of the atoms in which that constant appears. Such an index has the same structure as the standard graph encoding of feature structure constraints.

6. A Fixed-Point Theorem

We now turn to the theoretical justification of the bottom-up forward-chaining procedures sketched in the last section, and show that such methods will find a model for a set of SB formulae in clausal form if one exists. This section demonstrates that an SB formula in clausal form is satisfiable if and only if a bottom-up forward-chaining procedure finds a deductively closed set of atoms A. A similar theorem for the case in which all the clauses are Horn clauses is presented in Lloyd (1984); this section extends that work to arbitrary clauses.

It presents a characterization of the models of an arbitrary first-order formula φ in clausal form in terms of the least-fixed points of a set $\{T_{\varphi,\chi}\}$ of partial functions from Herbrand interpretations to Herbrand interpretations. These functions have the property that A is a Herbrand interpretation that satisfies φ if and only if the least-fixed point of at least one of them is a submodel of A.

For SB formulae this set of functions is finite and the least-fixed points are reached in a finitely bounded number of iterations. Since the procedures described in the last section calculate the least-fixed points of these functions, they can be used to determine the satisfiability of an arbitrary SB formula as well as all of its ground atomic consequences.

The functions $T_{\varphi,\chi}$ play a similiar rôle here to one that the transformation T_P plays in the least-fixed-point semantics of Horn clause programs. Informally, each function in the set $\{T_{\varphi,\chi}\}$ corresponds to one whole sequence of nondeterministic choices of disjuncts in non-Horn clauses that could be made during an iterative approximation of the least-fixed point. This section is based on Sections 5 and 6 of Chapter 1 of Lloyd (1984), to which the reader should turn for further details.

The fixed-point theorem holds for arbitrary first-order formulae in clausal form, but the set $\{T_{\varphi,\chi}\}$ is finite if and only if φ does not contain any function symbols, i.e., φ is an SB formula. Equality is not treated specially, so the formula φ must contain appropriate equality axioms, as mentioned above.

Let U be the Herbrand universe with respect to φ (i.e., the set of all terms that can be constructed using the constant and function symbols appearing in φ),[11] and let B_φ be the set of all ground atoms that can be formed using the predicate symbols of φ with elements of U as arguments. A Herbrand interpretation A is a subset of B_φ.

Note that the set of Herbrand interpretations 2^{B_φ} partially ordered by the subset relation forms a complete lattice. Further, U and hence B_φ are finite iff φ is an SB formula. (If φ is in clausal form but is not an SB formula then it must contain a function symbol, so its Herbrand universe U is infinite.)

11 As is standard, if φ has no constant symbols then it is necessary to take U to be a set consisting of a single constant, say $\{a\}$. See, e.g., Chang and Lee (1973) for details.

Definition
A Herbrand interpretation A **trivially falsifies** a set of clauses φ iff there is a ground instance $\sim \alpha'_1 \vee \cdots \vee \sim \alpha'_m$ of a negative clause $\sim \alpha_1 \vee \cdots \vee \sim \alpha_m$ in φ such that $\{\alpha'_1, \ldots, \alpha'_m\} \subseteq A$.

That is, a Herbrand interpretation trivially falsifies a set of clauses φ just in case some negative clause in φ is false in that interpretation. Clearly, any such interpretation cannot satisfy φ, but the converse does not hold: there are interpretations that do not trivially falsify φ but still do not satisfy φ because they do not satisfy one or more of the nonnegative clauses in φ.

We turn now to the nonnegative clauses in φ. The idea is that even if A does not satisfy a ground instance $\alpha'_1 \wedge \cdots \wedge \alpha'_m \rightarrow \beta'_1 \vee \cdots \vee \beta'_n$ of some nonnegative clause in φ, we can extend A so that it does so by adding one of the β'_i. The chief technical difficulty here is caused by the nondeterminism involved in deciding which of the β'_i to add, and a device called a "choice function" is introduced to choose, for each ground instance of a clause, which of the atoms in its consequent will be added to A if its antecedent is contained in A. A choice function for a clause $\alpha_1 \wedge \cdots \wedge \alpha_m \rightarrow \beta_1 \vee \cdots \vee \beta_n$ is therefore a function from all possible ways of grounding that clause to one of the β_i.

Definition
A **choice function** for a clause $c = (\alpha_1 \wedge \cdots \wedge \alpha_m \rightarrow \beta_1 \vee \cdots \vee \beta_n)$ is any function in $(V_c \rightarrow U) \rightarrow [1, \ldots, n]$, where V_c is the set of variables in the clause c.

That is, a choice function for a clause is a function from variable assignments to an integer representing one of the clause's consequents. It is so named because for each variable assignment (i.e., each way of grounding the variables in φ) it "chooses" an atom from the consequent of the clause. Horn clauses have only one choice function, and negative clauses have no choice functions at all. Note that since U is finite for SB formulae, there are a finite number of variable assignment functions for any SB clause and hence only a finite number of choice functions for any SB clause.

A choice function χ for a set of clauses φ is a function from φ to choice functions such that for each nonnegative clause c in φ, $\chi(c)$ is a choice function for c (the value that χ takes on negative clauses is ignored). Clearly, a choice function exists for every set of clauses.

Given a set of clauses φ and a choice function χ for φ, we define a function $F_{\varphi,\chi}$ from Herbrand interpretations to Herbrand interpretations as follows.

Definition
$F_{\varphi,\chi}$ is a function in $2^{\mathcal{B}_\varphi} \rightarrow 2^{\mathcal{B}_\varphi}$ that is defined as follows.

$$F_{\varphi,\chi}(A) = \{\theta(\beta_i) \quad : \quad \begin{array}{l} \text{for all nonnegative clauses} \\ c = (\alpha_1 \wedge \cdots \wedge \alpha_m \rightarrow \beta_1 \vee \cdots \vee \beta_n) \text{ in } \varphi \\ \text{and for all variable assignments } \theta : V_c \rightarrow U \text{ such that} \\ \theta(\alpha_1), \ldots, \theta(\alpha_n) \in A \text{ and } i = \chi(c)(\theta)\} \end{array}$$

Intuitively, $F_{\varphi,\chi}$ corresponds to one nondeterministic step in the 'bottom-up' construction of a Herbrand model for φ described in the previous section. If A makes the antecedent of some ground instance of some clause in φ true, then we use the choice function χ to pick an atom in the consequent of that ground clause and add it to the interpretation. Different choice functions χ represent different sequences of nondeterministic choices, and result in the construction of possibly different interpretations.

The following lemma, based directly on proposition 6.3 of Lloyd (1984), notes the continuity (and therefore the monotonicity) of $F_{\varphi,\chi}$.

Lemma 1
The function $F_{\varphi,\chi}$ is continuous. That is, if X is a directed subset of 2^{B_φ} (i.e., every finite subset of X has an upper bound in X) then $F_{\varphi,\chi}(\text{lub}(X)) = \text{lub}(F_{\varphi,\chi}(X))$.

Proof
Let X be any directed subset of 2^{B_φ}. Then $\{\alpha_1, \ldots, \alpha_m\} \subseteq \text{lub}(X)$ if and only if $\{\alpha_1, \ldots, \alpha_m\} \subseteq A$ for some $A \in X$. Then

$$
\begin{aligned}
& \beta \in F_{\varphi,\chi}(\text{lub}(X)) \\
\text{iff} \quad & c = (\alpha_1 \wedge \cdots \wedge \alpha_m \rightarrow \beta_1 \vee \cdots \vee \beta_n) \text{ is a nonnegative clause in } \varphi, \\
& \{\theta(\alpha_1), \ldots, \theta(\alpha_n)\} \subseteq \text{lub}(X), \text{ and } \beta = \theta(\beta_{\chi(c)(\Theta)}) \\
\text{iff} \quad & c = (\alpha_1 \wedge \cdots \wedge \alpha_m \rightarrow \beta_1 \vee \cdots \vee \beta_n) \text{ is a nonnegative clause in } \varphi, \\
& \{\theta(\alpha_1), \ldots, \theta(\alpha_n)\} \subseteq A \text{ for some } A \in X, \text{ and } \beta = \theta(\beta_{\chi(c)(\Theta)}) \\
\text{iff} \quad & \beta \in F_{\varphi,\chi}(A) \text{ for some } A \in X \\
\text{iff} \quad & \beta \in \text{lub}(F_{\varphi,\chi}(X)). \qquad \square
\end{aligned}
$$

The continuity of $F_{\varphi,\chi}$ immediately implies the convergence of the sequence $\langle F^i_{\varphi,\chi}(\varnothing)\rangle$; the value that it converges to is called the least-fixed point of $F_{\varphi,\chi}$, written $\text{lfp}(F_{\varphi,\chi})$. Note that if φ is in SB then there is an integer k such that $\text{lfp}(F_{\varphi,\chi}) = F^k_{\varphi,\chi}(\varnothing)$; this follows directly from the monotonicity of $F_{\varphi,\chi}$ and the finiteness of B_φ.

The function $F_{\varphi,\chi}$ and a condition requiring that the interpretation produced does not trivially falsify the set of clauses φ together define the partial function $T_{\varphi,\chi}$.

Definition
$T_{\varphi,\chi}$ is a partial function in $2^{B_\varphi} \rightrightarrows 2^{B_\varphi}$ that is defined as follows.

$T_{\varphi,\chi}(A) = F_{\varphi,\chi}(A)$ if $F_{\varphi,\chi}(A)$ does not trivially falsify φ, and is undefined otherwise.

Note that if the sequence $\langle T^i_{\varphi,\chi}(A)\rangle$ is defined for all i then $\langle T^i_{\varphi,\chi}(A)\rangle = \langle F^i_{\varphi,\chi}(A)\rangle$. $T_{\varphi,\chi}$ enjoys the following kind of monotonicity.

Lemma 2
Suppose $A \subseteq A'$. Then $T_{\varphi,\chi}(A)$ is defined if $T_{\varphi,\chi}(A')$ is defined, and $T_{\varphi,\chi}(A) \subseteq T_{\varphi,\chi}(A')$.

Proof
If A trivially falsifies φ then A' does too, so $T_{\varphi,\chi}(A)$ is defined if $T_{\varphi,\chi}(A')$ is defined. If $T_{\varphi,\chi}(A')$ is defined then $T_{\varphi,\chi}(A) = F_{\varphi,\chi}(A) \subseteq F_{\varphi,\chi}(A') = T_{\varphi,\chi}(A')$. $\square$

The following lemma shows that Herbrand models of φ contain fixed points of $T_{\varphi,\chi}$ for some choice function χ for φ.

Lemma 3
For all Herbrand interpretations A, $A \models \varphi$ iff there exists a choice function χ for φ such that $T_{\varphi,\chi}(A) \subseteq A$.

Proof
We begin first with the left-to-right component of the proof. If $A \models \varphi$, then A does not trivially falsify φ, so $T_{\varphi,\chi}(A)$ is defined. Now we show how to find for each satisfying interpretation A a choice function χ such that $T_{\varphi,\chi}(A) = A$. Since A satisfies φ, for every nonnegative clause $c = (\alpha_1 \wedge \cdots \wedge \alpha_m \rightarrow \beta_1 \vee \cdots \vee \beta_n)$ in φ and for every variable assignment function θ for the variables in c, if $\theta(\alpha_1), \ldots, \theta(\alpha_n) \in A$, then by the truth conditions for implication and disjunction, some $\theta(\beta_i) \in A$ as well. Thus, for all θ such that $\theta(\alpha_1), \ldots, \theta(\alpha_n) \in A$ let $\chi(c)(\theta)$ be any i such that $\theta(\beta_i) \in A$, and let $\chi(c)(\theta)$ take any permissible value otherwise. Hence $T_{\varphi,\chi}(A) = F_{\varphi,\chi}(A) = A$.

Now suppose $T_{\varphi,\chi}(A) \subseteq A$. Since $T_{\varphi,\chi}(A)$ is defined, A does not trivially falsify any negative clause in φ. Let $c = (\alpha_1 \wedge \cdots \wedge \alpha_m \rightarrow \beta_1 \vee \cdots \vee \beta_n)$ be any nonnegative clause in φ, and let $\theta \in V_c \rightarrow U$ be any variable assignment function for the variables in c. If $\theta(\alpha_1), \ldots, \theta(\alpha_n) \in A$ then $\theta(\beta_i) \in T_{\varphi,\chi}(A) \subseteq A$ as well, where $i = \chi(c)(\theta)$, so $A \models c$ and hence $A \models \varphi$. □

The following theorem shows that a formula is satisfiable if and only if the least-fixed point of at least one of the $T_{\varphi,\chi}$ exists. It justifies the decision procedures presented in the previous section, which operate by searching for such least-fixed points.

The proof actually establishes something stronger, viz., that every Herbrand model of φ is an extension of the least-fixed points of one or more of the $T_{\varphi,\chi}$. Thus an enumeration of all of the least-fixed points of the $T_{\varphi,\chi}$ yields all of the "minimal models" of φ (although it is not clear that these are in fact necessary for recognition or parsing, as discussed above).

Theorem
φ is satisfiable if and only if there exists a choice function χ for φ such that $\mathrm{lfp}(T_{\varphi,\chi})$ exists.

Proof
If $\mathrm{lfp}(T_{\varphi,\chi})$ exists then by Lemma 3, $\mathrm{lfp}(T_{\varphi,\chi}) = \varphi$. Now suppose A is a Herbrand interpretation that satisfies φ. Lemma 3 asserts the existence of a choice function χ such that $T_{\varphi,\chi}(A)$ exists and $T_{\varphi,\chi}(A) \subseteq A$. By Lemma 2 and the fixed point property noted above $\mathrm{lfp}(T_{\varphi,\chi})$ exists, since $\mathrm{lfp}(T_{\varphi,\chi}) = T^{\infty}_{\varphi,\chi}(\varnothing) \subseteq T^{\infty}_{\varphi,\chi}(A) \subseteq A$. □

It is important to recognize that these "minimal models" are in general not upward-closed: an extension A' of a model A can trivially falsify φ even though A does not. This is essentially Moshier's (1988) and Pereira's (1987) observation that in the presence of negation the set of models is not upwardly closed.

We conclude this section with the observation that the positive consequences of a formula φ can be "read off" its least-fixed points.

Corollary
φ is satisfiable iff for some choice function χ for φ, $\mathrm{lfp}(T_{\varphi,\chi})$ exists. Moreover, if $\beta = \beta_1 \vee \cdots \vee \beta_n$ is any disjunction of ground atoms, $\varphi \models \beta$ iff for all choice functions χ for φ such that $\mathrm{lfp}(T_{\varphi,\chi})$ exists, at least one of the β_i is in $\mathrm{lfp}(T_{\varphi,\chi})$.

7. Conclusion

The main goal of this paper was to demonstrate from a computational perspective that Schönfinkel-Bernays' formulae are a natural generalization of (boolean combina-

tions of) attribute-value feature structure constraints. From a computational complexity perspective we noted that the satisfiability problem for SB formulae with a bounded number of quantifiers is NP-complete, so it is no harder than the satisfiability problem for disjunctive attribute-value constraints.

From a more practical perspective, a semi-naive bottom-up evaluation strategy using union-find methods to handle equality generalizes the standard attribute-value "unification" algorithm to arbitrary SB constraints in clausal form. Because it treats standard attribute-value constraints in approximately the same way as the standard unification algorithm, and because it can incorporate the same kinds of indexing that the latter algorithm employs, the generalized algorithm should be able to determine the satisfiability of attribute-value constraints with approximately the same efficiency as the standard attribute-value unification algorithm.

In generalizing attribute-value constraints to SB formulae, we noted that in clausal form the SB formulae constitute a disjunctive extension to Datalog, and that the standard attribute-value unification algorithm is closely related to a version of semi-naive evaluation algorithm used to evaluate Datalog clauses. This offers another perspective on feature structure constraints; they can be seen as kinds of databases containing information about the linguistic structures they describe.

Perhaps the greatest weakness of this work is the lack of an efficient method for treating disjunctive constraints. The backtracking strategy suggested in the body of the paper can be extremely inefficient, even with 'toy' grammars. This problem is not unique to this approach; rather, it is endemic to most complex feature-based approaches to natural language processing, as evidenced by the volume of literature on the subject.

As discussed in Section 3, the satisfiability problem for SB formula with a fixed number of universal quantifiers is NP-hard, so all known algorithms require exponential time in the worst case, and unless P=NP no tractable general-purpose algorithm for determining the satisfiability of SB formulae exists. With present technology, the best we can hope for is an algorithm that performs adequately on the types of problems that we actually encounter.

Sometimes disjunctive constraints can be (automatically) transformed into nondisjunctive ones, thus avoiding the problem entirely. For example, Alshawi (1992) describes a technique attributed to Colmerauer for transforming disjunctions of finite-domain feature-value constraints into conjunctions. Kasper (1988) and Hegner (1991) point out that Horn clauses, although technically disjunctions, can be handled considerably more efficiently than general disjunctive constraints. The forward-chaining mechanisms that they propose for treating these constraints appear to be special cases of the semi-naive algorithm sketched in this paper.

Unfortunately, I know of no general adequate method for handling the disjunctive constraints that arise in real grammars with acceptable efficiency. The techniques discussed by Maxwell and Kaplan (1991, 1992) seem most directly compatible with the approach described in this paper, and the methods described by Kasper (1987b), Eisele and Dörre (1988), and Emele (1991) might have important applications as well.

Acknowledgment

I would like to thank Johan van Benthem, Bob Carpenter, Stephen Hegner, Ronald M. Kaplan, Edward Stabler and the participants of the feature structures seminar at the Institut für maschinelle Sprachverarbeitung, Universität Stuttgart, for their suggestions and comments. All responsibility for errors rests with me, of course.

References

Aït-Kaci, Hassan (1984). *A lattice theoretic approach to computation based on a calculus of*

partially ordered type structures. Doctoral dissertation, University of Pennsylvania.

Aït-Kaci, Hassan, and Podelski, Andreas (1993). "Towards a meaning of LIFE." *The Journal of Logic Programming* 16(3,4), 195–234.

Alshawi, Hiyan (1992). "Categories and rules." In *The Core Language Engine*, edited by Hiyan Alshawi, 41–60. MIT Press.

Barton, G. Edward; Berwick, Robert C.; and Ristad, Eric S. (1987). *Computational Complexity and Natural Language*. MIT Press.

Blackburn, Patrick (1991). "Modal logic and attribute-value structures." In *Modal Logic Colloquium '91*, edited by Maarten de Rijke. Dutch Project for Language, Logic and Information, Amsterdam.

Blackburn, Patrick, and Spaan, Edith (1992). "A modal perspective on the computational complexity of attribute value grammar." Logic Group Preprint Series No. 77, Department of Philosophy, University of Utrecht.

Blackburn, Patrick; Gardent, Claire; and Meyer-viol, Wilfried (1993). "Talking about trees." In *Proceedings, 6th European Meeting of the Asssociation for Computational Linguistics*. Utrecht, Holland.

Bresnan, Joan (1982). *The Mental Representation of Grammatical Relations*. MIT Press.

Carpenter, Bob (1991). "Typed feature structures: A generalization of first-order terms." In *Logic Programming, Proceedings of the 1991 International Symposium*, edited by Vijay Saraswat and Kazunori Ueda, 187–201. MIT Press.

Carpenter, Bob (1992). *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science 32. Cambridge University Press, Cambridge, England.

Carpenter, Bob, and Pollard, Carl (1991). "Inclusion, disjointness and choice: The logic of linguistic classification." In *Proceedings, 29th Annual Meeting of the Association for Computational Linguistics*, 9–16. Berkeley, CA.

Carpenter, Bob; Pollard, Carl; and Franz, Alex (1991). "The specification and implementation of constraint-based unification grammars." In *Proceedings, Second International Workshop on Parsing Technologies*. Cancun, Mexico.

Chang, Chin-Liang, and Lee, Richard Char-Tung (1973). *Symbolic Logic and Mechanical Theorem Proving*. Academic Press.

Chomsky, Noam (1986). *Knowledge of Language, Its Nature, Origins and Use*. Praeger.

Chomsky, Noam (1988). Some Notes on Economy of Derivation and Representation. ms. Massachusetts Institute of Technology.

Corman, Thomas H.; Leiserson, Charles E.; and Rivest, Ronald L. (1990). *Introduction to Algorithms*. MIT Press.

Dawar, Anuj, and Vijay-Shanker, K. (1990). "An interpretation of negation in feature structures." *Computational Linguistics* 16(1), 11–21.

Dörre, Jochen (1991). "The language of STUF." In *Text Understanding in LILOG: Integrating Computational Linguistics and Artificial Intelligence*, edited by Otthein Herzog and Claus-Rainer Rollinger, 39–50. Springer-Verlag.

Dörre, Jochen, and Eisele, Andreas (1990). "Feature logic with disjunctive unification." In *Proceedings, 13th International Conference on Computational Linguistics (COLING-90)*, 100–105. Helsinki, Finland.

Dörre, Jochen, and Eisele, Andreas (1991). "A comprehensive unification-based grammar formalism." DYANA Deliverable R3.1B, ESPRIT Basic Research Action BR3175.

Dörre, Jochen, and Rounds, William C. (1992). "On subsumption and semiunification in feature algebras." *Journal of Symbolic Computation* 13, 441–461.

Duffy, David (1991). *Principles of Automated Theorem Proving*. John Wiley and Sons.

Eisele, A., and Dörre, J. (1988). "Unification of disjunctive feature descriptions." In *Proceedings, 26th Annual Meeting of the Association for Computational Linguistics*, 286–294. Buffalo, New York.

Emele, Martin (1991). "Unification with lazy non-redundant copying." In *Proceedings, 29th Annual Meeting of the Association for Computational Linguistics*, 323–330. Berkeley, CA.

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

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

Hegner, Stephen J. (1991). "Horn extended feature structures: Fast unification with negation and limited disjunction." In *Proceedings, Fifth Conference of the European Chapter of the Association for Computational Linguistics*, 33–38. Berlin.

Höhfeld, Markus, and Smolka, Gert. (1988). "Definite relations over constraint languages." LILOG Report No. 53, IBM Deutschland.

Johnson, Mark (1988). *Attribute-Value Logic*

and the Theory of Grammar. CSLI Lecture Notes Series. University of Chicago Press.

Johnson, Mark (1990a). "Expressing disjunctive and negative feature constraints with classical first-order logic." In *Proceedings, 28th Annual Meeting of the Association for Computational Linguistics*, 173–179. Pittsburgh, PA.

Johnson, Mark (1990b). "Features, frames and quantifier-free formulae." In *Logic and Logic Grammars for Language Processing*, edited by Patrick Saint-Dizier and Stan Szpakowicz, 94–107. Ellis Horwood.

Johnson, Mark (1991a). "Features and formulae." *Computational Linguistics* 17(2), 131–152.

Johnson, Mark (1991b). "Logic and feature structures." In *Proceedings, International Joint Conference on Artificial Intelligence*. Sydney.

Johnson, Mark (in press a) "Attribute-value logic and natural language processing." In *Unification in Grammar*, edited by Jürgen Wedekind and Christian Rohrer. MIT Press.

Johnson, Mark (in press b) "Two ways of formalizing grammars." *Linguistics and Philosophy*.

Johnson, Mark, and Kay, Martin (1990). "Semantic operators and anaphora." In *Proceedings, 13th International Conference on Computational Linguistics (COLING-90)*, 17–27. Helsinki.

Johnson, Mark, and Klein, Ewan (1986). "Discourse, parsing and anaphora." In *Proceedings, 11th International Conference on Computational Linguistics*. Bonn.

Kamp, Hans (1981). "A theory of truth and semantic representation." In *Formal Methods in the Study of Language*, edited by J. A. G. Groenendijk, T. M. V. Janssem, and M. B. J. Stokhof, 277–322. Mathematical Centre Tracts, Amsterdam.

Kaplan, Ronald M., and Bresnan, Joan (1982). "Lexical-functional grammar, a formal system for grammatical representation." In *The Mental Representation of Grammatical Relations*, edited by Joan Bresnan, 173–281. MIT Press.

Kaplan, Ronald M., and Maxwell, John T. (1988a). "An algorithm for functional uncertainty." In *Proceedings, 12th International Conference on Computational Linguistics*, 297–302. Budapest, Hungary.

Kaplan, Ronald M., and Maxwell, John T. (1988b). "Constituent coordination in lexical-functional grammar." In *Proceedings, 12th International Conference on Computational Linguistics*, 297–302. Budapest, Hungary.

Kaplan, Ronald M., and Zaenen, Annie. (1989). "Long-distance dependencies, constituent structure and functional uncertainty." In *Alternative Conceptions of Phrase Structure*, edited by Mark Baltin and Anthony Krock, 17–42. Chicago University Press.

Kapur, D., and Musser, D. R. (1987). "Proof by consistency." *Artificial Intelligence* 31, 125–157.

Karttunen, Lauri (1984). "Features and values." In *Proceedings, International Conference on Computational Linguistics (COLING-1984)*, 28–33. Stanford University.

Kasper, Robert T. (1987a). *Feature structures: A logical theory with application to language analysis*. Doctoral dissertation, University of Michigan.

Kasper, Robert T. (1987b). "A unification method for disjunctive feature structures." In *Proceedings, 25th Annual Meeting of the Association for Computational Linguistics*, 235–242. Stanford University.

Kasper, Robert T. (1988). "Conditional descriptions in functional unification grammar." In *Proceedings, 26th Annual Meeting of the Association for Computational Linguistics*, 233–240. Buffalo, N.Y.

Kasper, Robert T., and Rounds, William C. (1986). "A logical semantics for feature structures." In *Proceedings, 24th Annual Meeting of the Association for Computational Linguistics*, 257–266. Columbia University, New York.

Kasper, Robert T., and Rounds, William C. (1990). "The logic of unification in grammar." *Linguistics and Philosophy* 13(1), 35–58.

Kay, Martin (1979). "Functional unification grammar." In *Proceedings, Fifth Annual Meeting of the Berkeley Linguistics Association*. Berkeley, CA.

Kay, Martin (1985a). "Parsing in functional unification grammar." In *Natural Language Parsing*, edited by D. R. Dowty, L. Karttunen, and A. M. Zwicky. Cambridge University Press.

Kay, Martin (1985b). "Unification in grammar." In *Natural Language Understanding and Logic Programming*, edited by V. Dahl and P. Saint-Dizier, 233–240. North Holland.

Keller, Bill (1991). *Feature logics, infinitary descriptions and the logical treatment of grammar*. Doctoral dissertation, University of Sussex.

Kowalski, Robert (1979). *Logic for Problem Solving*. North Holland.

Langholm, Tore (1989). "How to say no with feature structures." COSMOS Report

No. 13, Department of Mathematics, University of Oslo.

Lewis, Harry (1980). "Complexity results for classes of quantificational formulae." *JCSS* 21, 317–353.

Lewis, Harry, and Papadimitriou, Christos (1981). *Elements of the Theory of Computation.* Prentice-Hall, NJ.

Lloyd, John W. (1984). *Foundations of Logic Programming.* Springer-Verlag.

Lobo, Jorge; Minker, Jack; and Rajasekar, Arcot (1992). *Foundations of Disjunctive Logic Programming.* MIT Press.

Loveland, D. W. (1987). "Near-Horn Prolog." In *Logic Programming: Papers for the Fourth International Conference on Logic Programming,* edited by Jean-Louis Lassez, 456–469. MIT Press.

Marcus, Mitch; Hindle, Donald; and Fleck, Margaret M. (1983). "D-theory—talking about talking about trees." In *Proceedings, 21st Annual Meeting of the Association for Computational Linguistics,* 129–136. Cambridge, MA.

Maxwell, John T., and Kaplan, Ronald M. (1991). "A method for disjunctive constraint satisfaction." In *Current Issues in Parsing Technology,* edited by Masaru Tomita, 173–190. Kluwer Academic Publishers.

Maxwell, John T., and Kaplan, Ronald M. (1992). "The interface between phrasal and functional constraints." *Computational Linguistics* 19(4), 571–590.

Moshier, M. Drew (1988). *Extensions to unification grammar for the description of programming languages.* Doctoral dissertation, University of Michigan.

Moshier, M. Drew, and Rounds, William 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.

Nelson, G., and Oppen, D. C. (1980). "Fast decision procedures based on congruence closure." *J. ACM* 27(2), 245–257.

Partee, Barbara H.; ter Meulen, Alice; and Wall, Robert E. (1990). *Mathematical Methods in Linguistics.* Kluwer Academic Publishers.

Pereira, Fernando C. N. (1982). *Logic for natural language analysis.* Doctoral dissertation, University of Edinburgh.

Pereira, Fernando C. N. (1987). "Grammars and logics of partial information." In *Proceedings, International Conference on Logic Programming,* 989–1013. Melbourne, Australia.

Pollard, Carl, and Sag, Ivan A. (1987). *Information-Based Syntax and Semantics.* CSLI Lecture Notes Series. Chicago University Press.

Pollard, Carl, and Sag, Ivan A. (1992). *Head-Driven Phrase Structure Grammar.* CSLI Lecture Notes Series. Chicago University Press.

Rounds, William C. (1988). "LFP: A logic for linguistic descriptions and an analysis of its complexity." *Computational Linguistics* 14(4), 1–9.

Rounds, William C., and Manaster-Ramer, Alexis (1987). "A logical version of functional grammar." In *Proceedings, 25th Annual Meeting of the Association for Computational Linguistics,* 89–96. Stanford, CA.

Shieber, Stuart M. (1986). *An Introduction to Unification-Based Approaches to Grammar.* CSLI Lecture Notes Series. University of Chicago Press.

Shieber, Stuart M. (1992). *Constraint-Based Grammar Formalisms: Parsing and Type Inference for Natural and Computer Languages.* MIT Press.

Smolka, Gert. (1988). "A feature logic with subsorts." LILOG Report No. 33, IBM Deutschland GmbH.

Smolka, Gert. (1992). "Feature constraint logics for unification grammars." *The Journal of Logic Programming* 12(1,2), 51–87.

Ullman, Jeffrey D. (1988). *Principles of Database and Knowledge-Base Systems, Vol. I.* Computer Science Press.

Ullman, Jeffrey D. (1989). *Principles of Database and Knowledge-Base Systems, Vol. II: The New Technologies.* Computer Science Press.

Uszkoreit, Hans (1986). "Categorial unification grammar." In *Proceedings, 11th International Conference on Computational Linguistics,* 187–194. Bonn.

Vijay-Shanker, K. (1992). "Using descriptions of trees in a tree adjoining grammar." *Computational Linguistics* 18(4), 481–517.

Wall, Robert (1972). *Introduction to Mathematical Linguistics.* Prentice-Hall.

Zajac, Rémi (1992). "Inheritance and constraint-based grammar formalisms." *Computational Linguistics* 18(2), 159–182.

Appendix A: SB Formulae in Disjunctive Datalog Format

```
(1′)   Y = Z :- arc(X, A, Y), arc(X, A, Z).
(2′)   :- con(X), arc(X, A, Y).
(3′)   con(c).                                   (for each c in N)
(4′)   :- c1 = c2.                               (for each distinct pair c1, c2 in N)
(5′)   arc(X, person, 3rd) :- 3rd_sg(X).
(5″)   arc(X, number, singular) :- 3rd_sg(X).
(5‴)   3rd_sg(X) :- arc(X, person, 3rd), arc(X, number, singular).
(6′)   Y = singular; Y = plural :- arc(X, number, Y).
(7a′)  :- X < X.
(7b′)  :- X < Y, Y < X.
(7c′)  X < Z :- X < Y, Y < Z.
(8a′)  d(X, X) :- n(X).
(8a″)  n(X) :- d(X, X).
(8b′)  X = Y :- d(X, Y), d(Y, X).
(8c′)  d(X, Z) :- d(X, Y), d(Y, Z).
(9′)   n(root).
(9″)   d(root, Y) :- n(Y).
(10′)  X < Y; Y < X; d(X, Y); d(Y, X) :- n(X), n(Y).
(10″)  :- n(X), n(Y), X < Y, d(X, Y).
(10‴)  :- n(X), n(Y), X < Y, d(Y, X).
(12a′) n(X) :- X < Y.
(12a″) n(Y) :- X < Y.
(12b′) n(X) :- d(X, Y).
(12b″) n(Y) :- d(X, Y).
(7′)   d(X, Y) :- p(X, Y).
(7″)   :- p(X, Y), X = Y.
(7‴)   p(X, Y); X = Y :- d(X, Y).
(14′)  :- c(X, Y), d(X, Y).
(14″)  p(Z, Y) :- c(X, Y), p(Z, X).
(17′)  X = X.
(18′)  Y = X :- X = Y.
(19′)  arc(X1, A, Y) :- X = X1, arc(X, A, Y).
(20′)  arc(X, A1, Y) :- A = A1, arc(X, A, Y).
(21′)  arc(X, A, Y1) :- Y = Y1, arc(X, A, Y).
(22′)  con(C1) :- C = C1, con(C).
(23′)  3rd_sg(Y) :- X = Y, 3rd_sg(X).
```