

CHAPTER 1

# Feature Logics

William C. Rounds

*Artificial Intelligence Laboratory  
Department of Electrical Engineering and Computer Science  
University of Michigan  
Ann Arbor, Michigan 48109  
USA*

*Contents*



## 1. Introduction

### 1.1. Overview

Feature logics form a class of specialized logics which have proven especially useful in classifying and constraining the linguistic objects known as feature structures. Linguistically, these structures have their origin in the work of the Prague school of linguistics, followed by the work of Chomsky and Halle in *The Sound Pattern of English* [16]. Feature structures have been reinvented several times by computer scientists: in the theory of data structures, where they are known as record structures, in artificial intelligence, where they are known as frame or slot-value structures, in the theory of data bases, where they are called “complex objects”, and in computational linguistics, where they arose as systems of “registers” in augmented transition networks for natural language parsing, and, quite incidentally, from logic-based approaches to parsing such as definite-clause grammars. Work on the subject has thus involved contributions from linguistics, logic, and computer science.

Before discussing the logic of feature structures, it would be well to understand feature structures themselves, as they are the linguistic objects being constrained by the logic. I will try to say very briefly what perspective is taken by various linguistic theories and computational frameworks on feature structures.

We will then embark on a survey of the various logical systems that have been proposed to constrain feature structures in the setting of grammar. Each logical system, like the linguistic theories, takes a slightly different view of its models. Partly the view is taken because of the particular linguistic theory into which a given logic is embedded, and partly the view is taken because of the model-theoretic and logical tools which can be used to understand the nature of feature structures and the expressive power of the descriptive formalism.

After that, I will discuss various properties of the logics and their models, including mathematical ways of modeling feature structures. An important point to remember is that there may well be no unique way to view the structures. Just as we can view a Boolean algebra as either an abstract algebra, a partially ordered set, or as a topological space, we can think about feature structures with respect to their algebraic properties, their order-theoretic properties, and their connections to set theory. Perhaps most importantly, there is also a dual view of feature logics as either logics or as type systems. It is the mark of a robust class of structures that there are several ways to see them mathematically. I hope also that the applications in linguistics and computer science will become clear as we proceed.

The chapter ends with a linguistic application, concerning the theory commonly known as relational grammar, and then a short discussion of possible future directions for the theory of feature systems and their logics.

### 1.2. Some background

One aim of feature logic is to make precise sense of notations like the following.

This notation, called a *feature matrix*, or *attribute-value matrix*, is common to a number of linguistic theories. It became well-known first in phonology, through the influential work of Chomsky and Halle mentioned above. The descriptive notation continued in transformational theory [15], and became a *lingua franca* in the linguistic theories of Lexical-Functional Grammar (LFG) [39], Generalized Phrase-Structure Grammar (GPSG) [22], and Head-driven Phrase-Structure Grammar (HPSG) [58]. The notation (and extensions thereof) is also the subject matter of the computational frameworks Functional Unification Grammar (FUG) [42] and PATR-II [67], as well as mixed frameworks like Categorical Unification Grammar (see the Categorical Grammar chapter in this volume.)

What use is made of these structures by each of these theories and frameworks? In LFG, the structures are called *f-structures*, and are viewed, along with trees, as linguistic descriptions. The role of a grammar is to constrain the descriptions; so they are thought of as model-theoretic objects which a grammar constrains by virtue of the conditions placed on functional roles (like SUBJECT.) In GPSG, somewhat by contrast, the structures are regarded as giving an extended notion of linguistic category. So the above structure is a “more informative” kind of noun phrase. One aim of GPSG was to show that only a finite number of categories still suffice for natural language, so that in a weak sense, context-freeness of natural language still obtains. There was thus an effort made to limit the number of features (like SUBJECT) necessary for natural language description. On the way to this goal, though, GPSG introduced a logic for constraining the structures; one of the first explicit feature logics. (This was recognized as a true modal logic by Kracht [43].) Finally, the theory of HPSG views feature structures as satisfiers of constraints as well. Here feature structures are used both for phrasal information and for other functional information. The kinds of feature structures vary: use is made of feature structures with set and sequence values embedded in them.

As for the computational frameworks, feature structures are used in Kay’s FUG as representatives of linguistic *types*. We thus find the constraints of the grammar presented as type declarations, which means that disjunctive type restrictions are stated feature-theoretically. There is no separate use of a logic. We will see a formalization of this idea when we get to the order-theoretic aspects of the structures in Section 6. The other main computational framework, PATR-II, was instrumental in showing how *partial information* was encoded in feature structures, and how the *unification* of two structures encoded the sum of the information in both. It also emphasized the distinction between *extensional* versus *intensional* identity in the structures, by means of an explicit way to denote the sharing of structures. This kind of sharing is denoted in fig. 1.2 by the coreference markers like  $\boxed{1}$ . We should mention as well that the same ideas, though not the notation, appeared in the definite clause grammar (DCG) framework [55]. This system is rather obviously logic-based, as the control mechanism is a logic programming language. For complete motivation and examples, consult the chapter on unification-based frameworks by Martin Kay.

There are feature constraints in other formalisms: for example, the current Chomskyan system of principles and parameters (the descendant of government-binding

theory) [27], their role is somewhat limited. The general flavor of such systems is becoming increasingly like other constraint-based theories, though the details are quite different. In like manner, one uses feature constraints in the framework of tree adjoining grammars [38]. Although the primary focus of TAG is tree-based, and purely generative, the addition of features makes the task of describing and generating actual language fragments much simpler [74].

Two other significant contributions to the theory of features must be mentioned. One important influence is Shieber's work on constraints in grammatical formalisms, started in his thesis [68], and continued in his book [69]. Another is Mukai's work on the "Complex Indeterminate Language" CIL. [51]. This latter was one of the pioneering computational efforts, and the theory is equally interesting. A theoretical summary can be found in [52]; this work is one of a few references I know of that mentions *coinductive* techniques for the semantics of Horn clause programs with feature constraints.

Finally, credit for the first complete integration of "feature logic" and linguistic theory may, I think, go to Johnson and Postal's *arc pair grammar* [34]. This is a fully formalized – in first-order logic – version of earlier work of Postal and Perlmutter on relational grammar. Arc pair grammar, in retrospect, was well ahead of its time. The ideas of DCG, for example, had not even been enunciated. GPSG, although under development, did not have a full logic to go with it. Furthermore, the idea of *constraints* or *principles* are very clear, and in some sense precede Chomsky's current theory. I will cover, as an application of feature logic, a current version of this theory due to Johnson and Moss [33], called *stratified feature grammar*. To my mind this is the kind of synthesis of logic and linguistics that one hopes for when one proposes a theoretical framework like feature logic.

## 2. Formalizing feature systems

Our working hypothesis about objects like Fig. 1.2 is that it is a readable notation for what computer scientists call a *data structure*. Such a data structure (here specialized to feature structures) earns an ontological equality with things like tree structures, which are well-understood linguistic objects. The kind of information contained in a feature structure is then information about the constituents of a typical utterance. Such information is constrained by the grammar of a language. So the grammar functions as a logical description language, and the feature structures associated with an utterance are limited by the "assertional constraints" of the grammar.

As with many data structures, including tree structures, however, one can give a mathematical specification in terms of abstract sets of objects. So, for example, tree structures can be defined as a certain kind of partially ordered set, perhaps with a labeling function to add information to the nodes, and so forth. The advantage of this is that we do not have to commit ourselves to a notation in advance, and we may find that there are many more useful examples than we had anticipated. We therefore begin with a noncommittal approach to feature structures, which first

considers a type of algebraic structure, called a *feature system*. The class of feature structures themselves will be a subclass of this slightly more general class. It will eventually turn out that a feature structure can be considered as a feature system on its own, and also that the “set” of feature structures can be made into a feature system.

The general definition appears in Blackburn and Spaan [9], and Moshier [48]. It unifies the definitions found in Johnson [35], and Smolka [70]. We will later relate the definition to others found in the literature.

We begin with the notion of a signature:

**Definition 2.1.** *Let  $L$  be a set of feature names, and  $A$  be a set of sort names. We call the pair  $\langle L, A \rangle$  a feature signature.*

**Example 2.1.** Referring to fig. 1.2, typical elements of  $L$  would be names like AGR or NUM. A typical sort name might be “vp”.

**Definition 2.2** (Feature system). *A feature system of signature  $\langle L, A \rangle$  is a tuple*

$$\mathcal{A} = \langle D, \{f_l\}_{l \in L}, \{D_a\}_{a \in A} \rangle,$$

where for each feature name  $l$ ,  $f_l$  is a partial function on  $D$ , and for each sort name  $a$ ,  $D_a$  is a subset of  $D$ .

The term *feature algebra* has been used to refer to similar systems by Smolka [70]. We do not use that term here, because technically, an algebra over some signature usually consists only of total operations on the carrier  $D$ . Because we have predicates  $D_a$ , our systems are somewhat more ontologically complex than algebras. Blackburn and Spaan call these structures either *attribute value structures* or *Kripke models*. (The last terminology reflects the fact that the structures are used to interpret the modal versions of feature logic.)

**Remark** [. Notation] We write function symbols for features on the right, so that  $f(d)$  is written  $df$ . If  $f$  is defined at  $d$ , we write  $df \downarrow$ , and otherwise  $df \uparrow$ . We use  $p, q$  to denote strings of feature names, also known as *paths*, and if we write an equation  $dp = eq$  it is intended that the appropriate composed function on the left side is defined; so is the composed function on the right, and the two values are equal. (This is not the standard convention for partial function equality, which regards the equation as true if both functions are undefined.)

**Example 2.2** (*Various feature systems*).

- The **Term system**  $\mathbf{T}(\Sigma, X)$ . Let  $\Sigma$  be a ranked alphabet of function symbols<sup>1</sup>, and  $X$  a countable set of variables or *parameters*. We make this set into a feature system of signature  $\langle N, \Sigma \rangle$ , where  $N$  is the set of natural numbers. The elements

<sup>1</sup> Rank is the same as arity.

of  $\mathbf{T}(\Sigma, X)$  are first-order terms over  $\Sigma$  and  $X$ . The features are the natural numbers  $1, 2, \dots$ . The feature  $i$  gives us the  $i$ th argument term of a term, if it exists. For example, we have  $(\sigma(x, \tau(a, b)))_2 = \tau(a)$ , where  $\sigma$  has rank 3, and  $\tau$  rank 1. For  $D_\sigma$  we take  $\{\sigma(t_1, \dots, t_n) \mid t_i \in \mathbf{T}(\Sigma, X)\}$ . This example, due to Smolka [70], arises from logic programming, where terms are used as information bearers. It also gives a model of tree structures *qua* feature system, as the correspondence between ordered, labeled trees and first-order terms is well-understood. Our next example is a variation on this theme.

- The **typed tree system**  $\mathbf{T}(L, A)$ . Consider  $L$  and  $A$ , the set of labels and types, as alphabets. Then the domain  $D^{\mathbf{T}}$  of the tree system consists of all pairs  $(T, \theta)$ , where  $T$  is a nonempty, prefix-closed subset of  $L^*$ , and  $\theta$  is a partial function from  $T$  to  $A$ . If  $(T, \theta) \in D^{\mathbf{T}}$ , and  $f \in L$ , we define  $Tf = \{p \mid fp \in T\}$ , if this set is nonempty, and undefined otherwise. We also set  $(\theta f)(p) = \theta(fp)$ . In this system  $D_a = \{T \mid \theta(\lambda) = a\}$  for  $a \in A$ , where  $\lambda$  is the null string.
- The **typed Nerode system**  $\mathbf{N}(L, A)$ . The domain  $D$  of this system consists of triples  $(T, N, \theta)$ , where  $T$  is as above, and  $N$  is a right-invariant equivalence relation on  $T$  respecting  $\theta$ ; that is, if  $p$  and  $q$  are strings in  $T$ , with  $p N q$ , and  $pf \in T$  for a feature  $f$ , then (i)  $qf \in T$  and  $pf N qf$  (similarly if  $qf \in T$ ); (ii) if  $p N q$  and  $\theta(p) = a$  then  $\theta(q) = a$ . We define  $(T, N)f$  to be  $(Tf, Nf)$ , where  $Tf$  and  $\theta f$  are as above, and  $Nf$  is given by  $p (Nf) q$  if and only if  $fp N fq$ . We will use this system for representing abstract feature structures with structure-sharing.
- Consider a six-element domain consisting of Amy, Alice, Albert, and Anne, and the colors brown and red. Let  $A = \{\text{TALL, HUNGRY}\}$ . Take  $L$  to have one feature, HAIRCOLOR, with brown or red as a value. are atoms. The tall people are Amy and Albert. The hungry people are Anne, Albert, and Alice. Anne has brown hair, Amy and Albert have red hair, and Alice’s hair color is not known. So,  $(amy) \text{ HAIRCOLOR} = red$ , and so forth. This example is a more “real-life” one, which might arise in an AI situation.
- The **Feature Graph system**  $\mathbf{F}$ . The elements of this system are pairs  $(G, n)$ , where  $G$  is a finite directed graph, and  $n$  is a node of  $G$ . Nodes are taken from a fixed countable set; say the integers. Each arc is labeled with an element of  $L$ , and no two outgoing arcs are labeled with the same element of  $L$ . Nodes may optionally be labeled with elements of  $A$ . In this system, we interpret features  $f$  as follows: let  $(G, n)f$  be the graph  $(G, nf)$ , where  $nf$  is the unique node of  $G$  pointed to be the arc starting at  $n$  and labeled by  $f$ , if there is such an arc. We define, for  $a \in A$ , the set  $D_a$  to be the collection  $\{(G, n) \mid \text{the label of } n \text{ is } a\}$ .

We can think of attribute-value matrices as informal notations for feature graphs. Each submatrix of an AVM is in effect a node of the graph. Consider Fig. 1.2. Coreference markers like  $\boxed{1}$  at the end of feature names like SUBJ indicate that the arc labeled SUBJ should point to the (unique) submatrix which is also tagged

with that number, namely

$$\begin{bmatrix} \text{NUM} & \text{sing} \\ \text{PERS} & \text{3rd} \end{bmatrix}.$$

We continue with some useful concepts for the sequel.

**Definition 2.3.** A subsystem of a feature system  $\mathbf{A}$  consists of (i) a subset  $E$  of  $D^{\mathbf{A}}$  such that if  $f$  is a feature,  $d \in E$ , and  $df \downarrow$ , then  $df \in E$ , and (ii) subsets  $E_a$  of  $D_a$  such that  $E_a \subseteq E$ . We also define the principal subsystem  $P(d)$  generated by an element  $d$  of  $D^{\mathbf{A}}$ . The domain  $E(d)$  of this subsystem is the set  $\{d\pi \mid \pi \in L^* \wedge d\pi \downarrow\}$ , and  $E_a = D_a \cap E(d)$ . A feature system is point generated or principal if  $D = P(d_0)$  for some  $d_0 \in D$ . Another official name for such a system is feature structure.

We say that a feature system is finite if its domain is a finite set.

**Definition 2.4** (Homomorphism). A *homomorphism* between two feature systems  $\mathbf{A}$  and  $\mathbf{B}$  (of the same signature) is a total map  $\gamma$  between the two domains satisfying

(i) For any  $d \in D^{\mathbf{A}}$  and  $f \in L$ , if  $d\gamma \downarrow$  and  $df^{\mathbf{A}} \downarrow$ , then  $df^{\mathbf{A}}\gamma = d\gamma f^{\mathbf{B}}$ . (In particular,  $d\gamma f^{\mathbf{B}}$  is defined.)

(ii) Whenever  $d \in D_a^{\mathbf{A}}$ , we have  $d\gamma \in D_a^{\mathbf{B}}$ .

**Definition 2.5.** Let  $\mathbf{A}$  be a feature system. The *subsumption preorder*  $\sqsubseteq$  on  $\mathbf{A}$  is defined as follows:

$$d \sqsubseteq e \iff \text{there is a homomorphism } \gamma : P(d) \rightarrow P(e), \text{ with } d\gamma = e.$$

We say that  $d$  subsumes  $e$ .

### Examples.

- In the Term system, a term  $t$  subsumes a term  $u$  iff there is a substitution  $\theta$  such that  $t\theta = u$ .
- In the (Amy, Alice, etc.) system above, Alice subsumes Anne, and no other relationships hold other than the identity; for example Anne does not subsume Alice because Anne's hair color is instantiated, whereas Alice's is not defined; Anne does not subsume Albert because Albert has red hair and Anne's is brown; and Anne does not subsume Amy because Amy is not hungry.

The intuitive idea behind the definition of subsumption is that of information content. If an element  $d$  subsumes an element  $e$ , then in some sense  $e$  bears at least as much information as  $d$  does.

## 3. Feature Logics

We now introduce several logical languages which will be interpreted in variants of feature systems. All of these logics can be defined relative to the signature  $(L, A)$ .

### 3.1. Kasper-Rounds Logic

The language  $L(KR)$  was introduced in [40], [41] to solve the problem of expressing disjunctive information in feature structures, while at the same time capturing the constraints implicit in Shieber’s PATR-II [67]. The language  $L(KR)$  consists of *basic* and *compound* formulas. Assuming  $L$  and  $A$  as above, the basic formulas of  $L(KR)$  are as follows

- (Constants)  $a$  for each  $a \in A$ ;
- (Truth) The special formula *true*,
- (Path equations)  $p \doteq q$  for  $p, q \in L^*$ .

Then the compound formulas are given inductively. If  $\varphi$  and  $\psi$  are formulas, then so are

- $\varphi \wedge \psi$ ;
- $\varphi \vee \psi$ ;
- $l : \varphi$  for  $l \in L$ .

(In point of fact, Kasper and Rounds introduced, instead of path equations  $p \doteq q$ , finite sets  $E$  of paths as formulas, with the interpretation that all paths the set  $E$  were to be equal. We use the more readable notation here.)

A path equation models coreference constraints, as in PATR-II, and analogously in LFG. For example, we may have the PATR-II rule

$$VP \rightarrow V NP$$

with the constraint

$$V \text{ agr} \doteq NP \text{ agr}.$$

If we regard  $V$ ,  $NP$ , and “agr” as feature names, then this equation says that the “ $V$  feature” of the  $VP$  shares agreement information with the “ $NP$  feature”. Of course, in this case, the  $V$  and  $NP$  features refer to the actual constituents of the  $VP$ .

The semantics of Kasper-Rounds logic is straightforward given the concept of a feature system. Let  $\mathbf{A}$  be such a feature system, fixed for the discussion. Let  $d$  range over elements in  $D^{\mathbf{A}}$ . Then we say

- $d \models a$  if  $d \in D_a$ ;
- $d \models \textit{true}$  always;
- $d \models p \doteq q$  if  $dp = dq$ ;
- $d \models \varphi \wedge \psi$  if  $d \models \varphi$  and  $d \models \psi$ ;
- $d \models \varphi \vee \psi$  if  $d \models \varphi$  or  $d \models \psi$ ;
- $d \models l : \varphi$  if  $dl \models \varphi$  (implicitly  $dl$  is defined).

This definition is not quite the original semantics for the logic. There, the interpretation of a constant symbol  $a$  was required to be a set  $D_a$  consisting of just one

element, and distinct constants received distinct interpretations. (Such a feature system is called a *feature algebra* in [70], [19].)

Kasper and Rounds provided a number of basic results about their logic, including an (almost) complete axiom system. We defer discussion of completeness results until later, and point out the connections between logical formulas and unification.

Consider the feature graph system  $\mathbf{F}$ . An *isomorphism* of two graphs  $(G, n)$  and  $(H, m)$  is an ordinary graph isomorphism between the node sets of  $G$  and  $H$ , sending  $n$  to  $m$ . It is possible to show that the subsumption preorder on the feature graph algebra is a partial order, up to isomorphism. Further, if two graphs have an upper bound in this ordering, then they have a least such, up to isomorphism. This least upper bound (which can be calculated quite efficiently) is called the *unification* of the two graphs. (These results are due to Moshier [47].)

Any formula in Kasper-Rounds logic has a finite number of subsumption-minimal satisfying feature graphs, again up to isomorphism. The basic fact about the logic is then that if  $(G, n)$  is a minimal satisfier of the formula  $\varphi$ , and  $(H, m)$  of  $\psi$ , then the unification of  $(G, m)$  and  $(H, n)$  is a minimal satisfier of the conjunction  $\varphi \wedge \psi$ . This fact makes it possible to compute the minimal satisfiers of the conjunction of two disjunctive formulas by pairwise unifying the minimal satisfiers of the two separate formulas, and thus leads to an elucidation of the “semantics of disjunctive feature structures” by regarding a disjunctive structure not as a semantic entity, but instead, a disjunctive formula – a description of a set of non-disjunctive structures.

Another distinguishing characteristic of Kasper-Rounds logic is *persistence*. Because the logic contains no negations, we have the fact that if an element  $d$  satisfies  $\varphi$ , and  $d \sqsubseteq e$ , then  $e$  satisfies  $\varphi$  too. Combined with the results in the previous paragraph, this means that the set of satisfiers of any Kasper-Rounds formula can be described as the upward closure, in the subsumption preorder, of the set of minimal satisfiers.

Finally, a remark on the connection of Kasper-Rounds to logics of programming is in order. Since feature graphs are essentially nothing more than the graphs of deterministic finite automata, it follows that Kasper-Rounds logic formulas describe possible transition systems of this type, albeit in a very limited way. The formulas of the form  $l : \varphi$ , in particular, describe the state of a system after an “action”  $l$  has been performed. So we can think of  $l : \varphi$  as a kind of modal operator on formulas. This is the basic idea of dynamic logic [61], where  $l$  in general could be a large program. The direct progenitor of Kasper-Rounds logic is, in fact, a modal logic due to Hennessy and Milner [29], appropriate for describing nondeterministic transition systems.

### 3.2. Modal feature logics

Much has been made of the modal nature of Kasper-Rounds. Blackburn [8], Blackburn and Spaan [9], and Reape [63] have investigated this direction extensively. In their formulation, the symbols  $a$  can be thought of as special propositional variables, perhaps rewritten as  $p_a$ . The modality  $l : \varphi$  is written  $\langle l \rangle \varphi$ , and a negation opera-

tor  $\neg\phi$  is included. The use of the  $\langle \rangle$  notation is derived from the  $\diamond$  or possibility modality in standard modal logic.

A feature system  $\mathbf{A}$  can be rechristened a *Kripke model*, or *polyframe* in modal logic terminology. Now the elements  $d \in D$  of the structure are thought of as possible worlds, and the interpretation of the arc labels  $l$  as unary partial functions give several accessibility relations between the worlds.

Blackburn [8] shows how to treat path equations in the modal framework. He introduces *nominals*, an idea dating to Prior [62]. These are syntactic objects, written  $i, j, k$  (in effect constants in  $D$ ) which are always interpreted as singleton subsets of  $D$ . So, instead of having path equations, one has instead basic formulas  $i$ , where  $i$  is a nominal. To get the effect of the equation  $p \doteq q$ , one writes instead  $\langle p \rangle i \wedge \langle q \rangle i$ , where  $i$  is a (fresh) nominal.

The distinction between the use of nominals and the use of path equations may not be obvious at first sight. However, notice that the formula  $\langle p \rangle i \wedge \langle q \rangle i$  is not logically equivalent to the formula  $\langle p \rangle j \wedge \langle q \rangle j$ , where  $i$  and  $j$  are different nominals. This distinction shows up much more dramatically when the complexity issues for these logics are investigated. We will look at these problems below.

Blackburn and Spaan investigate various extensions of the basic modal formalism. The first of these is the *universal modality*  $\Box$ . Let  $\varphi$  be a formula of any of the types described so far. Then the semantics of  $\Box\varphi$  is simply described as follows. Fix a Kripke model  $\mathbf{A}$ , and let  $d \in D^{\mathbf{A}}$ . Then

$$d \models \Box\varphi \iff (\forall e \in D) (e \models \varphi).$$

The universal modality finds application in GPSG: consider the *feature co-occurrence restriction*

$$[\text{VFORM}] \supset [-\text{N}, +\text{V}].$$

This says that any category classified as a verb form must have a positive “noun” and a negative “verb” quality. Phrased in modal feature logic, this would read

$$\Box(\text{vform: true} \rightarrow \text{n: +} \wedge \text{v: -})$$

where VFORM, n, and v are attributes.

Another use of the modality would be to express the Head Feature Convention of GPSG and HPSG. This is a constraint which insists that the values of certain features occurring in constituents like verb phrases which are “headed” by verbs have the same values as features of the head as they do as features of the headed phrase. This would be expressed as a path equation in the scope of the universal modality.

Blackburn and Spaan cite the work of Evans [20], where the  $\Box$  modality is used in its dual form  $\diamond = \neg\Box\neg$  to express feature specification defaults. Specifically, the formula

$$(\diamond \text{ CASE: dat}) \rightarrow \text{CASE: dat}$$

is supposed to express that if it is consistent with known information that the case of a constituent is dative, then the case is dative. Unfortunately this use of the modality, when interpreted as the dual of the universal modality, does not express what is intended, as it says that if somewhere in a feature structure there is an occurrence of the CASE feature with a dative value, then there is an occurrence of the feature at the current level with the dative value. (Reparentesizing the formula does not help, either, as the reader may check.) It seems that a more sophisticated approach to specifying defaults is necessary.

Gazdar *et al.* [23] also define a *master modality*  $[*]\varphi$ . The conditions for this are

$$d \models [*]\varphi \iff d \models \varphi \wedge dl \models [*]\varphi \text{ for all } l \in L \text{ such that } dl \text{ is defined.}$$

This definition is recursive, but plainly its force is the following, as pointed out by Blackburn and Spaan:

$$d \models \Box\varphi \text{ if } (\forall e \in P(d)) (e \models \varphi)$$

where  $P(d)$  is the set of all worlds reachable from  $d$  by any path (that is, the principal subsystem generated by  $d$ ). This modality is very close to the universal modality in expressive power. The main interest is that it is defined recursively. This technique is of independent interest in feature logic, as we will see when we consider Moshier's fixed-point extensions. We should also mention that the master modality was studied in detail by Kracht [43], and preceded the universal modality historically.

### 3.3. Heterogeneous modalities

We briefly look at the idea of *set-valued* feature systems. The basic idea in these systems is to notice that a set can be pictured graphically like a transition system. The elementhood relation is of course a binary relation on sets, so has a graph. Allowing this graph to have cycles leads to the idea of a non-well-founded set, and to the work of Aczel on the anti-foundation axiom [2].

The simplest way to model feature structures with set values is to add a binary relation (representing set membership) to a feature system. If  $R$  is this binary relation, then by  $d R e$  we mean that  $e$  is one of the members of  $d$  considered as a set. So an element of a feature system may have *attributes* (represented by feature arcs) as well as *members* (represented by  $R$  arcs). Obviously the relation  $R$  need not be functional, and obviously conditions on  $R$  will be required to make the extended graphs have set-like properties; in particular, to satisfy an extensionality property when sets can be non-well-founded.

Ignoring these difficulties for the moment, one way to give a modal feature logic encompassing set values is to add a new modality  $\Box_s$  to our language. The semantics will be very simple:

$$d \models \Box_s\varphi \iff \text{for all } e \text{ with } d R e, e \models \varphi.$$

The logic obtained in this way is a combination of feature logic with the standard Kripke modal logic  $K$ , since there are no conditions on the relation  $R$ . Put another way, the following formulas are valid:

$$\Box_s \text{true} \leftrightarrow \text{true}$$

and

$$\Box_s (\varphi \wedge \psi) \leftrightarrow \Box_s \varphi \wedge \Box_s \psi.$$

This is observed by Moss [50] in his paper on completeness theorems for feature logics. It is not the full story, however, since the relation  $R$  is not arbitrary, but has to satisfy extensionality conditions; more properly, to get a set theory out, one must axiomatize the relation between graphs and the sets they picture.

Some of the uses of set values arise when one is modelling linguistic objects containing a number of components, but in which the particular order or enumeration of the components is irrelevant. Consider, for example, of a list of NP conjuncts such as “apples and oranges”. The situation in general is more complicated. Pollard and Moshier [57] study the use of set values in HPSG. Consider the sentence “She likes him”. This might be partially modelled by the following feature structure:

$$\left[ \begin{array}{l} \text{CONTENT} \left[ \begin{array}{l} \text{EXPERIENCER } \boxed{1} \\ \text{THEME } \boxed{2} \end{array} \right] \\ \text{CONTEXT } \{ \boxed{1} \left[ \begin{array}{l} \text{PERSON 3rd} \\ \text{NUMBER sing} \\ \text{GENDER fem} \end{array} \right], \boxed{2} \left[ \begin{array}{l} \text{PERSON 3rd} \\ \text{NUMBER sing} \\ \text{GENDER masc} \end{array} \right] \} \end{array} \right]$$

The value of the CONTEXT attribute is to be a set consisting of two parameters, each qualified to be of the appropriate person, number, and gender. These parameters are coreferential with the values of the EXPERIENCER and THEME attributes of the sentence.

Now we are faced with the problem of interpreting the “dotted brace” notation, and the notations within it. This is a problem independent of the logic used to describe such objects. The value of the CONTEXT attribute is a “set object”, related by the relation  $R$  to the values of the EXPERIENCER and THEME attributes. For Pollard and Moshier, there cannot be more than two values in the set which is the value of the CONTEXT attribute. But, if this set value is merely a “partial set”, one could hypothesize that potentially there could be more elements in the set. It all depends on the way one extends the idea of subsumption to set-valued attributes. We defer discussion until we have completed our survey of feature logics.

### 3.4. Reape's polyadic modal logic

We now come to a far-reaching generalization of feature logic. In fact, so much is contained in the generalization that the feature component of the logic is almost insignificant. This formalism is due to Reape ([63], especially chapter 4.)

Reape begins with the syntax of his logic, but it seems to be more perspicacious to start with the semantics. Assume that we are given a feature system  $\mathbf{A}$ . The set  $D^{\mathbf{A}}$ , in addition to being the domain of linguistic objects, may also possess additional mathematical structure. For example,  $D$  may be a Boolean algebra or a monoid. An example would be the case of set-valued feature structures, where we assume that the elements are sets (for the moment, let us say everything in  $D$  is a set). Then of course the set operations of union, intersection, and complement are available, as well as the membership relation.

Fix a feature signature as before, but augment the signature with function and relation symbols from some auxiliary set  $\mathcal{R}$ . So, in the set example, we use the signs for union, intersection, and complement, together with a function symbol  $\{\cdot\}$  which takes an individual into the set consisting of that individual. Now the innovation in Reape's formalism is to regard these new symbols as modalities, just as features were regarded before. The feature matrix of Moshier and Pollard above, for example, can be thought of as a conjunctive formula read as follows:

$$\text{content: experiencer: } x \wedge \text{ theme: } y \wedge \text{ context: } \varphi$$

where  $\varphi$  is the formula

$$\{x \wedge \text{ pers: } 3 \wedge \text{ num: sing} \wedge \text{ gender: f}\} \cup \{y \wedge \text{ pers: } 3 \wedge \text{ num: sing} \wedge \text{ gender: m}\}.$$

The difference between Reape's formalism and the structures of Moshier and Pollard then becomes one of the level at which one thinks of descriptions. For Moshier and Pollard, feature structures are semantic objects, to be further constrained by (an unspecified) logic. These semantic objects are themselves descriptive entities, furnishing partial information about an empirical domain. For Reape, the partiality enters at the level of syntax.

The full syntax of the Reape formalism is then as follows. Given a feature system  $\mathbf{A}$  and auxiliary signature  $\mathcal{R}$ , we allow all the KR formulas, together with the rules that if  $\varphi_1, \dots, \varphi_n$  are formulas, then so are

$$r(\varphi_1, \dots, \varphi_n),$$

for  $r$  an  $n$ -ary relation symbol, and

$$f(\varphi_1, \dots, \varphi_n)$$

for  $f$  an auxiliary function symbol.

Turning to the formal semantics, the auxiliary functions are interpreted over  $D^{\mathbf{A}}$  as usual: an  $n$ -ary function symbol is interpreted as an  $n$ -ary function on  $D$ .

However, a relation symbol  $r$  is interpreted as an  $(n + 1)$ -ary relation. The idea is that if  $(d_1, \dots, d_n, d)$  is in the interpretation of  $r$ , then the elements  $d_1, \dots, d_n$  stand in the relation  $r$  with respect to the current state  $d$ . An example will clarify this: instead of the union and set-former  $\{\cdot\}$  of the above example, suppose instead we have the element-of relation  $in(x, z)$ , indicating that  $x$  is a member of  $z$ . Then the above formula could be written

$$\text{content: experiencer: } x \wedge \text{ theme: } y \wedge \text{ context: } z \wedge in(x, z) \wedge in(y, z)$$

conjoined with the formula

$$x \wedge \text{pers:}3 \wedge \text{num: sing} \wedge \text{sex: f} \wedge y \wedge \text{pers:}3 \wedge \text{num: sing} \wedge \text{sex: m}.$$

(In fact, this new formula expresses something slightly different from the old one, in that it does not prevent the set  $z$  from having more members.)

Let the interpretation of  $r$  be  $I(r)$  and of  $f$  be  $I(f)$ . As usual let  $d \in d^{\mathbf{A}}$ .

- $d \models f(\varphi_1, \dots, \varphi_n)$  iff for some  $d_1, \dots, d_n \in D$ , we have  $I(f)(d_1, \dots, d_n) = d$ ;
- $d \models r(\varphi_1, \dots, \varphi_n)$  iff for some  $d_1, \dots, d_n \in D$ , we have  $I(r)(d_1, \dots, d_n, d)$ .

Notice the existential character of these polymodalities; also notice that if the formulas  $\varphi_i$  are nominals, then there will be a unique choice of the  $d_i$  in the above clauses. Finally, notice that if  $f$  is a unary function, then the interpretation of  $f(\varphi)$  will not be the same as if  $f$  were considered a feature, and the formula were  $f : \varphi$ . In the first case,  $d \models f(x)$  if for the element  $d_x$  interpreting the nominal  $x$ , we have  $d = f(d_x)$ . In the second case,  $d \models f : x$  if  $d_x = f(d)$ . That is to say, the auxiliary function symbols are used as *constructors*, while the features are used as *selectors*.

Reape provides an axiom system and proof of completeness for his system, but does not investigate notions like subsumption. This seems to be because his semantic objects are total ones, not partial objects as in Kasper-Rounds and Moshier-Pollard. In fact, the view of feature structures as partial objects is not important for many of the modal logics extending Kasper-Rounds. What these logics bring is a wealth of techniques for understanding expressiveness, completeness, compactness, and complexity.

**Bibliographic remarks on modal logics.** Much more is known on polyadic modal logic in general. They have been extensively studied by Goldblatt [24]. Decidability results for KR logic, and indeed the logics in Kracht [43] appear in much stronger form for *deterministic propositional dynamic logic* [7].

### 3.5. Negation and implication

Now we consider adding connectives to Kasper-Rounds logic which (in the partial-description view) destroy persistence, at least if they are given a classical interpretation. Syntactically, we allow to form negations  $\neg\varphi$  and  $\varphi \rightarrow \psi$ . The difficulty comes in giving the semantics.

One can, of course adopt the classical negation:

$$- d \models \neg f \text{ iff not } d \models f$$

with implication being defined as the standard material notion. This makes sense mathematically, especially if feature structures are regarded as total objects. However, consider what happens if feature structures are partial descriptions. Then if the feature  $l$  is not defined at a node  $d$ , it may be because we do not have the information at the present time that it is defined. Typically we get this behavior when we are parsing, and we do not know what the object of a verb might be, for example – even whether the verb will take an object. If we say  $\neg person : third$ , then according to the classical semantics, either the feature  $person$  is undefined at the current node, or it is defined, and the value is not “third”. We really would rather have our specification mean this second possibility. As we have indicated, if features were always total functions, this is what we would get, but we do not wish features always to be total.

A simple way out of this difficulty was proposed by Dawar and Vijay-Shanker [17]. The basic idea is to adopt a strong Kleene three-valued semantics for feature logic. To express this concisely, we recall the truth tables for the Boolean connectives proposed by Kleene:

$\wedge$	<b>t</b>	<b>f</b>	<b>u</b>	$\vee$	<b>t</b>	<b>f</b>	<b>u</b>	$p$	$\neg p$
<b>t</b>	<b>t</b>	<b>f</b>	<b>u</b>	<b>t</b>	<b>t</b>	<b>t</b>	<b>t</b>	<b>t</b>	<b>f</b>
<b>f</b>	<b>f</b>	<b>f</b>	<b>f</b>	<b>f</b>	<b>t</b>	<b>f</b>	<b>u</b>	<b>f</b>	<b>t</b>
<b>u</b>	<b>u</b>	<b>f</b>	<b>u</b>	<b>u</b>	<b>t</b>	<b>u</b>	<b>u</b>	<b>u</b>	<b>u</b>

in which **u** stands for “undefined.”

Let  $\mathbf{A}$  be a feature system. If  $d \in D^{\mathbf{A}}$ , and  $\varphi$  is a feature logic formula with negation, we can define the *valuation*  $\llbracket \varphi \rrbracket(d)$  recursively on the structure of  $\varphi$ :

- (i)  $\llbracket true \rrbracket(d) = t$ ;
- (ii)  $\llbracket a \rrbracket(d) = \begin{cases} t & \text{if } d \in D_a; \\ f & \text{otherwise.} \end{cases}$
- (iii)  $\llbracket p \doteq q \rrbracket(d) = \begin{cases} t & \text{if } dp = dq; \\ f & \text{if } dp \text{ and } dq \text{ are both defined and unequal;} \\ u & \text{otherwise.} \end{cases}$
- (iv)  $\llbracket l : \varphi \rrbracket(d) = \begin{cases} t & \text{if } \llbracket \varphi \rrbracket(dl) = t; \\ f & \text{if } \llbracket \varphi \rrbracket(dl) = f; \\ u & \text{otherwise.} \end{cases}$
- (v)  $\llbracket \varphi \wedge \psi \rrbracket(d) = \llbracket \varphi \rrbracket(d) \wedge \llbracket \psi \rrbracket(d)$ ;
- (vi)  $\llbracket \varphi \vee \psi \rrbracket(d) = \llbracket \varphi \rrbracket(d) \vee \llbracket \psi \rrbracket(d)$ ;
- (vii)  $\llbracket \neg \varphi \rrbracket(d) = \neg \llbracket \varphi \rrbracket(d)$

where of course the functions  $\wedge$ , and so forth, on the right side are the Kleene three-valued functions.

One would like, with this definition, to recapture the persistence property of

Kasper-Rounds logic augmented with negation. Stated again, if  $\llbracket \varphi \rrbracket(d) = t$ , and  $d \sqsubseteq e$ , then  $\llbracket \varphi \rrbracket(e) = t$ . Unfortunately this result does not quite go through, because of clause (iii) of the definition. To remedy this problem, Dawar and Vijay-Shanker replace the condition that  $dp$  and  $dq$  are both defined and unequal, by the condition that these elements are defined but not unifiable. Although this solves the problem, it requires that one literally interpret the logic in a feature system which supports unification, like the feature graph system or the Nerode system. The other problem with the three-valued interpretation above is that the formula  $\neg l : true$  is unsatisfiable, in fact logically equivalent to  $false = \neg true$ . This means that if we define  $\varphi \rightarrow \psi$  as the material conditional  $\neg \varphi \vee \psi$  then we get intuitively incorrect behavior. For example, suppose we want to assert that if the feature  $l$  is defined at  $d$ , then the value of the *case* feature must be *dative*. One would naturally write

$$l : true \rightarrow case : dat$$

but this formula is just equivalent to  $case : dat$ . Here the problem is that the material conditional is intuitively expressed, not with the strong Kleene negation, but with the weak version. A way out of the second difficulty is suggested by Dawar and Vijay-Shanker. In this approach, one looks not at the logic, but at the feature system definition itself. The idea is to specify in the semantics, the set of features which can *never* be defined at a node. If one does this, then the formula  $\neg l : true$ , for example, could be satisfiable by a node where the feature  $l$  is prohibited.

This idea generalizes: the book by Carpenter [14] shows how to assign types to nodes in such a way that the allowed features for a node are specified as part of the type. Typing also suggests a way out of the path equation persistence problem as well, and in fact Carpenter provides an elegant solution to the problem of negated path equations. We return to the logic of typed structures below.

### 3.6. Intuitionistic logic

The problem of saying that a feature can *never* become defined can, of course, be handled as a modal statement, and also by the idea implicit in intuitionistic logic that a negative statement is one that can be positively refuted. This chapter is not the place to review these well-known ideas, but this section is appropriate for mentioning the intuitionistic version of feature logic introduced by Moshier and Rounds [49], in fact to recapture the persistence property of Kasper-Rounds while at the same time dealing with the problems mentioned in the section on three-valued systems.

The syntax of feature logic remains the same in this section; we have Kasper-Rounds augmented with negation and implication (and the formula *false*.) However, we have a new kind of Kripke structure for interpreting formulae. We follow here the definition of Moss [50], as it simplifies the notions in Moshier-Rounds considerably. In further point of fact, this presentation simplifies again Moss' definitions, while retaining the essential flavor.

The basic idea is to interpret formulae over a Kripke structure, where now in addition to the accessibilities provided by the features, we use the subsumption relation  $\sqsubseteq$  as another accessibility notion. Moss does not work with the relation given above, but with a weaker notion: instead of using a homomorphism as a way to relate nodes  $d$  and  $e$ , he uses a more general relation. So let us say that a relation  $\preceq$  on the nodes of a feature system  $\mathbf{A}$  is an *approximation* if

$$d \preceq e \text{ and } dl \text{ is defined} \Rightarrow dl r el$$

and

$$d \preceq e \text{ and } d \in D_a \Rightarrow e \in D_a.$$

To deal with preservation of path equalities, we also require

$$d \preceq e \text{ and } dp = dq \Rightarrow ep = eq.$$

By a Kripke structure we now mean a structure  $(\mathbf{A}, \preceq)$  where  $\mathbf{A}$  is a feature system and  $\preceq$  is an approximation relation.

Instead of the satisfaction relation  $\models$  used above, we now consider a forcing relation  $\Vdash$  to interpret formulas. The clauses are as follows, where  $\mathbf{A}$  is fixed and  $d \in D^{\mathbf{A}}$ :

- $d \Vdash a$  if  $d \in D_a$ ;
- $d \Vdash true$  always;
- $d \Vdash false$  never;
- $d \Vdash p \doteq q$  if  $dp = dq$ ;
- $d \Vdash \varphi \wedge \psi$  if  $d \Vdash \varphi$  and  $d \Vdash \psi$ ;
- $d \Vdash \varphi \vee \psi$  if  $d \Vdash \varphi$  or  $d \Vdash \psi$ ;
- $d \Vdash l : \phi$  if  $dl \Vdash \phi$  (implicitly  $dl$  is defined);
- $d \Vdash \varphi \rightarrow \psi$  if for all  $e$  such that  $d \preceq e$ , if  $e \Vdash \varphi$ , then  $e \Vdash \psi$ .

We do not need to include negation in this definition, since  $\neg\varphi$  can be defined as  $\varphi \rightarrow false$ .

Now notice that, as in the standard intuitionistic logics, that the law of the excluded middle may fail. Indeed, the formula  $l : true$  is not forced by a node with the feature  $l$  undefined, but neither is the formula  $\neg(l : true)$ , as we may have another element  $e$  where the feature  $l$  is defined. But we now have persistence once again, and we do not have that  $l : true \rightarrow case : dat$  is logically equivalent to  $case : dat$  (here, logical equivalence means forced equivalently over all Kripke structures).

Once again we defer the properties of such a system to a later section, and continue with our survey of feature logics.

## 3.7. Fixed-point extensions

We now come to another extension of Kasper-Rounds logic which arises from the desire to define formulas recursively. An example of this was the master modality of Gazdar et al. [23] which we saw in Section 3.2. This extension is due to Moshier [47]. If we use a strong logic of this form, then many of the ideas in Kay’s *functional unification grammar* can be captured (for a preliminary attempt to do this, see Rounds and Manaster Ramer [65]). The point is that a grammar can be viewed as a logic incorporating a natural recursion construct.

In the following, we will ignore some of the formula syntax introduced by Moshier. Instead we will concentrate on the recursive nature of the logic. We augment our syntax with a set of *variables*  $V$ . An interpretation  $\mathbf{A}$  and an *environment*  $\rho : V \rightarrow \text{Pow}(D^{\mathbf{A}})$  will be required to interpret formulas. The idea of an environment is first that variables intuitively represent “places” in a formula where other formulas can be substituted. Second, an environment will tell us, for one of these variables, what the set of satisfying nodes is for the formula that it stands for.

The syntax of Moshier’s logic  $L(EKR1)$  is then the same as for Kasper-Rounds, with the addition of the following clauses:

- Every  $v \in V$  is a formula;
- If  $\varphi_1, \dots, \varphi_n, \varphi$  are formulas, and  $x_1, \dots, x_n \in V$ , then  $[x_1 \leftarrow \varphi_1, \dots, x_n \leftarrow \varphi_n]\varphi$  is a formula.

The idea of the recursion construct is that the formula  $\varphi$  will contain free occurrences of the  $x_i$ , and that successively repeating substitutions given in the bracketed part will lead to a kind of “infinitary” formula.

For example, consider the (as yet undefined) formula  $\langle l^* \rangle a$ , which, to be satisfied at a node, requires there to be some finite sequence of  $l$ ’s leading to a node satisfying  $a$ . This could be written as an infinitary formula

$$x = a \vee l : a \vee l : l : a \vee \dots$$

Equivalently, it seemingly obeys the “equation”

$$x = a \vee l : x.$$

The intent of this is captured by the  $L(EKR1)$  formula

$$[x \leftarrow (a \vee l : x)]x$$

which could be thought of as a programming instruction to iteratively replace  $x$  by  $a \vee l : x$ . It turns out, however, that a more elegant semantics for the recursion operator can be given not by a notion of substitution, but by a compositional semantic definition making elementary use of fixed-point notions from recursion theory and computer science. This is in fact the way recursive constructs typically are defined in programming language semantics; the idea was pioneered in computer

science by Scott [66]. The adaptation to feature logic follows work by Immerman [31], Vardi [72], and Blass and Gurevich [10].

The semantics of  $L(EKR1)$  can be given by simple equations, but “unpacking” these equations leads to complexities, so we will be slow in our explanations. The basic idea is to define, for a formula  $\varphi$  and environment  $\rho$ , a subset  $\llbracket \varphi \rrbracket(\rho)$  of  $D^{\mathbf{A}}$ , where as usual  $\mathbf{A}$  is a feature system. The set  $\llbracket \varphi \rrbracket(\rho)$  is the set of nodes which satisfy the formula  $\varphi$ , given that the set  $\rho(x)$  is the set of nodes satisfying the “placeholder” formula  $x$  occurring free in  $\varphi$ . We therefore define (given a feature system  $\mathbf{A}$ ) the *denotation*  $\llbracket \varphi \rrbracket$  as a function from environments to subsets of  $D^{\mathbf{A}}$ , by induction on the structure of  $\varphi$ :

- $\llbracket a \rrbracket(\rho) = D_a$ ;
- $\llbracket true \rrbracket(\rho) = D^{\mathbf{A}}$ ;
- $\llbracket p \doteq q \rrbracket(\rho) = \{d \in D^{\mathbf{A}} \mid dp = dq\}$ ;
- $\llbracket x \rrbracket(\rho) = \rho(x)$ ;
- $\llbracket l : \varphi \rrbracket(\rho) = \{d \in D^{\mathbf{A}} \mid dl \in \llbracket \varphi \rrbracket(\rho)\}$ ;
- $\llbracket \varphi \wedge \psi \rrbracket(\rho) = \llbracket \varphi \rrbracket(\rho) \cap \llbracket \psi \rrbracket(\rho)$ ;
- $\llbracket \varphi \vee \psi \rrbracket(\rho) = \llbracket \varphi \rrbracket(\rho) \cup \llbracket \psi \rrbracket(\rho)$ ;
- $\llbracket [x_1 \leftarrow \varphi_1, \dots, x_n \leftarrow \varphi_n] \varphi \rrbracket(\rho) = \llbracket \varphi \rrbracket(\rho^*)$ , where  $\rho^*$  is the *least fixed point* of a certain monotonic operator  $T[x_1 \leftarrow \varphi_1, \dots, x_n \leftarrow \varphi_n; \rho]$  from environments to environments. The definition of  $T$  is

$$T[x_1 \leftarrow \varphi_1; \dots; x_n \leftarrow \varphi_n; \rho](\tau)(x) = \begin{cases} \llbracket \varphi_1 \rrbracket(\tau)(x) & \text{if } x = x_1; \\ \vdots \\ \llbracket \varphi_n \rrbracket(\tau)(x) & \text{if } x = x_n; \\ \rho(x) & \text{otherwise.} \end{cases}$$

**Example 3.1.** Consider the formula  $\psi = [x_1 \leftarrow (a \vee l : x_1)]x_1$ . We calculate the meaning  $\llbracket \psi \rrbracket$  in the Feature Tree system (cf. ex. 2.2.) The formula is a fixpoint formula, satisfying the last clause of the inductive definition. So in this case, the definition of  $T$  reduces to:

$$T[x_1 \leftarrow (a \vee l : x_1); \rho](\tau)(x) = \begin{cases} \llbracket a \vee l : x_1 \rrbracket(\tau)(x) & \text{if } x = x_1; \\ \rho(x) & \text{otherwise.} \end{cases}$$

This in turn reduces to

$$T[x_1 \leftarrow (a \vee l : x_1); \rho](\tau)(x) = \begin{cases} \{a\} \cup \{t \mid t/l \in \tau(x)\} & \text{if } x = x_1; \\ \rho(x) & \text{otherwise.} \end{cases}$$

The least fixed point of this operator can be shown to be

$$\rho^*(x) = \begin{cases} \{a\} \cup \{t \mid (\exists n > 0) (t/(l^n) = a)\} & \text{if } x = x_1; \\ \rho(x) & \text{otherwise.} \end{cases}$$

The meaning of  $\psi$  is therefore

$$\llbracket x_1 \rrbracket(\rho^*) = \rho^*(x_1) = \{a\} \cup \{t \mid (\exists n > 0) (t/(l^n) = a)\}.$$

This is the set of all feature trees which are either the atom  $a$  or have a path of  $l$ 's of some nonzero length leading to the leaf node labeled  $a$ , which accords exactly with the intuitive “infinitary” semantics proposed above.

What are these “least fixed points”? We still have to explain them, and demonstrate their existence. We will be using the classical theorem, variously attributed to Tarski, Knaster, and Kleene, that a monotonic operator  $F$  on a complete lattice  $(R, \sqsubseteq)$  has a least fixed point: namely, an  $x \in R$  such that  $F(x) = x$  and for any other such  $y$ , we have  $x \sqsubseteq y$ . (To say that  $F$  is monotonic means that if  $w \sqsubseteq z$  then  $F(w) \sqsubseteq F(z)$ .)

In our case, we take the lattice  $R$  to be the set of all environments  $\rho$  from the set of variables  $X$  to  $\text{Pow}(D^{\mathbf{A}})$ , and the ordering

$$\rho \sqsubseteq \tau \text{ iff } \rho(x) \subseteq \tau(x) \text{ for all } x.$$

It remains to show that the operators  $T$  above are all actually monotonic. This is an easy consequence (in view of the definition of  $T$ ) of the fact that the meaning  $\llbracket \varphi \rrbracket$  is, for fixed  $\varphi$ , a monotonic function on environments. And the latter fact follows by induction on the structure of  $L(EKR1)$  formulas. Notice that here, by the way, is where the nonmonotonic operator of classical negation would invalidate the result.

The fixpoint existence theorem can be strengthened in special cases. If  $F$  is not only monotonic, but preserves all suprema of chains (i.e., is *continuous*) then there is a constructive way to calculate the least fixed point. Namely, the least fixed point  $x$  is given by

$$x = \bigvee_{i=1}^{\infty} F^i(-)$$

where  $-$  is the least element of the lattice  $R$ , and  $F^n$  is the  $n$ -fold composition of  $F$  with itself. This formula was used to calculate  $\rho^*$  in the example above.

There are many properties of the  $L(EKR1)$  system. These include the usual logical properties, of which perhaps the most interesting is Moshier’s result that  $L(EKR1)$  enjoys a finite model property. (Compare the analogous result, due to Fischer and Ladner [21] for propositional dynamic logic [61].) The result for  $L(EKR1)$  is possibly more complex, as more than just “regular” modalities can be introduced by fixpoint formulas.

Moshier applies his logic not only to the description of natural language syntax, but to computer language syntax as well. One example is the description of closed expressions of untyped lambda calculus. This formal language is not context-free, because of the problem of specifying variable binding. Its grammar, though, is naturally expressed in  $L(EKR1)$ . Another example is the collection of all closed

well-typable expressions of the untyped calculus. Here we must also ensure that occurrences of the same variable are all of the same type. However, not all programming languages are naturally specified in  $L(EKR1)$ . The specification of polymorphic types, as in the programming language **ML**, seems to require an extension of  $L(EKR1)$  which allows the relation of subsumption to be a primitive relation of the language. These issues are beyond the scope of the chapter, but see Shieber [69] for a discussion.

Recursion is also considered by Baader et al [5], and by Carpenter [14]. We turn now to a discussion of Carpenter’s general treatment of feature logic.

### 3.8. The logic of typed feature structures

Carpenter’s book *The Logic of Typed Feature Structures* [14, chapter 6 ff.] involves making assumptions about feature structures themselves, as well as about their logic.

Carpenter is concerned with the use of feature structures in HPSG, and orients his discussion to the use HPSG will make of so-called *types*. According to the general theory, knowledge types of linguistic objects is what is shared by speakers of a language. So the problem of grammar is to specify the ways in which tokens of types may be produced and recognized.

To put type constraints on feature structures, Carpenter first assumes that there is a certain ordering relation prespecified on the sort set  $A$ . The reasons for this are partly computational and partly to accord with one of the usual notions in artificial intelligence: that of an *inheritance hierarchy*. This notion was spelled out in general for programming languages by Cardelli and Wegner [13], where there are good reasons to associate types with each object. The first realization of typed feature structures themselves is probably due to Ait-Kaci [3], in which the computational reasons for type checking via inheritance are made quite clear.

We already have a definition of feature structure as a principal or point-generated feature system. (cf def. 2.3.) For the present, we only need to augment feature structures with a simple notion of type: when we say that a node  $d$  is of type  $a$ , where  $a \in A$ , then we mean that  $d \in D_a$ .

The technical condition placed on the type ordering  $\sqsubseteq$  on  $A$  is that it be a *bounded complete partial order*, or *Scott domain*. For now we avoid certain complications by assuming that  $A$  is finite. The conditions then are that there be a least element  $\perp$ , which stands for the “most general” or “least informative” type, and that any two elements which have an upper bound in the ordering have a least or most general such. In this setting, the most general types appear at the bottom of the ordering, contrary to the usual practice in artificial intelligence. An example of such an ordering might place the type *sign* at the bottom, with *phrase* as a subtype being more specific, and thus “higher” in the ordering. The condition on upper bounds is made so that the result of unifying structures will be uniquely defined when it is possible to unify them.

The second kind of restriction placed on feature structures is called an *appropri-*

*ateness condition.* For example, an attribute of *number* is not appropriate for the general type *sign*. But a feature which is appropriate for a type should be appropriate for any of its more specific subtypes. The idea of appropriateness is also a way of specifying negative information directly in feature structures. If a feature is not appropriate for a type, then it can never be defined at a node of a feature structure which has that type. Appropriateness is also a way of expressing at least some of the feature co-occurrence restrictions of GPSG.

Initially, adding the notion of appropriateness has little to do with feature logic itself. For example, Kasper-Rounds can be interpreted in the collection of typed feature structures. But it is clear that some of the validities of the logic will change. For instance, if a feature  $l$  is not appropriate for a type  $a$  then  $a \wedge l : true$  will be unsatisfiable.

Carpenter formalizes appropriateness restrictions as follows. Let  $(A, \sqsubseteq)$  be the partially ordered set of sorts (primitive types.) An *appropriateness specification* is a partial function  $Approp : L \times A \rightarrow A$  satisfying the following:

(i) For every feature  $l \in L$ , there is a most general type  $Intro(l)$  such that  $Approp(l, Intro(l))$  is defined;

(ii) If  $Approp(l, a)$  is defined, and  $a \sqsubseteq b$ , then  $Approp(l, b)$  is also defined and  $Approp(l, a) \sqsubseteq Approp(l, b)$ .

The intent of this definition is as follows. Think of the value of  $Approp(l, a)$  as the most general type of value taken by feature  $f$  given that its argument is of type  $a$ . If the value is undefined, this means that the feature  $l$  is not appropriate for the type  $a$ . Then, condition (i) says that for each feature, there is a most general type for which it is appropriate. So if we know that a feature structure satisfies  $f : true$ , we can infer the most general type that can occur at the root of that structure. The second condition says that if a feature is appropriate for a type, then this is passed on to subtypes, and further, that the type of feature values for more specific kinds of inputs must be more specific than the general class of outputs. This is a standard requirement for most inheritance system specifications.

It should perhaps be pointed out that appropriateness specifications are analogous to the notion of *database schemata* in relational database theory. Such schemata determine the “shapes” of relations occurring in any databases using them. So for example, employees have certain attributes, such as *name*, *salary*, and so forth. The attribute *weight* would perhaps not be appropriate.

Appropriateness conditions are not the only place where types and typed feature structures can be useful, however. Carpenter has many other applications. We concentrate here on just one such: *recursive type constraint systems* [14, chapter 15]. These constraint systems form an interesting class of logical formulae which are connected on the one hand to  $L(EKR1)$  formulae, and on the other to the various kinds of master modalities. We begin with an example.

**Example 3.2.** Suppose that *person* is a type, and consider the expression

$$person \Rightarrow adam \vee father: person.$$

Intuitively, this expresses a structured type, whose members are all those persons who have some chain of fathers leading back to Adam. In  $L(EKR1)$  this could be expressed by

$$[adam \vee father : x]x.$$

However, consider the closely related expression

$$person \Rightarrow employer : person.$$

We might wish the constraint to be satisfied by the cyclic feature structure consisting of one node of type *person*, with a single attribute *employer* pointing back to that same node. But the corresponding  $L(EKR1)$  formula denotes the empty set of feature structures. And indeed, these cyclic structures are excluded from the set of structures satisfying the Biblical *EKR1* formula above.

To allow such cyclic structures as solutions, Carpenter formalizes constraint systems as follows: Let  $L(KR)$  be the set of Kasper-Rounds formulae, and  $A$  be the set of sort symbols. A *constraint system* is then a mapping  $C : A \rightarrow L(KR)$ . To deal with inheritance correctly, we then extend  $C$  as follows:

$$C^*(a) = \bigcap_{b \sqsubseteq a} C(b).$$

Now let  $\mathbf{A}$  be a feature system, and  $d \in D^{\mathbf{A}}$ . Then  $d$  is said to be *resolved with respect to  $C$*  if for all  $p \in l^*$  for which  $dp$  is defined, we have

$$dp \models C^*(a) \text{ for all } a \text{ such that } dp \in D_a.$$

This notion of “solution” is clearly related to the master modality described above in the section on modal logics. Suppose that we allow KR formulas of type  $a \rightarrow \varphi$ , where  $\varphi$  contained no implication signs, and  $a \in A$ . Then we could translate  $C$  into a conjunction of formulas

$$C^T = \bigwedge_{a \in A} (a \rightarrow \varphi_a).$$

We then apply the master modality:

$$[*]C^T.$$

The semantics of  $a \rightarrow \varphi$  is straightforward: Given a feature system  $\mathbf{A}$ , and  $d \in D^{\mathbf{A}}$ , we say

$$d \models (a \Rightarrow \varphi) \iff d \notin D_a \text{ or } d \models \varphi.$$

If now  $a \sqsubseteq b$  implies that  $D_a \supseteq D_b$ , then  $d$  is resolved with respect to  $C$  iff  $d \models [*]C^T$ .

### 3.9. Attribute-value logics

We next turn to a brand of feature logic much more closely related to standard first-order logic. This begins with Johnson’s work on attribute-value logic, and is extended through the work of Smolka and others on “feature logic.” Of course we have been treating “feature logic” all along, but in this section the term is reserved for the family of logics in the Johnson and Smolka tradition.

Johnson’s work [35] focuses on the concept of *attribute-value structures* as the semantic space over which logics are to be interpreted. The shape of these structures differs from feature systems, so let us give Johnson’s own formulation:

**Definition 3.1.** *An attribute-value structure is a tuple  $\langle F, C, \delta \rangle$ , where  $F$  is a set (of attributes or features),  $C$  is a subset of  $F$ , and  $\delta$  is a partial function from  $F \times F$  into  $F$ . In addition  $\delta(c, f)$  is required to be undefined whenever  $c \in C$ .*

The idea is that  $\delta(a, f)$  represents the value of attribute  $f$  on the element  $a$ . Notice here that  $F$  plays the role of  $L$  above, and  $C$ , the set of *constants*, are something like the set  $A$  above, where  $A$  can only label terminal nodes of a structure. A big difference in the formulation is that features can themselves be values of  $\delta$ , a practice found in LFG, though not exemplified here.

Although  $\delta$  can be a partial function, the view of AVS in Johnson is that of totally instantiated objects. Accordingly, the subsumption relation plays little role in his structures. Partiality is only used to rule out inappropriate features on certain objects (among those, no feature is defined on a constant.) The interpretation of constants as constant types is also very restrictive: it is as if the sets  $D_c$  in feature systems were restricted to be singletons.

In Johnson’s view, then, the usual attribute-value matrices are only a pictorial representation of attribute-value structures. Coindexed points in an AVM are a representation of actual path identity. We illustrate with an example. Consider the AVM

$$\left[ \begin{array}{ll} \text{SUBJ} & \boxed{1} \text{ [PRED 'Tara' ]} \\ \text{OBJ} & \boxed{2} \text{ [PRED 'Louise' ]} \\ \text{TENSE} & \text{PRES} \\ \text{PRED} & \text{'paint' } \\ \text{ACTOR} & \boxed{1} \\ \text{RECIPIENT} & \boxed{2} \end{array} \right]$$

This is a pictorial representation of the following AVS:

- $F = \{ a, b, c, \text{paint}, \text{Tara}, \text{Louise}, \text{SUBJ}, \text{OBJ}, \text{PRED}, \text{TENSE}, \text{ACTOR}, \text{RECIPIENT}, \text{PRES} \}$ ;
- $C = \{ \text{paint}, \text{Tara}, \text{Louise}, \text{PRES} \}$ ;

- $\delta(a, \text{SUBJ}) = b$ ;
- $\delta(a, \text{OBJ}) = c$ ;
- $\delta(a, \text{TENSE}) = \text{PRES}$  ;
- $\delta(a, \text{PRED}) = \text{paint}$  ;
- $\delta(a, \text{ACTOR}) = b$ ;
- $\delta(a, \text{RECIPIENT}) = c$ ;
- $\delta(b, \text{PRED}) = \text{Tara}$  ;  $\delta(b, \text{PRED}) = \text{Louise}$ .

Johnson introduces a formal language which will be interpreted using attribute-value structures. The interpretation shows how the language can be considered a “constraint language” for feature systems, and it reflects an orientation towards first-order models, in that it uses a classical treatment of negation.

Fix  $C$ , a set of *constant symbols*, and  $V$ , an (infinite) set of variables. Then the following grammar defines  $L(C, V)$ . It has two syntactic categories: terms and formulas.

- Variables and constant symbols are terms;
- If  $t_1$  and  $t_2$  are terms, then  $t_1(t_2)$  is a term;
- If  $t_1$  and  $t_2$  are terms, then  $t_1 \approx t_2$  is a formula;
- The special constants **true** and **false** are formulas;
- The formulas are recursively closed under the standard Boolean connectives.

The semantics of  $\mathcal{L}$  is conditioned on having a way of interpreting constant symbols in an AVS  $(F, C, \delta)$ . This is an *injective* mapping

$$\xi : C \rightarrow F$$

The intent here is that distinct constants denote distinct constant elements of a structure, a typing notion common in LFG. Having said this, then the interpretation of terms and formulas is standard, modulo conventions on partiality. So, an interpretation is specified by giving  $\xi$  and an *environment* map  $\rho : V \rightarrow F$ . We first define the denotation  $\llbracket t \rrbracket$  of a term inductively.

- $\llbracket c \rrbracket(\rho) = \xi(c)$  for constant symbol  $c$ ;
- $\llbracket x \rrbracket(\rho) = \rho(x)$  for variable  $x \in V$ ;
- $\llbracket t_1(t_2) \rrbracket = \delta(\llbracket t_1 \rrbracket, \llbracket t_2 \rrbracket)$ , provided that the  $\llbracket t_i \rrbracket$  are defined and that  $\delta(\llbracket t_1 \rrbracket, \llbracket t_2 \rrbracket)$  is also defined; otherwise this quantity is undefined.

This semantics, somewhat reminiscent of the  $\lambda$ -calculus, is occasioned by the fact that attributes can themselves be values. Most of the time, terms of a form such as  $x(f)(g)(h)$  are considered, corresponding to applying the sequence of functions  $fgh$  to the variable quantity  $x$ . Then the semantics of the *satisfaction* relation is given, again relative to an interpretation  $I = (\xi, \rho)$ :

- $I \models \text{true}$  always and **false** never;
- $I \models t_1 = t_2$  if  $\llbracket t_1 \rrbracket = \llbracket t_2 \rrbracket$  and both values are defined;
- The satisfaction of Boolean combinations, including negation, is classical.

From the second and third items, we get that for a constant  $c$  and variable  $x$ , that  $I \models c(x) \neq c(x)$ , a somewhat perplexing result, but one which represents the desire to restrict the equality relation to asserting “defined equality”, and not to allow an ‘undefined element’ – as part of a model. This in turn reflects the desire to have linguistic objects (elements of a model) be *total descriptions*: the value of a feature should not be “undefined” if that feature is appropriate for a linguistic description. This requirement again has its source in LFG.

Johnson presents an axiomatic system for this logic, which he proves sound and complete in the usual sense of these terms. He also discusses the relationship of his logic to logics in the Kasper-Rounds style. In the course of this, he presents a translation of attribute-value logic to first-order logic of a restricted kind, and shows how techniques of first-order logic could be used to obtain results on compactness and complexity. Finally, he shows how to represent grammatical relations and constructions in the LFG style using his logic. This work thus represents a tying together of a full logical system with parts of a linguistic theory of syntax, one of the first such integrations, excepting perhaps the work of Montague [46], and Johnson and Postal [34].

### 3.10. Feature logics as constraint systems

The view of feature systems and feature logics as subsets of first-order logic is continued by Smolka [70], and Nebel and Smolka [53]. Once again we explain the class of intended models, then move to the syntax.

Smolka considers the concept of *feature algebra*. This can be seen as a specialization of the concept of feature system that we have been considering. To give the precise definition: a feature algebra is a feature system  $\mathbf{A}$  over  $L$  and  $A$  such that (i)  $D_a$  is a singleton for each  $a \in A$ ; (ii) if  $a \neq b$  then  $D_a \neq D_b$ ; (iii) if  $D_a = d$  then no feature is defined on  $d$ . Thus the elements of  $A$  denote primitive, unique atoms, and no feature can be defined on their interpretations.

Although a feature algebra can serve as a semantic model for a modal system, in the work of Smolka and Johnson it is considered more as a first-order structure. A feature is thought of as a relation which happens to be functional in nature. So, in the definition of Smolka’s feature logic, the interpretation of negation is classical. Smolka also considers the addition of quantifiers.

The syntax of constraints requires again the introduction of variables in some set  $V$ . Recall also that  $p$  and  $q$  denote paths: strings of feature symbols. A feature constraint is then one of the following:

- $xp \doteq yq$  for variables  $x, y$  (path equation);
- $xp \doteq a$  for atom  $a$  (path equation and constant);
- $xp \uparrow$  (divergence);
- Boolean combinations of constraints;
- $\exists x(\varphi)$  where  $\varphi$  is a constraint.

By this time it is not necessary to spell out the semantics of such a system. It is

done with respect to a feature algebra, using assignments from  $V$  to the elements of the carrier of the algebra. Quantification is expressed as usual in the standard Tarski semantics of first-order logic. Negation is treated classically.

The focus of Smolka's work is to give efficient algorithms for solving constraint systems, and not so much on the linguistic applications. He notes that it is possible to obtain a sound and complete axiom system for feature logic simply by regarding feature logic as a subsystem of first-order logic with equality, and by axiomatizing feature algebras as follows:

- For each set of distinct atoms  $\{a, b\}$ , add the constraint  $\neg(a \doteq b)$ .
- For each feature  $l$ , add the constraint

$$(xl \doteq y) \wedge (xl \doteq z) \rightarrow (y \doteq z).$$

- For each atom  $a$  and feature  $l$  add

$$\neg \exists x (al \doteq x).$$

A further section of Smolka's paper [70] incorporates Kasper-Rounds formulae into constraints. Suppose one takes the basic Kasper-Rounds logic over the set  $A$  and the feature set  $L$  where now  $A$  may be interpreted as a general set of sorts, and for simplicity we do not allow negation or path equations. Then with respect to a feature system  $\mathbf{A}$ , each KR formula denotes a subset of  $D^{\mathbf{A}}$ . So instead of thinking of a KR formula as a description of a linguistic object, we can think of it as defining an extended notion of type (i.e., an extension of the type system given by interpreting the sort symbols as subsets of  $D$ .) Let  $S$  and  $T$  denote KR formulas (Smolka's word is *feature term*). Then we may add to our basic syntax of constraints the constraint  $x : S$  where  $S$  is a feature term and  $x$  is a variable. Such a constraint will be satisfied in  $\mathbf{A}$  by an assignment  $\rho$  to the variable  $x$  iff  $\rho(x)$  is in the set  $\llbracket S \rrbracket$ , the set of elements "satisfying" the feature term  $S$ .

With sorts in place, Smolka considers the possibility of admitting recursive "definitions" of sorts; We compare his treatment with that of Carpenter, Moshier, and Gazdar *et al.* A *sort equation* is an expression

$$a \doteq S$$

where  $a$  is a primitive sort symbol, and  $S$  is a feature term. Of particular interest are *recursive* equations:

$$person \doteq father : person.$$

A sort equation may be considered as syntactic sugar for the constraint

$$\forall x (x : a \leftrightarrow x : S)$$

so that the notions of satisfiability and so forth carry over to constraint systems with sort equations. So, a recursive sort equation, like the one involving fatherhood

above, may or not be satisfied in a particular feature system. In fact, the given sort equation is satisfied by any feature system interpreting *person* as the empty set, as well as by any system interpreting *person* as a one-element set  $\{d\}$  with  $d$ 's father being  $d$  himself. On the other hand (admitting negation for the moment) the sort equation

$$a \stackrel{\cdot}{=} \neg a$$

is not satisfiable.

Using the theory of definite relations, as given in [30], Smolka shows that any definite system of recursive sort equations will have a “least model”. These notions need to be defined: “definite” refers to the feature term  $S$  on the right side of a sort equation. Each sort symbol in  $S$  must appear within the scope of an even number of negation signs. (We have ruled out negations for now, so that all of our feature terms are definite.) Then, the term “least model” refers to a partial ordering  $\preceq$  on feature systems. We say  $\mathbf{A} \preceq \mathbf{B}$  if  $A$  and  $B$  have the same domain  $D$  and the same interpretation of features, but that  $D_a^{\mathbf{A}} \subseteq D_a^{\mathbf{B}}$  for every sort symbol  $a$ . As expected, the proof of the least-model theorem is an application of the same fixed point results as in Moshier’s work. But Smolka’s notion of sort equation allows cyclic solutions, whereas Moshier’s notion rules them out by insisting on the least model as the meaning of the recursive definition of a predicate. These cyclic solutions are also allowed by Carpenter (who works in the algebra of feature structures) and by Gazdar.

We notice in passing that systems of sort definitions are crucial in the theory of *terminological logics* [53]. Concept definitions (the Tbox component of KL-ONE-like knowledge representation systems [12]) can be regarded as systems of sort equations. The idea of KR formulas as terms really occurred before the advent of KR formulae, in the work of Ait-Kaci [3]. Ait-Kaci’s  $\psi$ -terms correspond to KR formulae without disjunction. In fact, the discovery of KR logic can be seen as an attempt to work out the properties of equations between Ait-Kaci’s  $\epsilon$ -terms, which are  $\psi$ -terms with the addition of disjunction. Both of the systems of Ait-Kaci were intended to model the type system in KL-ONE.

#### 4. Logical properties: soundness and completeness

The logics we have been discussing up to now, whether modal, intuitionistic, or first-order, all are subject to the usual logicians’ questions about axiomatizability, soundness, completeness, and compactness. We will provide a representative sampling of results along this line, dividing between the three kinds of logical systems just mentioned. We will also present some work which unifies the various kinds of completeness results obtained for differing varieties of feature logics.

#### 4.1. Axiomatizations for Kasper-Rounds-style logics

Kasper and Rounds presented an axiomatization of their original logic, which, we recall, involved acyclic feature structures with no sorts except for atoms at the leaf nodes of these structures, and did not involve negation. We are not going to give this axiomatization here in detail, but will present instead an axiomatic system given by Moss for KR logic augmented with negation. Moss' technique and axiom system are both much more elegant than the original formulation. The technique is an adaptation of the standard Henkin model-theoretic construction common in first-order logic. It is also very common in modal logic, where it usually is called a *canonical model* construction. The propositional nature of Kasper-Rounds makes the construction considerably simpler than in the full first-order case.

Before we begin, we should note that validity of a formula is not what is usually required in applications. Constraint-solving problems arise much more naturally in natural-language systems; such problems usually refer to satisfiability. However, one may wish to use an equational proof system, such as the one we are about to discuss, to replace a constraint formula by a logically equivalent one. It is handy to know when this is in general a correct step, especially in the initial design of constraints.

In the following, we make no assumptions about the nature of  $A$ ; that is,  $D_a$  can be an arbitrary subset of  $D^{\mathbf{A}}$  for an interpretation  $\mathbf{A}$ . So there will be no laws of the logic involving the sorts. We consider a Kasper-Rounds language with negation and path equations, and we seek a complete axiom system. It turns out that our axioms can be taken to be equational – that is, in the form of equivalences between Kasper-Rounds formulae. The following is Moss' axiom system. By  $\varphi \supset \psi$  we mean the equation  $\varphi \wedge \psi = \varphi$ .

- (i) The standard Boolean algebra equations;
- (ii)  $(l : \varphi) \wedge (l : \psi) = l : (\varphi \wedge \psi)$ ;
- (iii)  $l : true = (l : \varphi) \vee (l : \neg\varphi)$ ;
- (iv)  $false = l : false$ ;
- (v)  $true = (\lambda \doteq \lambda)$  ( $\lambda$  is the null string);
- (vi)  $(p \doteq q) \supset (q \doteq p)$ ;
- (vii)  $(p \doteq q) \wedge (q \doteq r) \supset (p \doteq r)$ ;
- (viii)  $(p \doteq q) \wedge (p : \varphi) \supset q : \varphi$ ;
- (ix)  $p : (q \doteq r) = (pq \doteq pr)$ ;
- (x)  $(p \doteq q) \wedge pr : true \supset (pr \doteq qr)$ ;
- (xi)  $(p \doteq q) \supset p : true$ .

Here  $p, q$  are strings over  $L$  as usual.

The rules of proof for these axioms are the standard equational rules: the reflexive, symmetric, and transitive laws of  $=$ , together with the laws that make  $=$  into a congruence relation. Our objective is now to show that every equation between KR formulae that is valid in all feature systems is provable in the above axiom system. (The converse, soundness, is straightforward by induction on equational proof length.)

Moss' core construction involves the ideas of *consistency* and *maximal consis-*

*tency*. Say that a set  $S$  of KR formulae is *inconsistent* if there is a finite subset  $F$  of  $S$  such that the equation  $\bigwedge F \supset \text{false}$  is provable. Otherwise, say that  $S$  is *consistent*. Then  $S$  is *maximal consistent* if every proper superset of  $S$  is inconsistent. Finally, say that  $T$  is closed under deduction iff whenever  $\varphi \in T$  and  $\varphi = \psi$  is provable, then  $\psi \in T$ .

These definitions lead immediately to a completeness proof. The following is a combination of ideas from Moss' paper and from Moshier [48].

Let  $T$  be a set of KR formulae. We construct a feature system  $\mathbf{A}(T)$  based on  $T$ . The elements of  $D^{\mathbf{A}(T)}$  will be equivalence classes in a certain subset  $P_T$  of  $L^*$ . We define

$$P_T = \{p \mid p : \text{true} \in T\}.$$

Set  $p \equiv_T q$  iff  $(p \doteq q) \in T$ . Then we have

**Lemma 4.1.** *Suppose  $T$  is nonempty, consistent and closed under deduction. Then*

- (i)  $P_T$  is a nonempty, prefix-closed set of strings;
- (ii)  $\equiv_T$  is an equivalence relation on  $P_T$ ;
- (iii) If  $p \equiv_T q$  and  $pl \in P_T$  then  $pl \equiv_T ql$ ;
- (iv) If  $p \equiv_T q$  and  $p : a \in T$  then  $q : a \in T$ .

The proof appeals mostly to the laws involving path equations. For example,  $\varphi \supset t$  is always provable, so **true** is in any nonempty closed  $T$ . So, to see that  $P_T$  is prefix-closed, suppose  $(pr : \text{true}) \in T$ . Then  $(pr \doteq pr) \in T$ , so  $p : (r \doteq r) \in T$ . Since  $(r \doteq r) \supset \text{true}$  is provable, we have  $p : \text{true} \in T$ . The other assertions follow similarly.

Now let  $T$  be nonempty, consistent, and closed. Define a feature system  $\mathbf{A}(T)$  as follows: Take  $D^{\mathbf{A}(T)}$  to be the collection of equivalence classes of  $P_T$ . If  $[p]$  is such a class, let  $[p]l$  be  $[pl]$  if  $pl \in P_T$ , and otherwise undefined. Let  $D_a = \{[p] \mid p : a \in T\}$ . Then lemma 4.1 above implies that  $\mathbf{A}(T)$  is well-defined.

Our objective is now the following result. When we say  $(\mathbf{A}, d) \models \varphi$  that we mean that the object  $d$  satisfies  $\varphi$  in the structure  $\mathbf{A}$ .

**Lemma 4.2. (Truth lemma.)** *Suppose  $\varphi$  is a KR formula. Then for any maximal consistent set  $T$ , we have  $(\mathbf{A}(T), [\lambda]) \models \varphi$  if and only if  $\varphi \in T$ .*

The proof of this lemma is by induction on formulas  $\varphi$ . The base cases are direct from the definition of  $\mathbf{A}(T)$ , and the Boolean combinations follow from the Boolean laws. For the prefixing with labels case we need the next definitions and lemmas.

**Definition 4.3.** *Let  $T$  be a set of formulas, and  $l \in L$ . Define*

$$l^{-1}(T) = \{\varphi \mid l : \varphi \in T\}.$$

*Also, if  $\mathbf{A}$  is a feature system,  $d \in D^{\mathbf{A}}$ , and  $l \in L$ , then  $\mathbf{A}(d, l)$  is the principal subsystem  $P(dl)$  of  $\mathbf{A}$  generated by  $dl$ .*

**Lemma 4.4.** *If  $T$  is (maximal) consistent, and  $l \in P_T$  then  $l^{-1}(T)$  is also (maximal) consistent.*

For the proof, use the laws (2-4) involving prefixing.

**Lemma 4.5.** *Let  $T$  be a closed theory, and  $l \in P_T$ . Then  $\mathbf{A}(T)([\lambda], l)$  is isomorphic to  $\mathbf{A}(l^{-1}(T))$ .*

**Proof.** Let  $[p]$  be an equivalence class of  $\mathbf{A}(l^{-1}(T))$ . Let  $h([p]) = [lp]_T$ . It is easy to check that  $h$  is a well-defined bijection preserving structure.  $\square$

Now we can finish the proof of Lemma 4.2. We only check the case of  $l : \phi$ . We have

$$\begin{aligned} (\mathbf{A}(T), [\lambda]_T) \models l : \varphi &\iff (\mathbf{A}(T), [l]) \models \varphi \\ &\iff (\mathbf{A}(T)([\lambda]_T, l), [\lambda]_T) \models \varphi \\ &\iff \mathbf{A}(l^{-1}(T), [\lambda]_{l^{-1}(T)}) \models \varphi \\ &\iff \varphi \in l^{-1}(T) \text{ (by induction)} \\ &\iff l : \varphi \in T. \end{aligned}$$

The completeness of Moss' system now follows as a corollary of the truth lemma.

**Theorem 4.6.** *Let  $S$  be a set of KR formulas. If for all  $(\mathbf{A}, d)$  such that  $d \models \sigma$  for all  $\sigma \in S$ , we have  $(\mathbf{A}, d) \models \varphi$ , then  $S \vdash \varphi$ .*

**Proof.** . Suppose not. Then  $S \cup \{\neg\varphi\}$  is consistent. By Zorn's lemma, choose a maximal consistent  $T$  containing  $S \cup \{\neg\varphi\}$ . By the truth lemma,  $(\mathbf{A}(T), [\lambda]) \models \sigma$  for all  $\sigma \in S$ , but  $(\mathbf{A}(T), [\lambda])$  does not satisfy  $\varphi$ , in contradiction to the hypothesis of the theorem.  $\square$

#### 4.2. Axiomatizing intuitionistic feature logic

Moss also uses similar model-theoretic apparatus to axiomatize his intuitionistic feature logic. Instead of presenting his axiom system, though, we move to a more general proof-theoretic setting, following the work of Moshier [48]. This involves the notion of a *sequent* and the notion of a Gentzen-style proof. (See [71] for a general introduction.)

The basic construct of a Gentzen-style proof system is the sequent. This we define to be a pair of finite sets of KR formulae  $(?, \Delta)$ , also written  $? \vdash \Delta$ . The intuitive meaning of such a sequent is that the conjunction of the formulas in  $?$  implies the disjunction of the formulas in  $\Delta$ , in all feature systems. (We use sets instead of

sequences of formulas to shorten assertions of commutativity and idempotence.) Then we have proof rules of the following forms.

– Unary schemata: of the form

$$\frac{?_0 \vdash \Delta_0}{?_1 \vdash \Delta_1}$$

– Binary schemata: of the form

$$\frac{?_0 \vdash \Delta_0 \quad ?_1 \vdash \Delta_1}{?_2 \vdash \Delta_2}$$

These are the general forms of any rule; a proof system will be a collection of such schemata together with a collection *Init* of initial sequents. A proof of a sequent  $? \vdash \Delta$  will then be a finite tree whose root is labeled with  $? \vdash \Delta$  and whose leaves are labeled with initial sequents. Typically the root of such a tree is at the bottom of the page.

Next we list rule schemata for the actual proof system  $F$  of feature logic. The first two are *structural schemata* which generally are included in any sequent-style proof system. Here LW and RW stand for the left and right *weakening* rules.

$$\begin{array}{ll} \text{LW:} & \frac{? \vdash \Delta}{?, \varphi \vdash \Delta} & \text{RW:} & \frac{? \vdash \Delta}{? \vdash \Delta, \varphi} \\ \text{Cut:} & \frac{? \vdash \Delta, \varphi \quad \varphi, ?' \vdash \Delta'}{?, ?' \vdash \Delta, \Delta'} \end{array}$$

By  $?, \phi$  we really mean  $? \cup \{\phi\}$ , and by  $?, \gamma'$  we mean  $? \cup ?'$ . Also we omit braces around singleton sets. If we have blank space on the left or right of the sequent sign  $\vdash$ , then this means the empty set of formulae.

The second set of rules is for the propositional connectives. For intuitionistic calculus we include conjunction, disjunction, and implication.

– Conjunction:

$$\begin{array}{ll} \text{Left:} & \frac{\varphi, ? \vdash \Delta}{\varphi \wedge \psi, ? \vdash \Delta} & \frac{\psi, ? \vdash \Delta}{\varphi \wedge \psi, ? \vdash \Delta} \\ \text{and} & & \\ \text{Right:} & \frac{? \vdash \Delta, \varphi \quad ? \vdash \Delta, \psi}{? \vdash \Delta, \varphi \wedge \psi} \end{array}$$

– Disjunction:

$$\begin{array}{ll} \text{Right} & \frac{? \vdash \Delta, \varphi}{? \vdash \Delta, \varphi \vee \psi} & \frac{? \vdash \Delta, \psi}{? \vdash \Delta, \varphi \vee \psi} \\ \text{and} & & \\ \text{Left} & \frac{?, \varphi \vdash \Delta \quad ?, \psi \vdash \Delta}{?, \varphi \vee \psi \vdash \Delta} \end{array}$$

– Implication:

$$\frac{\varphi, ? \vdash \psi}{? \vdash \varphi \supset \psi}$$

and

$$\frac{? \vdash \Delta, \varphi \quad \psi, \Pi \vdash \Lambda}{\varphi \supset \psi, ?, \Pi \vdash \Delta, \Lambda}.$$

We also include rules for dealing with the prefixing operation. If  $p$  is a path, then by  $p : ?$  we mean the set  $\{p : \phi \mid \phi \in ?\}$ . Then the rule for prefixing reads

$$\frac{? \vdash \Delta}{p \doteq q, p : ? \vdash q : \Delta}.$$

Besides the rules of inference for sequents, we have rules in *Init*. These include the *logical sequents* ( $\vdash t$ ), ( $f \vdash$ ), and ( $\varphi \vdash \varphi$ ) for all atomic formulas  $\varphi$ . The also include the sequents particular to feature logic:

- (i) ( $\vdash \lambda \doteq \lambda$ ), where  $\lambda$  is the null path;
- (ii) ( $p \doteq q, q \doteq r \vdash p \doteq r$ );
- (iii) ( $p \doteq q \vdash q \doteq p$ );
- (iv) ( $p : (q \doteq r) \vdash pq \doteq pr$ );
- (v) ( $pq \doteq pr \vdash p : (q \doteq r)$ );
- (vi) ( $p \doteq q, pr : t \vdash pr \doteq qr$ ).

What do *soundness* and *completeness* mean in the setting of sequent calculi?

We need a notion of “valid sequent.” If we were considering classical feature logic, then we could say that a sequent  $? \vdash \Delta$  is valid if for all feature systems  $\mathbf{A}$  (both finite and infinite) and  $d \in D^{\mathbf{A}}$ , we have that whenever  $d \models \varphi$  for all  $\varphi \in ?$ , then for some  $\psi \in \Delta$ ,  $d \models \psi$ . Since we work intuitionistically, we use instead Moss’ notion of forcing from Section ?.

Recall that a relation  $\preceq$  on the nodes of a feature system  $\mathbf{A}$  is an *approximation* if

$$d \preceq e \text{ and } dl \text{ is defined } \Rightarrow dl \preceq el$$

and

$$d \preceq e \text{ and } d \in D_a \Rightarrow e \in D_a.$$

To deal with preservation of path equalities, we also require

$$d \preceq e \text{ and } dp = dq \Rightarrow ep = eq.$$

A Kripke structure is a pair  $(\mathbf{A}, \preceq)$  where  $\mathbf{A}$  is a feature system and  $\preceq$  is an approximation relation.

Instead of the satisfaction relation  $\models$  used above, we consider a forcing relation  $\Vdash$  to interpret formulas. The clauses are as follows, where  $(\mathbf{A}, \preceq)$  is fixed and

$d \in D^{\mathbf{A}}$ :

- $d \Vdash a$  if  $d \in D_a$ ;
- $d \Vdash \text{true}$  always;
- $d \Vdash \text{false}$  never;
- $d \Vdash p \doteq q$  if  $dp = dq$ ;
- $d \Vdash \varphi \wedge \psi$  if  $d \Vdash \varphi$  and  $d \Vdash \psi$ ;
- $d \Vdash \varphi \vee \psi$  if  $d \Vdash \varphi$  or  $d \Vdash \psi$ ;
- $d \Vdash l : \phi$  if  $dl \Vdash \varphi$  (implicitly  $dl$  is defined);
- $d \Vdash \varphi \rightarrow \psi$  if for all  $e$  such that  $d \preceq e$ , if  $e \Vdash \varphi$ , then  $e \Vdash \psi$ .

We can now define *validity* of a sequent: a sequent  $? \vdash \Delta$  is valid if for all Kripke structures  $(\mathbf{A}, \preceq)$  and  $d \in D^{\mathbf{A}}$ , we have that whenever  $d \Vdash \varphi$  for all  $\varphi \in ?$ , then for some  $\psi \in \Delta$ ,  $d \Vdash \psi$ .

It is straightforward to show that the sequent proof system given above for feature systems is sound: Every provable sequent is valid. This follows by induction on the size of proof trees. The interesting question is the converse one. Moshier actually considers a strengthening of the converse. For this result, we need some preliminary notions.

The key to the result is the following definition.

**Definition 4.7. (Saturation)** Let  $T$  be a set of intuitionistic KR formulas. We say that  $T$  is *saturated* if for any sequent  $? \vdash \Delta$ , such that  $? \subseteq T$  and  $? \vdash \Delta$  is provable, we have that  $\Delta \cap T \neq \emptyset$ .

The intuitive idea of this definition is that a saturated theory is rich enough to have witnesses for valid disjunctions: if a disjunctive formula is in a saturated closed theory, then one of the disjuncts is already in the theory. In fact, a *prime* theory in intuitionistic propositional calculus is a theory  $T$  such that whenever  $(\varphi \vee \psi) \in T$ , then one of  $\varphi$  or  $\psi$  must be in  $T$  (additionally  $T$  is closed under deduction and does not contain **false**.) Notice also that in the classical setting, saturation corresponds to maximal consistency.

We need to relate saturated theories to models. In the setting of intuitionistic KR, we will take a model to be a tuple  $((\mathbf{A}, \preceq), d)$ , where  $\mathbf{A}$  is a feature system,  $\preceq$  is an approximation relation on  $D^{\mathbf{A}}$ , and  $d \in D^{\mathbf{A}}$ . Then we have the following results.

**Lemma 4.8. (Saturation lemma.)** Suppose that  $(S, T)$  is a pair of sets of formulae, such that for every sequent  $(? \vdash \Delta)$  with  $? \subseteq S$  and  $\Delta \subseteq T$ , we have that  $? \vdash \Delta$  is not provable. Then there is a saturated set  $S' \supseteq S$  such that  $S' \cap T = \emptyset$ .

We omit the proof. It involves some observations on provability using the cut rule, and an application of Zorn's lemma.

**Lemma 4.9. (Completeness criterion)** Suppose that for every saturated  $T$  there is a model  $M = ((\mathbf{A}, \sqsubseteq), d)$  such that  $T = \{\varphi \mid ((\mathbf{A}, \preceq), d) \models \varphi\}$ . Then every valid sequent is provable.

The lemma is immediate, because if there is a valid but unprovable  $? \vdash \Delta$ , we can apply the saturation lemma. We get a saturated  $T$  containing  $?$  and excluding  $\Delta$ . By the hypothesis, there is an  $M$  such that  $T = \{\varphi \mid M \models \varphi\}$ . But such a model violates the validity of  $? \vdash \Delta$ .

**Lemma 4.10.** (*Truth lemma for Kripke structures*) *There is a “universal” Kripke structure  $(\mathbf{U}, \preceq)$  such that for each saturated  $T$ , there is a  $d \in D^{\mathbf{U}}$  such that*

$$T = \{\varphi \mid ((\mathbf{U}, \preceq), d) \models \varphi\}.$$

We only sketch the proof of the lemma. We use the same equivalence relations  $\equiv_T$  on strings as in the classical case. Then we can define a universal feature system  $\mathbf{U}$  and approximation relation  $\preceq$  as follows.

- (i)  $D^{\mathbf{U}}$  is the collection of pairs  $(T, [p]_T)$ , where  $T$  is a saturated theory and  $[p]_T$  is an equivalence class of  $T$ ;
- (ii)  $(T, [p]_T)l = [pl]_T$  provided  $pl : true \in T$ ;
- (iii)  $D_a = \{(T, [p]_T) \mid p : a \in T\}$ ;
- (iv)  $(T, [p]_T) \preceq (T', [p']_{T'})$  iff for some  $u \in L^*$ ,  $up' = p$  and  $u^{-1}(T) \subseteq T'$ , where as before

$$u^{-1}(T) = \{\varphi \mid u : \varphi \in T\}.$$

Then the basic claim is: for all formulas  $\varphi$ , all  $p$  and all saturated  $T$ ,

$$((\mathbf{U}, \preceq), [p]_T) \models \varphi \iff p : \varphi \in T,$$

from which the lemma follows, taking  $p = \lambda$ . The claim is proved again by induction on  $\varphi$ . This time, instead of appealing to the equational proof system, we appeal to the sequent system; in particular *Init* and certain sequents easily provable from *Init*. The inductive cases follow from the proof rules for sequents involving the various connectives. Saturatedness is used to handle the case of implication. We give the proof for one direction of this case.

Suppose that  $p : (\varphi \supset \psi)$  is not in  $T$ , where  $p \in P_T$ . Then every sequent  $(? \vdash \Delta)$  such that  $? \subseteq p^{-1}(T)$  and  $\Delta = \{\psi\}$  is not provable, because  $p : true$  is in  $T$ . By the Saturation Lemma, there is a saturated  $T' \supseteq p^{-1}(T)$  so that  $\varphi \in T'$  but  $\psi \notin T'$ . So  $((\mathbf{U}, \preceq), (T, [p]_T))$  does not force  $\varphi \supset \psi$ .

**Remark M.** oshier’s techniques apply in a variety of other completeness proofs. He shows, for example, how to handle various assumptions about appropriateness conditions. He also deals with issues involving compactness and completeness, in a general setting for sequent proof systems which do not even involve feature logic. So I have included some of his work here, to point out a general logical technique which in the case of feature logic yields esthetically pleasing completeness results.

As a historical note, the axiomatization of Kasper and Rounds is an equational one, and in fact was the starting point for the discovery of feature logic in the first

place. In work leading to the axiom system, the symbol  $\wedge$  replaced the symbol  $\oplus$ , standing for “unification”. That is, Kasper and Rounds were thinking of expressions involving disjunctive feature types, and simply wrote equations expressing desired validities between those expressions. The unification  $\varphi \oplus \psi$  intuitively stood for the least type subsumed by both types  $\varphi$  and  $\psi$ . Only after these equations were written out did it become clear that they could be read as validities of feature logic, and that  $\oplus$  was really  $\wedge$ .

### 4.3. Axiomatizing other forms of feature logic

After the excursion into sequent structures, we should also point out that standard methods of logic can be made to yield completeness results for feature logic. The idea is simply to choose a completeness technique from modal or first-order logic, and then to employ a translation method from feature logic into one of these systems, apply the completeness result, and infer a corresponding result for feature logic. Of course the term “feature logic” is ambiguous here, intentionally so. As an example, recall Smolka’s constraint feature logic [70].

A feature constraint is one of the following:

- $xp \doteq yq$  for variables  $x, y$  (path equation);
- $xp \doteq a$  for atom  $a$  (path equation and constant);
- $xp \uparrow$  (divergence);
- Boolean combinations of constraints;
- $\exists x(\varphi)$  where  $\varphi$  is a constraint.

The translation to first-order logic (with equality) is fairly simple. First axiomatize feature algebras as follows:

- For each set of distinct atoms  $\{a, b\}$ , add the constraint  $\neg(a \doteq b)$ .
- For each feature  $l$ , add the constraint

$$(xl \doteq y) \wedge (xl \doteq z) \rightarrow (y \doteq z).$$

- For each atom  $a$  and feature  $l$  add

$$\neg \exists x(al \doteq x).$$

Let  $F$  be the smallest set of constraints subject to these rules. Then the class of feature algebras, regarded as first-order structures, is exactly the class of structures satisfying every constraint of  $F$ .

We thus need to translate arbitrary feature constraints into formulas of first-order logic. Basic constraints (no connectives or quantifiers) are handled, for example, by replacing  $xp \doteq yq$  by a conjunction of atomic constraints of the form  $wl \doteq v$ , where  $w$  and  $v$  (and so forth) are fresh variables, and then existentially quantifying over these new variables. So, for example,  $xfg \doteq ygh$  would transform into

$$(\exists uvw)(xf \doteq u \wedge ug \doteq v \wedge yh \doteq v).$$

Then each atomic constraint is a syntactic variant of the first-order formula  $l(w, v)$  where  $l$  is a binary relation symbol. This relation is forced to be a partial function by the axiomatization  $F$  of feature algebras. Recall that atoms  $a$  are interpreted as singletons, so that they become constants in the translation to first-order logic. The translation of nonatomic constraints is then routine.

We get a sound and complete axiom system for constraints by choosing the axioms of  $F$  for nonlogical axioms, and adopting the usual first-order proof rules. The difficulty with this approach, of course, is that to work with arbitrary proof rules (such as generalization), proofs may need to go outside of the system of (translated) constraints. Thus such a deduction calculus need not be entirely self-contained. But it may be useful for deriving some of the other standard metamathematical properties of constraint theories such as compactness, decidability, or complexity.

Johnson's book [35] uses similar techniques for his attribute-value system. We give an example of his techniques below, in the section on complexity issues. Johnson addresses the issues of *compactness* as well in his book. He shows from the translation into first-order logic that this attribute-value logic is compact (if every finite subset of a set of formulae is satisfiable, then so is the whole set.) Compactness is a desirable property of a logic in that (informally) if a logic is not compact, then it does not admit of a finite axiom system; in general, infinitary proof rules tend to be required. Feature logics tend to be very "low-level" logics, so they do not often run into this problem. However, Moshier's recursive  $L(EKR1)$  is not in fact compact.

Finally, we mention modal techniques. Reape's polyadic modal logic, for example, admits of complete axiomatizations relative to axiomatizing the arbitrary mathematical structures to which elements of a feature system belong. One thus gets a heterogeneous axiom system involving features and other classes of models (sets are just one example.) As we have mentioned above, Kracht [43] relates feature logics to other systems of modal logic, and adduces complete axiom systems as a result.

## 5. Complexity and decidability issues

The natural questions to answer here have to do with the satisfiability and validity problems for various feature logics. First let us ask where such questions might arise.

If one works in the constraint-based parsing mode, then one is led to ask whether or not a given string has a parse. This can be interpreted in several ways. The most obvious way to answer the question in many unification-based formalisms is to use the context-free base grammar to guide the search, in that a successful parse of a string will require the string to be the yield of some skeleton tree from the base grammar. Efficient methods, in particular chart-parsing methods, can be employed for this purpose. If the string has no underlying tree, then it has no parse. If it does have a tree, then one can (as in LFG) build a system of constraints among the feature paths. Here we find a special form of the satisfiability problem, for conjunctions of path equations in feature logic. These conjunctions can be solved efficiently

by what is known as the unification algorithm, or in Smolka's terms, an incremental constraint-solving method. These algorithms are quite efficient, requiring linear time, or close-to-linear time for their implementation.

In practice, of course, life is not that simple. There are just too many possible parse trees which could potentially yield a given string. In general there could be infinitely many, because of productions of the form  $A \rightarrow A$ . Even if such productions are eliminated, we still face an exponentially difficult brute force elimination procedure. For this reason, chart parsing algorithms are typically interleaved with unification algorithms to reduce exponential search whenever possible. But at least, for reasonably well-designed grammars, there in principle is an algorithm for recognizing grammatical correctness.

There are, however, other sources of complexity – for example, when one has disjunctive feature information specified by the grammar; in particular, in the lexicon. (Think of multiple meanings for various words.) In this case, a natural constraint-solving method would try to extend unification to the case of disjunctive feature structures. It therefore becomes relevant to understand what the semantics of such structures is; and we have seen that disjunctive structures can be equivalently represented as disjunctive formulas. So, finding whether or not two disjunctive structures can be unified reduces to the question of satisfiability of two conjoined formulas, each containing disjunctions.

The complexity of this problem was determined by Kasper and Rounds. For KR formulas without negation, or even path equations, the satisfiability problem is *NP*-complete. This means that there is a nondeterministic algorithm that can guess a satisfying solution to a formula, in polynomial time in the size of the formula. But further, if there is a corresponding deterministic algorithm, then the complexity class *P* is equal to the class *NP*, a result widely conjectured to be untrue.

What about the validity problem for such a logic? With classical negation, the problem is easily seen to be “*co-NP*-complete.” That is, validity is the complement of a problem in *NP*, and if the problem is actually polynomial, then the class *P* is equal to *co-NP*.

When we move to more complicated versions of feature logic, things get more interesting. Blackburn and Spaan [9] have studied what happens when nominals are allowed, when one considers master modalities, and universal modalities. Since these modal logics admit classical negation, complexity results for validity are also not hard to derive.

One of Blackburn and Spaan's more surprising results is that the complexity of satisfiability for KR logic augmented with nominals and the universal modality is exponential-time complete, while the corresponding problem when we replace nominals by path equations is  $\Pi_1^0$ -complete; that is, co-semidecidable and  $\Pi_1^0$ -hard: every such problem reduces to this one.

The proof of the latter result is not too difficult. Blackburn and Spaan use a *tiling problem* [28] known to be hard for  $\Pi_1^0$  [64]. Here I will prove undecidability of a slightly stronger result, using a technique of Smolka [70]. This involves the idea of *word problems for Thue systems*.

We need some preliminary definitions on such word problems. Given a finite

alphabet  $L$ , consider the class  $\mathcal{S}$  of semigroups finitely generated by  $L$ . An *equation* is a two-element set  $\{p, q\}$ , where  $p$  and  $q$  are nonempty strings over  $L$ . The *word problem* for semigroups is the following. Given a finite set  $E = \{e_1, \dots, e_n\}$  of equations, and a *test equation*  $e = \{p, q\}$ , is there a semigroup  $S \in \mathcal{S}$  satisfying all equations in  $E$  but not the test equation  $e$ ? Analogously, the word problem for finite semigroups asks the same question when  $\mathcal{S}$  is now the class of finite semigroups generated by  $L$ . Gurevich has shown that the word problem for finite semigroups is undecidable [26]; this result holds with  $|L| = 2$ .

**Theorem 5.1.** *Suppose  $|L| = 2$ . It is undecidable whether a formula  $\varphi$  in KR logic augmented by path equations and the universal modality is satisfiable in a finite feature system over  $L$ .*

**Proof.** We reduce the word problem for finite semigroups to our desired one. Let  $L = \{f, g\}$ . Suppose given  $E$  and test equation  $e = \{p, q\}$ . Form the KR formula

$$\varphi_{E,e} = \Box[f : true \wedge g : true \wedge \bigwedge_{\{p_i, q_i\} \in E} (p_i \doteq q_i)] \wedge \neg(p \doteq q).$$

(Note that the scope of  $\Box$  does not include the last subformula.) We claim that there is a finite feature system  $\mathbf{A}$  and  $d \in D^{\mathbf{A}}$  so that  $d \models \varphi_{E,e}$  if and only if there is a finite semigroup  $S$  generated by  $L$  satisfying the equations in  $E$  but not the equation  $e$ .

To see one direction of the claim, let  $\mathbf{A}$  and  $d$  be given satisfying  $\varphi_{E,e}$ . Then  $f$  and  $g$ , interpreted as functions on  $P(d)$ , are total. Since  $D$  is finite, the finite semigroup  $S$  generated by composing these functions (acting on  $P(d)$ ) is a finite semigroup. By the universal modality, and by the third subformula of  $\varphi$ , the semigroup  $S$  satisfies the equations  $E$ . By the last subformula,  $S$  does not satisfy  $e$ .

Conversely, let  $S$  be a finite semigroup satisfying the condition. Adjoin an identity element  $\lambda$  to  $S$  if there is not one there already. Then  $S$  becomes a feature system under the feature definitions

$$sl = s * l,$$

where the  $*$  on the right is multiplication in the semigroup. Choose the element  $d \in S$  to be the identity element. Then trivially  $d \models \varphi_{E,e}$ . This proves the theorem.  $\square$

We remark that the use of negation in this result can be eliminated if we assume two disjoint sort symbols  $a$  and  $b$ , and replace the last subformula by  $p : a \wedge q : b$ . Of course a strictly positive formula, with no assumptions on sorts, can always be satisfied in a one-element system. Also, with negation, no sort symbols are necessary.

The above proof is an adaptation of one by Dörre and Rounds [19], showing the unsolvability of *subsumption constraints*, a concept of Smolka which we will cover

later in the section. The technique also leads to a proof, by further reductions, to a proof of the unsolvability of the *semi-unification* problem for *rational terms*. As such, this proof is an application of the theory of feature systems to a problem in computer science, arising from the need to perform type inference.

The original result of Blackburn and Spaan shows undecidability of the problem for finite or infinite feature systems, not just finite ones. Membership of that problem in  $\Pi_1^0$  is shown by noting that the class of valid formulas can in fact be recursively enumerated. (An easy way to see this is to translate into first-order logic.) For just the case of finite structures, it is easy to see that the problem is in  $\Sigma_1^0$ , and it is not hard to check from Gurevich's result that the problem is complete for this class.

Now we turn to the case of nominals and the universal modality. Recall that nominals are special atomic sort symbols which can have only one node satisfying them. The force of the path equation  $p \doteq q$  can be captured weakly by  $p : i \wedge q : i$ , where  $i$  is a fresh nominal. However, one would really like to say (and Smolka allows one to say) that  $p \doteq q$  is expressed by  $(\exists i)(p : i \wedge q : i)$ . This is not allowed in  $L(KR)$  with nominals, and this is the source of the lessened difficulty of satisfiability, in the presence of the universal modality.

The proof that satisfiability of  $L(KR)$  with the universal modality and nominals is in exponential time is accomplished by a finite model argument: If a formula  $\varphi$  is satisfiable, then it is satisfiable in a model with at most  $c^n$  nodes, where  $c > 1$  is a fixed constant, and  $n$  is the size of  $\varphi$ . This result follows from a filtration argument, which is fairly standard in modal logic. Given  $\varphi$ , one finds the set of all subformulas, and closes this under negation. Call this set  $Cl(\varphi)$ . Then if  $\mathbf{A}$  is any structure and  $d \in d^{\mathbf{A}}$  with  $d \models \varphi$ , we can construct an equivalence relation  $\equiv_{\mathbf{A}}$  on  $D$  by

$$d \equiv_{\mathbf{A}} e \iff (\forall \psi \in Cl(\varphi))(d \models \psi \iff e \models \psi).$$

The equivalence classes of this relation can then naturally be made into a structure satisfying  $\varphi$ , and there are at most exponentially many classes. It is only necessary to check that nominals denote singleton sets in the quotient structure.

The above argument only establishes a nondeterministic exponential time bound for satisfiability, as the decision procedure based on it requires guessing of the small model. The refinement of the argument to get a deterministic exponential procedure is more complex, and we omit the details here. Finally, the proof that the satisfiability problem is hard for exponential time requires an embedding of the corresponding problem for *propositional dynamic logic* [21]. Details are again omitted.

The questions of satisfiability in a *finite* model for KR with path equations and the universal modality seems not to have been studied. Notice that the model constructed in the proof above is infinite. Below, though, I provide a short proof of undecidability in this case using a method of Smolka's. Let us then turn to some complexity results for the Smolka-Johnson class of attribute-value logics.

We recall Smolka's notion of *constraint*:

- $xp \doteq yq$  for variables  $x, y$  (path equation);

- $xp \doteq a$  for atom  $a$  (path equation and constant);
- $xp \uparrow$  (divergence);
- Boolean combinations of constraints;
- $\exists x(\varphi)$  where  $\varphi$  is a constraint.

From what we have seen on path equations and nominals, the following class of constraints is natural:

**Definition 5.2. (Existential prenex constraints)** *A constraint is in existential prenex form (EPF) if it is of the form*

$$\exists x_1 \cdots \exists x_n \varphi,$$

where the constraint  $\varphi$  is quantifier-free.

Smolka gives a decision procedure, based on syntactic transformations, for checking satisfiability of EPF constraints. It relies on transforming the matrix of the constraint to disjunctive normal form, and in effect eliminating the initial existential quantifiers. Eventually, satisfiability reduces to considering *feature clauses* which are conjunctions of atomic constraints. Such a conjunction is said to have a solution if there is an assignment to its free variables (with respect to some feature system) making all of the conjuncts true. Existence of such a solution is shown to be decidable in at most quadratic time, again by a transformation method into what is called solved form.

Similar methods apply to *sorted* constraints, which, as we recall, allow constraints of the form  $x : S$  where  $S$  is a feature term (Kasper-Rounds formula, Ait-Kaci  $\psi$ -term) over the sort alphabet  $A$ .

Smolka also provides an undecidability result for general feature logic, where universal quantification is allowed over variables involving path equations. This result is very much analogous to the undecidability result of Blackburn and Spaan for the universal modality. The technique was in fact shown in Theorem 5.1. This theme is continued in [19]. There it is shown that systems of constraints including path constraints and *subsumption constraints*, of the form  $xp \sqsubseteq yq$ , where  $x$  and  $y$  are variables and  $p$  and  $q$  are paths, have an undecidable satisfiability problem (for the definition of  $\sqsubseteq$ , see def. 2.4.) A recent difficult positive decidability result on so-called *functional uncertainty constraints* has been shown by Backofen [6]. The form of such constraints is  $x\alpha = y$ , where  $\alpha$  is a regular expression over the feature alphabet  $L$ . The interpretation in a feature system is that some path given by the expression leads from the element denoted by  $x$  to that denoted by  $y$ .

As a final excursion into complexity properties, we consider a method due to Johnson [37], [36] This relies on a “correspondence” theory between feature constraints and first order logic. The idea is to express natural feature constraints on first-order terms, but to rely on general decidability results for first order systems to derive them for feature constraint systems.

The observation in [37] is that feature constraints often translate into a special decidable subclass of first-order formulas: the *Schönfinkel-Bernays* class *SB*. A

formula is in *SB* iff it is of the form

$$\exists x_1 \dots x_m \forall y_1 \dots y_n \varphi,$$

where  $\varphi$  contains no function symbols or quantifiers.

Following the methodology of his thesis and book, Johnson shows how to translate various assertions about features into the class *SB*. The first observation is that the functional nature of features can be captured by a formula of the class. In the following the three-place relation  $\text{arc}(x, y, z)$  denotes a “feature arc” labeled by  $y$  connecting the nodes  $x$  and  $z$ . Then functionality is captured by the usual axiom

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

This is an *SB* formula.

Other constraints similarly can be translated. We content ourselves with examples.

(i) *Path equations*.

$$\text{sem} : \text{swim} \wedge \neg[\text{agr} : (\text{num} : \text{sing} \wedge \text{person} : 3)]$$

translates to

$$\begin{aligned} \text{arc}(c, \text{sem}, \text{swim}) \wedge \exists v \text{arc}(c, \text{agr}, v) \\ \wedge \neg[\text{arc}(v, \text{num}, \text{sing}) \wedge \text{arc}(v, \text{person}, 3)] \end{aligned}$$

Perhaps the constant  $c$  should really be a free variable, but if the same “anonymous” constant is used to denote the “initial node” of a feature system, satisfiability of a conjunction of constraints in feature logics will reduce to satisfiability of a conjunction of *SB* formulae with no loss of expressive power.

(ii) *Extensionality*. This constraint says that no two distinct elements of the same sort have identical attribute values.

$$\begin{aligned} \forall x, y, u, v \quad & A(x) \wedge \text{arc}(x, \text{num}, u) \wedge \text{arc}(x, \text{person}, v) \\ & \wedge A(y) \wedge \text{arc}(y, \text{num}, u) \wedge \text{arc}(y, \text{person}, v) \\ & \rightarrow x = y \end{aligned}$$

(iii) *Definitions of predicates*. Suppose we have a formula which defines the predicate  $A$ :

$$\forall x (A(x) \leftrightarrow \Phi(x)).$$

where, say,  $\Phi$  is a quantifier-free formula not involving  $A$  (the quantifier-free condition can be relaxed.) As it stands, the definitional formula is not in *SB*. But if we are only concerned with satisfiability, then the formula may be replaced by the “one-sided” *SB* formula

$$\forall x (A(x) \rightarrow \Phi(x)).$$

It is then not hard to show that when the occurrences of  $A$  appear only positively inside another constraint  $\Psi$ , then the two-sided formula

$$\Psi(A) \wedge \forall x (A(x) \leftrightarrow \Phi(x))$$

is satisfiable if and only if the corresponding one-sided formula is. So, we still have a decision method for satisfiability in this case.

Johnson shows how to define certain formulae involving sets using the last item above. This entails the decidability of satisfiability for formulae involving such predicates, in the time necessary for the SB class. As to the complexity of deciding this problem, if the number of universal quantifiers is fixed, then the problem is *NP*-complete. The problem is in general complete for polynomial space if the number of such quantifiers is not fixed [44]. Johnson notes, however, that most of the linguistic uses require only fixed numbers of the quantifiers. This method thus does not lead to more complexity than in the original KR logic.

## 6. Order-theoretic aspects of feature structures

The unification method, alluded to in previous sections, is an efficient algorithm for constraint solving. Is there anything more to the method than this? Why is it that grammars with augmented with feature constraints are called “unification grammars”? We will look at ordering relations in feature systems to answer the question. In the process we will learn more about feature logics in the disguise of disjunctive type specifications. One of the principal tools we use is the theory of complete partially ordered sets, or *domain theory*. This theory is generally about constructive information-theoretic ordering relations. Only a few of the results are needed for our applications. On the other hand, feature systems provide a very nice illustration of some of the domain-theoretic constructions.

Historically, Pereira and Shieber [54] were the first to suggest using domain theory to give a semantics of grammars. Their work preceded the introduction of feature logic, and appeared at about the same time as Ait-Kaci’s work on what might be called feature types. Ait-Kaci uses domain-theoretic methods to construct disjunctive types ( $\epsilon$ -terms) from conjunctive types ( $\psi$ -terms.) Pereira and Shieber suggest that types might be modeled by a so-called *powerdomain* construction.

Johnson [35] also gives an interesting discussion on partiality issues for feature structures. His view, as we have mentioned, is that feature logic, without loss, can restrict itself to talking about total models. But the comparisons he makes between the two views are informative. In particular, partiality for Johnson, Smolka, and in fact the modal logicians resides in the logical theory, not in the models. Fenstad’s chapter in this Handbook discusses the distinctions.

Carpenter’s book on typed feature structures and their logic contains a detailed review of domain-theoretic ideas needed to understand these constructions. Here I will not be so thorough, but will use constructions as necessary to show how they arise in the task of modelling, say, disjunction. But we need to begin by

examining how ordering is associated with the class of feature structures. For this, recall the definition of feature structure as a principal feature system as in def. 2.3, and the definition of subsumption between two structures  $\mathbf{A}$  and  $\mathbf{B}$ , by means of a homomorphism of their domains (cf def. 2.4.) In case  $\mathbf{A}$  and  $\mathbf{B}$  are feature structures, we require the mapping to preserve the generators.

**Example 6.1.** Let  $D^{\mathbf{A}} = \{d_0, d_1, d_2\}$ , with  $d_0f = d_1$  and  $d_0g = d_2$ . Let  $D^{\mathbf{B}} = \{e_0, e_1\}$  with  $e_0f = e_0g = e_1$ . Then  $\mathbf{A} \sqsubseteq \mathbf{B}$  via the map  $\gamma$  sending  $d_0$  to  $e_0$  and  $d_1$  and  $d_2$  to  $e_1$ .

It is easy to see that if there are two homomorphisms  $\gamma : D^{\mathbf{A}} \rightarrow D^{\mathbf{B}}$  and  $\gamma' : D^{\mathbf{B}} \rightarrow D^{\mathbf{A}}$ , then these are mutual inverses, so that  $\mathbf{A}$  and  $\mathbf{B}$  are isomorphic. (The functional nature of the features makes this true.)

If  $\mathbf{A} \sqsubseteq \mathbf{B}$ , we regard  $\mathbf{B}$  as *at least as informative* as  $\mathbf{A}$ . In example 6.1, the structure  $\mathbf{B}$  is more informative, as it identifies two potentially different feature values. In this example, the structure  $\mathbf{B}$  is *extensional*, as no two distinct elements have exactly the same features with the same values. This is not the case for  $\mathbf{A}$ , as the elements  $d_0$  and  $d_1$  are distinct but indistinguishable. (We do not have space in the chapter to pursue the interesting topic of extensionality. Some discussion appears in Carpenter [14, chapter 8].)

The collection of isomorphism classes of feature structures is thus partially ordered by  $\sqsubseteq$ . In fact a better representation is available. Moshier, in his thesis [47], shows that this collection is order-isomorphic to the partially ordered set of *abstract feature structures*. Such an abstract structure is a triple  $(P, N, V)$ , where  $P$  is a nonempty, prefix-closed subset of  $L^*$ ,  $N$  is a right-invariant equivalence relation<sup>2</sup> (a Nerode relation) on  $P$ , and  $V$  is a relation from  $P$  to  $A$  respecting the relation  $N$ ; that is, if  $p Va$  and  $p N q$  then  $q Va$ .

Let  $(\mathbf{N}, \sqsubseteq)$  be the poset of abstract feature structures, where  $(P_1, N_1, V_1) \sqsubseteq (P_2, N_2, V_2)$  iff  $P_1 \subseteq P_2$ ,  $N_1 \subseteq N_2$ , and  $V_1 \subseteq V_2$  (this last as graphs of relations.) This poset is almost the same as the typed Nerode feature system in Example 2.2. It differs only in that  $V$  is relational, not functional.

Here, for example, is the representation of a concrete structure  $(\mathbf{A}, d_0)$ . We let

$$P = \{p \mid d_0p \downarrow\}; N = \{(p, q) \mid d_0p = d_0q\}; \text{ and } V = \{(p, a) \mid d_0p \in D_a\}.$$

What kind of partially ordered set is the set of abstract feature structures? It turns out that (with proper assumptions on sorts, which we ignore for the moment) that  $\mathbf{N}$  is a *Scott domain*. We review the necessary terminology.

**Definition 6.1. (Domain-theoretic definitions)** A directed subset of a partial order  $(U, \sqsubseteq)$  is a non-empty set  $X \subseteq U$  such that for every  $x, y \in X$ , there is a  $z \in X$  such that  $x \sqsubseteq z$  and  $y \sqsubseteq z$ . A complete partial order (cpo) is a partial order which has a bottom element and least upper bounds of directed sets. A subset

<sup>2</sup> Recall from Section 2 that a relation  $N$  is right-invariant if whenever  $d N e$  and  $dl \downarrow$  then  $el \downarrow$  and  $dl N el$ .

$X \subseteq U$  is bounded (or compatible, consistent) if it has an upper bound in  $U$ . An isolated (or compact, finite) element  $x$  of  $U$  is one such that whenever  $x \sqsubseteq \bigsqcup X$  with  $X$  directed, we also have  $x \sqsubseteq y$  for some  $y \in X$ . A cpo is algebraic if each element of which is the least upper bound of a set of isolated elements. A cpo is  $\omega$ -algebraic if it is algebraic and the set of isolated elements is countable. A Scott domain is an  $\omega$ -algebraic cpo in which every compatible subset has a least upper bound. Scott domains are somewhat more anonymously termed “bounded-complete partial orders” (BCPOs).

It is not too hard, using the set-theoretic definition of the partial order in  $\mathbf{N}$ , to check that  $\mathbf{N}$  in fact is a Scott domain, and that the finite feature structures are the compact elements. Notice that a finite feature structure is represented in  $\mathbf{N}$  as a triple  $(P, N, V)$  where the equivalence relation  $N$  is of finite index.

Now it is possible to give an abstract characterization of *unification*. If  $x$  and  $y$  are elements of  $\mathbf{N}$ , and  $\{x, y\}$  is bounded above, then the least upper bound  $x \sqcup y$  exists, and is called the *unification* of  $x$  and  $y$ . One can likewise form the least upper bound of any nonempty subset of  $\mathbf{N}$  that is bounded above. Note also that the least upper bound of finite elements must be finite in any Scott domain; and one can find the *greatest lower bound* of an arbitrary nonempty subset. This latter is called the *generalization* of the subset.

Actually, the Scott domain determined by the class of all abstract feature structures, with no assumptions about the ordering of sorts, is uninteresting, because any two feature structures will be unifiable. In fact, the domain turns into a *complete lattice*, with the top element being the full tree on  $L$ , with all paths equivalent. So, we generally make sort assumptions, the most common of which is to assume a Scott ordering (also denoted by  $\sqsubseteq$ ) on the sort symbols. We also typically limit the class of feature structures so that a given node is assigned a special type, which is thought of as the “most specific” type for that node. The following, for example, is Carpenter’s definition of typed feature structure.

**Definition 6.2.** A *typed feature structure* is a triple  $(\mathbf{A}, d_0, \theta)$ , where  $(\mathbf{A}, d_0)$  is a feature structure, and  $\theta : D^{\mathbf{A}} \rightarrow A$  is a total function. A node  $d \in D_a$  if and only if  $a \sqsubseteq \theta(d)$ .

If  $\mathbf{A}$  is a typed structure, then the sets  $D_a$  respect the ordering on sorts, but more general sorts denote bigger subsets: if  $a \sqsubseteq b$ , then  $D_a \supseteq D_b$ . (The reverse inclusion need not hold: Suppose archy is the only bug-author. He might be the only bug and the only author, but the type of authors need not subsume the type of bugs.)

For typed feature structures, a different definition of homomorphism is used: if  $\gamma : D^{\mathbf{A}} \rightarrow D^{\mathbf{B}}$ , then one requires  $\theta(d) \sqsubseteq \theta(d\gamma)$ . But this definition, in the case of typed structures, is really the same as the one we gave at the beginning of the section.

**Proposition 6.3.** For typed feature structures,

$$\theta^{\mathbf{A}}(d) \sqsubseteq \theta^{\mathbf{B}}(d\gamma) \iff \{a \mid d \in D_a^{\mathbf{A}}\} \subseteq \{a \mid d\gamma \in D_a^{\mathbf{B}}\}.$$

**Proof.** Assume the left side of the condition. Let  $a$  be such that  $d \in D_a^{\mathbf{A}}$ . Then  $a \sqsubseteq \theta(d)$ , so  $a \sqsubseteq \theta(d\gamma)$ . Therefore  $d\gamma \in D_a^{\mathbf{B}}$ . Conversely, let  $a = \theta^{\mathbf{A}}(d)$ . Then  $a \in \{a \mid d \in D_a^{\mathbf{A}}\}$ . So  $a \in \{a \mid d\gamma \in D_a^{\mathbf{B}}\}$ . This means  $a \sqsubseteq \theta(d\gamma)$  as required.  $\square$

If we assume that all feature structures are typed, then we may equivalently represent the domain  $\mathbf{N}$  of abstract feature structures over  $A$  and  $L$  as the set of all triples  $f = (P, N, \theta)$  where  $P$  and  $N$  are as before, but now  $\theta : P \rightarrow A$ . The ordering relation  $f_1 \sqsubseteq f_2$  still requires set containments for the  $P$  and  $N$  components but now requires  $\theta_1(p) \sqsubseteq \theta_2(p)$  for all  $p \in P_1$ .

We assume for the rest of the section that feature structures are typed and that the underlying sort set  $A$  is a Scott domain. We can now tie feature logics to domain theory as follows. Consider Kasper-Rounds logic with path equations, but with only basic formulas and those involving prefixing, but not conjunction, negation, or disjunction. Call such formulas *extended basic*. For a feature structure  $(\mathbf{A}, d_0)$ , put

$$Th(\mathbf{A}) = \{\varphi \mid d_0 \models \varphi\}.$$

where the  $\varphi$  are extended basic. It is straightforward to show the following:

**Theorem 6.4.**  $\mathbf{A} \sqsubseteq \mathbf{B}$  if and only if  $Th(\mathbf{A}) \subseteq Th(\mathbf{B})$ .

**Theorem 6.5.** Every satisfiable extended basic formula  $\varphi$  has a least satisfier  $\mathbf{A}_\varphi$  in the subsumption order, up to isomorphism. In fact this satisfier is finite.

We recall that KR logic without negation is *persistent*, so this last result tells us that an extended basic formula  $\varphi$  determines a *principal filter* in the poset  $\mathbf{N}$ . In fact it is a *compact filter*, being generated by a compact element.

When we allow conjunctions and disjunctions, we get the following result. It was originally shown by Kasper and Rounds, but only for the special kinds of structure they were considering.

**Theorem 6.6.** The collection of satisfiers of any KR formula without negation is a finite union of principal compact filters in the domain of feature structures.

**Theorem 6.7.** Every compact element  $\mathbf{A}$  in the domain of feature structures has a corresponding formula  $\varphi_{\mathbf{A}}$  such that the principal filter generated by  $\mathbf{A}$  is the filter of satisfiers of  $\varphi_{\mathbf{A}}$ .

These results have the following significance: they show that negation-free KR logic, with no modalities other than prefixing, and with path equations, is in some sense the “natural” finite logic for the domain of feature structures. The reason is that every domain (in fact algebraic cpo) has a “natural” positive logic associated with it. This logic is given by the *Scott topology* of the domain.

**Definition 6.8.** Let  $(U, \sqsubseteq)$  be a cpo. A set  $V \subseteq U$  is said to be **Scott open** if (i)  $U$  is upward-closed, and (ii) for any directed  $D \subset U$ , we have  $\bigsqcup D \in V$  iff  $U \cap D \neq \emptyset$ .

(By “upward-closed”, we mean that if  $x \in U$  and  $x \sqsubseteq y$ , then  $y \in U$ .)

One checks readily that under this definition of “open”, that the collection of open subsets of a cpo form a topological space. Such a space must contain  $\emptyset$  and  $U$ , and be closed under finite intersections and arbitrary unions. Furthermore, we can regard open sets as being “properties” of domain elements. The definition says that if an element has a certain property, then we can discover that the property holds by testing a “sequence” of finite elements which “converges” to the given element. (In general, sequence really means directed set, and “converges to an element” means that the element is the least upper bound of the set.) After a finite time, we find that the element does indeed have the property. Such properties are sometimes called “affirmable” [73].

It is straightforward to prove the following in any algebraic cpo  $U$ .

**Theorem 6.9** (Compactness in the Scott topology). *(i) For each finite element  $f \in U$ , the principal compact filter  $\uparrow f = \{u : f \sqsubseteq u\}$  is open.*

*(ii) Every open set  $V$  is the union of the principal compact filters generated by the compact elements of  $V$ .*

*(iii) Every compact open set  $X$  is a finite union of such compact principal filters. (Compact here means the topological usage: every covering of  $X$  by open sets has a finite subcovering.)*

We want to make a “logic” out of open sets in the domain  $\mathbf{N}$ . To do this, we make the set  $\mathbf{N}$  into a feature system. If  $(P, N, \theta) \in \mathbf{N}$ , and  $l \in L$ , then define  $(P, N, V)l = (P/l, N/l\theta/l)$ , where (i)  $P/l = \{q \mid lq \in P\}$ ; (ii)  $q N/l r$  iff  $lq N lr$ , and (iii)  $\theta/l(q) = \theta(lq)$ . For  $a \in A$  we let  $D_a = \{(P, N, \theta) \mid \theta(\lambda) \sqsubseteq a\}$ . This in fact is the *Nerode system* in Example 3.2 of section 3.

Now we can make a “logic” by using the following “syntax”.

- $\mathbf{N}$  is a property;
- For  $p, q \in L^*$ , the set  $\{f \in \mathbf{N} \mid fp = fq\}$  is a property;
- For  $a \in A$ , the set of feature structures  $\{(P, N, \theta) \mid a \sqsubseteq \theta(\lambda)\}$  is a property;
- The empty set is a property;
- If  $X \subseteq \mathbf{N}$  is a property and  $l \in L$ , then  $\{f : fl \in X\}$  is a property;
- The union and intersection of two properties is a property.

“Satisfaction” in this “logic” is just membership: a feature structure  $f \models X$  if and only if  $f \in X$ .

More properly, we define the class of properties to be the least class of subsets of  $\mathbf{N}$  closed under the above operations. Then our results above tell us that the class of properties coincides with the sets definable as models of positive KR formulas, and further that these are exactly the compact open subsets of the domain of feature structures.

**The Smyth powerdomain.** We have not yet completed our promised task: to show how disjunctive types are modeled using domain-theoretic methods. Each finite feature structure (compact element) represents a conjunctive type. But we have not yet presented a domain construction such that an element of the domain

represents a *disjunctive* type. One might think that the generalization of two feature structures represented the disjunction of their types, because the unification represents the conjunction. But this is easily seen to be incorrect, as generalization does not distribute over unification.

Fortunately we have done most of the work necessary to introduce disjunctive types, because we already have disjunction in our logic. The collection of all finite unions of compact principal filters is the set of compact elements of  $\mathbf{N}$ . And each such finite union is really determined by a collection (antichain) of pairwise-incomparable compact elements; namely, the generators which do not subsume each other. So, we should take these collections as representatives of disjunctive types. Then the question becomes: how to order such antichains?

Remember that an antichain  $\{f_1, \dots, f_n\}$  represents the type of an object which is more specific than at least one of the  $f_i$ . We can thus increase the informativeness of the antichain by removing one or more of the elements, or by increasing the informativeness of one or more of the elements. So now if  $X$  and  $Y$  are compact antichains, we set  $X \sqsubseteq_S Y$  (Smyth subsumption) iff for all  $g \in Y$ , there is some  $f \in X$  with  $g \sqsubseteq f$ . It is easy to see that  $\sqsubseteq_S$  is a partial order on antichains.

This construction actually does most of the work for us when we want to calculate with disjunctive types. It is possible to show (see, for example, Carpenter, chapter 12) that the Smyth subsumption relation on compact antichains actually gives us a distributive lattice. So then, for example, to unify  $X$  and  $Y$  we form the set  $\{x \sqcup y \mid x \in X; y \in Y\}$  and remove non-minimal elements. The formula is expected given that the distributive law holds in the logic of the domain of feature structures. But in fact the construction is general.

As an aside, exactly these same methods were first used by Ait-Kaci [3] to pass from conjunctive types to disjunctive ones. Ait-Kaci's syntax for conjunctive types is called the calculus of  $\psi$ -terms. In effect these are identifiable with, not so much conjunctive Kasper-Rounds formulae, but conjunctive Kasper-Rounds formulae without path equations and with nominals. So, for example, the type of people whose father is their employer is notated by Ait-Kaci as

$$Person(father \Rightarrow y; employer \Rightarrow y).$$

The translation of this should be clear; use *father* and *employer* as attributes, *Person* as a sort symbol, and *y* as a nominal. (These terms are very flexible. They form the basic data types in Ait-Kaci's logic programming language LIFE [4].) One can pass to disjunctive types ( $\epsilon$ -terms) by introducing a connective for disjunctions and proceeding to give equational rules as suggested by the logic of feature structures.

In effect we have now constructed the set of compact elements of the Smyth powerdomain of the domain of feature structures. To generate the "complete" complete partial order "determined" by these compact elements, we use a domain-theoretic technique known as *ideal completion*. This is a general technique which always produces an algebraic cpo from a partially ordered set of elements, in such a way that all existing meets and joins are preserved. The details in general are as follows. Given a partially ordered set  $(X, \preceq)$ , we define an *ideal* of  $X$  to be a nonempty,

directed, and downward-closed subset  $I$  of  $X$ . (The last condition just says that if  $x \in X$  and  $u \preceq x$  then  $u \in X$ .) Then the ideal completion of  $(X, \preceq)$  is the collection of ideals of  $X$ , partially ordered by inclusion of sets.

We thus form the ideal completion of our set of compact Smyth elements, and arrive at the Smyth or *upper* powerdomain. It is possible to prove that if one starts with a Scott domain, then the Smyth powerdomain is again a Scott domain. And one can use other orderings on sets of domain elements to get new powerdomains. Two other very common constructions, for example, are the *Hoare* (lower) and the *Plotkin* (convex) powerdomains. See Gunter and Scott [25] for a good survey; other references are Abramsky [1], or Zhang [76] for the details of passing to a logic from a domain.

At this point, we have digressed pretty much from linguistics. To return thereto, and to close out the subsection, we sketch the details of another powerdomain construction due to Pollard and Moshier [57]. It is one way in which *set values* might be handled order-theoretically.

Consider the following sentence.

He thinks he is smart.

The sentence has a *set* of referents, which might occur as the value of a CONTEXT feature. What should be the type for this set? The two occurrences could be coreferential or not. Otherwise, the two occurrences have identical features: third person masculine singular. Here is a feature structure proposed by Pollard and Moshier to account for the above sentence.

$$\left[ \begin{array}{l} \text{CONTENT} \left[ \begin{array}{l} \text{EXPERIENCER} \boxed{1} \\ \text{THEME} \boxed{2} \end{array} \right] \\ \text{CONTEXT} \left\{ \boxed{1} \left[ \begin{array}{l} \text{PERSON} \quad 3\text{rd} \\ \text{NUMBER} \quad \text{sing} \\ \text{GENDER} \quad \text{masc} \end{array} \right], \boxed{2} \left[ \begin{array}{l} \text{PERSON} \quad 3\text{rd} \\ \text{NUMBER} \quad \text{sing} \\ \text{GENDER} \quad \text{masc} \end{array} \right] \right\} \end{array} \right]$$

The value of the CONTEXT attribute is to be a set consisting of two parameters, each qualified to be of the same type. These parameters are coreferential with the values of the EXPERIENCER and THEME attributes of the sentence. For sake of simplicity let us focus only on this “set node”, ignoring coreferences within the structure:

$$\left\{ \left[ \begin{array}{l} \text{PERSON} \quad 3\text{rd} \\ \text{NUMBER} \quad \text{sing} \\ \text{GENDER} \quad \text{masc} \end{array} \right], \left[ \begin{array}{l} \text{PERSON} \quad 3\text{rd} \\ \text{NUMBER} \quad \text{sing} \\ \text{GENDER} \quad \text{masc} \end{array} \right] \right\}$$

We would like a “type” corresponding to this “set”. Notice that since we have listed two identical feature structures in the “set”, by the usual set theory extensionality laws, the set would only contain one element. So none of the usual

powerdomain constructors will work, because the basis of compact generators is a set (antichain) of compact elements. Another problem is to make sure that no more than two individuals, but at least one individual, is represented.

Pollard and Moshier solve this problem by proposing a new powerdomain construction. One begins with the collection  $K(U)$  of compact elements of the underlying domain  $(U, \sqsubseteq)$  as before. However, for basis elements of the Pollard-Moshier powerdomain, one uses the collection of finite *multisets*  $X$  over  $K$ . (A finite multiset simply allows repetition of an element.) The relation  $\sqsubseteq_{PM}$  is then the following:  $X \sqsubseteq_{PM} Y$  iff there is an onto function  $\mu : X \rightarrow Y$ , such that

$$(\forall x \in X) (x \sqsubseteq \mu(x)).$$

This relation is reflexive and transitive, though not symmetric. But the ideal completion construction still works to give an algebraic cpo. In the example of the two pronoun occurrences, we see that the given multiset models those sets of individuals of the same type, consisting of either one or two members.

Our presentation of the Pollard-Moshier construction leaves out many other details of their set-valued feature structures. The point here is that not only does domain theory lead to new tools for understanding linguistic constructions, but also that linguistic examples suggest new definitions and results in the theory of domains.

## 7. An Application

In this last technical section of the chapter we give an illustration of the use of feature logic: an application of the ideas to the topic of *multistratal grammar*. This subject emerged in the 1970's as *relational grammar*, principally through the work of Postal and Perlmutter (see, for example, Perlmutter [56] or Postal and Joseph [60] for representative work.)

Multistratal grammar harks back to the early days of transformational theory. As we recall, the notion of a construction like “passive” in some intuitive sense identifies the surface subject *John* in the sentence *John was hit by Bill* with the “deep” or “initial” direct object. But whereas transformational theory represented this intuition by means of derivations, relational grammar took the position that a proper linguistic structure for a passive sentence would in fact record the information that the surface subject was once the direct object. The “stratal diagrams” of relational grammar are an attempt to do this, for example.

A consequence of this view of linguistic objects is that one can directly enunciate constructions like “passive” and “raising” as rules of grammar, in such a way that inspection of the structure guarantees that the rules have been satisfied. This is an idea which we see now in HPSG, but HPSG is *monostratal*, because there is no notion of an “earlier” stratum. Current versions of GB, by the same intuitive classification, could be said to be *bistratal* in that some constraints apply before the “move- $\alpha$ ” transformation has taken place, and some after.

We will not dwell in this short exposition on the details of relational diagrams, or on the graphical versions formalized in *arc-pair grammar* [34] or multigraph grammar [59]. Instead we will illustrate a current version of the theory, due to Johnson and Moss [33]. *Stratified feature grammar* is a representation of multistratal grammar which integrates feature logic with the multistratal point of view. Feature logic is useful here because it allows for the natural expression of constructions working across strata, without having to adapt first-order logic for this purpose. It also shows how it might be possible to use some of the computational algorithms available in unification-based systems to actually parse in a multistratal style. And in fact, such a parser has been implemented [32].

What are the essential features of stratified feature grammars? There are two main innovations.

- To be able to capture the multistratal point of view, Johnson and Moss introduce structure on the label alphabet  $L$ . In typical linguistic analyses, labels stand for grammatical relations like “head”, “comp”, “patient”, and the like. Call these feature names “R-signs” for the moment. Then, the labels on feature arcs in SFG are *sequences* of R-signs, standing for the role that a given constituent may have played at different strata. For example,  $[2, 1] : \textit{glass}$  tells us that the word “glass” is initially a direct object (2) and finally a subject (1). The label  $[2, 1]$  is not a path, but a single member of the label alphabet.
- Persistence in feature logics is in fact a persistent problem for linguistic analyses. A feature structure may, for example, satisfy  $\textit{agr} : \textit{per} : 3$ , but if no restrictions are placed on the typing system, a feature structure with extra completely irrelevant arcs like  $\textit{employer} : \textit{IBM}$  will also be allowed by the logic. But we certainly wish to rule out such irrelevant arcs. This is the reason that appropriateness conditions, for example, are introduced in Carpenter’s typing system. Other systems place global conditions on models: typically that feature structures satisfying a formula be minimal in the subsumption ordering.

In SFG, a new idea, called *justification*, is introduced to rule out, among other things, extraneous arcs. The difference between justification and a global minimality condition is that justification applies locally. In the example above, if  $\textit{agr} : \textit{per} : 3$  were a “rule”, then it only would justify itself, or perhaps naturally co-occurring features, and not an irrelevant arc. So, the idea of justification is that each “local” piece of data occurring in a feature structure must be justified by some rule of the grammar, where “rule” is interpreted as a feature logic formula.

SFG independently adopts conventions that we have already seen in other systems of feature logic. Rules of the grammar are formulas of (stratified) feature logic, and a grammar is a conjunction of such rules. A stratified feature structure will satisfy a grammar only if at each node, all rules are satisfied. So, something like a universal modality is in effect, and rules typically have the form  $a \rightarrow \psi$ , where  $a$  is a fairly simple type, and  $\psi$  is a more complicated formula. Justification, in addition, requires the data at each node of a feature structure to be justified by some rule. (This is a very imprecise rendering of the actual definition.) Word ordering is handled in

SFG by *precedence constraints* on feature arcs. We will ignore word ordering in this overview.

Our presentation is by example, building on the illustrations of Johnson and Moss.

### Examples.

- Consider the sentence *Joe breaks the glass*. This is diagrammed as a feature structure in Figure 2. We present an (oversimplified) SFG lexical rule, stated in logical format, which accounts for the sentence:

$$\textit{breaks} \rightarrow [\textit{Head}]^{-1} : \{[\textit{Cat}] : S \wedge [1] : \textit{true} \wedge [2] : \textit{true}\}.$$

This rule states that the head verb “breaks” occurs as the value of the attribute “Head” in a structure which also requires a subject [1] and a direct object [2]. It actually is a logic formula, but with a new “inverse label” modal connective. The semantics of such formulas is in general

$$d \models l^{-1} : \varphi \text{ iff there is a unique } e \in D \text{ such that } el = d \text{ and } e \models \varphi.$$

We have other lexical entries for “glass” and “Joe”. The entry for “Joe”, for example, could be

$$\textit{Joe} \rightarrow \textit{true}.$$

This rule seems to be logically superfluous. However, it *justifies* the occurrence of “Joe” in Figure 2. And in a more detailed example, an entry would provide agreement and morphological information.

- Now we show how simple constructions combine with lexical rules. It also begins to explain the function of the stratified feature labels. Consider the “unaccusative” sentence *Glass breaks*. This is diagrammed in Figure 3. In this case, we need to recognize that “break” is a verb that may have no explicit agent. So an additional entry might be

$$\textit{breaks} \rightarrow [\textit{Head}]^{-1} : \{[\textit{Cat}] : S \wedge \neg[1] : \textit{true} \wedge [2] : \textit{true}\}.$$

(The open and closed bracketing will be explained in a moment.)

Next, there is a rule which embodies the unaccusative construction. This is a formula **Unacc**, defined to be

$$\neg[1] : \textit{true} \rightarrow \{(2) : \textit{true} \rightarrow (2, 1) : \textit{true}\}.$$

This rule says that “if there is no initial subject, then assuming there is an initial object, that object can be “advanced” to be a subject at the next stratum. (In point of fact, these constructions are given by so-called “extension formulas”,

which have yet another semantics. But the formula above serves to show the idea.)

The feature structure in Figure 3 satisfies both these formulas. The reason it satisfies the lexical rule is that the label [2] occurring in the lexical entry for “breaks” *extends to* the label [2, 1] occurring in the feature structure. Similarly (2, 1) in the unaccusative rule extends to [2, 1] in the feature structure. This explains the open and closed parentheses enclosing sequences of features; there is a natural partial ordering  $\sqsubseteq$  (not, by the way, a Scott ordering) of the stratified label set  $L$ . For example  $[2, 1] \sqsubseteq [2, 1, 0]$  and  $[2, 1]$ , but  $[2, 1]$  does not extend to anything but itself. The definition of satisfaction for such an ordered label set now reads

$$d \models l : \varphi \text{ iff for some unique } f \text{ with } l \sqsubseteq f, df \models \varphi.$$

It might be argued that the unaccusative rule is superfluous, since it is logically implied by the right side of the lexical rule. But the lexical rule does not *justify* the occurrence of [2, 1]. It only justifies the occurrence of a [2]. The unaccusative rule does justify this occurrence.

- Here is another example showing the Dative construction. Consider the sentence *Joe gave Mary tea*. This appears in Figure 4. An entry for “gave” might be

$$\text{gave} \rightarrow [\text{Head}]^{-1} : \{[\text{Cat}] : S \wedge [1] : \text{true} \wedge [2] : \text{true} \wedge [3] : \text{true}\}.$$

We also have a rule for Dative:

$$(3) : \text{true} \wedge (2) : \text{true} \rightarrow (3, 2) : \text{true} \wedge (2, 8) : \text{true}.$$

This rule is an extension formula, and should not technically be read as an implication. The reasons are too detailed to cover here. The intent is certainly an implicational one: if there is an initial indirect object (3) and an initial direct object, then the indirect object gets “advanced” to a direct object, and the direct object becomes a “chômeur” (8).

- Constructions can interact. Consider a final example : *Mary was given tea by Joe*. This appears in Figure 5.

In this example nominals  $x, y$  are used to indicate sharing, though SFG uses path equations. The example shows a nontrivial clause structure (the value of  $[\text{Comp}]$ ) and the interaction of Dative and Passive. I will not attempt to give all the rules involved in this structure. One other aspect of it needs to be pointed out: the occurrence of the “0” sign in the feature labels like  $[1, 8, 0]$  and  $[0, \text{Marked}]$ . “0” is a special “null” relational sign. When it occurs at the end of a sequence, it means that the value of that attribute plays no role in the surface form of the sentence. Likewise, when 0 occurs at the beginning of a sequence, it means that the constituent plays no role in the “initial” or predicate-argument structure of the sentence.

The full notions of predicate-argument structure and surface-structure can also be

illustrated by the last example. To get the predicate-argument structure for “Mary was given tea by Joe” we delete from the structure in Fig. 5 all arcs beginning with a 0. In the remaining arcs, we remove every sign except the first. The result is the following.

$$\left[ \begin{array}{l} [Cat] \ S \\ [Head] \ was \\ \\ [Comp] \ \left[ \begin{array}{l} [Cat] \ VP \\ [Head] \ given \\ [3] \ Mary \\ [2] \ tea \\ [1] \ Joe \end{array} \right] \end{array} \right]$$

Similarly, to get the surface structure, we remove all arcs ending in 0, and remove all signs in the remaining arcs except the last. We obtain

$$\left[ \begin{array}{l} [Cat] \ S \\ (1) \ Mary \\ [Head] \ was \\ \\ [Comp] \ \left[ \begin{array}{l} [Cat] \ VP \\ [Head] \ given \\ (8) \ tea \\ (8) \ \left[ \begin{array}{l} [Cat] \ PP \\ [Flag] \ by \\ [Marked] \ Joe \end{array} \right] \end{array} \right] \end{array} \right]$$

(Notice that there are two (8)’s in the complement, so this last structure is actually a set-valued structure.) From the surface structure, it is finally possible to get a terminal sentence. For this, the surface structure is required to be a tree. Ordering of the words is handled by the precedence relations, a part of SFG we do not cover here.

We also do not go into the full details of the SFG concept of *justification*. The data in a feature system justified by a formula of SFG logic can be given recursively simultaneously with the notion of satisfaction. Suffice it to say that this means that Boolean connectives now cannot be interdefined, even though the definition of satisfaction in SFG uses the classical interpretation of negation and implication. Intuitively speaking, justification introduces *nonmonotonicity* into feature logic. But a full explication of this has not, as far as I know, been undertaken.

To sum up: SFG extends ordinary feature logic by introducing stratified (in general partially ordered) labels, and by introducing some new connectives. It introduces non-persistence via the concept of justification. Rules are lexically based, and satisfaction is required to obey a “universal-modality” principle in that each node of

a feature structure must satisfy all rules. In addition, each “core” node of a feature structure must be justified by some rule.

## 8. Conclusion and future work

I have attempted in the chapter to show how methods of logic, used initially to understand feature-theoretic representations in language, can lead to a development of the mathematics of those representations, and even further, to applications not only in linguistics, but to logic and computer science. In my view, there is still much work to be done both in applications and the theory itself.

Computer science applications that should be pursued more thoroughly, for example, include (i) the use of feature-theoretic ideas in the area of terminological logics for artificial intelligence, and (ii) applications to the design of data base systems and data base query languages. This last application has in a sense a long history, since record structures in the typical relational data bases are just flat feature structures. I will not attempt here to recount specific work on the subject. With regard to (i), we have already alluded to the work of Nebel and Smolka [53]. Work more in the spirit of domain theory, attempting to model subsumption in terminological systems, has been carried out by Dionne, Mays, and Oles [18].

Another extension of feature theory which is needed for linguistic applications is to give a treatment of *non-monotonic* phenomena. We have mentioned feature specification defaults in GPSG, for example. Johnson shows in [37] how some of the non-monotonic phenomena in LFG can be modeled using first-order *circumscription* [45]. Some results on unification of nonmonotonic structures can be found in [75], which also refers to other work like that of [11]. Evans gives an extensive proposal for GPSG in [20]. A full treatment for feature logic has yet to appear, however. (The Handbook chapter by Thomason expands on these issues.)

A third objective for feature theory is to reconcile the distinction between the “total object” perspective of Smolka and Johnson, and the “partial object” view stemming from Kasper-Rounds, and earlier, from Pereira and Shieber. In fact, the theory to date raises more questions than it answers about the fundamental status of linguistic objects. If these objects can be partial, then what is their epistemological status? Are they objectively existing entities? If so, then they should not have undefined appropriate attributes. If we allow such undefined attributes, then our objects may live in a kind of “third world” of descriptions, about which we speak with our grammars. Or, they may be regarded from a situation-theoretic perspective, which allows us the flexibility to look at worlds where not all issues are settled. These are quite general issues, as the chapter on partiality by Fenstad shows. The interesting point is that the issues are exemplified by such easily visualized objects as feature structures. So, we finally need to contrast feature *theory* with feature *logic*. As with ordinary mathematics, the subject matter is at least, if not more, interesting than the formal language we use to describe it.

## References

- [1] S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51, 1991.
- [2] P. Aczel. *Non-Well-Founded Sets*. Center for Study of Language and Information 14, 1988.
- [3] H. Ait-Kaci. *A Lattice-Theoretic Approach to Computation based on a Calculus of Partially Ordered Type Structures*. PhD thesis, University of Pennsylvania, 1984.
- [4] H. Ait-Kaci and A. Podelski. Is there a meaning to life? In *Proceedings of the International Conference on Logic Programming*, 1990.
- [5] F. Baader, H. J. Bürckert, B. Nebel, W. Nutt, and G. Smolka. On the expressivity of feature logics with negation, functional uncertainty, and sort equations. Technical Report RR-91-01, DFKI, 1991.
- [6] R. Backofen. On the decidability of functional uncertainty. In *Proceedings of the 31st Annual meeting of the Association for Computational Linguistics*, pages 201–208, 1993.
- [7] M. Ben-Ari, J. Halpern, and A. Pnueli. Deterministic propositional dynamic logic: finite models, complexity, and completeness. *Journal of Computer and System Science*, 25:402–417, 1982.
- [8] P. Blackburn. Modal logic and attribute value structures. In M. de Rijke, editor, *Diamonds and defaults*. Kluwer, 1993.
- [9] P. Blackburn and E. Spaan. A modal perspective on the computational complexity of attribute value grammar. *Journal of Logic, Language, and Information*, 2:129–169, 1993.
- [10] A. Blass and Y. Gurevich. Existential fixed-point logic. In E. Börger, editor, *Computation Theory and Logic*, pages 20–36. Springer-Verlag Lecture Notes 270, 1987.
- [11] G. Bouma. Feature structures and nonmonotonicity. *Computational Linguistics*, 18:183–203, 1992.
- [12] Ronald J. Brachman and J. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.
- [13] L. Cardelli and P. Wegner. On understanding types, data abstractions, and polymorphism. *Computing Surveys*, 17(4):471–522, 1985.
- [14] B. Carpenter. *The Logic of Typed Feature Structures*. Cambridge University Press, 1992.
- [15] N. Chomsky. *Aspects of the Theory of Syntax*. MIT Press, 1965.
- [16] N. Chomsky and M. Halle. *The Sound Pattern of English*. Harper and Row, 1957.
- [17] A. Dawar and K. Vijay-Shanker. A three-valued interpretation of negation in feature structure descriptions. In *Proceedings of 27th Annual meeting of the Association for Computational Linguistics*, 1991.
- [18] R. Dionne, E. Mays, and F. J. Oles. A non well founded approach to terminological cycles. In *Proceedings of Tenth National Conference on Artificial Intelligence: AAAI 92*, pages 761–766, 1992.
- [19] J. Dörre and W. Rounds. On subsumption and semiunification in feature algebras. *J. Symbolic Computation*, 13:441–461, 1992.
- [20] R. Evans. Towards a formal specification for defaults in gpsg. In E. Klein and J. van Benthem, editors, *Categories, Polymorphism, and Unification*. Centre for Cognitive Science, Edinburgh, 1988.
- [21] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Science*, 18(2):194–211, 1979.
- [22] G. Gazdar, E. Klein, G. Pullum, and I. Sag. *Generalized Phrase Structure Grammar*. Harvard university Press, 1985.
- [23] G. Gazdar, G. Pullum, R. Carpenter, E. Klein, T. Hukari, and T. Levine. Category structures. *Computational Linguistics*, 14:1–19, 1988.
- [24] R. I. Goldblatt. Varieties of complex algebras. *Annals of Pure and Applied Logic*, 44:173–242, 1989.
- [25] C. A. Gunter and D. S. Scott. Semantic domains. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, volume B: Formal Models and Semantics*, chapter 12, pages 633–674. Elsevier, 1990.

- [26] Y. Gurevich. The word problem for certain classes of semigroups. *Algebra and Logic*, 5:25–35, 1966.
- [27] L. Haegeman. *Introduction to government and binding theory*. Basil Blackwell, 1991.
- [28] D. Harel. Dynamic logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic, vol. II*. Reidel, 1984.
- [29] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the Association for Computing Machinery*, 32(1):137–161, 1985.
- [30] M. Höhfeld and G. Smolka. Definite relations over constraint languages. Technical Report LILOG-REPORT 53, IBM Deutschland, Stuttgart, 1988.
- [31] N. Immerman. Relational queries computable in polynomial time. In *Proceedings of the 14th ACM Symposium on the Theory of Computing*, pages 147–152, 1982.
- [32] D. Johnson, A. Meyers, and L. Moss. A unification-based parser for relational grammar. In *Proc. 31st Annual Meeting of the Association for Computational Linguistics*, pages 97–104, 1993.
- [33] D. Johnson and L. Moss. Some formal properties of stratified feature grammars. *Annals of Mathematics and Artificial Intelligence*, 8, 1993.
- [34] D. E. Johnson and P. M. Postal. *Arc Pair Grammar*. Princeton University Press, 1980.
- [35] M. Johnson. *Attribute-Value Logic and the Theory of Grammar*. Center for Study of Language and Information, 1988.
- [36] M. Johnson. Features and formulae. *Computational Linguistics*, 17:131–152, 1991.
- [37] M. Johnson. Logic and feature structures. In *Proceedings of IJCAI 91*, pages 992–996, 1991.
- [38] A. K. Joshi and Yves Schabes. Tree adjoining grammars and lexicalized grammars. In *Tree Automata and Languages*. Elsevier, 1992.
- [39] R. Kaplan and J. Bresnan. Lexical-functional grammar: A formal system for grammatical representations. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, 1982.
- [40] R. Kasper and W. Rounds. A logical semantics for feature structures. In *Proc. of 24th Meeting of the Association for Computational Linguistics*, pages 257–266, 1986.
- [41] R. Kasper and W. Rounds. The logic of unification in grammar. *Linguistics and Philosophy*, 13:33–58, 1990.
- [42] M. Kay. Functional grammar. In C. Chiarello, editor, *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society*, pages 142–158, 1979.
- [43] M. Kracht. On the logic of category definitions. *Computational Linguistics*, 15:111–113, 1989.
- [44] H. Lewis. Complexity results for classes of quantificational formulae. *Journal of Computer and System Science*, 21:317–353, 1980.
- [45] J. McCarthy. Circumscription – a form of non-monotonic reasoning. In M. Ginsberg, editor, *Readings in Nonmonotonic Reasoning*, pages 145–152. Morgan Kaufman, 1987.
- [46] R. Montague. The proper treatment of quantification in ordinary english. In R. Thomason, editor, *Formal Philosophy: Selected Writings of Richard Montague*, pages 247–270. Yale University Press, 1974.
- [47] M. A. Moshier. *Extensions to Unification Grammar for the Description of Programming Languages*. PhD thesis, University of Michigan, 1988.
- [48] M. A. Moshier. Completeness theorems for logics of feature structures. *Annals of Mathematics and Artificial Intelligence*, 8, 1993.
- [49] M. A. Moshier and W. Rounds. A logic for partially specified data structures. In *Proceedings of 14th ACM Symposium on Principles of Programming Languages*, 1987.
- [50] L. Moss. Completeness theorems for logics of feature structures. In Y. Moschovakis, editor, *Proceedings of MSRI Workshop on Logic from Computer Science*. Springer-Verlag, 1991.
- [51] K. Mukai. A system of logic programming for linguistic analysis. Technical Report TR-540, ICOT, Tokyo, 1990.
- [52] K. Mukai. Clp(afa): Coinductive semantics of horn clauses with compact constraints. In J. Gawron, G. Plotkin, and S. Tutiya, editors, *Situation Theory and its Applications, vol. 2*,

- pages 179–214. Center for Study of Language and Information, 1991.
- [53] B. Nebel and G. Smolka. Representation and reasoning with attributive descriptions. In K. H. Bläsius, U. Hedstüch, and C-R. Rollinger, editors, *Sorts and Types in Artificial Intelligence*, pages 112–139. Springer-Verlag Lecture notes in Artificial Intelligence 418, 1989.
  - [54] F. Pereira and S. Shieber. The semantics of grammar formalisms seen as computer languages. In *Proceedings of 10th International Conference on Computational Linguistics: COLING 84*, 1984.
  - [55] F. Pereira and D. H. D. Warren. Definite clause grammars for language analysis: A survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278, 1980.
  - [56] D. Perlmutter. *Studies in Relational Grammar I*. University of Chicago Press, 1983.
  - [57] C. Pollard and M. A. Moshier. Unifying partial descriptions of sets. In P. Hansen, editor, *Vancouver Studies in Cognitive Science: vol. I*. University of British Columbia Press, 1990.
  - [58] C. Pollard and I. Sag. *Information-Based Syntax and Semantics: Volume I - Fundamentals*. CSLI Lecture Notes 13, Chicago University Press, 1987.
  - [59] P. M. Postal. *Studies of Passive Clauses*. State University of New York Press, 1986.
  - [60] P. M. Postal and B. Joseph, editors. *Studies in Relational Grammar, III*. University of Chicago Press, 1990.
  - [61] V. Pratt. Semantical considerations on floyd-hoare logic. In *Proceedings of the 17th IEEE Symposium on Foundations of Computer Science*, 1976.
  - [62] A. Prior. *Past, Present, and Future*. Oxford University Press, 1967.
  - [63] M. Reape. *Introduction to Semantics of Unification-based Grammar Formalisms*. Kluwer, 1994.
  - [64] R. Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Math.*, 12:177–209, 1971.
  - [65] W. Rounds and A. Manaster Ramer. A logical version of functional grammar. In *Proceedings of the 25th Annual Conference of the Association for Computational Linguistics*, 1987.
  - [66] Dana S. Scott. Domains for denotational semantics. In *Lecture Notes in Computer Science 140*, 1982.
  - [67] S. Shieber. The design of a computer language for linguistic information. In *Proceedings of 12th COLING*, pages 211–215, 1986.
  - [68] S. Shieber. *Parsing and Type Inference for Natural and Computer languages*. PhD thesis, Stanford University, 1989.
  - [69] S. Shieber. *Constraint-Based Grammar Formalisms: Parsing and Type Inference for Natural and Computer Languages*. MIT Press, 1992.
  - [70] G. Smolka. Feature constraint logics for unification grammars. *Journal of Logic Programming*, 12:51–87, 1992.
  - [71] Gaisi Takeuti. *Proof Theory*. North Holland, 1987.
  - [72] M. Vardi. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on the Theory of Computing*, pages 137–146, 1982.
  - [73] S. Vickers. *Topology via Logic*. Cambridge University Press, 1989.
  - [74] K. Vijay-Shanker. *A Study of tree-adjoining grammars*. PhD thesis, University of Pennsylvania, 1987.
  - [75] M. Young. Non-monotonic sorts for feature structures. In *Proceedings of Tenth National Conference on Artificial Intelligence: AAAI 92*, pages 596–601, 1992.
  - [76] Guo-Qiang Zhang. *Logic of Domains*. Birkhauser, Boston, 1991.

$$\left[ \begin{array}{l} \text{vp} \\ \text{AGR } \boxed{1} \\ \text{SUBJ } \boxed{1} \end{array} \left[ \begin{array}{l} \text{NUM } \text{sing} \\ \text{PERS } \text{3rd} \end{array} \right] \right]$$

Figure 1. A feature matrix.

$$\left[ \begin{array}{l} [\textit{Cat}] \textit{ S} \\ [1] \textit{ Joe} \\ [\textit{Head}]\textit{breaks} \\ [2] \textit{ the glass} \end{array} \right]$$

Figure 2. *Joe breaks the glass.*

$$\left[ \begin{array}{l} [\textit{Cat}] \textit{ S} \\ [2, 1] \textit{ glass} \\ [\textit{Head}]\textit{breaks} \end{array} \right]$$

Figure 3. *Glass breaks.*

$$\left[ \begin{array}{l} [\textit{Cat}] \textit{ S} \\ [1] \textit{ Joe} \\ [\textit{Head}]\textit{gave} \\ [3, 2] \textit{ Mary} \\ [2, 8] \textit{ tea} \end{array} \right]$$

Figure 4. *Joe gave Mary tea.*

$$\left[ \begin{array}{l}
 [Cat] \ S \\
 [0, 1] \ (x) \ Mary \\
 [Head] \ was \\
 [Comp] \ \left[ \begin{array}{l}
 [Cat] \ VP \\
 [Head] \ given \\
 [3, 2, 1, 0](x) \\
 [2, 8] \ tea \\
 [1, 8, 0] \ (y) \\
 [0, 8] \ \left[ \begin{array}{l}
 [0, Cat] \ PP \\
 [0, Flag] \ by \\
 [0, Marked](y) \ Joe
 \end{array} \right]
 \end{array} \right]
 \end{array} \right]$$

Figure 5. *Mary was given tea by Joe.*