EXPRESSING DISJUNCTIVE AND NEGATIVE FEATURE CONSTRAINTS WITH CLASSICAL FIRST-ORDER LOGIC.

Mark Johnson,
Cognitive and Linguistic Sciences, Box 1978,
Brown University, Providence, RI 02912.
mj@cs.brown.edu

ABSTRACT

In contrast to the "designer logic" approach, this paper shows how the attribute-value feature structures of unification grammar and constraints on them can be axiomatized in classical first-order logic, which can express disjunctive and negative constraints. Because only quantifier-free formulae are used in the axiomatization, the satisfiability problem is $\mathcal{NP}$-complete.

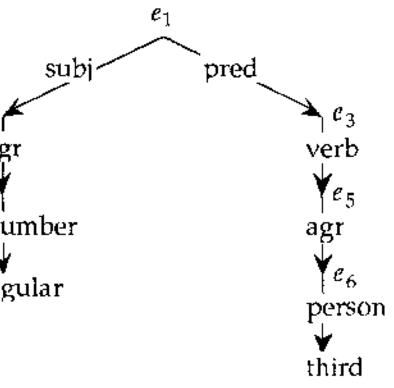
INTRODUCTION.

Many modern linguistic theories, such as Lexical-Functional Grammar [1], Functional Unification Grammar [12] Generalized Phrase-Structure Grammar [6], Categorial Unification Grammar [20] and Head-driven Phrase-Structure Grammar [18], 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 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 parents and its immediate descendants. 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 nature of the satisfying feature structures is of secondary importance.

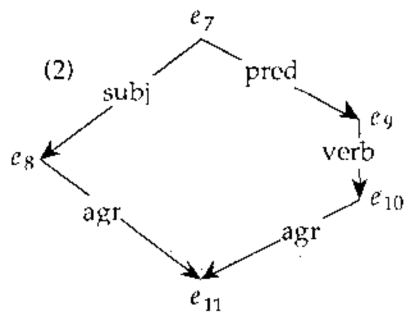
Most work on unification-based grammar (including the references cited above) has adopted a type of feature structure called an *attribute-value structure*. The elements in an attribute-value structure come in two kinds: *constant elements* and *complex elements*. Constant elements are atomic entities 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) and any 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. The set of attribute-value structures partially ordered by the subsumption relation (together with an additional entity $\top$ that every attribute-value structure subsumes) forms a lattice, and the join operation on this lattice is called the unification operation [19].

Example: (from [16]). *The attribute-value structure* (1) *has six complex elements labelled* $e_1 \ldots e_6$ *and two constant elements, singular and third. The complex element* e_1 *has two attributes, subj and pred, the value of which are the complex elements* e_2 *and* e_3 *respectively.*

(1)

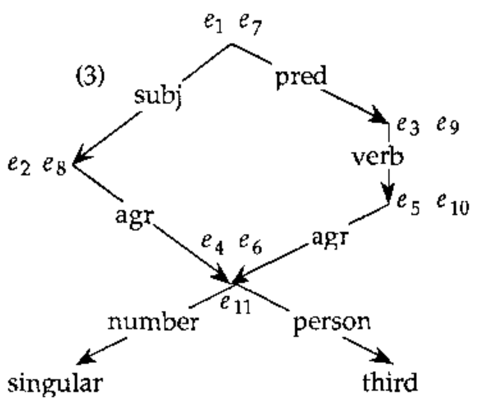


(2)



The unification of elements e_1 *of* (1) *and* e_7 *of* (2) *results in the attribute-value structure* (3), *the* *minimal structure in the subsumption lattice which subsumes both* (1) *and* (2).

(3)



If constraints on attribute-value structures are restricted to conjunctions of equality constraints (i.e. requirements that the value of a path of attributes is equal to a constant or the value of another path) then the set of satisfying attribute-value structures is the principal filter generated by the minimal structure that satisfies the constraints. The generator of the satisfying principal filter of the conjunction of such constraints is the unification of the generators of the satisfying principal filters of each of the conjuncts. Thus the set of attribute-value structures that simultaneously satisfy a set of such constraints can be characterized by computing the unification of the generators of the corresponding principal filters, and the constraints are satisfiable iff the resulting generator is not $\top$ (i.e. $\top$ represents unification failure). Standard unification-based parsers use unification in exactly this way.

When disjunctions and negations of constraints are permitted, the set of satisfying attribute-value structures does not always form a principal filter [11], so the simple unification-based technique for determining the satisfiability of feature structure constraints must be extended. Kasper and Rounds [11] provide a formal framework for investigating such constraints by reviving a distinction originally made (as far as I am aware) by Kaplan and Bresnan [10] between the *language* in which feature structure constraints are expressed and the structures that satisfy these constraints. Unification is supplanted by conjunction of constraints, and disjunction and negation appear only in the constraint language, not in the feature structures themselves (an exception is [3] and [2], where feature bundles may contain negative arcs).

Research in this genre usually follows a general pattern: an abstract model for feature structures and a specialized language for expressing constraints on such structures are "custom-crafted" to treat some problematic feature constraint (such as negative feature constraints). Table 1 sketches some of the variety of feature structure models and constraint types that previous analyses have used.

This paper follows Kasper and Rounds and most proposals listed in Table 1 by distinguishing the constraint language from feature structures, and restricts disjunction and negation to the constraint language alone. It

Table 1: Constraint Languages and Feature Structure Models.

Author	Model of Feature Structures	Constraint Language Features
Kaplan and Bresnan [10]	Partial functions	Disjunction, negation, set-values
Pereira and Shieber [17]	Information Domain $F = [\, A \to F \,] + C$	
Kasper and Rounds [11]	Acyclic finite automata	Disjunction
Moshier and Rounds [14]	Forcing sets of acyclic finite automata	Intuitionistic negation
Dawar and Vijayashankar [3]	Acyclic finite automata	Three truth values, negation
Gazdar, Pullum, Carpenter, Klein, Hukari and Levine [7]	Category structures	Based on propositional modal logic
Johnson [9]	"Attribute-value structures"	Classical negation, disjunction...

(A1) For all constants c and attributes a, $a(c) = \perp$.

(A2) For all distinct pairs of constants c_1, c_2, $c_1 \neq c_2$.

(A3) For all attributes a, $a(\perp) = \perp$.

(A4) For all constants c, $c \neq \perp$.

(A5) For all terms u, v, $u \approx v \leftrightarrow (u = v \wedge u \neq \perp)$

Figure 1: The axiom schemata that define attribute-value structures .

differs by not proposing a custom-built "designer logic" for describing feature structures, but instead uses standard first-order logic to axiomatize attribute-value structures and express constraints on them, including disjunctive and negative constraints. The resulting system is a simplified version of Attribute-Value Logic [9] which does not allow values to be used as attributes (although it would be easy to do this). The soundness and completeness proofs in [9] and other papers listed in Table 1 are not required here because these results are well-known properties of first-order logic.

Since both the axiomatizion and the constraints are actually expressed in a decidable class of first-order formulae, viz. quantifier-free formulae with equality,[1] the decidability of feature structure constraints follows trivially. In fact, because the satisfiability problem for quantifier-free formulae is $\mathcal{NP}$-complete [15] and the relevant portion of the axiomatization and translation of constraints can be constructed in polynomial time, the satisfiability problem for feature constraints (including negation) is also $\mathcal{NP}$-complete.

Axiomatizing Attribute-Value Structures

This section shows how attribute-value structures can be axiomatized using first-order quantifier-free formulae with equality. In the next section we see that equality and inequality constraints on the values of the attributes can also be expressed as such formulae, so systems of these constraints can be solved using standard techniques such as the Congruence Closure algorithm [15], [5].

The elements of the attribute-value structure, both constant and complex, together with an additional element $\perp$ constitute the domain of individuals of the intended interpretation. The attributes are unary partial functions over this domain (i.e. mappings from elements to elements) which are always undefined on constant elements. We capture this partiality by the standard technique of adding an additional element $\perp$ to the domain to serve as the value 'undefined'. Thus $a(x) = \perp$ if x does not have an attribute a, otherwise $a(x)$ is the value of x's attribute a.

We proceed by specifying the conditions an interpretation must satisfy to be an attribute-value structure. Modelling attributes with functions automatically requires attributes to be single-valued, as required.

Axiom schema (A1) describes the properties of constants. It expresses the requirement that constants have no attributes.

Axiom schema (A2) requires that distinct constant *symbols* denote distinct *elements* in any satisfying model. Without (A2) it would be possible for two distinct constant elements, say *singular* and *plural*, to denote the same individual.[2]

Axiom schema (A3) and (A4) state the properties of the "undefined value" $\perp$. It has no attributes, and it is distinct from all of the constants (and from all other elements as well – this will be enforced by the translation of equality constraints).

This completes the axiomatization. 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. The claim is that *any* interpretation which satisfies all of these

[1] The close relationship between quantifier-free formulae and attribute-value constraints was first noted in Kaplan and Bresnan [10].

[2] Such a schema is required because we are concerned with satisfiability rather than validity (as in e.g. logic programming).

axioms is an attribute-value structure; i.e. (A1) – (A4) constitute a *definition* of attribute-value structures.

Example (continued): *The interpretation corresponding to the attribute-value structure (1) has as its domain the set D = {$e_1, \ldots, e_6$, singular, third, $\perp$}. The attributes denote functions from D to D. For example, agr denotes the function whose value is $\perp$ except on e_2 and e_5, where its values are e_4 and e_6 respectively. It is straight-forward to check that all the axioms hold in the three attribute-value structures given above.*

In fact, any model for these axioms can be regarded as a (possibly infinite and disconnected) attribute-value feature structure, where the model's individuals are the elements or nodes, the unary functions define how attributes take their values, the constant symbols denote constant elements, and $\perp$ is a sink state.

EXPRESSING CONSTRAINTS AS QUANTIFIER-FREE FORMULAE.

Various notations are currently used to express attribute-value 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 $(x\ a) = y$ in PATR-II [19], as $(x\ a) = y$ in LFG [10], and as $x(a) \approx y$ in [9], for example. At the risk of further confusion we use another notation here, and write the constraint requiring that the value of attribute a of x is y as $a(x) \approx y$. This notation emphasises the fact that attributes are modelled by functions, and simplifies the definition of '$\approx$'.

Clearly for an attribute-value structure to satisfy the constraint $u \approx v$ then u and v must denote the same element, i.e. $u = v$. However this is not a sufficient condition: $num(x) \approx num(y)$ is not satisfied if $num(x)$ or $num(y)$ is $\perp$. Thus it is necessary that the arguments of '$\approx$' denote identical elements distinct from the denotation of $\perp$.

Even though there are infinitely many instances of the schema in (A5) (since there are infinitely many terms) this is not problematic, since $u \approx v$ can be regarded as an *abbreviation* for $u = v \wedge u \neq \perp$.

Thus equality constraints on attribute-value structures can be expressed with quantifier-free formulae with equality. We use classically interpreted boolean connectives to express conjunctive, disjunctive and negative feature constraints.

Example (continued): *Suppose each variable x_i denotes the corresponding e_i, $1 \leq i \leq 11$, of (1) and (2). Then $subj(x_1) \approx x_2$, $number(x_4) \approx singular$ and $number(agr(x_2)) \approx number(x_4)$ are true, for example. Since e_4 and e_5 are distinct elements, $x_8 \approx x_{11}$ is false and hence $x_8 \not\approx x_{11}$ is true. Thus "$\not\approx$" means "not identical to" or "not unified with", rather than "not unifiable with".*

Further, since $agr(x_1) = \perp$, $agr(x_1) \approx agr(x_1)$ is false, even though $agr(x_1) = agr(x_1)$ is true. Thus $t \approx t$ is not a theorem because of the possibility that $t = \perp$.

SATISFACTION AND UNIFICATION

Given any two formulae ϕ and φ, the set of models that satisfy both ϕ and φ is exactly the set of models that satisfy $\phi \wedge \varphi$. That is, the conjunction operation can be used to describe the intersection of two sets of models each of which is described by a constraint formula, even if these satisfying models do not form principal filters [11] [9]. Since conjunction is idempotent, associative and commutative, the satisfiability of a conjunction of constraint formulae is independent of the order in which the conjuncts are presented, irrespective of whether they contain negation. Thus the evaluation (i.e. simplification) of constraints containing negation can be freely interleaved with other constraints.

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 elements $a(e)$ and $a(e')$ for all attributes a that are defined on both e and e'. The constraint $x \approx x'$ implies $a(x) \approx a(x')$ in exactly the same circumstances; i.e. when $a(x)$ and $a(x')$ are both distinct from $\perp$. Unification fails either when two different constant elements are to be unified, or when a complex element and a constant element are unified (i.e. *constant-constant clashes* and *constant-complex clashes*). The constraint $x \approx x'$ is unsatisfiable under exactly the same circumstances. $x \approx x'$ is unsatisfiable when x and x' are also required to satisfy $x \approx c$ and $x' \approx c'$ for distinct constants c, c', since $c \neq c'$ by axiom schema (A2). $x \approx x'$ is also unsatisfiable when x and x' are required to satisfy $a(x) \approx t$ and $x' \approx c'$

for any attribute a, term t and constant c', since $a(c') = \bot$ by axiom schema (A3).

Since unification is a technique for determining the satisfiability of conjunctions of atomic equality constraints, the result of a unification operation is exactly the set of atomic consequences of the corresponding constraints. Since unification fails precisely when the corresponding constraints are unsatisfiable, failure of unification occurs exactly when the corresponding constraints are equivalent to *False.*

Example (continued): *The sets of satisfying models for the formulae* (1′) *and* (2′) *are precisely the principal filters generated by* (1) *and* (2) *above.*

(1′) $subj(x_1) \approx x_2 \wedge agr(x_2) \approx x_4 \wedge$
$number(x_4) \approx singular \wedge pred(x_1) \approx x_3 \wedge$
$verb(x_3) \approx x_5 \wedge agr(x_5) \approx x_6 \wedge$
$person(x_6) \approx third$

(2′) $subj(x_7) \approx x_8 \wedge agr(x_8) \approx x_{11} \wedge pred(x_7) \approx x_9 \wedge$
$verb(x_9) \approx x_{10} \wedge agr(x_{10}) \approx x_{11}$

Because the principal filter generated by the unification of e_1 and e_7 is the intersection of the principal filters generated by (1) *and* (2), *it is also the set of satisfying models for the conjunction of* (1′) *and* (2′) *with the formula* $x_1 \approx x_7$ (3′).

(3′) $subj(x_1) \approx x_2 \wedge agr(x_2) \approx x_4 \wedge$
$nmber(x_4) \approx singular \wedge pred(x_1) \approx x_3 \wedge$
$verb(x_3) \approx x_5 \wedge agr(x_5) \approx x_6 \wedge$
$person(x_6) \approx third \wedge subj(x_7) \approx x_8 \wedge$
$agr(x_8) \approx x_{11} \wedge pred(x_7) \approx x_9 \wedge$
$verb(x_9) \approx x_{10} \wedge agr(x_{10}) \approx x_{11} \wedge x_1 \approx x_7$.

The satisfiability of a formula like (3′) *can be shown using standard techniques such as the Congruence Closure Algorithm* [15], [5]. *In fact, using the substitutivity and transitivity of equality,* (3′) *can be simplified to* (3″). *It is easy to check that* (3) *is a satisfying model for both* (3″) *and the axioms for attribute-value structures.*

(3″) $subj(x_1) \approx x_2 \wedge agr(x_2) \approx x_4 \wedge$
$number(x_4) \approx singular \wedge person(x_4) \approx third \wedge$
$pred(x_1) \approx x_3 \wedge verb(x_3) \approx x_5 \wedge agr(x_5) \approx x_4 \wedge$
$x_1 \approx x_7 \wedge x_2 \approx x_5 \wedge x_3 \approx x_9 \wedge x_5 \approx x_{10} \wedge$
$x_4 \approx x_6 \wedge x_4 \approx x_{11}$.

The treatment of negative and disjunctive constraints is straightforward. Since negation is interpreted classically, the set of satisfying models do not always form a filter (i.e. they are not always upward closed [16]). Nevertheless, the quantifier-free language itself is capable of characterizing exactly the set of feature structures that satisfy any boolean combination of constraints, so the failure of upward closure is not a fatal flaw of this approach.

At a methodological level, I claim that after the mathematical consequences of two different interpretations of feature structure constraints have been investigated, such as the classical and intuitionistic interpretations of negation in feature structure constraints [14], it is primarily a linguistic question as to which is better suited to the description of natural language. I have been unable to find any linguistic analyses which can yield a set of constraints whose satisfiablity varies under the classical and intuitionistic interpretations, so the choice between classical and intuitionistic negation may be moot.

For reasons of space the following example (based on Pereira's example [16] demonstrating a purported problem arising from the failure of upward closure with classical negation) exhibits only negative constraints.

Example: *The conjunction of the formulae*

$number(agr(x)) \approx singular$

and

$agr(x) \approx y \wedge \sim (pers(y) \approx 3rd \wedge$
$number(y) \approx singular)$

can be simplified by substitution and transitivity of equality and boolean equivalences to

(4′) $agr(x) \approx y \wedge number(y) \approx singular \wedge$
$pers(y) \not\approx 3rd.$

This formula is satisfied by the structure (4) *when x denotes e and y denotes f. Note the failure of upward closure, e.g.* (5) *does not satisfy* (4′), *even though* (4) *subsumes* (5).

(4)
$$\begin{tikzcd}
e \arrow[d, "\text{agr}"] \\
f \arrow[d, "\text{number}"] \\
\text{singular}
\end{tikzcd}$$

(5)
$$\begin{tikzcd}
& e \arrow[d, "\text{agr}"] & \\
& f \arrow[dl, "\text{number}"'] \arrow[dr, "\text{pers}"] & \\
\text{singular} & & \text{3rd}
\end{tikzcd}$$

However, if (4′) is conjoined with pers(agr(x)) ≈ 3rd the resulting formula (6) is unsatisfiable *since it is equivalent to (6′), and 3rd ≉ 3rd is unsatisfiable.*

(6) agr(x) ≈ y ∧ number(y) ≈ singular ∧ pers(y) ≉ 3rd ∧ pers(agr(x)) ≈ 3rd.

(6′) agr(x) ≈ y ∧ number(y) ≈ singular ∧ pers(y) ≈ 3rd ∧ 3rd ≉ 3rd.

CONCLUSION

This paper has shown how attribute-value structures and constraints on them can be axiomatized in a decidable class of first-order logic. The primary advantage of this approach over the "designer logic" approach is that important properties of the logic of the feature constraint language, such as soundness, completeness, decidability and compactness, follow immediately, rather than proven from scratch. A secondary benefit is that the substantial body of work on satisfiability algorithms for first-order formulae (such as ATMS-based techniques that can efficiently evaluate some disjunctive constraints [13]) can immediately be applied to feature structure constraints.

Further, first-order logic can be used to axiomatize other types of feature structures in addition to attribute-value structures (such as "set-valued" elements) and express a wider variety of constraints than equality constraints (e.g. subsumption constraints). In general these extended systems cannot be axiomatized using only quantifier-free formulae, so their decidability may not follow directly as it does here. However the decision problem for sublanguages of first-order logic has been intensively investigated [4], and there are decidable classes of first-order formulae [8] that appear to be expressive enough to axiomatize an interesting variety of feature structures (e.g. function-free universally-quantified prenex formulae can express linguistically useful constraints on "set-valued" elements).

An objection that might be raised to this general approach is that classical first-order logic cannot adequately express the inherently "partial information" that feature structures represent. While the truth value of any formula with respect to a model (i.e. an interpretation and variable assignment function) is completely determined, in general there will be many models that satisfy a given formula, i.e. a formula only partially identifies a satisfying model (i.e. attribute-value structure). The claim is that this partiality suffices to describe the partiality of feature structures.

BIBLIOGRAPHY

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

2. Dawar, A. and K. Vijayashanker. *Three-Valued Interpretation of Negation in Feature Structure Descriptions.* University of Delaware Technical Report 90-03. 1989.

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

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

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

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

7. Gazdar, G., G. K. Pullum, R. Carpenter, E. Klein, T. E. Hukari and R. D. Levine. "Category Structures." *Computational Linguistics.* 14.1: 1 – 20, 1988.

8. Gurevich, Y. "The Decision Problem for Standard Classes." *JSL.* 41.2: 460-464, 1976.

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

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

11. Kasper, R. T. and W. C. Rounds. "A logical semantics for feature structures", in *The Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics,* Columbia University, New York, 1986, 257-266.

12. Kay, M. "Unification in Grammar," in *Natural Language Understanding and Logic Programming,* Dahl and Saint-Dizier ed., 1985 North Holland. Amsterdam, The Netherlands.

13. Maxwell, J. T., III and R. Kaplan. "An Overview of Disjunctive Constraint Satisfaction", in *International Workshop on Parsing Technologies,* Pittsburgh, PA., 1989, 18 – 27. Carnegie Mellon.

14. Moshier, M. D. and W. C. Rounds. "A logic for partially specified data structures", in *The ACM Symposium on the Principles of Programming Languages,* Munich, Germany, 1987, Association for Computing Machinery.

15. Nelson, G. and D. C. Oppen. "Fast Decision Procedures based on Congruence Closure." *J. ACM.* 27.2: 245–257, 1980.

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

17. Pereira, F. C. N. and S. M. Shieber. "The semantics of grammar formalisms seen as computer languages", in *COLING-84,* Stanford University, 1984, 123-129. The Association for Computational Linguistics.

18. Pollard, C. and I. Sag. *Information-based Syntax and Semantics, Volume 1.* CSLI Lecture Notes. 1987 Chicago University Press. Chicago.

19. Shieber, S. M. *An Introduction to Unification-based Approaches to Grammar.* CSLI Lecture Notes Series. 1986 University of Chicago Press. Chicago.

20. Uszkoreit, H. "Categorial unification grammar", in *COLING-86,* 1986, 187–194.