A Family of Decidable Feature Logics which Support HPSG-Style Set and List Constructions[†]

Stephen J. Hegner

Department of Computer Science and Electrical Engineering
Votey Building
University of Vermont
Burlington, VT 05405 USA
hegner@emba.uvm.edu

Abstract. A desirable goal of constraint-based parsing is that the whole process should be one of pure algorithmic constraint satisfaction; the implementor should not need to specify any control information, or be aware of how the underlying system implements control. Unfortunately, most existing tools are Turing complete, and hence require additional control information by virtue of their computational power. In this work, a first step towards automatic cosntraint-based parsing of HPSG is provided, in the form of a family of decidable (for satisfiability) logics in which set and list constructions may be expressed in a uniform fashion, and constraints such as the Nonlocal Feature Principle may be recaptured succinctly.

1 Introduction

1.1 Motivation and Overview

In [22, p. 10], it is argued that parsing of (at least a sizeable fragment of) Head-Driven Phrase-Structure Grammar (hereafter HPSG) should be decidable. Under the assumption of decidability, parsing may be performed via purely algorithmic *constraint satisfaction.* The user provides only declarative constraints, and need not supply any control information. In practice, the situation is not so ideal. Existing tools for working with feature logics in general, and HPSG in particular, such as ALE [1], CUF [5], and TFS [24], are *Turing complete*, meaning that they are general enough to allow representation of any computational process, including undecidable ones. While some simple problems may be solved within these frameworks without the specification of control information, this is not the case for many of the essential constraints of HPSG. Thus, it is up to the user, at least in part, to supply parts of the parsing algorithm (or at least to respect how the underlying system implements control).

[†] Portions of this work were prepared while the author was a visiting research fellow at the Department of Informatics, University of Bergen, supported by a grant from the Norwegian Research Council,.

In the work reported here, as a step towards fully automatic parsing of HPSG, a family of decidable feature logics which support some of the key constructs of HPSG is presented. These logics are focused particularly upon the constructs which require sets and/or lists, and have two notable features which distinguish them from other efforts known to the author.

Sets and lists are treated uniformly, using the same construction, and duplicate values are supported. In HPSG, both sets and lists occur in fundamental ways. For example, in filler-gap constructions, the SLASH binding feature consists of an *unordered* multiset of structures, i.e., a set in which duplicates are allowed. While there is no order structure on the elements of the multiset, the possibility that several of the elements will be identical can occur, and must be supported in any formalism. On the other hand, the value of COMP-DTRS is always a (totally ordered) *multilist* of signs, i.e., a list in which the same element may occur more than once. Since multisets and multilists are are variations of the same idea (a collection of objects with order structure), it seems natural that they should be represented using similar constructs. Among other benefits, this would imply that multisets and multilists will be represented by feature structures of similar size and depth, and corresponding operations on multisets and multilists will be of the same complexity. Furthermore, partially ordered sets, should they be needed, may easily be introduced.

The model structure of the associated logic is one of a traditional feature structure, augmented to support multicollections. The sentences necessary to represent (set-oriented) binding dependencies such as the *nonlocal feature principle*, as well as list-oriented dependencies such as the *subcategorization principle*, are expressible. In particular, operations such as union and disjoint union, as well as list concatenation, are expressible. To the best of the author's knowledge, this is the only work to treat multisets and lists uniformly within a decidable feature-structure framework. Manandhar [17], Carpenter [3], and Moshier and Pollard [19] have all developed elegant decidable feature logics which supports set constructions. However, neither supports lists as a fundamental type. In [14], Kepser gives a proof that the feature logic of King [15] is decidable, but this logic does not provide any special constructs for sets or lists; rather, lists must constructed as trees of features, with sets taken to be equivalence classes of lists.

Decidability is proven by mapping the logic into a sorted first-order logic, and then using variants of well-known decidability results in that context. By using this approach, the great wealth of knowledge regarding decidability of first-order theories [6] may be brought to bear on the problem. The approach used in this work employs a typed generalization of the Schönfinkel-Bernays quantificational class (no $\exists$ quantifier occurs within the scope of a $\forall$ quantifier) to many-sorted logics. The types of the logic are constructed by forming a cartesian product of the underlying types of HPSG (called *sorts* in [22]) and a group of three special types used to distinguish ordinary values from multicollections. Roughly speaking, in this generalization, whenever a quantifier ordering of the form $(\forall x)(\exists y)$ occurs, the types of x and y must be such that infinite recursive construction of terms is impossible, and so the

Herbrand universe remains finite. It is shown that the desired class of HPSG constraints may be expressed within this class.

1.2 Prerequisites and Scope

It is assumed that the reader has a reasonable knowledge of order structures [4] first-order logic in general and the expansion theorem in particular [16, Ch, 9], and many-sorted logics [8], as well as some acquaintance with notation for regular languages [16, Ch. 2], and the logical representation of feature structures [7]. For an understanding of the examples, as well as for the motivation for this work, some acquaintance with HPSG [21], [22] would prove very helpful.

Because of space constraints, it has been necessary to limit background material, condense the number and scope of examples, and to limit proofs to sketches of the techniques.

2 Basic Concepts

2.1 Multicollections and Multicollection-Extended Feature Structures

2.1.1 Partially ordered sets. Partially ordered sets (posets) will usually be represented by boldface roman letters, with the underlying set denoted by the corresponding non-bold roman letter. Unless otherwise stipulated, the associated order is denoted by $\leq$. For a set L, $\mathsf{Poset}(L)$ denotes the set of all posets whose underlying set is a subset of L. Consult [4] for more comprehensive information.

2.1.2 Multicollections. The notion of a *multiset*, i.e., a set in which an element may occur several times, is well known [23]. A *multilist* is defined similarly. The construct which we employ to recapture multisets and multilists is termed a *multicollection*, which is a set S which is indexed by a poset of *tags*. Formally, *multicollection* is a triple $\mathbf{C} = (S, \mathbf{P}, f)$ in which S is a set, called the *base set*, $\mathbf{P}$ is a poset, called the *tag set*, and $f : P \to S$ is a surjective function, called the *tagging function.* $\mathbf{C}$ is a *multiset* (resp. *multilist*) if $\mathbf{P}$ is a trivial partial order (resp. total order).

As a simple example, let $S = \{a, b, c\}$, let $P = \{x_1, x_2, x_3, x_4\}$, and let $f : P \to S$ be defined by $x_1 \mapsto a$, $x_2 \mapsto b$, $x_3 \mapsto c$, and $x_4 \mapsto b$. Then, with $\leq$ the trivial partial order in which $x_i \leq x_j \Rightarrow i = j$, $\mathbf{C}$ is the multiset $\{a, b, c, b\}$ in which b occurs twice, and with $\leq$ the partial order generated by $x_1 \leq x_2 \leq x_3 \leq x_4$, then $\mathbf{C}$ is the multilist $[a\ b\ c\ b]$, in which b occurs in both the second and in the fourth position. With $\leq$ the ordering generated by $x_1 \leq x_2$ and $x_1 \leq x_3$, a multicollection which is neither a multiset nor a multilist is obtained.

2.1.3 Multicollection contexts and systems. In this work, multicollections will live at certain nodes of a feature structure, and the multicollections at distinct nodes must be related to each other in a particular way. Specifically, the set of tags must be global, although the ordering relationship among the tags may be local. To recapture this idea, the following notion is employed. A *multicollection context* is a triple $\mathbf{K} = (L, S, I)$ of sets in which L is the *global tag set*, S is the *object set*, and I is the *context set.* A multicollection system over $\mathbf{K}$ is a set of multicollections whose tagging functions agree whenever they overlap. In other words, in a multicollection system, the tag alone determines the associated member of the base set; it does not matter which of the multicollections in the system is considered. Formally, a *multicollection system* over $\mathbf{K}$ is a pair $\mathbf{D} = (\gamma, \eta)$ with $\gamma : L \to S$ a total function, called the *global tagging function*, and $\eta : I \to \mathsf{Poset}(L)$ a total function, called the *context function.* The global tagging function gives the *global* association (i.e., over all multicollections in the system) of tags to elements, while the context function gives the *local* ordering within each multicollection. Thus, while tagging is global to the entire system, ordering is local to the particular multicollection.

As an abuse of notation, we sometimes use the notation $\eta(I)$ to denote the underlying set of the poset. Context will always make it clear which is meant, the entire poset or just the underlying set. The multicollection system $\mathbf{D}$ defines the family $\mathbf{Sys}(D) = \{\mathbf{C}_i = (S_i, \mathbf{P}_i, f_i) \mid i \in I\}$ of multicollections in which $\mathbf{P}_i = \eta(i)$, $S_i = \gamma(P_i)$, and $f_i = \gamma_{|P_i}$. (The notation $\gamma_{|P_i}$ identifies a function with the same action as γ, but with its domain restricted to P_i.)

These rather involved constructs are best understood within the context of of a multicollection-extended feature structure (2.1.5 below), and so an example which illustrates these ideas is deferred until that point.

2.1.4 Feature contexts. A feature context is just a set of parameters which underlie the feature structures of a given context. Formally, an *untyped feature context* is a pair $\mathcal{C} = (F, A)$ in which F is a finite set, called the set of *features*, and A is a finite set, called the set of *atoms.* As a notational convention, throughout the rest of this section, an untyped feature context $\mathcal{C} = (F, A)$ is fixed.

2.1.5 Multicollection-extended feature structures. Within the current literature, there are several different formalisms for recapturing feature structures. A popular one is the so-called *Kasper-Rounds* [13] representation, in which the feature structure is modelled as a finite-state automaton. An extension of the Kasper-Rounds formalism which integrates multicollections into the structures is employed in this work. Formally, a *multicollection-extended feature structure* (*MEFS* for short) is an eight-tuple $M = (Q, \delta, \alpha, q_o, L, C, \gamma, \eta)$ in which:

(mefs-i) Q is a finite set, called the set of *object states.*

(mefs-ii) C is a finite set, called the set of *multicollection states.*

(mefs-iii) L is a finite set, called the set of *object indices.*

(mefs-iv) $\delta : Q \times F \to Q \cup C$ is a partial function, called the *state-transition function.*

(mefs-v) $\alpha : Q \to A$ is an injective partial function, called the *constant-assignment function.*

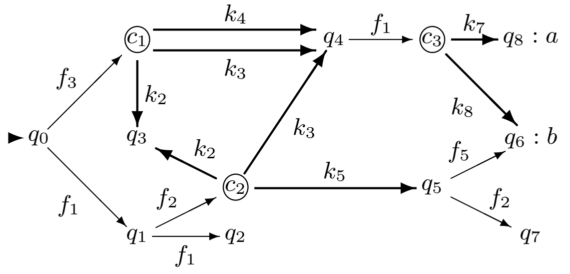
(mefs-vi) (γ, η) is a multicollection system over (L, Q, C).

(mefs-vii) $q_o \in Q$ is called the *initial state.*

(mefs-viii) The sets Q, C, L, and F are pairwise disjoint.

(mefs-ix) Given $q \in Q$, if $\alpha(q)\downarrow$, then $\delta(q,e)\uparrow$ for all $e \in F$. (Notation: $f(x)\downarrow$ (resp. $f(x)\uparrow$) means that $f(x)$ is defined (resp. undefined)).

Figure 1 depicts an example, which should help to clarify these ideas, as well as those of 2.1.3. There are two flavors of states, *object states* (the q_i's



Order information at collection nodes:
At c_1: $k_4 \leq k_2 \leq k_3$.
At c_2: No nontrival ordering.
At c_3: $k_7 \leq k_8$.

Fig. 1. An Example MEFS

in the example) and *multicollection states* (the c_i's in the example, circled for emphasis), and two types of edges, *feature edges* (labelled with members of the underlying set F of features, f_i's in the example) and *tag edges* (shown in bold lines, and labelled with object indices, k_i's in the example). Each object state is similar to a state in an ordinary feature structure, while each multicollection state is the root of a multicollection. The feature edges are defined by the transition function δ, the definition of tag edges involves the multicollection system, and will be addressed shortly. The root node is q_0 in the example and is labelled by the short ingoing arrowhead.

At a given multicollection node c, the base set of the multicollection is just the set of all $q \in Q$ such that there is an edge from c to q. The tag set is the set of all $k \in L$ such that there is an edge from c with label k. The tagging function identifies the association of tag edge labels to the states at the end of the edge. In the example, at c_1, the base set is $\{q_3, q_4\}$, the tag set is $\{k_2, k_3, k_4\}$, and the tagging function assigns $k_2 \mapsto q_3$, $k_3 \mapsto q_2$, and $k_3 \mapsto q_4$. The ordering on the elements of the tag set is specified separately, to the right of the graph; in this case $k_4 \leq k_2 \leq k_3$.

A critical property of MEFS's is that *sinks of tag edges are global values*; that is, all tag edges labelled with the same tag name terminate at the same node. Thus, the k_3 edge emanating from c_1 and that emanating from c_2 **must** point to the same node. This is recaptured succinctly within the definition of a multicollection system (2.1.3); the global tagging function is γ, with each local tagging function a restriction of it. On the other hand, note that *tag ordering is local*; for example, $k_2 \leq k_3$ at c_1, while k_2 and k_3 are incomparable at c_2. Indeed, the multicollections at c_1 and c_3 are multilists, while that at c_2 is a multiset.

The sink of a tag edge must be an object node, so that multicollections cannot be nested directly. However, an element of a multicollection may have a feature which identifies another multicollection (e.g., f_1 from q_4 to c_3). This simplifies the mathematical aspects, and does not seem to impose any major roadblocks in knowledge representation.

$\mathsf{MEFS}(\mathcal{C})$ denotes the set of all MEFS's over $\mathcal{C}$.

2.1.6 The extended transition function. In the classical Kasper-Rounds formalism, the extended transition function is just the function of that name which is derived from the associated automaton. In the case of an MEFS, the definition is somewhat more complex, because the function γ of the multicollection system embedded within the MEFS is used to define transitions out of tag nodes. Formally, let $M = (Q, \delta, \alpha, q_o, L, C, \gamma, \eta)$ be a MEFS. Define the *extended transition function* for M to be the partial function $(\delta+\gamma)^* : (Q \cup C) \times (F \cup L)^* \to Q \cup C$ given by the following conditions.

(i) For each $q \in Q \cup C$, $(\delta + \gamma)^*(q, \epsilon) = q$. ($\epsilon$ denotes the empty string.)

(ii) For each $q \in Q$, $f \in F$, $(\delta + \gamma)^*(q, f)\downarrow$ iff $\delta(q, f)\downarrow$; $(\delta + \gamma)^*(q, f) = \delta(q, f)$ in this case.

(iii) For each $c \in C$, $k \in L$, $(\delta + \gamma)^*(c, k)\downarrow$ iff $k \in \eta(c)$; $(\delta + \gamma)^*(c, k) = \gamma(k)$ in this case.

(iv) For each $q \in Q$ and $k \in L$, $(\delta + \gamma)^*(q, k)\uparrow$.

(v) For each $c \in C$, $f \in F$, $(\delta + \gamma)^*(c, f)\uparrow$.

(vi) For any $q \in Q \cup C$, $\omega \in (F \cup L)^*$, $b \in F \cup L$, $(\delta+\gamma)^*(q, \omega \cdot b)\downarrow$ iff $(\delta+\gamma)^*(q, \omega)\downarrow$ and $\delta((\delta + \gamma)^*(q, \omega), b)\downarrow$, and then $(\delta + \gamma)^*(q, \omega \cdot b) = \delta((\delta + \gamma)^*(q, \omega), b)$.

Notice that the definition of $(\delta + \gamma)^*$ does not depend upon the ordering of elements in the multicollection. In terms of the example of Fig. 1, $(\delta + \gamma)^*$ may be determined from the graph alone, without any reference to the ordering table to its right. The string $\rho \in (F \cup L)^*$ is an *actual path* for M if $(\delta + \gamma)^*(q_o, \rho)\downarrow$. The set of all actual paths for M is denoted $\mathsf{ActPath}(M)$.

2.2 Typed Multicollection-Extended Feature Structures

HPSG is founded upon typed feature structures. Thus, to recapture constraints within that framework, it is imperative that the notion of an MEFS be extended to a typed domain. While it is often the case that one starts with *partial* type systems, an adequate theory generally requires that it be extensible to a total one. For details on conditions under which such extensions are possible, consult [9]. In this report, we shall confine attention to the situation in which the order structure is total.

2.2.1 Bounded semilattices and type hierarchies. A common framework for modelling type hierarchies, and the one appropriate for HPSG, is that of a *bounded meet semilattice*, which is a quadruple $\mathbf{T} = (T, \bot, \top, \sqcap)$ in which T is a set, called the *underlying set*; $\sqcap : T \times T \to T$ is a total function, called the *meet operator*, which is associative, commutative, and idempotent; $\bot \in T$ (resp. $\top \in T$) is called the *least element*, (resp. *greatest element*); $\bot \sqcap x = \bot$ and $\top \sqcap x = x$ for all $x \in T$. The symbol $\sqsubseteq$ is used to denote the order relation induced by the semilattice. For more information on semilattices, consult [4].

For the purposes of this paper, a *type hierarchy* is a bounded, finite meet semilattice. Throughout the rest of this paper, we let $\mathbf{T}$ be a type hierarchy.

2.2.2 Typed feature contexts and typed MEFS's. A typed feature context plays the same role in the definition of typed MEFS's that an ordinary feature context plays in the definition of ordinary MEFS's. Formally, a *typed feature context* is a quadruple $\mathcal{C} = (F, A, \mathbf{T}, \mu)$, in which F is a finite set, called the set of *features*; A is a finite set, called the set of *atoms*; $\mathbf{T}$ is a type hierarchy; and $\mu : A \to T \setminus \{\bot\}$ is a total function, called the *constant typing function.*

The definition of a typed MEFS provided here is patterned after that of Carpenter [2], which details the extension of the Kasper-Rounds representation to the typed context. Formally, let $\mathcal{C}$ be a typed feature context over $\mathbf{T}$. A *typed MEFS* over $\mathcal{C}$ is a nine-tuple $M = (Q, \delta, \alpha, q_o, L, C, \gamma, \eta, \theta)$ in which

(tmefs-i) $(Q, \delta, \alpha, q_o, L, C, \gamma, \eta)$ is an MEFS.

(tmefs-ii) $\theta : Q \cup C \cup L \to T$ is a total function, called the *typing function.*

Thus, object states (Q), multicollection states (C), and object indices (L) all have type. This typing function is subject to the following constraints.

(tmefs-iii) For all $c \in C$ and $k \in L$, $k \in \eta(C)$ implies that $\theta(k) \sqsubseteq \theta(c)$.

(tmefs-iv) For all $k \in L$ and $q \in Q$, $\gamma(k) = q$ implies that $\theta(q) \sqsubseteq \theta(k)$.

(tmefs-v) For all $q \in Q$, $\alpha(q)\downarrow$ implies that $\theta(q) \sqsubseteq \mu(\alpha(q))$.

The collection of all typed MEFS's over $\mathcal{C}$ is denoted $\mathsf{TMEFS}(\mathcal{C})$. The *extended transition function* $(\delta+\gamma)^*$ for a typed MEFS is defined exactly as for an untyped MEFS.

2.3 The Logic of Typed MEFS's

2.3.1 The logical sort system of a type hierarchy. The *base sort system* for MEFS's is the triple $\mathsf{BaseSort} = \{\mathsf{Obj}, \mathsf{Coll}, \mathsf{Tag}\}$. The *syntactic sort system generated by* $\mathbf{T}$, denoted $\mathbf{CESort}(\mathbf{T}) = (\mathsf{CESort}(\mathbf{T}), \bot, \top, \sqcap)$, is the meet semilattice defined by the Cartesian product $((T \setminus \{\bot\}) \times \mathsf{BaseSort}) \cup \{\top, \bot\}$. The meet operator $\sqcap$ is defined by $(t_1, s_1) \sqcap (t_2, s_2) = (t_1 \sqcap t_2, s_1)$ if both $t_1 \sqcap t_2 \neq \bot$ and $s_1 = s_2$; otherwise $(t_1, s_1) \sqcap (t_2, s_2) = \bot$. The symbols $\bot$, $\top$, and $\sqcap$ have double duty, associated with both the underlying type hierarchy $\mathbf{T}$ and the syntactic sort system $\mathbf{CESort}(\mathbf{T})$. Similarly, since $(t, s) \sqcap \top = (t, s)$ for any $(t, s) \in \mathbf{T} \setminus \{\bot\} \times \mathsf{BaseSort}$, $(\top, s)$ will often be abbreviated to s; e.g., $\mathsf{Tag} = (\top, \mathsf{Tag})$. This should cause no confusion, since context will always make clear which interpretation is correct. Also, $\sqsubseteq$ is used to denote the order relation on $\mathbf{CESort}(\mathbf{T})$, as well as the order relation on $\mathbf{T}$.

2.3.2 Variable contexts and typed MEFS feature contexts. An *MEFS variable context* over $\mathbf{T}$ is a $\mathsf{CESort}(\mathbf{T}) \setminus \{\top, \bot\}$-indexed set $\mathcal{V} = \{\mathbf{V}_r \mid r \in \mathsf{CESort}(\mathbf{T}) \setminus \{\top, \bot\}\}$ of variables such that, for $r_1 \neq r_2$, $\mathbf{V}_{r_1} \cap \mathbf{V}_{r_2} = \emptyset$.

To aid in recognizing the type of a variable, the following convention is used. For $t \in T$, variables in $\mathbf{V}_{(t,\mathsf{Obj})}$ are usually written as lowercase letters from the end of the alphabet, with $t \in T$ written as a superscript. Subscripts may be used to distinguish variables, if necessary. Examples include x^t, y^t, z^t_1, and z^t_2. A similar convention is used for variables in $\mathbf{V}_{(t,\mathsf{Coll})}$, except that uppercase letters are used. Examples include X^t, Y^t, Z^t_1, and Z^t_2. Variables in $\mathbf{V}_{(t,\mathsf{Tag})}$ are represented by the special letter ℓ, with subscript and superscript conventions as in the other two cases. Examples include ℓ^t, ℓ^t_1, and ℓ^t_2. In each of these cases, if $t = \top$, then we may omit the superscript entirely, and write, e.g., x, X_1, or ℓ.

For $s \in \mathsf{BaseSort}$, $\mathbf{V}_s$ denotes the set $\bigcup\{\mathbf{V}_{(t,s)} \mid t \in T \setminus \{\bot\}\}$. The symbol $\mathbf{V}$ denotes $\mathbf{V}_{\mathsf{Obj}} \cup \mathbf{V}_{\mathsf{Coll}} \cup \mathbf{V}_{\mathsf{Tag}}$. When we have a variable of an unspecified type, that is, an element of $\bigcup \mathcal{V}$, the Greek letter ν, possibly with a subscript, will be used to represent it.

A *typed MEFS context with variables* is a pair $\mathcal{K} = (\mathcal{C}, \mathcal{V})$ in which $\mathcal{C} = (F, A, \mathbf{T}, \mu)$ is a typed feature context and $\mathcal{V}$ is a MEFS variable context over $\mathbf{T}$. Throughout the rest of this paper, unless noted to the contrary, we let $\mathcal{K} = (\mathcal{C}, \mathcal{V})$ be a typed MEFS context with variables.

2.3.3 Description paths and feature terms. Description paths generalize the feature terms of the Kasper-Rounds framework. The generalization must account for both the edge labelling and the order structure of the embedded multicollections. It is important to note that while feature names are part of the underlying language (as embodied in the underlying typed MEFS context $\mathcal{K}$), tag names are not. The language of description paths uses tag variables (members of $\mathbf{V}_{\mathsf{Tag}}$) within paths to represent tag edges; these variables must then be bound by the logical expression in which the description path is used.

A *simple description path* (or just *SD path*) over $\mathcal{K}$ is any element of the regular set $(F \cdot (\mathbf{V}_{\mathsf{Tag}} + \epsilon))^*$. A simple description path may be thought of as a path through a feature structure, but with tags replaced by tag variables. The restrictions that two tags may not occur in a row, and that the root of a feature structure must be an object state, are built into the regular expression. For example, $f_3\ell_1 f_1\ell_2$ is a simple description path which "fits" the example of Fig. 1, in the sense that upon substituting k_4 for ℓ_1 and k_7 for ℓ_2, an actual path of the MEFS is obtained.

A *description path* over $\mathcal{K}$ is a simple description, followed by an optional *terminator*. As in the Kasper-Rounds formalism, a terminator may be of the form $:x$ or $:a$, in which $x \in \mathbf{V}_{\mathsf{Obj}}$ and $a \in A$. Thus, $f_3\ell_1 f_1\ell_2 : x$ and $f_3\ell_1 f_1\ell_2 : a$ are description paths. For a simple description path which ends with an element of F, a terminator of the form $: X$ is also permitted, with $X \in \mathbf{V}_{\mathsf{Coll}}$. Thus, $f_3\ell_1 f_1 : X$ is a description path, but $f_3\ell_1 f_1\ell_2 : X$ is not. Finally, for a simple description path ending in an element of F, a terminator may be of the form $[\ell_1 \leq \ell_2]$ is also permitted, so that $f_3\ell_1 f_1[\ell_2 \leq \ell_3]$ is a description path. The set of all description paths over $\mathcal{K}$ is denoted $\mathsf{DPath}(\mathcal{K})$.

Informally, the set of feature terms is the closure of the set of description paths under the usual logical connectives. Formally, the set of *typed collection extended feature terms*, denoted $\mathsf{FT}(\mathcal{K})$, is the smallest set such that $\{\top, \bot\} \cup \mathsf{DPath}(\mathcal{K}) \subseteq \mathsf{FT}(\mathcal{K})$, and whenever $\varphi_1, \varphi_2 \in \mathsf{FT}(\mathcal{K})$, then $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $(\neg\varphi_1) \in \mathsf{FT}(\mathcal{K})$ as well. As per usual mathematical conventions, parentheses may be dropped in feature terms when no confusion can result, so one may write terms such as $(\varphi_1 \wedge \neg\varphi_2 \wedge \varphi_3)$, for example.

2.3.4 Assignments. Let $M = (Q, \delta, \alpha, q_o, L, C, \gamma, \eta, \theta) \in \mathsf{TMEFS}(\mathcal{C})$. A $(\mathcal{K}, M)$-assignment associates with each variable an object of the appropriate sort. More formally, a $(\mathcal{K}, M)$-*assignment* is a function $\beta : \mathbf{V} \to Q \cup C \cup L$ with the property that $\beta(\mathbf{V}_{\mathsf{Obj}}) \subseteq Q$; $\beta(\mathbf{V}_{\mathsf{Coll}}) \subseteq C$; $\beta(\mathbf{V}_{\mathsf{Tag}}) \subseteq L$; and for each $t \in T \setminus \{\bot\}$ and $s \in \mathsf{BaseSort}$, $\theta(\beta(\mathbf{V}_{(t,s)})) \sqsubseteq t$. $\mathsf{Asgn}(\mathcal{K}, M)$ denotes the set of all $(\mathcal{K}, M)$-assignments.

As noted above, description paths may not contain tags; they may only contain variables with tag types. The application of an assignment generates a "true" path through a typed MEFS by replacing tag variables with true tags. Given $\rho \in \mathsf{SDPath}(\mathcal{K})$, $\beta\langle\rho\rangle$ denotes the string which is obtained by replacing each variable $\nu \in \mathbf{V}_{\mathsf{Tag}}$ occurring in ρ with $\beta(\nu)$. For example, if $\rho = f_3\ell_1 f_1[\ell_2 \leq \ell_3]$ and β is any assignment which maps $\ell_1 \mapsto k_4$, $\ell_2 \mapsto k_7$, and $\ell_3 \mapsto k_8$, then $\beta\langle\rho\rangle = f_3 k_4 f_1[k_7 \leq k_8]$, a "ground term" (not a legal feature term) which may be interpreted as true or false in a given structure (true in the case of Fig. 1).

2.3.5 Satisfiability. Let $M = (Q, \delta, \alpha, q_o, L, C, \gamma, \eta, \theta) \in \mathsf{TMEFS}(\mathcal{C})$. Informally, for a given feature term ρ, $\mathsf{Sat}(M, \rho)$ is the set of all truth assignments β for which $\beta\langle\rho\rangle$ is a "ground term" which is interpreted as true in M. For example, with M as given in Fig. 1 and ρ and β as defined at the end of 2.3.4 above, $\beta \in \mathsf{Sat}(M, \rho)$, while any β' which does not satisfy either $\ell_1 \mapsto k_4$ or $\ell_1 \mapsto k_3$ is not in $\mathsf{Sat}(M, \rho)$. Formally, for any feature term φ, $\mathsf{Sat}(M, \varphi)$ is a subset of $\mathsf{Asgn}(\mathcal{K}, M)$, defined in cases as follows.

(i) $\mathsf{Sat}(M, \rho) \Leftrightarrow \beta\langle\rho\rangle \in \mathsf{ActPath}(M)$.

(ii) $\mathsf{Sat}(M, \rho{:}\nu) \Leftrightarrow (\beta\langle\rho\rangle \in \mathsf{ActPath}(M)$ and $(\delta + \gamma)^*(q_o, \beta\langle\rho\rangle) = \beta(\nu))$.

(iii) $\mathsf{Sat}(M, \rho{:}a) \Leftrightarrow (\beta\langle\rho\rangle \in \mathsf{ActPath}(M)$ and $\alpha((\delta + \gamma)^*(q_o, \beta\langle\rho\rangle)) = a)$.

(iv) $\mathsf{Sat}(M, \rho[\ell_1 \leq \ell_2]) \Leftrightarrow (\beta\langle\rho\rangle \in \mathsf{ActPath}(M)$ and $(\delta + \gamma)^*(q_o, \rho) \in C$ and $\beta(\ell_1) \leq_{\eta((\delta+\gamma)^*(q_o,\rho))} \beta(\ell_2))$.

(v) For $\varphi_1, \varphi_2 \in \mathsf{FT}(\mathcal{K})$, $\mathsf{Sat}(M, (\varphi_1 \wedge \varphi_2)) = \mathsf{Sat}(M, \varphi_1) \cap \mathsf{Sat}(M, \varphi_2)$.

(vi) For $\varphi_1, \varphi_2 \in \mathsf{FT}(\mathcal{K})$, $\mathsf{Sat}(M, (\varphi_1 \vee \varphi_2)) = \mathsf{Sat}(M, \varphi_1) \cup \mathsf{Sat}(M, \varphi_2)$.

(vii) For $\varphi \in \mathsf{FT}(\mathcal{K})$, $\mathsf{Sat}(M, (\neg\varphi)) = \mathsf{Asgn}(\mathcal{K}, M) \setminus \mathsf{Sat}(M, \varphi)$.

For a set $\Phi \subseteq \mathsf{FT}(\mathcal{K})$, define $\mathsf{Sat}(M, \Phi) = \bigcap\{\mathsf{Sat}(M, \varphi) \mid \varphi \in \Phi\}$.

2.3.6 Variable substitution. To formalize the action of quantifiers, it is necessary to formalize the idea of altering a $(\mathcal{K}, M)$-assignment on exactly one variable. Formally, let $M = (Q, \delta, \alpha, q_o, L, C, \gamma, \eta, \theta) \in \mathsf{TMEFS}(\mathcal{C})$, let $\beta \in \mathsf{Asgn}(\mathcal{K}, M)$, let $s \in \mathsf{BaseSort}$, and let $r \in Q \cup C \cup L$, subject to the constraint that if $s = \mathsf{Obj}$, then $r \in Q$; if $s = \mathsf{Coll}$, then $r \in C$; if $s = \mathsf{Tag}$, then $r \in L$. Then, for $t \in T \setminus \{\bot\}$ and $\nu_1 \in \mathbf{V}_{(t,s)}$, define $\beta[\nu_1 \leftarrow r] \in \mathsf{Asgn}(\mathcal{K}, M)$ by

$$\beta[\nu_1 \leftarrow r](\nu_2) = \begin{cases} r & \text{if } \nu_1 = \nu_2; \\ \beta(\nu_2) & \text{otherwise.} \end{cases}$$

provided that $\theta(r) \sqsubseteq t$. If $\theta(r) \not\sqsubseteq t$, then $\beta[\nu_1 \leftarrow r]$ is undefined.

2.3.7 Quantified feature terms. The quantification of feature terms proceeds in a manner virtually identical to that for many-sorted first-order logic. A *typed MEFS quantifier* is a symbol of the form $\forall_\tau$ or $\exists_\tau$, with $\tau \in \mathsf{CESort}(\mathbf{T}) \setminus \{\top, \bot\}$. The *sort* of the quantifier is r. As a notational abbreviation, for $s \in \mathsf{BaseSort}$, we will sometimes write $\forall_s$ (resp. $\exists_s$) for $\forall_{(\top,s)}$ (resp. $\exists_{(\top,s)}$).

A *typed MEFS quantifier term* is a string of the form $(Q\nu)$ in which Q is a typed MEFS quantifier and $\nu \in \mathbf{V}$, with Q and ν of the same sort. (In that which follows, the quantifier subscript may be dropped when the sort may be determined from the variable; thus, $(\forall_{(t,\mathsf{Obj})} x^t)$ may be abbreviated to $(\forall x^t)$, and $(\exists_{(\top,\mathsf{Tag})}\nu)$ may be abbreviated to $(\exists\nu)$. These conventions will be used in the sorted first-order logic of Sec. 3.1 as well.)

A *typed MEFS quantifier string* is a (possibly empty) sequence of typed MEFS quantifier terms. A typed MEFS quantifier string is *clean* if no variable in $\mathbf{V}$ occurs in more than one of its quantifier terms.

A *quantified typed feature term* is a string φ of the form $\xi\psi$, in which ξ is a clean quantifier string and ψ is a typed MEFS feature term. In this case, we call ξ the *prefix* of φ and denote it by $\mathsf{Prefix}(\varphi)$. Likewise, we call ψ the *matrix* of φ and denote it by $\mathsf{Matrix}(\psi)$. The set of all quantified feature terms (over the context $\mathcal{K}$) is denoted $\mathsf{QFT}(\mathcal{K})$. We identify $\mathsf{FT}(\mathcal{K})$ with the subset of $\mathsf{QFT}(\mathcal{K})$ consisting of all quantified feature terms whose prefix is empty.

A variable $\nu \in \mathbf{V}$ is *free in* $\varphi \in \mathsf{QFT}(\mathcal{K})$ if it occurs in $\mathsf{Matrix}(\varphi)$, but not in $\mathsf{Prefix}(\varphi)$. A quantified typed feature term is a *sentence* if it contains no free variables. $\mathsf{QFS}(\mathcal{K})$ denotes the set of elements of $\mathsf{QFT}(\mathcal{K})$ which are sentences.

2.3.8 Satisfiability for typed quantified feature terms. In the following, let $M = (Q, \delta, \alpha, q_o, L, C, \gamma, \eta, \theta) \in \mathsf{TMEFS}(\mathcal{C})$, let $t \in T$ and let $\varphi \in \mathsf{QFT}(\mathcal{K})$.

$\mathsf{Sat}(M, (\exists_{(t,\mathsf{Obj})} x^t)\varphi) = \{\beta \in \mathsf{Asgn}(\mathcal{K}, M) \mid (\exists q \in Q)(\beta[x^t \leftarrow q] \in \mathsf{Sat}(M, \varphi))\}$.

$\mathsf{Sat}(M, (\exists_{(t,\mathsf{Tag})} X^t)\varphi) = \{\beta \in \mathsf{Asgn}(\mathcal{K}, M) \mid (\exists c \in C)(\beta[X^t \leftarrow c] \in \mathsf{Sat}(M, \varphi))\}$.

$\mathsf{Sat}(M, (\exists_{(t,\mathsf{Coll})} \ell^t)\varphi) = \{\beta \in \mathsf{Asgn}(\mathcal{K}, M) \mid (\exists k \in L)(\beta[\ell^t \leftarrow k] \in \mathsf{Sat}(M, \varphi))\}$.

$\mathsf{Sat}(M, (\forall_{(t,\mathsf{Obj})} x^t)\varphi) = \{\beta \in \mathsf{Asgn}(\mathcal{K}, M) \mid (\forall q \in Q)(\beta[x^t \leftarrow q] \in \mathsf{Sat}(M, \varphi))\}$.

$\mathsf{Sat}(M, (\forall_{(t,\mathsf{Tag})} X^t)\varphi) = \{\beta \in \mathsf{Asgn}(\mathcal{K}, M) \mid (\forall c \in C)(\beta[X^t \leftarrow c] \in \mathsf{Sat}(M, \varphi))\}$.

$\mathsf{Sat}(M, (\forall_{(t,\mathsf{Coll})} \ell^t)\varphi) = \{\beta \in \mathsf{Asgn}(\mathcal{K}, M) \mid (\forall k \in L)(\beta[\ell^t \leftarrow k] \in \mathsf{Sat}(M, \varphi))\}$.

2.3.9 Proposition — characterization of sentences. *Let* $M \in \mathsf{TMEFS}(\mathcal{C})$ *and* $\varphi \in \mathsf{QFS}(\mathcal{K})$*. Then either* $\mathsf{Sat}(M, \varphi) = \mathsf{Asgn}(\mathcal{K}, M)$ *or else* $\mathsf{Sat}(M, \varphi) = \emptyset$*.*

Proof. The proof is similar to that for first-order logic, as may be found in [18, 11.6]. □

2.3.10 Example: The Nonlocal Feature Principle of HPSG. The *Nonlocal Feature Principle* of HPSG [22, p. 164] is representative of the class of constraints that this theory targets for representation. Figure 2 depicts the abstract setting in which this constraint may be described. In general, the edges labelled by subscripted λ's may represent sequences of feature edges, rather than single edges. The bolder edges, labelled with subscripted k's, are tag edges. For any S, let $\mathbf{C}_S$ denote the multicollection rooted at node c_S. It is assumed

$$\begin{tikzcd}
 & & c_m & & c_{\overline{hd}} & & \\
 & & & q_{hd} \arrow[ur, "\lambda_{hd_{tb}}"] \arrow[dr, "\lambda_{hd_{ih}}"] & & & \\
 {} \arrow[r] & q_0 \arrow[uur, "\lambda_m"] \arrow[r, "\lambda_d"] & q_d \arrow[ur, "\lambda_{hd}"] \arrow[ddrr, "\lambda_{cd}"] & & c_{hd} & q_{cd_1} \arrow[r, "\lambda_{cdx}"] & c_{cd_1} \\
 & & & & & q_{cd_2} \arrow[r, "\lambda_{cdx}"] & c_{cd_2} \\
 & & & & c_{cd} \arrow[uur, "k_{cd_1}"] \arrow[ur, "k_{cd_2}"] \arrow[dr, "k_{cd_n}"] & \vdots \quad \vdots & \vdots \\
 & & & & & q_{cd_n} \arrow[r, "\lambda_{cdx}"] & c_{cd_n}
\end{tikzcd}$$

Fig. 2. Abstract Setting for the Nonlocal Feature Principle of HPSG

that $\mathbf{C}_{cd}$ is a multilist, while all other multicollections are multisets. The overall constraint may be expressed succinctly, if somewhat informally, as

$$\mathbf{C}_m = (\mathbf{C}_{hd} \cup (\bigcup_{i=1}^{n} \mathbf{C}_{cd_i})) \setminus \mathbf{C}_{\overline{hd}}$$

In other words, the multiset node c_m is the (multiset) union of the multisets at node c_{hd} and at nodes c_{cd_i}, $1 \le i \le n$, less the elements in the multiset at node $c_{\overline{hd}}$. The fact that $\mathbf{C_{cd}}$ is a multilist (as opposed to a multiset) is of no consequence to this abstraction, although it is certainly important in the linguistic model of HPSG.

For those familiar with HPSG, here is a concretization of the paths of Fig. 2, at least for common situations. These definitions assume that q_0 is of type *headed structure.*[1]

$$\begin{aligned}
\lambda_d &= \text{DTRS} \\
\lambda_{hd} &= \text{HEAD-DTR} \\
\lambda_{cd} &= \text{COMP-DTRS} \\
\lambda_m = \lambda_{hd_{ih}} = \lambda_{cdx} &= \text{SYNSEM} \mid \text{NONLOCAL} \mid \text{INHERITED} \mid \text{SLASH} \\
\lambda_{hd_{tb}} &= \text{SYNSEM} \mid \text{NONLOCAL} \mid \text{TO-BIND} \mid \text{SLASH}
\end{aligned}$$

A quantified feature term which represents this constraint is the following.

$$(\forall \ell_{\tau_1})(\exists \ell_{\tau_2})$$
$$(\lambda_m \ell_{\tau_1} \Leftrightarrow ((\lambda_d \lambda_{hd} \lambda_{hd_{ih}} \ell_{\tau_1} \ \vee \ \lambda_d \lambda_{cd} \ell_{\tau_2} \lambda_{cdx} \ell_{\tau_1}) \ \wedge \ \neg \lambda_d \lambda_{hd} \lambda_{hd_{tb}} \ell_{\tau_1})) \qquad (1)$$

[1] This example covers the case in which the only daughters of a headed structure, other than the head daughter, are the complement daughters. However, a headed structure may have other forms of daughters, including a marker daughter, an adjunct daughter, and a filler daughter. For simplicity, such daughters are not modelled here, although the extension of the example described here to include these cases is completely straightforward.

In the context of HPSG, τ_1 is the type *local*, and τ_2 is the type *sign*.

Strictly speaking, the symbol $\Leftrightarrow$ is not allowed in quantified feature terms, but may easily be eliminated in the usual way. It remains to express the order constraints. The following two sentences recapture that $\mathbf{C}_m$ is a multiset and that $\mathbf{C}_{cd}$ is a multilist, respectively. The other multisets are characterized similarly.

$$(\forall \ell_1)(\forall \ell_2)(\lambda_m[\ell_1 \leq \ell_2] \Rightarrow \lambda_m[\ell_2 \leq \ell_1]) \tag{2}$$

$$(\forall \ell_1)(\forall \ell_2)((\lambda_d\lambda_{cd}\ell_1 \vee \lambda_d\lambda_{cd}\ell_2) \Rightarrow (\lambda_d\lambda_{cd}[\ell_1 \leq \ell_2] \vee \lambda_d\lambda_c[\ell_2 \leq \ell_1])) \tag{3}$$

3 Decidability

3.1 Embedding into a First-Order Logic

The approach to establishing satisfiability which is taken in this paper is that of embedding the typed MEFS feature logic into a typed first-order logic. This approach has the distinct advantage that much is known about techniques for establishing decidability for satisfiability of first-order logics, and that wealth of knowledge may be drawn upon in establishing the desired result.

3.1.1 Some essential notation. The *rank* of a relation symbol is a sequence defining the sorts of its arguments. The set of all domain elements of sort τ which occur in the model M is denoted $\mathsf{Dom}(M, \tau)$. The value that symbol X assumes under interpretation M is denoted X^M. For a set Φ of sentences, $\mathsf{Mod}(\Phi)$ denotes the set of all models of Φ, while $\mathsf{Mod}_f(\Phi)$ denotes the set of all finite models of Φ. Because we work with two distinct logics, there are two distinct notions of satisfiability. As already defined in 2.3.6 and 2.3.9, $\mathsf{Sat}(M, \varphi)$ denotes the set of $(\mathcal{K}, M)$-assignments which satisfy the typed feature term φ with respect to the typed MEFS M. On the other hand, $\mathsf{FOSat}(M, \varphi)$ denotes the set of first-order truth assignments which satisfy the (many-sorted) first-order formula φ with respect to the first-order model M, in the logic $\mathsf{Logic}(\mathcal{K})$ defined in 3.1.2 below.

3.1.2 The first-order logic of a typed MEFS context. The *many-sorted first-order logic corresponding to* $\mathcal{K}$, denoted $\mathsf{Logic}(\mathcal{K})$, is the first-order logic, with equality, which is defined as follows.

(i) The set of sorts of $\mathsf{Logic}(\mathcal{K})$ is precisely $\mathsf{CESort}(\mathbf{T})$.

(ii) The language contains precisely the following relation symbols.
- For each $f \in F$, there is a relation symbol Attr_f, with rank $(\mathsf{Obj}, \mathsf{Obj})$.
- For each $f \in F$, there is a relation symbol AttrC_f, with rank $(\mathsf{Obj}, \mathsf{Coll})$.
- For each $a \in A$, there is a relation symbol Const_a, with rank (Obj).
- There is a relation symbol $\mathsf{TagOrder}$, with rank $(\mathsf{Coll}, \mathsf{Tag}, \mathsf{Tag})$.
- There is a relation symbol TagVal, with rank $(\mathsf{Tag}, \mathsf{Obj})$.
- There is a relation symbol $\mathsf{TagUsed}$, with rank (Tag).

(iii) There is one constant symbol, $\mathsf{InitState}$, of type Obj.

(iv) There are no non-nullary function symbols.

3.1.3 First-order representation of an MEFS. Let $M = (Q, \delta, \alpha, q_o, L, C, \gamma, \eta, \theta)$ be a typed MEFS. The *first-order representation of* M, denoted $\mathsf{FO}(M)$, is the interpretation in $\mathsf{Logic}(\mathcal{K})$ which describes M. Informally, $\mathsf{Attr}_f^{\mathsf{FO}(M)}(q_1, q_2)$ (resp. $\mathsf{AttrC}_f^{\mathsf{FO}(M)}(q, c)$) means that there is a feature edge in M from q_1 to q_2 (resp. from q_1 to c) labelled f. $\mathsf{TagOrder}^{\mathsf{FO}(M)}(c, k_1, k_2)$ records that $k_1 \leq k_2$ at node c. $\mathsf{TagVal}^{\mathsf{FO}(M)}(k, q)$ records that tag k is assigned the value at q, and that the types of a and q are compatible. $\mathsf{TagUsed}^{\mathsf{FO}(M)}(k)$ just records that the tag k is used in M. The formal specification is as follows.
(i) $\mathsf{Dom}(\mathsf{FO}(M), (t, \mathsf{Obj})) = \{q \in Q \mid \theta(q) \sqsubseteq t\}$.
(ii) $\mathsf{Dom}(\mathsf{FO}(M), (t, \mathsf{Coll})) = \{c \in C \mid \theta(c) \sqsubseteq t\}$.
(iii) $\mathsf{Dom}(\mathsf{FO}(M), (t, \mathsf{Tag})) = \{k \in L \mid \theta(k) \sqsubseteq t\}$.
(iv) $\mathsf{Attr}_f^{\mathsf{FO}(M)}(q_1, q_2)$ iff $q_1, q_2 \in Q$ and $\delta(q_1, f) = q_2$.
(v) $\mathsf{AttrC}_f^{\mathsf{FO}(M)}(q, c)$ iff $q \in Q$, $c \in C$, and $\delta(q, f) = c$.
(vi) $\mathsf{Const}_a^{\mathsf{FO}(M)}(q)$ iff $q \in \mathsf{Dom}(\mathsf{FO}(M), (\mu(a), \mathsf{Obj}))$ and $\alpha(q) = a$.
(vii) $\mathsf{TagOrder}^{\mathsf{FO}(M)}(c, k_1, k_2)$ iff $c \in C$, $k_1, k_2 \in \eta(c)$, and $k_1 \leq_{\eta(c)} k_2$.
(viii) $\mathsf{TagVal}^{\mathsf{FO}(M)}(k, q)$ iff $k \in L$, $q \in Q$, and $\gamma(k) = q$.
(ix) $\mathsf{TagUsed}^{\mathsf{FO}(M)}(k)$ iff $k \in L$ and there is a $c \in C$ such that $\delta(c, k)\downarrow$.
(x) $\mathsf{InitState}^{\mathsf{FO}(M)} = q_o$.

3.1.4 The axiom system. To ensure that a structure over the logic $\mathsf{Logic}(\mathcal{K})$ represents a typed MEFS, it is necessary to enforce certain axioms. The *first-order axiom system* $\mathsf{FAxioms}(\mathcal{K})$ for the typed MEFS context $\mathcal{K}$ is defined to be the following set.
(i) For each $a \in A$:
  (i-a) $(\forall x)(\forall y)((\mathsf{Const}_a(x) \wedge \mathsf{Const}_a(y)) \Rightarrow (x = y))$.
(ii) For each $f \in F$:
  (ii-a) $(\forall x)(\forall y)(\forall z)((\mathsf{Attr}_f(x, y) \wedge \mathsf{AttrC}_f(x, z)) \Rightarrow \perp)$.
  (ii-b) $(\forall x)(\forall y)(\forall z)((\mathsf{Attr}_f(x, y) \wedge \mathsf{Attr}_f(x, z)) \Rightarrow y = z)$.
  (ii-c) $(\forall x)(\forall y)(\forall z)((\mathsf{AttrC}_f(x, y) \wedge \mathsf{AttrC}_f(x, z)) \Rightarrow y = z)$.
(iii) $(\forall X)(\forall \ell_1)(\forall \ell_2)$
$$(\mathsf{TagOrder}(X, \ell_1, \ell_2) \Rightarrow (\mathsf{TagOrder}(X, \ell_1, \ell_1) \wedge \mathsf{TagOrder}(X, \ell_2, \ell_2))).$$
(iv) $(\forall X)(\forall \ell_1)(\forall \ell_2)(\forall \ell_3)$
$$((\mathsf{TagOrder}(X, \ell_1, \ell_2) \wedge \mathsf{TagOrder}(X, \ell_2, \ell_3)) \Rightarrow \mathsf{TagOrder}(X, \ell_1, \ell_3)).$$
(v) $(\forall X)(\forall \ell_1)(\forall \ell_2)((\mathsf{TagOrder}(X, \ell_1, \ell_2) \wedge \mathsf{TagOrder}(X, \ell_2, \ell_1) \Rightarrow \ell_1 = \ell_2))$.
(vi) $(\forall X)(\forall \ell)(\mathsf{TagOrder}(X, \ell, \ell) \Rightarrow \mathsf{TagUsed}(\ell))$.
(vii) $(\forall \ell)(\exists y)(\mathsf{TagUsed}(\ell) \Rightarrow \mathsf{TagVal}(\ell, y))$.
(viii) $(\forall \ell)(\forall x)(\forall y)((\mathsf{TagVal}(\ell, x) \wedge \mathsf{TagVal}(\ell, y)) \Rightarrow x = y)$.
Condition (i) states that constant a can be associated with at most one object state; (ii) states that there is at most one edge leaving a given object node with a given feature label; conditions (iii)-(v) state that $\mathsf{TagOrder}$ is a partial order; condition (vi) states that any tag at node X is used; condition (vii) states that the value associated with a given tag is global.

3.1.5 Proposition. *If* $M \in \mathsf{TMEFS}(\mathcal{C})$*, then* $\mathsf{FO}(M) \in \mathsf{FAxioms}(\mathcal{K})$. $\square$

The proof of 3.1.5 is a straightforward verification. Thus, the first order representation of any typed MEFS satisfies the axioms of 3.1.4.

3.1.6 Representation of a feature term as a first-order formula. To establish an equivalence between the logic of typed MEFS's and these axioms, it must also be shown that any (finite) model of these axioms represents an MEFS. This task is somewhat intricate, and requires a substantial dose of complex notation. To simplify the presentation in this abbreviated paper, the idea of this representation will be illustrated, with the formal details left to the reader. The idea is a natural extension of that introduced for ordinary feature structures in [10], [11]. Consider the feature term $\varphi = f_1 f_2 \ell_1 f_1[\ell_2 \leq \ell_3]$, which might be used in the representation of the example of Fig. 1. One first-order representation is

$$\begin{aligned}(\exists x_0)(\exists x_1)(\exists x_2)(\exists x_3)(\exists x_4)(\exists X_1)(\exists X_2)\ (&\mathsf{InitState} = x_0 \wedge \mathsf{Attr}_{f_1}(x_0, x_1) \wedge \\ &\mathsf{AttrC}_{f_2}(x_1, X_1) \wedge \mathsf{Attr}_{f_1}(x_2, X_2) \wedge \mathsf{TagVal}(\ell_1, x_2) \wedge \mathsf{TagVal}(\ell_2, x_3) \wedge \\ &\mathsf{TagVal}(\ell_3, x_4) \wedge \mathsf{TagOrder}(X_1, \ell_1, \ell_1) \wedge \mathsf{TagOrder}(X_2, \ell_2, \ell_3))\end{aligned} \tag{4}$$

Similarly, a first-order formula representing $\varphi' = f_1 f_2 \ell_1 f_1 : a$ is

$$\begin{aligned}&(\exists x_0)(\exists x_1)(\exists x_2)(\exists x_3)(\exists X_1)(\exists X_2) \\ &(\mathsf{InitState} = x_0 \wedge \mathsf{Attr}_{f_1}(x_0, x_1) \wedge \mathsf{AttrC}_{f_2}(x_1, X_1) \wedge \mathsf{Attr}_{f_1}(x_2, X_2) \wedge x_3 = a \wedge \\ &\mathsf{TagVal}(\ell_1, x_2) \wedge \mathsf{TagVal}(\ell_2, x_3) \wedge \mathsf{TagOrder}(X_1, \ell_1, \ell_1) \wedge \mathsf{TagOrder}(X_2, \ell_2, \ell_2))\end{aligned} \tag{5}$$

These two examples illustrate most of the key ideas of the representation. A variable, of the appropriate sort, is introduced for each node which the path requires. The predicates have the meanings identified in 3.1.3. Note that some predicates are implied by the axioms of 3.1,4, and need not be included explicitly. For example, $\mathsf{TagOrder}(X_2, \ell_2, \ell_2)$, $\mathsf{TagOrder}(X_2, \ell_3, \ell_3)$, $\mathsf{TagUsed}(\ell_1)$, $\mathsf{TagUsed}(\ell_2)$, and $\mathsf{TagUsed}(\ell_3)$ may be added as conjuncts to the representation of φ without altering the semantics. Nonetheless, since all such representations are equivalent in the presence of the schema of 3.1.4, the terminology "the" first order representation of φ, notationally $\mathsf{FO}(\varphi)$, will be used in that which follows.

All quantifiers introduced in this representation are existential. In translating a feature term, any quantifiers on that term are placed outside of the the scope of these translation quantifiers. Thus, for quantified feature term, $\psi = (\forall \ell_1)(\exists \ell_2)(\exists \ell_3)(f_1 f_2 \ell_1 f_1[\ell_2 \leq \ell_3])$, the full quantifier prefix, to be applied to the conjunct of atoms identified in (4) above, would be $(\forall \ell_1)(\exists \ell_2)(\exists \ell_3)(\exists x_0)(\exists x_1)(\exists x_2)(\exists x_3)(\exists x_4)(\exists X_1)(\exists X_2)$. The string $(\forall \ell_1)(\exists \ell_2)(\exists \ell_3)$ is $\mathsf{Prefix}(\psi)$, and the string $(f_1 f_2 \ell_1 f_1[\ell_2 \leq \ell_3])$ is $\mathsf{Matrix}(\psi)$, in the notation of 2.3.7. $\mathsf{FO}(\varphi)$ is then defined to be $\mathsf{Prefix}(\varphi) \cdot \mathsf{FO}(\mathsf{Matrix}(\psi))$. Note that $\mathsf{FO}(\mathsf{Matrix}(\psi)) = \varphi$ is the formula of (4). The full formal result relating satisfiability of quantified feature terms to the satisfiability of first-order formulas is the following.

3.1.7 Proposition. *Let* $\varphi \in \mathsf{QFT}(\mathcal{K})$*, and let* $M \in \mathsf{TMEFS}(\mathcal{C})$*. Then* $\mathsf{Sat}(M, \varphi) = \mathsf{FOSat}(\mathsf{FO}(M), \mathsf{FO}(\varphi))$*.* □

The proof of 3.1.7 is a tedious but straightforward inductive argument. We are now in a position to provide the complement of 3.1.3; namely the definition of a canonical typed MEFS associated with a first-order model.

3.1.8 The canonical MEFS of a first-order model. Let $\mathcal{I} \in \mathsf{Mod}_f(\mathsf{FAxioms}(\mathcal{K}))$. Define the *canonical typed MEFS corresponding to* $\mathcal{I}$, denoted $\mathsf{TMEFS}(\mathcal{I}) = (Q^{\mathcal{I}}, \delta^{\mathcal{I}}, \alpha^{\mathcal{I}}, q_o^{\mathcal{I}}, L^{\mathcal{I}}, C^{\mathcal{I}}, \gamma^{\mathcal{I}}, \eta^{\mathcal{I}}, \theta^{\mathcal{I}})$, as follows. Start by defining the relation $\mathsf{AttrL}^{\mathcal{I}}$ of rank $(\mathsf{Coll}, \mathsf{Obj})$ by $\mathsf{AttrL}^{\mathcal{I}}(X, x)$ iff $(\exists \ell)(\mathsf{TagOrder}(X, \ell, \ell) \wedge \mathsf{TagVal}(\ell, x))$. Then let $\mathsf{Attr}^{\mathcal{I}}$ to be the binary relation defined by $(\bigcup_{e \in F} \mathsf{Attr}_e^{\mathcal{I}}) \cup (\bigcup_{e \in F} \mathsf{AttrC}_e^{\mathcal{I}}) \cup \mathsf{AttrL}^{\mathcal{I}}$. Next, let $\overline{\mathsf{Attr}}^{\mathcal{I}}$ be the transitive closure of Attr. We are now set to make the definitions.

(a) $Q^{\mathcal{I}} = \{q \in \mathsf{Dom}(\mathcal{I}, \mathsf{Obj}) \mid \overline{\mathsf{Attr}}^{\mathcal{I}}(\mathsf{InitState}, q)\}$.

(b) $C^{\mathcal{I}} = \{x \in \mathsf{Dom}(\mathcal{I}, \mathsf{Coll}) \mid \overline{\mathsf{Attr}}^{\mathcal{I}}(\mathsf{InitState}, X)\}$.

(c) $L^{\mathcal{I}} = \{\ell \in \mathsf{Dom}(\mathcal{I}, \mathsf{Tag}) \mid (\exists X \in C^{\mathcal{I}})(\mathsf{TagOrder}(X, \ell, \ell))\}$.

(d) $\delta^{\mathcal{I}}$ is defined by $\delta^{\mathcal{I}}(q, e)\!\downarrow$ iff $q \in Q^{\mathcal{I}}$ and either $(\exists r \in Q^{\mathcal{I}})(\mathsf{Attr}_e^{\mathcal{I}}(q, r))$ or else $(\exists c \in C^{\mathcal{I}})(\mathsf{AttrC}_e^{\mathcal{I}}(q, c))$. In this case, $\delta^{\mathcal{I}}(q, e)$ is defined to be the unique r or c found by this construction. The uniqueness follows from rule (ii) of 3.1.4.

(e) For $q \in Q^{\mathcal{I}}$, $\alpha^{\mathcal{I}}(q)\!\downarrow$ iff $(\exists a \in A)(\mathsf{Const}_a(q))$, and then $\alpha^{\mathcal{I}}(q) = a$.

(f) For $k \in L^{\mathcal{I}}$, $\gamma^{\mathcal{I}}(k)$ is defined iff $(\exists q \in Q^{\mathcal{I}})(\mathsf{TagVal}(k, q))$ holds. In this case, the value of $\gamma^{\mathcal{I}}(k)$ is this q.

(g) For $c \in C^{\mathcal{I}}$, $\eta^{\mathcal{I}}(c)$ is defined to be the poset whose underlying set is $\{k \mid \mathsf{TagOrder}^{\mathcal{I}}(c, k, k)\}$, with $k_1 \leq_c k_2$ iff $\mathsf{TagOrder}^{\mathcal{I}}(c, k_1, k_2)$.

(h) The typing function $\theta^{\mathcal{I}}$ is defined as follows.
- For $q \in Q^{\mathcal{I}}$, $\theta^{\mathcal{I}}(q) = \sqcap\{t \in T \mid q \in \mathsf{Dom}(\mathcal{I}, (t, \mathsf{Obj}))\}$.
- For $c \in C^{\mathcal{I}}$, $\theta^{\mathcal{I}}(c) = \sqcap\{t \in T \mid c \in \mathsf{Dom}(\mathcal{I}, (t, \mathsf{Coll}))\}$.
- For $k \in L^{\mathcal{I}}$, $\theta^{\mathcal{I}}(k) = \sqcap\{t \in T \mid k \in \mathsf{Dom}(\mathcal{I}, (t, \mathsf{Tag}))\}$.

3.1.9 Proposition. *Let* $\mathcal{I} \in \mathsf{Mod}_f(\mathsf{FAxioms}(\mathcal{K}))$*. Then* $\mathsf{TMEFS}(\mathcal{I}) \in \mathsf{TMEFS}(\mathcal{C})$*, and for any* $\varphi \in \mathsf{QFT}(\mathcal{K})$*,* $\mathsf{Sat}(\mathsf{TMEFS}(\mathcal{I}), \varphi) = \mathsf{FOSat}(\mathcal{I}, \mathsf{FO}(\varphi))$*.* □

The proof is once again a tedious but straightforward induction. The finiteness of $\mathcal{I}$ is essential, since a set of sentences in $\mathsf{Logic}(\mathcal{K})$ may certainly have infinite models, which will by definition not be typed MEFS's. In 3.2.5, an applicable condition on a set of sentences which ensures that the existence of a model implies the existence of finite model will be established.

3.2 Decidability of the First-Order Theory

3.2.1 The Schönfinkel-Bernays class. In single-sorted first-order logic, the *Schönfinkel-Bernays* class (or *SB class*, for short) of sentences, without function symbols, has the property that no existential quantifier occurs within the scope of a universal quantifier. Thus, in prenex normal form, such a sentence has the general form $(\exists x_1)\cdots(\exists x_m)(\forall y_1)\cdots(\forall y_n)\varphi$. It is well known that any sentence in this class (and hence any finite set of such sentences) is decidable for satisfiability [6, p. 212]. The proof rests upon the fact that the Herbrand universe [16, 9.4] of the functional form of the sentence (i.e., the sentence obtained by Skolemizing all existential variables) involves no non-nullary function symbols, and hence is finite with a predetermined size bound.

The SB class has been used to establish the decidability of a logic for feature structures [10], [11]. However, in the context of this paper, both the axiom system (3.1.4) and typical constraints to be modelled (2.3.11) involve formulas outside outside of the SB class. Fortunately, it is possible to extend the idea of the SB class to many-sorted logics in such a way that existential quantifiers may lie with in the scope of universal ones, yet decidability is preserved.

3.2.2 Quantifier classes in many-sorted logics. In a single-sorted logic, a quantifier language is just a set of strings over the alphabet $\{\forall, \exists\}$. The quantifier language associated with the SB class is thus the regular set $\exists^*\forall^*$. The extension of the notion of quantifier language to the many-sorted case is identical, save that the quantifiers are tagged.

A *quantifier language* over a set S of sorts (e.g., $S = \mathsf{CESort}(\mathbf{T})$) is a set of strings $\mathcal{Q}$ over the alphabet $\{\forall_\tau \mid \tau \in S\} \cup \{\exists_\tau \mid \tau \in S\}$. The *closure* of $\mathcal{Q}$ is $\overline{\mathcal{Q}} = \{Q^1_{\tau_1}Q^2_{\tau_2}..Q^k_{\tau_k} \mid (\forall i :\ 1 \leq i \leq k)(Q^i \in \{\forall, \exists\}$ and $\exists \tau'_i \in S$ with $\tau_i \sqsubseteq \tau'_i$ and $Q^1_{\tau'_1}Q^2_{\tau'_2}..Q^k_{\tau'_k} \in \mathcal{Q})\}$. The closure of $\mathcal{Q}$ thus contains all strings obtained by replacing type subscripts on quantifiers by more specific types. $\mathcal{Q}$ is *closed* if $\mathcal{Q} = \overline{\mathcal{Q}}$. Given a formula φ in prefix-matrix form, the *quantifier string* of φ is the string $\mathsf{QuantStr}(\varphi)$ obtained from $\mathsf{Prefix}(\varphi)$ by deleting all parentheses and variables (but preserving or restoring type markers on quantifiers). For example, in $\mathsf{Logic}(\mathcal{K})$, both $(\forall_{(s_1,\mathsf{Obj})}x^{s_1})(\exists_{(s_2,\mathsf{Coll})}Y^{s_2})(\exists_{(\top,\mathsf{Tag})}\ell^\top)$ and its abbreviation $(\forall x^{s_1})(\exists Y^{s_2})(\exists \ell)$ (see 2.3.8(b)) become $\forall_{(s_1,\mathsf{Obj})}\exists_{(s_2,\mathsf{Coll})}\exists_{(\top,\mathsf{Tag})}$. Given an S-sorted first-order logic $\mathcal{L}$, the *quantifier class* for the quantifier language $\mathcal{Q}$, denoted $\mathsf{QuantClass}(\mathcal{Q}, \mathcal{L})$, is the set of all sentences φ in the language of $\mathcal{L}$ in prefix-matrix form such that $\mathsf{QuantStr}(\varphi) \in \mathcal{Q}$.

3.2.3 Functional forms, the Herbrand universe and expansion. The ideas of this paragraph, for the single-sorted case, are discussed in [16, Sec. 9.4]. Only the items necessary to extend these concepts to the many-sorted context are presented here. Given a sentence, the *functional form* is obtained by Skolemizing all existential variables. A key difference is that, in the many-sorted case, functions have rank. For example, the Skolemization of axiom 3.1.4(vii) yields $(\forall \ell)(\mathsf{TagUsed}(\ell) \Rightarrow \mathsf{TagVal}(\ell, f_y(\ell)))$. The Skolem function f_y has rank $\mathsf{Obj} \to \mathsf{Tag}$; that is, the single argument must be of sort Obj, and the result will be of sort Tag.

The *Herbrand universe* consists of all formal terms obtained by composition of all functions, Skolem and otherwise. If there is no term of a given maximal sort s, a special constant term a_s is introduced and added to the Herbrand universe. For $\mathsf{Logic}(\mathcal{K})$ and the axioms of 3.1.4, there is one constant symbol from the language $\mathsf{InitState}$ (of sort Obj), and just one Skolem function, the f_y defined above. The composite term $f_y(\mathsf{InitState})$ is of type Tag. It is necessary to add one additional constant symbol a_{Coll}, corresponding to the maximal sort Coll, to ensure that the Herbrand universe contains at least one element of each sort. Notice, though, that because of the rank constraints, the function f_y cannot compose recursively, and so the Herbrand universe for 3.1.4 is finite, consisting of just the three terms $\{\mathsf{InitState}, f_y(\mathsf{InitState}), a_{\mathsf{Coll}}\}$.

The *Herbrand expansion* consists of all ground terms of the underlying formulas formed by using the Herbrand universe as the underlying domain space. If the Herbrand universe is finite and of determinable size, so too will be the Herbrand expansion of a finite set of formulas (such as those of 3.1.4). Since a set of sentences has a model iff its Herbrand expansion does, and a model of the Herbrand expansion is a model of the original set of sentences, finiteness of the Herbrand expansion provides not only a framework for establishing the existence of a model, but for its finiteness as well. We now turn to the issue of determining conditions under which the Herbrand universe remains finite and of determinable size.

3.2.4 Generalized SB quantifier languages. Let S be a set of sorts and $\mathcal{Q}$ be a quantifier language over S. Define $R_{\mathcal{Q}} \subseteq S \times S$ by $(\tau_1, \tau_2) \in R_{\mathcal{Q}}$ iff there is a string $\sigma \in \mathcal{Q}$ with $\forall_{\tau_1}$ preceding $\exists_{\tau_2}$. Define $R_{\mathcal{Q}}^{\sqsubseteq} = \{(\tau_1, \tau_2) \mid (\exists \tau_3)(\exists \tau_4)((\tau_3, \tau_4) \in R_{\mathcal{Q}}$ and $\tau_1 \sqsubseteq \tau_3$ and $\tau_2 \sqsubseteq \tau_4)\}$, with $R_{\mathcal{Q}}^{\sqsubseteq +}$ its transitive closure. Call $\mathcal{Q}$ a *generalized SB-quantifier class* if $(\tau_1, \tau_2) \in R_{\mathcal{Q}}^{\sqsubseteq +}$ implies $\tau_1 \sqcap \tau_2 = \bot$.

3.2.5 Theorem. *Let S be a set of sorts, $\mathcal{L}$ an S-sorted first-order logic with no non-nullary function symbols and only finitely many constant symbols, and $\mathcal{Q}$ a generalized SB-quantifier class. Then* $\mathsf{QuantClass}(\mathcal{Q}, \mathcal{L})$ *is decidable for satisfiability, and any finite $\Phi \subseteq$* $\mathsf{QuantClass}(\mathcal{Q}, \mathcal{L})$ *which has a model has a finite model.*

Proof sketch. The proof hinges upon establishing that the Herbrand universe is finite, and of determinable size. Once that is established, we may make use of the fact that a formula is satisfiable iff its Herbrand expansion is [16, Thm. 9.4.1].

To establish that the Herbrand universe is finite. it suffices to show that for no function symbol f is it possible to have a term of the form $f(\ldots, t, \ldots)$ for which f occurs in t. Let $\mathcal{Q}$ denote the quantifier class for the given set of sentences, and assume that t is a term involving a chain of function symbols $(f_0, f_1, \ldots f_k)$, in which $f_0 = f$ and for each i, $0 \le i \le k-1$, a term of the form $f_i(-)$ occurs as the $\alpha(i)^{th}$ argument to f_{i-1}. Let τ_i denote the range sort of f_i, and let $\tau_{\alpha(i)}$ denote the domain type in f_i of the argument position in which the f_{i+1} term occurs. Then $(\tau_{\alpha(i)}, \tau_i) \in R_{\mathcal{Q}}^{\sqsubseteq +}$ for $0 \le i \le k-1$, and since $\tau_{i+1} \sqsubseteq \tau_{\alpha(i)}$, it follows that $(\tau_{i+1}, \tau_i) \in R_{\mathcal{Q}}^{\sqsubseteq}$ for $0 \le i \le k-1$. Thus $(\tau_k, \tau_0) \in R_{\mathcal{Q}}^{\sqsubseteq +}$. Thus $f_k \ne f_0$, and so no terms can involve recursive applications of any function symbol. The number of distinct terms is finite and strictly bounded, whence the Herbrand expansion has the same property. Decidability, as well as finiteness of the model, follows as outlined above.

A final point to consider is the decidability of a finite set of sentences, as opposed to a single sentence. However, it is straightforward to establish that $\mathsf{QuantClass}(\mathcal{Q}, \mathcal{L})$ must be closed under conjunctions, since the relation $R_{\mathcal{Q}}^{\sqsubseteq+}$ is constructed from all strings in $\mathcal{Q}$, and not just a single string. In other words, the elements of $R_{\mathcal{Q}}^{\sqsubseteq+}$ generated by a "best" prefix of $\varphi_1 \wedge \varphi_2$ are also generated by $\mathsf{Prefix}(\varphi_1)$ and $\mathsf{Prefix}(\varphi_2)$. □

3.2.6 A family of decidable classes of typed MEFS feature logics. *Let $\mathcal{Q}_1$ be a generalized SB quantifier class which is a subset of $(\{\forall_{(s,\mathsf{Tag})} \mid s \in T \setminus \{\bot\}\} \cup \{\exists_{(s,\mathsf{Tag})} \mid s \in T \setminus \{\bot\}\})^*$, and let $\mathcal{Q}_2 = (\{\exists_{(s,\mathsf{Obj})} \mid s \in T \setminus \{\bot\}\} \cup \{\exists_{(s,\mathsf{Coll})} \mid s \in T \setminus \{\bot\}\})^*$. Then any finite family of quantified typed feature sentences over the quantifier class $\mathcal{Q}_1 \cdot \mathcal{Q}_2 = \{\sigma_1 \cdot \sigma_2 \mid \sigma_1 \in \mathcal{Q}_1$ and $\sigma_2 \in \mathcal{Q}_2\}$ is decidable for satisfiability.*

Proof sketch. At first glance, this may appear to be a simple application of 3.2.5. However, 3.2.5 is a theorem about first-order logic, while this result concerns typed feature logic. Let $\varphi \in \mathsf{QFS}(\mathcal{K})$ be such that $\mathsf{QuantStr}(\varphi) \in \mathcal{Q}_1 \cdot \mathcal{Q}_2$. The process of translation of φ to $\mathsf{FO}(\varphi)$, as described in 3.1.6, introduces additional quantifiers. In the translation, feature terms are replaced with the equivalent formulas identified in 3.1.6. There are two key points to note. First of all, the quantifiers which are added will lie within the scope (and thus to the right of) the quantifiers of φ. Second. all quantifiers in the replacement terms are existential and of a subsort of either Obj or else of Coll. Some of the original feature terms may be negated, which complements the affected quantifiers to universal. However, these feature terms do not lie in one another's scope, so, for each term in the translation, the quantifiers will be either all existential or else all universal. The universal quantifiers of this added sequence may thus be moved inside of the existential ones, resulting in a quantifier sequence of the form $\exists_{\tau_1}\exists_{\tau_2}..\exists_{\tau_{k_1}}\forall_{\upsilon_1}\forall_{\upsilon_2}..\forall_{\upsilon_{k_2}}$, with each τ_i and each υ_i of one of the forms (s, Obj) or (s, Coll), for some $s \in T \setminus \{\bot\}$. In other words, if we define $\mathcal{Q}_3 = (\{\forall_{(s,\mathsf{Obj})} \mid s \in T \setminus \{\bot\}\} \cup \{\forall_{(s,\mathsf{Coll})} \mid s \in T \setminus \{\bot\}\})^*$, then this added quantifier sequence will be in $\mathcal{Q}_2 \cdot \mathcal{Q}_3$. The translated first order formula will thus have a total quantifier sequence in $\mathcal{Q}_1 \cdot \mathcal{Q}_2 \cdot \mathcal{Q}_2 \cdot \mathcal{Q}_3$, which reduces to just $\mathcal{Q}_1 \cdot \mathcal{Q}_2 \cdot \mathcal{Q}_3$. It is clear that $(\mathcal{Q}_1 \cdot \mathcal{Q}_2 \cdot \mathcal{Q}_3)^{\sqsubseteq+} = (\mathcal{Q}_1 \cdot \mathcal{Q}_2)^{\sqsubseteq+}$, since there are no existential quantifiers to the right of any of the universal quantifiers in $\mathcal{Q}_3$. We are thus left with the problem of analyzing the original quantifier class $\mathcal{Q}_1 \cdot \mathcal{Q}_2$. It is a straightforward verification that this is an SB quantifier class, thus completing the proof. □

3.2.7 Example: decidability of the Nonlocal Feature Principle. The constraints of the Nonlocal Feature Principle of HPSG are expressed by equations (1)–(3), in 2.3.11. To establish decidability, these constraints must be combined with the axiom system $\mathsf{FAxioms}(\mathcal{K})$ of 3.1.4, and the resulting set of typed feature sentences tested for satisfiability. It is easy to see that the conditions of 3.2.6 applies to this set, rendering it satisfiable.

4 Limitations and Further Directions

The work abstracted here is only a beginning of an effort towards fully automated parsing of HPSG, for several reasons. First of all, certain critical, but not all, set and list operations are expressible in a decidable quantifier class of our language. Further work is necessary to establish a universal framework for expressing a wider variety of set and list constructions.

Second, the only specialized constructions which are considered are those based upon sets and lists. Other equally important forms, such as *linear precedence constraints*, are not addressed.

Third, while the logic presented is decidable for the representation of constraints at a single level, it is not decidable for what Kepser [14] calls *grammaticality*; that is, it is not decidable for recursively embedded constraints. For example, the Nonlocal Feature Principle must hold at every node which is type both *phrase* and *headed-structure*, and not just at one particular node. (It should be pointed out the neither the logic of King [15] nor the logic of Manandhar [17] are shown to be decidable for grammaticality, and in all likelihood they do not have this property.) It is the belief of this author that the decidability of grammaticality cannot be established without some reference to input size; decidability of grammaticality must be tied to size-bounding information computed from the input string, in a spirit similar to off-line parsing of LFG [20] (but without reference to a context-free skeleton, of course). Johnson [12] does provide a system, based upon first-order logic, in which grammaticality is established via off-line parsability, but the formalism is closer to LFG than to HPSG, in that is is built around a context-free skeleton. Furthermore, it does not support types or lists as an explicit construct.

Fourth, the expression of even simple constraints is quite tedious. Thus, at a very minimum, some enclosing package of syntactic sugar which makes is relatively simple to represent key constraints in a simple fashion must be developed.

Finally, it must be acknowledged that decidability does not necessarily imply tractability. Once a decidable logic is identified, the issue of finding means of incorporating it into computationally tractable algorithms remains. Nonetheless, it is possible to envision embedding a decidable logic, such as the one described here, into a larger, more general system such as CUF, TFS, or ALE. This would enable a larger class of constraints to be solved without explicit specification of control information.

References

1. B. Carpenter. ALE: The Attribute Logic Engine user's guide. Technical report, Carnegie Mellon University, Laboratory for Computational Linguistics, 1992.
2. B. Carpenter. *The Logic of Typed Feature Structures.* Cambridge University Press, 1992.
3. B. Carpenter. Set descriptions for logic programming. Paper presented at the 3rd ASL/LSA Conference on Logic and Language. Also referenced under the title: An attribute-value logic for sets, 1993.

4. B. A. Davey and H. A. Priestly. *Introduction to Lattices and Order.* Cambridge University Press, 1990.
5. M. Dorna. The CUF user's manual. Technical report, Institut für maschinelle Sprachverarbeitung, Universität Stuttgart, 1994.
6. B. Dreben and W. D. Goldfarb. *The Decision Problem: Solvable Classes of Quantificational Formulas.* Addison-Wesley, 1979.
7. J. E. Fenstad, T. Langholm, and E. Vestre. Representations and interpretations. In M. Rosner and R. Johnson, editors, *Computational Linguistics and Formal Semantics*, pages 31–95. Cambridge University Press, 1992.
8. J. H. Gallier. *Logic for Computer Science.* John Wiley and Sons, 1986.
9. S. J. Hegner. Distributivity in incompletly specified type hierarchies: Theory and computational complexity. In J. Dörre, editor, *Computational Aspects of Constraint-Based Linguistic Description II, DYANA-2, ESPRIT Basic Research Project 6852, Deliverable R1.2B*, pages 29–120. DYANA, 1994. Also available as LLI Technical Report No. 4, University of Oslo, Department of Linguistics.
10. M. Johnson. Features and formulas. *Computational Linguistics*, 17:131–151, 1991.
11. M. Johnson. Computing with features as formulas. *Computational Linguistics*, 20(1):1–25, 1994.
12. M. Johnson. Two ways of formalizing grammar. *Linguistics and Phil.*, 17, 1994.
13. R. T. Kasper and W. C. Rounds. The logic of unification in grammar. *Linguistics and Phil.*, 13:35–58, 1990.
14. S. Kepser. A satisfiability algorithm for a typed feature logic. Sonderforschungsberiech 340, Teilprojekt B4, Universities of Stuttgart and Tübingen, 1993.
15. P. King. *A Logical Formalism for Head-Driven Phrase Structure Grammar.* PhD thesis, University of Manchester, 1989.
16. H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall, 1981.
17. S. Manandhar. An attributive logic of set descriptions and set operations. In *Proceedings of the 32nd Annual Meeting of the ACL*, pages 255–262, 1994.
18. J. D. Monk. *Mathematical Logic.* Springer-Verlag, 1976.
19. M. A. Moshier and C. J. Pollard. The domain of set-valued feature structures. *Linguistics and Phil.*, 17:607–631, 1994.
20. F. C. N. Pereira and D. H. D. Warren. Parsing as deduction. In *Proceedings of the 21st Annual Meeting of the ACL*, pages 137–144, 1983.
21. C. Pollard and I. A. Sag. *Information-Based Syntax and Semantics*, volume 1: Fundamentals of *CSLI Lecture Notes No. 13.* CSLI, 1987.
22. C. Pollard and I. A. Sag. *Head-Driven Phrase Structure Grammar.* University of Chicago Press, 1994.
23. S. Sahni. *Concepts in Discrete Mathematics.* Camelot Publishing, 1985.
24. R. Zajac. Notes on the Typed Feature System, Version 4, January 1991. Technical report, Universität Stuttgart, Institut für Informatik, Project Polygloss, 1991.