Computational and Structural Aspects of Openly Specified Type Hierarchies

Stephen J. Hegner

Umeå University
Department of Computing Science
S-901 87 Umeå, Sweden
hegner@cs.umu.se
http://www.cs.umu.se/~hegner

Abstract. One may identify two main approaches to the description of type hierarchies. In *total specification*, a unique hierarchy is described. In *open specification*, a set of constraints identifies properties of the hierarchy, without providing a complete description. Open specification provides increased expressive power, but at the expense of considerable computational complexity, with essential tasks being NP-complete or NP-hard. In this work, a formal study of the structural and computational aspects of open specification is conducted, so that a better understanding of how techniques may be developed to address these complexities. In addition, a technique is presented, based upon Horn clauses, which allows one to obtain answers to certain types of queries on open specifications very efficiently.

1. Introduction

In recent years, grammatical formalisms based upon constraint-based parsing within the context of typed feature structures have become central within computational linguistics, the most prominent undoubtedly being HPSG [22]. As a result, numerous computational frameworks specifically designed for constraint-based reasoning on typed feature logics have emerged, among them ALE [5], TFS [24], and CUF [7]. Likewise, systems for representing and managing lexical information in a hierarchical fashion have appeared in recent years [3]. To function efficiently, all of these frameworks must first and foremost be capable of managing the associated type hierarchy effectively.[1]

All type hierarchies share some basic properties, such as the ability to express inheritance. However, beyond that, there is relatively little agreement as

[1] By "type hierarchy," we mean the static hierarchy of (parameterless) types which underlies the typing mechanism of the feature structures. We do not include the recursively specified types which also form an integral part of grammars which are specified using such systems. In CUF [7], the static types are called *types*, and the (often parameterized) recursive types are called *sorts*, but there is no universal agreement, and other terminology, or even the opposite terminology, is often used. Pollard and Sag [22], for example, use the term *sort* to characterize that which we call type.

to which properties are appropriate. It seems clear that these difference arise not because some systems embody the "correct" formalism while others do not, but rather because each system places distinct expectations upon the hierarchy. As with all forms of knowledge representation [19], there is a fundamental tradeoff between expressiveness and tractability in formalization of type hierarchies. In this paper, we examine aspects of the tradeoff which occurs when one moves from total specification, in which the entire hierarchy is specified explicitly, to constraint-based open specification[2] in which the actual hierarchy is not completely specified, but rather may be of the models of a set of constraints.

It is fair to say that total specification is used far more frequently than is open specification. Indeed, the only system which we know of which supports open specification is CUF.[3] It appears to be the case, at least in part, that open specification has been avoided because the computational overhead is perceived to be too high. This perception is reinforced by the fact that that the question of model existence in such a context is NP-complete [14]. Nonetheless, we feel that open specification can be a useful tool in some contexts, and it is therefore important to understand more about its representational aspects and computational complexity. In this paper, we take some steps towards this end.

The paper is organized as follows. In Sec. 2, a brief summary of the representational aspects of completely specified hierarchies is presented. Such a summary is important because even within that context, there is a critical distinction between distributive and nondistributive hierarchies which must be understood to appreciate fully the various aspects of open specification, since the latter is carried out in a distributive context. In Sec. 3, the basic ideas and structural aspects of open specification are presented. In Sec. 4, representations which are essential for the process of computing solutions to an open specification are developed. Finally, in Sec. 5, some efficient techniques for identifying properties of models in open specifications, based upon Horn clauses, are developed,

This paper may be viewed as a complement to [14]. In that work, the focus was largely upon developing the machinery necessary to show the satisfaction problem to be NP-complete. In this paper, we focus more on structural and algorithmic issues.

2. The Rôle of Distributivity

This paper is largely about the open specification of distributive hierarchies. However, to put that work in perspective, it is first necessary to identify the rea-

[2] In [21, Sec. 3.1], the term *open specification* is used in a different way, in a discussion contrasting ALE to Troll [12]. We see this terminological overloading as a non-issue, provided authors are careful to indicate which definitions are in force in their writings.

[3] In his work Aït-Kaci [1], [2] has used a crown construction to complete hierarchies which may not have all glb's. However, this technique always associates a unique completion with a specification. Thus, we regard it as a means of extracting a total specification from an incomplete specification, rather than one of open modelling.

sons why distributivity is important. In this section, we present a brief overview of the rationale for requiring distributivity in ordinary, complete hierarchies.

2.1 Bounded and distributive lattices A lattice is *bounded* if it has a greatest element $\top$ and a least element $\bot$. It is always assumed that $\top$ and $\bot$ are distinct, so under this definition, a lattice must have at least two different elements. As a notational convention, a boldface symbol (e.g., $\mathbf{L}$) will denote the entire algebraic structure, while the roman symbol L will denote just the underlying set of elements. Thus, we write $\mathbf{L} = (L, \vee, \wedge, \top, \bot)$. A lattice is *distributive* [13, p. 30] if it satisfies $(a{\vee}b){\wedge}c = (a{\wedge}c){\vee}(b{\wedge}c)$ for all elements a, b, and c.

2.2 General semantics for total type hierarchies The syntactic component of a type hierarchy is just a finite bounded lattice (not necessarily distributive). However, a type hierarchy has a semantics as well, which identifies a collection of objects, together with a specification of which objects belong to which types. More formally, let $\mathbf{L} = (L, \vee, \wedge, \top, \bot)$ be a finite bounded lattice. A *type semantics* for $\mathbf{L}$ is a pair $\mathbf{S} = (\mathfrak{U}, \mathfrak{I})$, in which $\mathfrak{U}$ is a nonempty set, called the *universe of objects*, and $\mathfrak{I} : L \to \mathbf{2}^{\mathfrak{U}}$ is a function which associates a subset of $\mathfrak{U}$ to each type in L, subject to the conditions that $\mathfrak{I}(\top) = \mathfrak{U}$, $\mathfrak{I}(\bot) = \emptyset$, and for $\tau_1, \tau_2 \in L,\ \ \tau_1 \leq \tau_2$ implies $\mathfrak{I}(\tau_1) \subseteq \mathfrak{I}(\tau_2)$. The last condition expresses the critical concept of *inheritance*: if $\tau_1 \leq \tau_2$, then every instance of τ_1 is also an instance of τ_2, and so every property which applies to an object of type τ_2 also applies to (i.e., is *inherited by*) every object of type τ_1.

The semantics $(\mathfrak{U}, \mathfrak{I})$ *separates* $\{\tau_1, \tau_2\} \in L$ if $\mathfrak{I}(\tau_1) \neq \mathfrak{I}(\tau_2)$, and is *totally separating* if it separates every pair $\{\tau_1, \tau_2\} \in L$. Note that $\{\top, \bot\}$ must be separated by any semantics $(\mathfrak{U}, \mathfrak{I})$, since $\mathfrak{U}$ is required to be nonempty.

Formally, a type hierarchy is a pair $\mathfrak{H} = (\mathbf{L}, (\mathfrak{U}, \mathfrak{I}))$ in which $\mathbf{L}$ is a finite bounded lattice and $(\mathfrak{U}, \mathfrak{I})$ is a semantics for $\mathbf{L}$.

2.3 Natural meet semantics The fundamental operation in constraint-based parsing strategies is *unification*; the unification of two objects x and y results in a an object $x \sqcup y$ which has all of the attributes of both x and y. In the context of typed unification, this means in particular that $x \sqcup y$ is of both the type of x and the type of y. Thus, if x is known to be of type τ_x, and y of type τ_y, then $x \sqcup y$ must be of type $\tau_x \wedge \tau_y$. It follows that a type hierarchy which is used for constraint-based parsing must have the following property: A type hierarchy $\mathfrak{H} = (\mathbf{L}, (\mathfrak{U}, \mathfrak{I}))$ satisfies the *natural meet semantics* if the following condition is satisfied for every $\tau_1, \tau_2 \in L$:

$$\mathfrak{I}(\tau_1 \wedge \tau_2) = \mathfrak{I}(\tau_1) \cap \mathfrak{I}(\tau_2) \qquad \text{(nat-}\wedge\text{)}$$

A lattice $\mathbf{L}$ is said to *admit natural meet semantics* if there is a semantics $(\mathfrak{U}, \mathfrak{I})$ for $\mathbf{L}$ which satisfies condition (nat-$\wedge$) above.

2.4 Existence of natural meet semantics *Every bounded lattice* $\mathbf{L} = (L, \vee, \wedge, \top, \bot)$ *admits a totally separating separating natural meet semantics.*

Proof For each $\tau \in L \setminus \bot$, associate a distinct element x_τ, and then put $\mathfrak{U} = \{x_\tau \mid \tau \in L\}$. Define $\mathfrak{I} : L \to \mathfrak{U}$ by $\tau \mapsto \{x_\sigma \mid \sigma \leq \tau\}$. It is easy to see that this semantics satisfies the condition (sem-$\wedge$), and it is totally separating by construction. $\square$

2.5 The role of distributivity In the systems ALE [5], TFS [24], and the ACQUILEX LKB [6], among others, the type hierarchies are not required to be distributive. In [4, pp. 15-17], Carpenter argues that type hierarchies should not be distributive. On the other hand, the CUF system [7] requires a distributive hierarchy. We shall now attempt to sort out how these apparently disparate points of view can exist.

Unification itself makes no use of the join operation; as outlined above, only the meet operation is used. Even though a bounded finite meet semilattice will necessarily have joins of any elements [13, Ch. 1, Sec. 3, Lem. 14], these joins are not used in the unification process, and so it is unnecessary to assign any computationally significant semantics to them. In short, only the natural meet semantics is relevant. Since the concept of distributivity can make sense only in a context in which there is a meaningful join as well as meet, distributivity plays no formal rôle in these contexts. As shown in 2.4, it is always possible to assign a complete meet semantics to a finite bounded lattice, regardless of any further properties such as distributivity or modularity.

CUF, on the other hand, was designed to support *disjunctive unification* [8], in which alternative parses are support via a meaningful semantics on the join operation. This indeed requires a distributive lattice.

2.6 Natural join semantics and natural semantics A type hierarchy $\mathfrak{H} = (\mathbf{L}, (\mathfrak{U}, \mathfrak{I}))$ satisfies the *natural join semantics* if the following condition is satisfied for every $\tau_1, \tau_2 \in L$:

$$\mathfrak{I}(\tau_1 \vee \tau_2) = \mathfrak{I}(\tau_1) \cup \mathfrak{I}(\tau_2) \qquad \text{(nat-}\vee\text{)}$$

The type hierarchy $\mathfrak{H}$ is said to satisfy the *natural semantics* if it satisfies both condition (sem-$\wedge$) and (sem-$\vee$). Similarly, a lattice $\mathbf{L}$ is said to *admit natural semantics* if there is a semantics for $\mathbf{L}$ which satisfies both (sem-$\wedge$) and (sem-$\vee$).

The definitions of *separating* and *totally separating* natural semantics are analogous to those which apply to meet semantics.

2.7 Birkhoff-Stone representation of distributive lattices A lattice $\mathbf{L}$ is called a *ring of sets* if there is a set S (called the *basis*) with the property that every $x \in L$ is a subset of S, and, furthermore, that for every $x, y \in L$, $x \vee y = x \cup y$ and $x \wedge y = x \cap y$. In other words, join is union and meet is intersection. In the case of a bounded lattice, it suffices to take $\top = S$. Such a lattice has a "built-in" semantics; let S be the universe of objects, with the set of objects associated with an element $x \in L$ just x itself.

The representation theorem of Birkhoff and Stone [13, Ch. 2, Sec. 1, Thm. 19] states that *a lattice is distributive iff it is isomorphic to a ring of sets*. It is furthermore possible to require that the ring be *nonredundant*, in the precise sense that if $s_1, s_2 \in S$, then there is some $x \in L$ which contains one of $\{s_1, s_2\}$, but not both. (Otherwise, one of $\{s_1, s_2\}$ could be removed, with the resulting lattice isomorphic to the original one.)

2.8 Existence and algorithmic aspects *Let* $\mathbf{L} = (L, \vee, \wedge, \top, \bot)$ *be an arbitrary finite bounded lattice, and let* $n = \mathsf{Card}(L)$ *denote the cardinality of* L.

(a) $\mathbf{L}$ *admits a totally separating natural semantics iff it is distributive. This question is decidable in time* $\Theta(n^3)$.

(b) *It is decidable in time* $\Theta(n^3)$ *whether or not* $\mathbf{L}$ *admits a natural semantics.*

(c) *If* $\mathbf{L}$ *admits a natural semantics, it may be constructed in time* $\Theta(n^3)$.

Proof The characterization is a consequence of the Birkhoff-Stone representation theorem. The $\Theta(n^3)$ algorithm arises from the fact that the distributive law involves triples of elements; it is only necessary to check each such triple in turn. The question of establishing a natural semantics involves construction of an appropriate quotient lattice via the congruence defined by nondistributive components. The details are not presented in this paper. The important point, however, is that the construction may be carried out in deterministic polynomial time. □

3. Open Specification of Type Hierarchies

In open specification, a constraint-based specification replaces a complete description of the hierarchy. At least one existing system, CUF [7], has taken this approach. In this section, some basic issues surrounding such specifications are examined, including condition for existence of a model, size of models of models, and characterization of canonical models. Because the principal interest in such a framework lies within the domain of distributive hierarchies, attention has been restricted to that case.

3.1 Augmentation and interpretations A set of *clean types* is one which contains neither $\top$ nor $\bot$. For such a set, define $\mathsf{Aug}_{\bot}(P) = P \cup \{\bot\}$, $\mathsf{Aug}_{\top}(P) = P \cup \{\top\}$, and $\mathsf{Aug}(P) = P \cup \{\bot, \top\}$. An *interpretation* of P is a pair $I = (\mathbf{L}, f)$ in which $\mathbf{L} = (L, \vee, \wedge, \top, \bot)$ is a bounded distributive lattice and $f : \mathsf{Aug}(P) \to L$ is a function which is subject to the conditions that $f(\bot) = \bot$ and $f(\top) = \top$. The collection of all interpretations of P is denoted $\mathsf{Interp}(P)$.

3.2 Open specifications Let P be any finite set of clean types. The system of *constraints over* P is defined to be the smallest set $\mathsf{Constraints}(P)$ satisfying the following conditions.

(c-$\leq$): If $\tau_1 \in \mathsf{Aug}_{\top}(P)$, $\tau_2 \in \mathsf{Aug}_{\bot}(P)$, then $(\tau_1 \leq \tau_2) \in \mathsf{Constraints}(P)$.

(c-$\wedge$): If $\tau \in \mathsf{Aug}(P)$ and $S \subseteq P$ is nonempty, then $(\bigwedge S = \tau) \in \mathsf{Constraints}(P)$.

(c-$\vee$): If $\tau \in \mathsf{Aug}(P)$ and $S \subseteq P$ is nonempty, then $(\bigvee S = \tau) \in \mathsf{Constraints}(P)$.

(c-$\neq$): If $\tau_1 \in P$, $\tau_2 \in \mathsf{Aug}(P)$, then $(\tau_1 \neq \tau_2) \in \mathsf{Constraints}(P)$.

(c-Atom): If $\tau \in P$, then $\mathsf{Atom}(\tau) \in \mathsf{Constraints}(P)$.

An interpretation $I = (\mathbf{L}, f)$ *satisfies* the constraint $\varphi \in \mathsf{Constraints}(P)$, written $I \models \varphi$, if the appropriate rule given below is satisfied.

(sat-$\leq$): $I \models (\tau_1 \leq \tau_2)$ iff $f(\tau_1) \leq f(\tau_2)$.

(sat-$\wedge$): $I \models (\bigwedge S = \tau)$ iff $\bigwedge\{f(\sigma) \mid \sigma \in S\} = f(\tau)$.

(sat-$\vee$): $I \models (\bigvee S = \tau)$ iff $\bigvee\{f(\sigma) \mid \sigma \in S\} = f(\tau)$.
(sat-$\neq$): $I \models (\tau_1 \neq \tau_2)$ iff $f(\tau_1) \neq f(\tau_2)$.
(sat-Atom): $I \models \mathsf{Atom}(\tau)$ iff $f(\tau)$ is an atom in $\mathbf{L}$ (i.e., $\sigma \leq f(\tau)$ implies $\sigma = f(\tau)$ or $\sigma = \bot$).

An *open specification* is a pair (P, Φ), in which P is a finite set of clean types and $\Phi \subseteq \mathsf{Constraints}(P)$. A *model* of (P, Φ) is an interpretation $I = (\mathbf{L}, f : \mathsf{Aug}(P) \to L)$ for which $I \models \varphi$ holds for each $\varphi \in \Phi$. A model $(\mathbf{L}, f)$ of (P, Φ) is *finite* if L is a finite set. The set of all models of (P, Φ) is denoted $\mathsf{Mod}(P, \Phi)$.

It is convenient to partition the constraints into two classes. Elements of the first three categories listed above ($\{\leq, \wedge, \vee\}$) are called *positive constraints*, while elements of the last two ($\{\neq, \mathsf{Atom}\}$) are called *negative constraints*. If $\Phi \subseteq \mathsf{Constraints}(P)$, then Φ^+ denotes the positive constraints over P, and Φ^- the negative constraints. Similarly, $\mathsf{Constraints}^+(P)$ (resp. $\mathsf{Constraints}^-(P)$) denotes the set of positive (resp. negative) constraints contained in $\mathsf{Constraints}(P)$.

It is important to note that other forms of constraints may be realized easily, even though they are not explicitly in this set. For example, a constraint of the form $(\tau_1 = \tau_2)$ is equivalent to the pair of constraints $\{(\tau_1 \leq \tau_2), (\tau_2 \leq \tau_1)\}$. Likewise, the constraint $(\tau_1 < \tau_2)$ may be viewed as an abbreviation for the set $\{(\tau_1 \leq \tau_2), (\tau_1 \neq \tau_2)\}$.

3.3 Canonical models In general, an open specification has a multitude of models. This situation arises for two reasons. First, it is always possible to augment a model with all sorts of extraneous types, without violating any constraints. For example, if $P_1 \subseteq P_2$, and Φ is any set of constraints dealing only with symbols in P_1, then any model of (P_2, Φ) defines a model of (P_1, Φ). Second, a model may impose a constraint which is not mandated by the specification. For example, for the specification $(\{a, b\}, \emptyset)$, a model in which $(a = b)$ holds is quite permissible, even though it is not required to hold in all models.

The question thus arises as to whether, for a given open specification (P, Φ), there is a canonical model which (a) does not introduce any extraneous types, and (b) does not force any constraints not shared by all models. The answer is "yes," provided that Φ does not contain any constraints of the form $\mathsf{Atom}(\tau)$. Such a canonical model is the *initial model*, described as follows.

Let (P, Φ) be an open specification, and let $(\mathbf{L}_1, f_1 : \mathsf{Aug}(P) \to L_1)$ and $(\mathbf{L}_2, f_2 : \mathsf{Aug}(P) \to L_2)$ be models of (P, Φ). A *morphism* $h : M_1 \to M_2$ is just a bounded lattice homomorphism $h : \mathbf{L}_1 \to \mathbf{L}_2$ with the property that $f_2 = h \circ f_1$. A model $N = (\mathbf{K}, i)$ for (P, Φ) is *initial* if, for every model $M = (\mathbf{L}, f)$ of (P, Φ), there is a unique morphism $h : N \to M$. It is a standard result from category theory that such initial constructions, when they exist, are unique up to isomorphism [16, Chap. 4, Sec. 7]. Thus, we may speak of *the* initial or canonical model. In this paper, the term *canonical model* will be taken to be synonymous with initial model. We prefer the term canonical model because it conveys a sense of its representational power, while initial model conveys a purely algebraic property.

3.4 The bounded distributive lattice of crowns over P The definition of canonical model is abstract; however, it is useful to have a basic understanding of its structural and combinatorial aspects. To this end, we provide a concrete construction, beginning with the situation $(P, \emptyset)$, in which there are no constraints. In that case, the canonical model is represented by the distributive lattice whose elements are $\top$, together with all expressions built up from elements of P using $\wedge$ and $\vee$, subject to equivalence via the distributive laws. The idea of how such expressions are represented parallels closely the idea of *disjunctive normal form* (*DNF*) from propositional logic [10, pp. 48-49]. Just as any formula in propositional logic may be converted to one in DNF (using the distributivity of the corresponding operations), so too may any expression in a distributive lattice be converted to an equivalent one in such a form. Specifically, we work with expressions of the following form,

$$(a_{11} \wedge a_{12} \wedge .. \wedge a_{1n_1}) \vee (a_{21} \wedge a_{22} \wedge .. \wedge a_{2n_2}) \vee .. \vee (a_{m1} \wedge a_{m2} \wedge .. \wedge a_{mn_m})$$

in which the a_{ij}'s are elements of P.

Such representations of individual elements in a lattice can easily be confused with expressions involving the lattice operations $\wedge$ and $\vee$. Therefore, an alternate notation is adopted. If $C_i = \{a_{ij} \mid 1 \le j \le n_i\}$, then the *set representation* of the above expression is $\{C_1, C_2, .., C_m\}$. Thus, in the set representation, each element other than $\top$ is represented by a set of subsets of elements of P.

There is one further problem; namely an element may have more than one representation. For example $(a \wedge b \wedge c) \vee (a \wedge b)$ and $(a \wedge b)$ are equivalent. To remedy this, we must disallow expressions in which the types in one of the disjuncts is a subset of those in another. This leads to a representation using *co-chains*, or *crowns.*

Let $T = \{C_1, C_2, .., C_n\}$ be a set of subsets of P. Call T a *crown* if for no two distinct indices i and j is it the case that $C_i \subseteq C_j$. Let $\mathsf{Crown}(P)$ denote the set of all crowns of P, and let $\mathsf{Crown}_\top(P)$ denote $\mathsf{Crown}(P) \cup \{\top\}$.

It is easy to see that the elements of $\mathsf{Crown}_\top(P)$ form a bounded distributive lattice, under natural operations. Specifically, the empty set $\emptyset$ is the bottom element $\perp$ of the lattice, and further operations are defined as follows.

(cr-$\vee$): $\{C_1, C_2, .., C_n\} \vee \{D_1, D_2, .., D_m\} =$
$$\mathsf{Crownify}(\{C_1, C_2, .., C_n, D_1, D_2, .., D_m\}).$$
(cr-$\wedge$): $\{C_1, C_2, .., C_n\} \wedge \{D_1, D_2, .., D_m\} =$
$$\mathsf{Crownify}(\{C_1 \cap D_j \mid 1 \le i \le n \text{ and } 1 \le j \le m\}).$$
(The operation $\mathsf{Crownify}$ converts its argument into a crown by removing all subsumed subsets.) The lattice so constructed shall be denoted $\mathsf{CrownLat}(P)$.

3.5 Theorem *For any set P, $(\mathsf{CrownLat}(P), \iota : P \to \mathsf{Crown}_\top(P))$, with $\iota : \tau \mapsto \{\{\tau\}\}$, is a canonical model over $(P, \emptyset)$.*

Proof The proof is a standard free algebra construction [13, Ch. 1, Sec. 5]. □

3.6 Example Let $P = \{a, b\}$. Then $\mathsf{Crown}(P) = \{\{\emptyset\}, \{\{a\}\}, \{\{b\}\}, \{\{a, b\}\}, \{\{a\}, \{b\}\}\}$. The corresponding free lattice is leftmost in Fig. 1. (The other two lattices will be considered in 3.10 below.) This example also illustrates why there is a distinct element assigned to be $\top$, rather than just taking $\top$ to be the largest crown consisting of all maximal subsets of P. The condition that $\top$ is the join of all elements in P is a constraint, and not a condition which must be satisfied in all models.

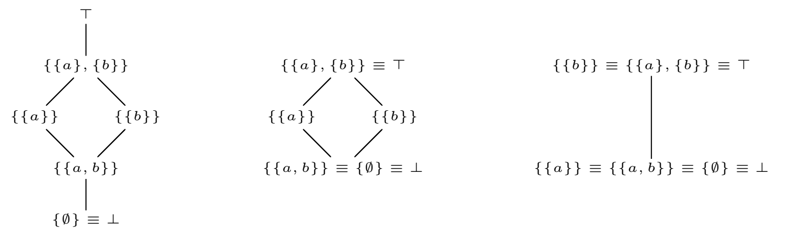
$\top$
$\{\{a\},\{b\}\}$
$\{\{a\}\}$
$\{\{b\}\}$
$\{\{a,b\}\}$
$\{\emptyset\} \equiv \bot$
$\{\{a\},\{b\}\} \equiv \top$
$\{\{a\}\}$
$\{\{b\}\}$
$\{\{a,b\}\} \equiv \{\emptyset\} \equiv \bot$
$\{\{b\}\} \equiv \{\{a\},\{b\}\} \equiv \top$
$\{\{a\}\} \equiv \{\{a,b\}\} \equiv \{\emptyset\} \equiv \bot$

Fig. 1. Three canonical lattices over $P = \{a, b\}$.

3.7 Combinatorics of the crown construction The size of the set of crowns grows very rapidly. For $P = \{a, b, c\}$, $\mathsf{Crown}_\top(P)$ has eighteen elements. Specifically, $\mathsf{Crown}(\{a,b,c\}) = \{\{\emptyset\}, \{\{a\}\}, \{\{b\}\}, \{\{c\}\}, \{\{a,b\}\}, \{\{a,c\}\}, \{\{b,c\}\}, \{\{a,b,c\}\}, \{\{a\},\{b\}\}, \{\{a\},\{c\}\}, \{\{b\},\{c\}\}, \{\{a,b\},\{c\}\}, \{\{a,c\},\{b\}\}, \{\{b,c\},\{a\}\}, \{\{a,b\},\{a,c\}\}, \{\{a,b\},\{b,c\}\}, \{\{a,c\},\{b,c\}\}, \{\{a\},\{b\},\{c\}\}\}$.

In general, the number of elements in $\mathsf{Crown}(P)$ is much greater than 2^n, although less than 2^{2^n}. Thus, *explicit construction of the canonical lattice for* (P, Φ) *is generally highly impractical.* This is true even when $\Phi \neq \emptyset$, as shown below. Generally, although the canonical lattice will be smaller, it will still be very large.

3.8 Coalescing constraints and quotient lattices We now turn to the problem of characterizing the canonical lattice over an open specification (P, Φ) in which Φ is not empty. A *congruence* on $\mathbf{L} = (L, \vee, \wedge, \top, \bot)$ is an equivalence relation $\equiv$ on L with the property that whenever $x_1 \equiv x_2$ and $y_1 \equiv y_2$ hold, then $x_1 \vee y_1 = x_2 \vee y_2$ and $x_1 \wedge y_1 = x_2 \wedge y_2$. It is well known that if (and only if) $\equiv$ is a congruence relation, the equivalence classes $L/\!\equiv$ of L form a lattice $\mathbf{L}/\!\equiv$ under the induced operations [13, p. 21].

It is straightforward to show that any set of positive constraints on P defines a congruence in a natural way. For $\tau \in P$, define $\mathsf{ECrown}(\tau) = \{\{\tau\}\}$, with $\mathsf{ECrown}(\bot) = \{\emptyset\}$ and $\mathsf{ECrown}(\top) = \top$. Then, for $\Phi \subseteq \mathsf{Constraints}^+(P)$, define $\equiv_\Phi$ to be the finest congruence relation on $\mathsf{Crown}_\top(P)$ which includes the following identifications.

($\equiv_\Phi$-$\leq$): If $(\tau_1 \leq \tau_2) \in \Phi$ with $\tau_1, \tau_2 \in P$, then $\{\{\tau_1\},\{\tau_2\}\} \equiv_\Phi \{\{\tau_2\}\}$.
If $(\top \leq \tau) \in \Phi$, then $\{\{\tau\}\} \equiv_\Phi \top$.
If $(\tau \leq \bot) \in \Phi$, then $\{\{\tau\}\} \equiv_\Phi \{\emptyset\}$.

($\equiv_\Phi$-$\wedge$): If $(\bigwedge S = \tau) \in \Phi$, then $\{S\} \equiv_\Phi \mathsf{ECrown}(\tau)$.

($\equiv_\Phi$-$\vee$): If $(\bigvee S = \tau) \in \Phi$, then $\{\{\sigma\} \mid \sigma \in S\} \equiv_\Phi \mathsf{ECrown}(\tau)$.

Define $f_{\equiv_\Phi} : \mathsf{Aug}(P) \to \mathsf{Crown}_\top(P)/\!\equiv_\Phi$ by $\tau \mapsto [\{\{\tau\}\}]_{\equiv_\Phi}$, $\bot \mapsto [\{\emptyset\}]_{\equiv_\Phi}$, and $\top \mapsto [\top]_{\equiv_\Phi}$.

We are now in a position to crystallize when and how an open specification has a model. For technical reasons, we first address the case in which there are no constraints of the form $\mathsf{Atom}(\tau)$.

3.9 Theorem — characterization of satisfiability *Let (P, Φ) be any open specification with the property that Φ contains no constraints of the form $\mathsf{Atom}(\tau)$. Then the following conditions are equivalent.*

(a) *$\mathsf{Mod}(P, \Phi)$ is nonempty.*

(b) *(P, Φ) has a finite canonical model, which is given by $(\mathsf{Crown}_{\top}(P)/{\equiv_{\Phi}},\ f_{\equiv_{\Phi}})$.*

(c) *$\equiv_{\Phi}$ contains more than one equivalence class and, for each $(\tau_1 \neq \tau_2) \in \Phi^-$, $\{\{\tau_1\}\}$ and $\{\{\tau_2\}\}$ lie in distinct equivalence classes of $\equiv_{\Phi^+}$.*

Proof First of all, assume that Φ contains only positive constraints. If $\equiv_{\Phi}$ contains only one equivalence class, then $\bot$ and $\top$ must collapse to the same element, which violates the definition of a bounded lattice. Thus, no model can exist. If there is more than one equivalence class, then the free algebra exists, as specified, using standard techniques for the construction of free lattices [13, Ch. 1, Sec. 5].

For the general case, note that an inequality constraint of the form $(\tau_1 \neq \tau_2)$ does not alter the canonical model, but it may prevent its existence. More specifically, one first computes the canonical model for the positive constraints, and then tests to see whether the inequality constraints are satisfied in that canonical model. The condition identified in (c) exactly recaptures this situation. □

3.10 Examples As in 3.6, let $P = \{a, b\}$, but now let $\Phi = \{(\bigvee\{a, b\} = \top), (\bigwedge\{a, b\} = \bot)\}$. The corresponding canonical lattice is in the middle of Fig. 1. If we add the constraint $(a \neq b)$ to Φ, this does not change the canonical model at all, since this constraint is already satisfied. Adding the constraint $(a \leq \bot)$ results in the rightmost lattice.

3.11 Dealing with atomic constraints The results of 3.9 do not address atomic constraints (i.e., $\mathsf{Atom}(\tau)$), because the situation surrounding such constraints is much more difficult. For example, let $P = \{a, b\}$, and let $\Phi = \{\mathsf{Atom}(a), \mathsf{Atom}(b)\}$. At first glance, it might appear that the leftmost lattice in Fig. 2 is an initial model for (P, Φ). However, this is not the case. The rightmost lattice in Fig. 2 also satisfies these constraints, yet it is not a homomorphic image of the one on the left, since on the left $[\{\{a\}\}] \wedge [\{\{b\}\}] = \bot$, yet on the right $[\{\{a\}\}]$ and $[\{\{b\}\}]$ are the same element, and so this value is also the meet. This type of behavior is typical of constraints such as $\mathsf{Atom}(\tau)$; initial models often do not exist.

From 3.9, we can deduce that in the absence of atomic constraints, when there is a model, there is a finite model. This is an extremely important result, since infinite hierarchies pose all sorts of conceptual and computational difficulties. Fortunately, this finite result remains valid even in the presence of atomic constraints, as shown below. It result must not be viewed as trivial or frivolous.

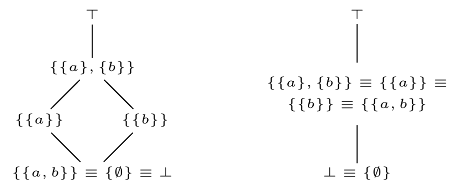

Fig. 2. Two candidates for initial lattices with constraints $\Phi = \{\mathsf{Atom}(a), \mathsf{Atom}(b)\}$.

There exist many classes of lattices (e.g., modular lattices) for which initial models may be infinite. See [13, Ch. 1, Sec. 5, Exer. 12]. From a computational point of view, it is indeed fortunate that the distributive model of a type hierarchy does not share this shortcoming, even in the presence of atomic constraints.

3.12 Theorem — finiteness of models *Let (P, Φ) be a finite open specification.*

(a) *If (P, Φ) has a model, then it has a finite model.*

(b) *If (P, Φ) has a canonical model, then this initial model is finite.*

Proof In both cases, the proof depends upon the observation that in any (not necessarily finite) distributive lattice, the sublattice generated by a finite set is itself finite. This is easily seen from the crown construction (3.4); under distributivity, there is only a finite number of distinct expressions which may be built up from a finite number of elements. Thus, if (P, Φ) has an infinite model, just extract the sublattice generated by $P \cup \{\top, \bot\}$; it is guaranteed to be finite. This also shows that any initial model must be finite, since the part not generated by $P \cup \{\top, \bot\}$ must be extraneous. □

3.13 Complementation In many frameworks, including that of CUF [7], complementation is an implicit operation. Thus, every type τ has a complementary type $\overline{\tau}$ which satisfies the conditions $\tau \vee \overline{\tau} = \top$ and $\tau \wedge \overline{\tau} = \bot$. For reasons of space limitation, we have not developed explicit results for this extended framework. However, with minor modifications, all of the theoretical results of this section carry through in the presence of complementation. Particularly, in the crown construction of 3.4, the sets of conjuncts (i.e., the C_i's) involve symbols of the form a_{ij} and of the form $\overline{a_{ij}}$, subject to the condition that no set may contain both a and $\overline{a}$. Needless to say, the size of the canonical model is even larger in the presence of implicit complements. In a general context, further information on the effect of including implicit complementation may be found in [14].

4. Basic Computational Techniques

The results on the size of the canonical model identified in 3.7 suggest that explicit construction of this lattice from an open specification is unrealistic, except in the most limited of circumstances. Fortunately, it is not generally necessary to provide an explicit construction of the whole hierarchy. Often, it is sufficient to know whether a given open specification admits a model; that is, that it is not inconsistent. However, even for that question, we have the following rather negative result, which is proven in [14].

4.1 NP-completeness *The question of whether a finite open specification* (P, Φ) *has a model is NP-complete, even when attention is restricted to positive sets of constraints.* $\square$

Despite this negative result, it is important to identify basic solution techniques. In practice, NP-complete problems are dealt with effectively all the time using a a variety of strategies. Later, we shall look at some of these approaches, but first some basic results must be established.

4.2 Two-element interpretations The two-element lattice, denoted $\mathbf{2}$, contains $\{\bot, \top\}$ as its only elements. It is trivially distributive.

Let (P, Φ) be an open specification. A *two-element interpretation* for (P, Φ) is an interpretation of the form $(\mathbf{2}, f)$; thus $f : P \to \{\top, \bot\}$. The set of all two-elements interpretations for (P, Φ) is denoted $\mathsf{Interp}_{\mathbf{2}}(P, \Phi)$. The set of two-element models is denoted $\mathsf{Mod}_{\mathbf{2}}(P, \Phi)$.

4.3 Adequacy of two-element models *A positive open specification* (P, Φ) *has a model iff it has a two-element model.*

Proof Let $(\mathbf{L},\ g : P \to L)$ be a model of (P, Φ). In view of the Birkhoff-Stone representation theorem (2.7), we may take $\mathbf{L}$ to be a ring of sets. Let s be any element from this ring which appears in some members of L, but not in all. Then, it is easily verified that $(\mathbf{2},\ g_s : P \to \{\top, \bot\})$ with $g_s : \tau \mapsto \top$ if $s \in g(\tau)$ and $g_s : \tau \mapsto \bot$ otherwise, is a two-element model. The converse is trivial. $\square$

4.4 Limitations of two-element models Two-element models are sufficient if one is interested only in positive constraints. However, they are inadequate for negative constraints. For example, let $P = \{\tau_1, \tau_2, \tau_3\}$, and let $\Phi = \{(\tau_1 \neq \tau_2), (\tau_1 \neq \tau_3), (\tau_2 \neq \tau_3)\}$. Then (P, Φ) cannot have a two-element model, because τ_1, τ_2, and τ_3 must be distinct elements in the underlying lattice. Fortunately, it is possible to combine two-element models to obtain larger models which satisfy both positive and negative constraints, as we now show.

4.5 Product interpretations and representations A key notion in the construction of models for a general set of constraints is that of the *product model.* Let (P, Φ) be an open specification, and let $\mathcal{I} = \{(\mathbf{L}_j,\ g_j : \mathsf{Aug}(P) \to L_j) \mid j \in J\}$ be a finite set of interpretations of (P, Φ). Define the product interpretation $\prod \mathcal{I}$ to be $(\prod_{j \in J} \mathbf{L}_j,\ g : \mathsf{Aug}(P) \to \prod_{j \in J} L_j)$. $\prod_{j \in J} \mathbf{L}_j$ is the product lattice, which is easily verified to be distributive.

Conversely, given an interpretation $I = (\mathbf{L},\ g : \mathsf{Aug}(P) \to L)$, it is possible to construct a product interpretation which satisfies the same constraints. In view of 2.3.3, $\mathbf{L}$ may be taken to be a nonredundant ring of sets. Let S be the basis for such a ring. For each $s \in S$, define a two-element semantics $I_s = (\mathbf{2}, g_s : \mathsf{Aug}(P) \to \{\bot, \top\})$ with $g_s : \tau \mapsto \top$ if $s \in g(\tau)$ and $\tau \mapsto \bot$ otherwise, for $\tau \in P$. Then $\prod_{s \in S} I_s$ satisfies the same constraints as I.

We are now able to provide the main characterization theorem for existence and structure of models of open specifications.

4.6 Characterization of models of arbitrary open specifications *An open specification (P, Φ) is satisfiable iff there is a finite nonempty family $\mathcal{I} \subseteq \mathsf{Interp}_{\mathbf{2}}(P, \Phi)$ satisfying the following conditions.*

(a) *Each $\varphi \in \Phi^+$ is satisfied by every $I \in \mathcal{I}$.*

(b) *Each $\varphi \in \Phi^-$ of the form $(\tau_1 \neq \tau_2)$ is satisfied by least one $I \in \mathcal{I}$.*

(b) *Each $\varphi \in \Phi^-$ of the form $\mathsf{Atom}(\tau)$ is satisfied by exactly one $I \in \mathcal{I}$.*

□

4.7 Model characterization via satisfiability in propositional logic We now turn to the question of computing two-element models for (P, Φ). The simplest and most direct approach is to reduce the problem to one of satisfiability in a propositional logic. Associate with P a propositional logic whose propositional letters are $\{\mathfrak{r}_\tau \mid \tau \in \mathsf{Aug}(P)\}$. To each $\varphi \in \mathsf{Constraints}(P)$ is associated a propositional formula $\mathfrak{F}(\varphi)$, according to the table below. For a set Φ of constraints, define $\mathfrak{F}(\Phi) = \{\mathfrak{F}(\varphi) \mid \varphi \in \Phi\}$.

Constraint φ	Associated Logical Formula $\mathfrak{F}(\varphi)$
$\tau_1 \leq \tau_2$	$\mathfrak{r}_{\tau_1} \Rightarrow \mathfrak{r}_{\tau_2}$
$\bigvee_{i=1}^n \tau_i = \tau$	$\mathfrak{r}_{\tau_1} \vee \mathfrak{r}_{\tau_2} \vee \ldots \vee \mathfrak{r}_{\tau_n} \Leftrightarrow \mathfrak{r}_\tau$
$\bigwedge_{i=1}^n \tau_i = \tau$	$\mathfrak{r}_{\tau_1} \wedge \mathfrak{r}_{\tau_2} \wedge \ldots \wedge \mathfrak{r}_{\tau_n} \Leftrightarrow \mathfrak{r}_\tau$
$\tau_1 \neq \tau_2$	$\neg(\mathfrak{r}_{\tau_1} \Leftrightarrow \mathfrak{r}_{\tau_2})$
$\mathsf{Atom}(\tau)$	$\mathfrak{r}_\tau$

Each *stable* interpretation I of the propositional logic (i.e., a truth assignment to each proposition, with $\mathfrak{r}_\top$ true and $\mathfrak{r}_\bot$ false) naturally defines a two-element interpretation $\mathfrak{A}^{-1}(I) = (\mathbf{2}, g_I : \mathsf{Aug}(P) \to \{\bot, \top\})$ of P via $g_I : \tau \mapsto \top$ if $\mathfrak{r}_\tau$ is true in I, and $\tau \mapsto \bot$ otherwise. Thus, *the two-element models of (P, Φ) are in natural bijective correspondence with the stable models of $\mathfrak{F}(\Phi)$.*

4.8 Queries Given an open specification (P, Φ), we may wish to know whether certain other constraints are implied by this specification. For example, if Φ is the partial specification of a hierarchy, we may wish to know whether Φ forces $\tau_1 = \tau_2$ to hold. A *query* is a question of the form "Does $\Phi \models \varphi$ hold?" with $\Phi \subseteq \mathsf{Constraints}(P)$ and $\varphi \in \mathsf{Constraints}(P)$. This question could can be posed formally as the two queries $\Phi \models (\tau_1 \leq \tau_2)$ and $\Phi \models (\tau_2 \leq \tau_1)$. Unfortunately, the complexity of answering queries is co-NP-complete [11, Sec. 7.1].

4.9 Theorem – complexity of query processing *Let P be any finite set. The question of whether $\Phi \models \varphi$ for $\Phi \subseteq \mathsf{Constraints}(P)$ and $\varphi \in \mathsf{Constraints}(P)$ is co-NP-complete, in the size of P.*

Proof First of all, note that $\Phi \models \varphi$ holds iff $\Phi \cup \{\neg\varphi\}$ is unsatisfiable, where $\neg\varphi$ is the negation of the constraint φ, with the obvious semantics. The set $\mathsf{Constraints}(P)$, together with logical negations of its elements, will be called the set of *extended constraints* over P.

Now deciding whether a set of extended constraints is satisfiable is at least as difficult as deciding whether or not a set of ordinary constraints (i.e., a subset of $\mathsf{Constraints}(P)$) is satisfiable. Thus, in view of 4.1, it is NP-hard. On the other hand, it is also in NP, since we may guess at a solution and then test it in linear time. Thus, the problem of deciding the *satisfiability* of $\Phi \cup \{\varphi\}$ is NP-complete. Consequently, the problem of deciding the *un*satisfiability of such a set is co-NP-complete. □

5. Efficient Consistency Checking using Horn Clauses

The results presented in the previous section paint a fairly negative picture of the tractability issues surrounding maintenance of type hierarchies which are openly specified. In particular, it may not be feasible to conduct a complete check of the admissibility of a specification. Under such circumstances, there are two tacts which may be taken. First, one may look for an efficient strategy which solves a limited number of problem instances completely. Second, one may look for an efficient strategy which works on any problem instance, but yields only partial information. In this section, we take the latter approach.

Horn clauses form an important class of sentences for a variety of reasons [20]. Particularly, in propositional logic, they admit very efficient inference mechanisms. While the best known inference algorithms for general propositional logic run in exponential time in the worst case, Horn-clause inference may be performed in linear time [9], [15]. In that which follows, we show how to use a Horn-clause formulation to detect forced equivalences in open specifications (that is, constraint sets Φ for which $\Phi \models (\tau_1 = \tau_2)$). The result is not a mere query mechanism, but a procedure which generates a list of such equivalences. This technique improves substantially upon an earlier one presented in [14, Sec. 2.1], in simplicity, in improved computational complexity, and in extensibility.

5.1 Horn clauses In a propositional logic, a *Horn clause* is one in which at most one of the literals is positive. Usually, we will write the Horn clause $\neg p_1 \vee \neg p_2 \vee .. \vee \neg p_n \vee q$ in rule form, as $p_1 \wedge p_2 \wedge .. \wedge p_n \Rightarrow q$. If there is no positive literal, we will write $p_1 \wedge p_2 \wedge .. \wedge p_n \Rightarrow \mathbf{F}$, with $\mathbf{F}$ representing the proposition which is always false. Simple clauses consisting of one positive literal (e.g., p) are referred to as *facts*; the atom name itself will represent such clauses. The empty clause will be represented by $\mathbf{F}$.

For any set Ψ of Horn clauses, define $\mathsf{Facts}(\Psi)$ to be the set of all facts which are semantic consequences of Ψ. $\mathsf{Facts}(\Psi)$ may be computed in time which is linear in the size of the clause set [9], [15].

We will also work with conjunctions of Horn clauses, as though they were clauses themselves. Thus, a formula such as $(p \wedge q \Rightarrow r \wedge s)$ is to be regarded as an abbreviation for the conjunction $(p \wedge q \Rightarrow r) \wedge (p \wedge q \Rightarrow s)$.

5.2 Exclusive-or constraints As noted above, inference on sets of Horn clauses is very fast; the best algorithms are linear in the size of the input set. Since the problem of determining satisfiability of an open specification is NP-complete (see 4.1 above), we certainly cannot expect Horn clauses to be a vehicle for the complete description of open specifications. Rather, some additional forms of representation must be employed to recapture completely such specifications.

A propositional formula of the form $p \oplus q$ is called an *exclusive-or constraint*, or *xor-constraint*, for short. The formula $p \oplus q$ is equivalent to $(p \vee q) \wedge (\neg p \vee \neg q)$. Such a constraint expresses the restriction that exactly one of two alternatives must be true. In the work reported in this section, the constraints identified in the table of 4.7 will be re-expressed using a combination of Horn clauses and xor-constraints. Such a representation carries the advantage that the "tractable" part of the representation (the Horn clauses) is completely separated from the "intractable" part (the xor-constraints). Effective computational techniques may then focus on the Horn part, looking for inconsistencies in the specification. While such a technique will not find all inconsistencies, it can find many, as we shall see.

5.3 The Horn clauses associated with an open specification Let P be any finite set of clean types. Define two $\mathsf{Aug}(P)$-indexed sets of propositions, as follows: $\mathsf{Prop}_{\Uparrow}(P) = \{\mathfrak{p}_\tau \mid \tau \in \mathsf{Aug}(P)\}$; $\mathsf{Prop}_{\Downarrow}(P) = \{\mathfrak{q}_\tau \mid \tau \in \mathsf{Aug}(P)\}$. $\mathsf{Prop}_{\Updownarrow}(P)$ denotes $\mathsf{Prop}_{\Uparrow}(P) \cup \mathsf{Prop}_{\Downarrow}(P)$. Relative to a two-element interpretation $I = (\mathbf{2}, f)$, think of $\mathfrak{p}_\tau$ as representing the statement $f(\tau) = \top$, and $\mathfrak{q}_\tau$ representing $f(\tau) = \bot$.

Now let Φ be a set of constraints over P. For each $\varphi \in \Phi$, associate two sets of Horn clauses, over $\mathsf{Prop}_{\Uparrow}(P)$ and $\mathsf{Prop}_{\Downarrow}(P)$, according to the table below.

φ	$\mathsf{Rules}_{\Uparrow}(\{\varphi\})$	$\mathsf{Rules}_{\Downarrow}(\{\varphi\})$
$(\tau_1 \leq \tau_2)$	$(\mathfrak{p}_{\tau_1} \Rightarrow \mathfrak{p}_{\tau_2})$	$\mathfrak{q}_{\tau_2} \Rightarrow \mathfrak{q}_{\tau_1}$
$(\bigvee_{i=1}^{n} \tau_i = \tau)$	$(\mathfrak{p}_{\tau_1} \Rightarrow \mathfrak{p}_{\tau}),\ ..,\ (\mathfrak{p}_{\tau_n} \Rightarrow \mathfrak{p}_{\tau})$	$(\mathfrak{q}_{\tau} \Rightarrow \mathfrak{q}_{\tau_1}),\ ..,\ (\mathfrak{q}_{\tau} \Rightarrow \mathfrak{q}_{\tau_n}),$ $(\mathfrak{q}_{\tau_1} \wedge \mathfrak{q}_{\tau_2} \wedge .. \wedge \mathfrak{q}_{\tau_n} \Rightarrow \mathfrak{q}_{\tau})$
$(\bigwedge_{i=1}^{n} \tau_i = \tau)$	$(\mathfrak{p}_{\tau} \Rightarrow \mathfrak{p}_{\tau_1}),\ ..,\ (\mathfrak{p}_{\tau} \Rightarrow \mathfrak{p}_{\tau_n}),$ $(\mathfrak{p}_{\tau_1} \wedge \mathfrak{p}_{\tau_2} \wedge .. \wedge \mathfrak{p}_{\tau_n} \Rightarrow \mathfrak{p}_{\tau})$	$(\mathfrak{q}_{\tau_1} \Rightarrow \mathfrak{q}_{\tau}),\ ..,\ (\mathfrak{q}_{\tau_n} \Rightarrow \mathfrak{q}_{\tau})$
$(\tau_1 \neq \tau_2)$	$(\mathfrak{p}_{\tau_1} \wedge \mathfrak{p}_{\tau_2} \Rightarrow \mathbf{F})$	$(\mathfrak{q}_{\tau_1} \wedge \mathfrak{q}_{\tau_2} \Rightarrow \mathbf{F})$
$\mathsf{Atom}(\tau)$	$(\mathfrak{p}_{\tau})$	$(\mathfrak{q}_{\tau} \Rightarrow \mathbf{F})$

For a set $\Phi \subseteq \mathsf{Constraints}(P)$, define

$$\mathsf{Rules}_{\Uparrow}(\Phi) = \{(\mathfrak{p}_\top),\ (\mathfrak{p}_\bot \Rightarrow \mathbf{F})\} \cup (\bigcup_{\varphi \in \Phi} \mathsf{Rules}_{\Uparrow}(\{\varphi\}))$$

$$\mathsf{Rules}_{\Downarrow}(\Phi) = \{(\mathfrak{q}_\bot),\ (\mathfrak{q}_\top \Rightarrow \mathbf{F})\} \cup (\bigcup_{\varphi \in \Phi} \mathsf{Rules}_{\Downarrow}(\{\varphi\})).$$

These sets of clauses are called the $\Uparrow$-*rules* (resp. $\Downarrow$-*rules)* for Φ. The notation is suggestive of the semantics of these rules. The $\Uparrow$-rules express closure conditions on the elements of P which must be true in a two-element model $I = (\mathbf{2}, f)$, while the $\Downarrow$-rules express similar conditions on elements which must be false. Consider the generic constraint $(\bigvee_{i=1}^{n} \tau_i = \tau)$. If it holds, several things are implied. First of all, for each i, $\tau_i \leq \tau$. Thus, for any i, if $f(\tau_i) = \top$, then it must be that $f(\tau) = \top$ also. This condition is recaptured by the rule $(\mathfrak{p}_{\tau_i} \Rightarrow \mathfrak{p}_{\tau})$. Similarly, if $f(\tau) = \bot$, then $f(\tau_i) = \bot$ must hold as well, and this is recaptured by the rule $(\mathfrak{q}_{\tau} \Rightarrow \mathfrak{q}_{\tau_i})$. Furthermore, if $f(\tau_i) = \bot$ for each i, then the join condition mandates that $f(\tau) = \bot$ also; this is recaptured by the rule $(\mathfrak{q}_{\tau_1} \wedge \mathfrak{q}_{\tau_2} \wedge .. \wedge \mathfrak{q}_{\tau_n} \Rightarrow \mathfrak{q}_{\tau})$. Note that there is no corresponding $\Uparrow$-rule in this case. The situation for a constraint of the form $(\bigwedge_{i=1}^{n} \tau_i = \tau)$ is completely analogous.

The rules in $\{(\mathfrak{p}_{\top}),\ (\mathfrak{p}_{\bot} \Rightarrow \mathbf{F})\}$ and in $\{(\mathfrak{q}_{\bot}),\ (\mathfrak{q}_{\top} \Rightarrow \mathbf{F})\}$ are called *bound rules*, because they assert that $f(\top) = \top$ and $f(\bot) = \bot$, respectively, for any two-element model $(\mathbf{2}, f)$ of (P, Φ).

In addition to the $\Uparrow$-rules and $\Downarrow$-rules, there are xor-constraints which assert that for any $\tau \in P$, exactly one of $f(\tau) = \bot$ and $f(\tau) = \bot$ must hold for any given two-element model $(\mathbf{2}, f)$. The *XOR constraints*, *rule set*, and *total representation* are defined as follows.

$$\begin{aligned}
\mathsf{XOR}(P) &= \{\mathfrak{p}_{\tau} \oplus \mathfrak{q}_{\tau} \mid \tau \in P\} \\
\mathsf{Rules}(\Phi) &= \mathsf{Rules}_{\Uparrow}(\Phi) \cup \mathsf{Rules}_{\Downarrow}(\Phi) \\
\mathsf{TotalRep}(P, \Phi) &= \mathsf{Rules}(\Phi) \cup \mathsf{XOR}(P)
\end{aligned}$$

Finally, given a two-element interpretation $I = (\mathbf{2}, f)$ of (P, Φ), define the *fact set* of I as $\mathsf{FactSet}(f) = \{\mathfrak{p}_x \mid x \in f^{-1}(\top)\} \cup \{\mathfrak{q}_x \mid x \in f^{-1}(\bot)\}$. Then, using the semantics which have been outlined above, it is easy to establish the following alternative to 4.7.

5.4 Characterization of two-element models *Let (P, Φ) be an open specification, and let $I = (\mathbf{2}, f) \in \mathsf{Interp}_{\mathbf{2}}(P, \Phi)$. Then $I \in \mathsf{Mod}_{\mathbf{2}}(P, \Phi)$ iff the set $\mathsf{FactSet}(f) \cup \mathsf{TotalRep}(P, \Phi)$ of propositional formulas is satisfiable.* $\Box$

5.5 Example Let $P = \{\kappa_i \mid 1 \leq i \leq 6\}$, and let $\Phi = \{(\bigvee\{\kappa_1, \kappa_2\} = \kappa_5), (\bigvee\{\kappa_2, \kappa_3\} = \kappa_5), (\bigvee\{\kappa_1, \kappa_3\} = \kappa_4), (\bigwedge\{\kappa_2, \kappa_3\} = \kappa_6), (\bigwedge\{\kappa_1, \kappa_6\} = \bot), ((\kappa_5 \leq \kappa_4)\}$. The table below shows the associated rules.

$\mathsf{Rules}_{\Uparrow}(\Phi)$	$\mathsf{Rules}_{\Downarrow}(\Phi)$
$\mathfrak{p}_{\kappa_1} \Rightarrow \mathfrak{p}_{\kappa_4} \wedge \mathfrak{p}_{\kappa_5}$	$\mathfrak{q}_{\kappa_2} \Rightarrow \mathfrak{q}_{\kappa_6}$
$\mathfrak{p}_{\kappa_2} \Rightarrow \mathfrak{p}_{\kappa_5}$	$\mathfrak{q}_{\kappa_3} \Rightarrow \mathfrak{q}_{\kappa_6}$
$\mathfrak{p}_{\kappa_3} \Rightarrow \mathfrak{p}_{\kappa_4} \wedge \mathfrak{p}_{\kappa_5}$	$\mathfrak{q}_{\kappa_4} \Rightarrow \mathfrak{q}_{\kappa_1} \wedge \mathfrak{q}_{\kappa_3} \wedge \mathfrak{q}_{\kappa_5}$
$\mathfrak{p}_{\kappa_5} \Rightarrow \mathfrak{p}_{\kappa_4}$	$\mathfrak{q}_{\kappa_5} \Rightarrow \mathfrak{q}_{\kappa_1} \wedge \mathfrak{q}_{\kappa_2} \wedge \mathfrak{q}_{\kappa_3}$
$\mathfrak{p}_{\kappa_6} \Rightarrow \mathfrak{p}_{\kappa_2} \wedge \mathfrak{p}_{\kappa_3}$	$\mathfrak{q}_{\kappa_1} \wedge \mathfrak{q}_{\kappa_2} \Rightarrow \mathfrak{q}_{\kappa_5}$
$\mathfrak{p}_{\kappa_2} \wedge \mathfrak{p}_{\kappa_3} \Rightarrow \mathfrak{p}_{\kappa_6}$	$\mathfrak{q}_{\kappa_2} \wedge \mathfrak{q}_{\kappa_3} \Rightarrow \mathfrak{q}_{\kappa_5}$
$\mathfrak{p}_{\kappa_1} \wedge \mathfrak{p}_{\kappa_6} \Rightarrow \mathfrak{p}_{\bot}$	$\mathfrak{q}_{\kappa_1} \wedge \mathfrak{q}_{\kappa_3} \Rightarrow \mathfrak{q}_{\kappa_4}$
$\mathfrak{p}_{\top}$	$\mathfrak{q}_{\top} \Rightarrow \mathbf{F}$
$\mathfrak{p}_{\bot} \Rightarrow \mathbf{F}$	$\mathfrak{q}_{\bot}$

Note that the rules $(\mathfrak{p}_{\bot} \Rightarrow \mathfrak{p}_{\kappa_1})$, $(\mathfrak{p}_{\bot} \Rightarrow \mathfrak{p}_{\kappa_6})$, $(\mathfrak{q}_{\kappa_1} \Rightarrow \mathfrak{q}_{\top})$, and $(\mathfrak{q}_{\kappa_6} \Rightarrow \mathfrak{q}_{\top})$ are not included in the table, even though they are formally members of the appropriate rule set. Since $\mathfrak{p}_\perp$ is always false, and $\mathfrak{q}_\top$ is always true, they are trivial tautologies, and so may be omitted.

Next, define the two-element interpretations $I_j = (\mathbf{2}, f_j)$ for $1 \leq j \leq 7$ according to the following table.

j	$f_j^{-1}(\top)$	$f_j^{-1}(\perp)$
1	$\{\kappa_3, \kappa_4, \kappa_5\}$	$\{\kappa_1, \kappa_2, \kappa_6\}$
2	$\{\kappa_2, \kappa_4, \kappa_5\}$	$\{\kappa_1, \kappa_3, \kappa_6\}$
3	$\emptyset$	$\{\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5, \kappa_6\}$
4	$\{\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5, \kappa_6\}$	$\emptyset$
5	$\{\kappa_1, \kappa_4\}$	$\{\kappa_2, \kappa_3, \kappa_5, \kappa_6\}$
6	$\{\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5\}$	$\{\kappa_6\}$
7	$\{\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5, \kappa_6\}$	$\emptyset$

I_1, I_2, and I_3, are easily verified to be models of (P, Φ). Indeed, it is not difficult to see that they are the only two-element models. On the other hand, I_4 fails to be a model, since $(\mathfrak{p}_{\kappa_1} \wedge \mathfrak{p}_{\kappa_6} \Rightarrow \mathfrak{p}_\perp)$ would then mandate that $\perp \in f_4^{-1}(\top)$, an impossibility. I_5 fails to be a model of (P, Φ), since the rule $(\mathfrak{p}_{\kappa_1} \Rightarrow \mathfrak{p}_{\kappa_4} \wedge \mathfrak{p}_{\kappa_5})$ mandates that $\kappa_5 \in f_5^{-1}(\top)$, contradicting $\kappa_5 \in f_5^{-1}(\perp)$. Similarly, I_6 is not a model of (P, Φ), since the rule $(\mathfrak{p}_{\kappa_2} \wedge \mathfrak{p}_{\kappa_3} \Rightarrow \mathfrak{p}_{\kappa_6})$ mandates that $\kappa_6 \in f_6^{-1}(\top)$, contradicting $\kappa_6 \in f_6^{-1}(\perp)$. Finally, I_7 is not a model, since the rule $(\mathfrak{p}_{\kappa_1} \wedge \mathfrak{p}_{\kappa_6} \Rightarrow \mathfrak{q}_\perp)$ requires that at least one of $\{\kappa_1, \kappa_6\}$ lie in $f_7^{-1}(\perp)$.

Now let $\Phi' = \Phi \cup \{(\kappa_4 \neq \kappa_5)\}$. Then $\mathsf{TotalRep}(P, \Phi') = \mathsf{TotalRep}(P, \Phi) \cup \{(\mathfrak{p}_{\kappa_4} \wedge \mathfrak{p}_{\kappa_5} \Rightarrow \mathbf{F}), (\mathfrak{q}_{\kappa_4} \wedge \mathfrak{q}_{\kappa_5} \Rightarrow \mathbf{F})\}$. It is easy to see that (P, Φ) has no model. Indeed, $\{(\mathfrak{q}_{\kappa_5} \Rightarrow \mathfrak{q}_{\kappa_1} \wedge \mathfrak{q}_{\kappa_2} \wedge \mathfrak{q}_{\kappa_3}), (\mathfrak{q}_{\kappa_1} \wedge \mathfrak{q}_{\kappa_3} \Rightarrow \mathfrak{q}_{\kappa_4})\} \models (\mathfrak{q}_{\kappa_5} \Rightarrow \mathfrak{q}_{\kappa_4})$, and since $(\mathfrak{q}_{\kappa_4} \Rightarrow \mathfrak{q}_{\kappa_5})$ also holds, this means that for any two-element model $(\mathbf{2}, f)$, $\kappa_4 \in f^{-1}(\perp)$ iff $\kappa_5 \in f^{-1}(\perp)$. Thus, $f(\kappa_4) = f(\kappa_5)$. This is impossible, hence there can be no model of (P, Φ).

5.6 Static consistency checking Let (P, Φ) be an open specification. For any set $X \subseteq \mathsf{Prop}_{\Updownarrow}(P)$, $\mathsf{Facts}(X \cup \mathsf{Rules}(\Phi))$ may be computed very efficiently — in time proportional to the size of Φ. The idea behind static consistency checking is to compute $\mathsf{Facts}(X \cup \mathsf{Rules}(\Phi))$ for each member X of a judiciously chosen set of subsets of $\mathsf{Prop}_{\Updownarrow}(P)$. From this computation, many properties of solutions may be detected. One such example is provided by (P, Φ) of 5.5. The condition $(\kappa_4 = \kappa_5)$ holds in every model, as the failure of (P, Φ') to have a model confirms. In 5.5, this failure was shown via a direct proof, which essentially posed $(\kappa_4 = \kappa_5)$ as a query. To check each such condition separately would require a great deal of computational resources. With a static consistency check, on the other hand, we can identify a large number of such conditions at one time. We now develop the machinery to perform such checks systematically. First, observe the following result, which follows immediately from the definition of the fact set.

5.7 Utility of fact closure *Let (P, Φ) be an open specification, and let $X \subseteq$ $\mathsf{Prop}_{\Updownarrow}(P)$. Then* $\mathsf{Rules}(\Phi) \models (\bigwedge X) \Rightarrow (\bigwedge \mathsf{Facts}(X \cup \mathsf{Rules}(\Phi)))$. *In other words, the process of computing* $\mathsf{Facts}(X \cup \mathsf{Rules}(\Phi))$ *may essentially be viewed as one of* *applying the rules in* $\mathsf{Rules}(\Phi)$ *to* X*. (Note:* $\bigwedge$ *denotes logical conjunction here, not lattice join.)* □

5.8 Aggregate complexity for fact closure *Let* (P, Φ) *be an open specification, and let* $\mathbf{S}$ *be a set of subsets of* $\mathsf{Prop}_{\Updownarrow}(P)$*. Then the set of sets* $\{\mathsf{Facts}(S \cup \mathsf{Rules}(\Phi)) \mid S \in \mathbf{S}\}$ *may be computed in time* $\Theta(n + r{\cdot}s + r{\cdot}\log(r))$*, with* n *the cardinality of* P*,* s *is the number of set in* $\mathbf{S}$*, and* r *is the sum of the lengths of the rules in* $\mathsf{Rules}(\Phi)$*).*

Proof The proof rests largely upon results found in [9] and [15]. The process is broken into two steps. There is a total of $2(n+2)$ propositions in $\mathsf{Prop}_{\Uparrow}(P) \cup \mathsf{Prop}_{\Downarrow}(P)$. Assign each proposition a natural-number tag in $\{0, .., 2n+3\}$. This takes time $\Theta(n)$. Next, sort the propositions in each antecedent set of each clause. This takes time $\Theta(r{\cdot}\log(r))$.

After this preconditioning, the computation of each $\mathsf{Facts}(S \cup \mathsf{Rules}(\Phi))$ takes just $\Theta(r)$ time, using the techniques in the above-cited references. The total time for all elements of S is thus $\Theta(s{\cdot}r)$. Combining these, the running time for the entire algorithm is $\Theta(n + r{\cdot}\log(r) + s{\cdot}r)$. □

5.9 Singleton Associations Let (P, Φ) be an open specification, and let $p \in \mathsf{Prop}_{\Updownarrow}(P)$. The *singleton associates* of p, denoted $\mathsf{SingAsc}(p)$, is just $\mathsf{Facts}(\{p\} \cup \mathsf{Rules}(\Phi))$. In words, $\mathsf{SingAsc}(p)$ is the set of all propositions in $\mathsf{Prop}_{\Updownarrow}(P)$ which can be deduced from p alone, using the rules in $\mathsf{Rules}(\Phi)$.

5.10 Example Let (P, Φ) be as in 5.5. Here are the singleton associates.

x	$\mathsf{SingAsc}(\mathfrak{p}_x)$	$\mathsf{SingAsc}(\mathfrak{q}_x)$
κ_1	$\{\mathfrak{p}_{\kappa_1}, \mathfrak{p}_{\kappa_4}, \mathfrak{p}_{\kappa_5}\}$	$\{\mathfrak{q}_{\kappa_1}\}$
κ_2	$\{\mathfrak{p}_{\kappa_2}, \mathfrak{p}_{\kappa_4}, \mathfrak{p}_{\kappa_5}\}$	$\{\mathfrak{q}_{\kappa_2}, \mathfrak{q}_{\kappa_6}\}$
κ_3	$\{\mathfrak{p}_{\kappa_3}, \mathfrak{p}_{\kappa_4}, \mathfrak{p}_{\kappa_5}\}$	$\{\mathfrak{q}_{\kappa_3}, \mathfrak{q}_{\kappa_6}\}$
κ_4	$\{\mathfrak{p}_{\kappa_4}\}$	$\{\mathfrak{q}_{\kappa_1}, \mathfrak{q}_{\kappa_3}, \mathfrak{q}_{\kappa_4}, \mathfrak{q}_{\kappa_5}\}$
κ_5	$\{\mathfrak{p}_{\kappa_4}, \mathfrak{p}_{\kappa_5}\}$	$\{\mathfrak{q}_{\kappa_1}, \mathfrak{q}_{\kappa_2}, \mathfrak{q}_{\kappa_3}, \mathfrak{q}_{\kappa_4}, \mathfrak{q}_{\kappa_5}\}$
κ_6	$\{\mathfrak{p}_{\kappa_2}, \mathfrak{p}_{\kappa_3}, \mathfrak{p}_{\kappa_6}\}$	$\{\mathfrak{q}_{\kappa_6}\}$
$\bot$	$\{\mathfrak{p}_{\bot}\}$	$\{\mathfrak{q}_{\bot}\}$
$\top$	$\{\mathfrak{p}_{\top}\}$	$\{\mathfrak{q}_{\top}\}$

The information that $(\kappa_4 = \kappa_5)$ is easily recovered from these data. Indeed, note that $\mathfrak{q}_{\kappa_5} \in \mathsf{SingAsc}(\mathfrak{p}_{\kappa_4})$ and that $\mathfrak{q}_{\kappa_4} \in \mathsf{SingAsc}(\mathfrak{p}_{\kappa_5})$. In light of 5.7, this means that both $(\mathfrak{q}_{\kappa_4} \Rightarrow \mathfrak{q}_{\kappa_5})$ and $(\mathfrak{q}_{\kappa_5} \Rightarrow \mathfrak{q}_{\kappa_4})$ are logical consequences of $\mathsf{Rules}(\Phi)$. Thus, for any $(\mathbf{2}, f) \in \mathsf{Mod}_{\mathbf{2}}(\Phi)$, $f(\kappa_4) = \bot$ iff $f(\kappa_4) = \bot$. Thus, it must be the case that $f(\kappa_4) = f(\kappa_5)$. We now develop a means of discovering such associations systematically.

5.11 The static equivalence Define the relation $\preceq^1_\Phi$ on $\mathsf{Prop}_{\Updownarrow}(\Phi)$ by $p \preceq^1_\Phi q$ iff $q \in \mathsf{SingAsc}(p)$. The transitive closure of $\preceq^1_\Phi$ is denoted by $\overline{\preceq^1_\Phi}$. The equivalence relation $\equiv^1_\Phi$ places into a single equivalence class all elements which lie in the same cycle in $\overline{\preceq^1_\Phi}$. Formally, $p \equiv^1_\Phi q$ iff $p \overline{\preceq^1_\Phi} q$ and $q \overline{\preceq^1_\Phi} p$.

In the above example, only $\mathfrak{q}_{\kappa_4} \equiv^1_\Phi \mathfrak{q}_{\kappa_5}$.

5.12 The complexity of determining static equivalence *Let (P, Φ) be an open specification. Then, with n, r, and s defined as in 5.8, there is an algorithm which computes $\equiv^1_\Phi$ from (P, Φ) in worst-case time $\Theta(n^3 + n \cdot r + r \cdot \log(r))$.*

Proof There are $2n + 4$ $(= O(n))$ distinct sets of the form $\mathsf{SingAsc}(p)$, two for each element of $\mathsf{Aug}(P)$. Transitive closure has the same computational complexity as matrix multiplication [18, 10.3.6]; thus, the equivalence relation $\equiv^1_\Phi$ may be computed in time $\Theta(n^3)$ from $\{\mathsf{Facts}(\{p\} \cup \mathsf{Rules}(\Phi)) \mid p \in \mathsf{Prop}_{\Updownarrow}(P)\}$. Thus, in view of 5.8, the total complexity is $\Theta(n^3) + \Theta(n + r \cdot s + r \cdot \log(r)) = \Theta(n^3 + n \cdot r + r \cdot \log(r))$. □

The above bound is a substantial improvement over $\Theta(n^4 \cdot r \cdot \log(r))$ which was reported for the algorithm in [14, Sec. 2.1]. The general idea also lends itself to extension, as outlined below.

5.13 Higher-level static consistency checking Although the technique just described detects many element equivalences, it cannot find them all. As a concrete example, consider the open specification (P, Φ) with $P = \{\tau_i \mid 1 \leq i \leq 5\}$, and $\Phi = \{(\tau_i \vee \tau_j = \tau_4) \mid 1 \leq i < j \leq 3\} \cup \{(\tau_i \wedge \tau_j = \tau_5) \mid 1 \leq i < j \leq 3\}$. It is not difficult to see that all five elements of P must be collapsed to the same value in any model. However, this fact is not detected by the static equivalence algorithm of 5.11. It can, however, be detected with a higher-level static consistency check, which works with sets of the form $\mathsf{Facts}(X, \Phi)$, with X a subset of $\mathsf{Prop}_{\Updownarrow}(P)$ of size at most two. Thus, the technique of static consistency checking may be extended. Indeed, if we work with all sets of the form $\mathsf{Facts}(X, \Phi)$ for X any subset of $\mathsf{Prop}_{\Updownarrow}(P)$, then consistency checking may be made complete. Unfortunately, the computational complexity which results when all subsets of P are considered yields no advantage over direct satisfaction testing. What may prove promising, though, is to work with all subsets of a small size bound, say two or three. The complexity is still quite manageable, yet many inconsistencies may be detected. This approach is not elaborated further here.

6. Conclusions and Further Directions

Open specification clearly imposes a substantial computational burden. Therefore, a decision to employ it must be measured carefully. As implied by the work in Sec. 2, the first question to ask is whether or not natural semantics are needed for both meet and join. If only natural meet semantics is needed, then open specification, as described in this paper, is not an issue.[4] However, if one is interested manipulating general classes of representations which involve disjunction, then it may be a necessity, although alternatives to managing limited disjunction have been proposed and implemented [12]. We cannot and do not address the

[4] It is unclear whether the idea of open specification of meet-only hierarchies is interesting, or nontrivial. We know of no existing work on the topic, and no systems which embody the idea.

adequacy of such approaches rather we proceed under the assumption that join semantics, and hence distributive hierarchies, are desired.

As illustrated by the construction of 3.7, complete representation of a distributive hierarchy will generally be infeasible. (See also [14, Sec. 0] for some examples.) Therefore, it would seem that open specification is the only alternative. Although we have presented an efficient method for determining certain implied constraints in Sec. 5, such techniques, by themselves, cannot counterbalance the NP-hardness of the underlying problems. Rather, techniques for tackling these underlying problems directly must be developed. Fortunately, NP-hard problems are very common, and there is a substantial body of research on computational methods [17]. However, most of these techniques deal with optimization problems, in which the notion of an approximate answer makes sense. On the other hand, as made clear in 4.7, the questions involved in open specification are related closely to satisfiability questions for logical formulas; such problems do not admit useful notions of approximation. Fortunately, there is an active body of research on such problems, as well as a library of tools known *SATLIB – The Satisfiability Library*, which is available on the world-wide web. Our next steps in addressing the problems of open specification must clearly be experimental ones, and will proceeds as follows.

1. Direct solution of the associated logic problems, as characterized in 4.7, will be addressed using SATLIB tools such as GSAT [23], a tool which is effective in the solution of large satisfiability problems.
2. A study of techniques related to re-use. One of the features of the satisfiability problems surrounding open specification is that not one, but a whole family of satisfiability problems (one for each two-element model) must be obtained. It is clear from the results of Sec. 4, and 4.7 in particular, that the members of the families of formulas to be solved are closely related; they may differ in only slightly. Therefore, techniques which solve whole families of related formulas in a fashion more efficient than solving each individually must be developed. As far as we know, such techniques are not part of current work on the subject.
3. The alternate characterization of two element models presented in 5.4 suggests that techniques which address re-use in the specific context of Horn and XOR-constraints, may prove useful. As far as we know, SATLIB-style results for such specially conditioned formulas do not exist at this time.

References

[1] H. Aït-Kaci. An algebraic semantics approach to the effective resolution of type equations. *Theoret. Comput. Sci.*, 45:293–351, 1986.

[2] H. Aït-Kaci, R. Boyer, P. Lincoln, and R. Nasr. Efficient implementation of lattice operations. *ACM Trans. Programming Languages and Systems*, 11:115–146, 1989.

[3] T. Briscoe, V. de Paiva, and A. Copestake, editors. *Inheritance, Defaults, and the Lexicon.* Cambridge University Press, 1993.

[4] B. Carpenter. *The Logic of Typed Feature Structures.* Cambridge University Press, 1992.

[5] B. Carpenter and G. Penn. ALE: The Attribute Logic Engine user's guide, Version 3.1 Beta. Technical report, Bell Laboratories and Universität Tübingen, 1998.

[6] V. de Paiva. Types and constraints in the LKB. In T. Briscoe, V. de Paiva, and A. Copestake, editors, *Inheritance, Defaults, and the Lexicon*, pages 164–189. Cambridge University Press, 1993.

[7] J. Döerre and M. Dorna. CUF – a formalism for linguistic knowledge representation. In J. Dörre, editor, *Computational Aspects of Constraint-Based Linguistic Description, DYANA-2 Deliverable R.1.2.A*, pages 3–22. ESPRIT, 1993.

[8] J. Dörre and A. Eisele. Feature logic with disjunctive unification. In *Proceedings of the COLING 90, Volume 2*, pages 100–105, 1990.

[9] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn clauses. *J. Logic Programming*, 3:267–284, 1984.

[10] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman, 1979.

[12] D. Gerdemann and P. J. King. The correct and efficient implementation of appropriateness conditions for typed feature structures. In *Proceedings of COLING-94*, pages 956–960, 1994.

[13] G. Grätzer. *General Lattice Theory.* Academic Press, 1978.

[14] S. J. Hegner. Distributivity in incompletely 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 R1.2.B*, pages 29–120. DYANA, 1994.

[15] S. J. Hegner. Properties of Horn clauses in feature-structure logic. In C. J. Rupp, M. A. Rosner, and R. L. Johnson, editors, *Constraints, Languages and Computation*, pages 111–147. Academic Press, 1994.

[16] H. Herrlich and G. E. Strecker. *Category Theory.* Allyn and Bacon, 1973.

[17] D. S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing, 1997.

[18] E. Horowitz, S. Sahni, and S. Rajasekaran. *Computer Algorithms.* Computer Science Press, 1998.

[19] H. J. Levesque and R. J. Brachman. Expressiveness and tractability in knowledge representation. *Computational Intelligence*, 3:78–93, 1987.

[20] J. A. Makowsky. Why Horn formulas matter in computer science: Initial structures and generic examples. *J. Comput. System Sci.*, 34:266–292, 1987.

[21] W. D. Meurers. On implementing an HPSG theory–aspects of the logical architecture, the formalization, and the implementation of head-driven phrase structure grammars. In E. W. Hinrichs, W. D. Meurers, and T. Nakazawa, editors, *Partial-VP and Split-NP Topicalization in German – An HPSG Analysis and its Implementation*, volume 58 of *Arbeitspapiere des SFB 340.* University of Tübingen, 1994.

[22] C. Pollard and I. A. Sag. *Head-Driven Phrase Structure Grammar.* University of Chicago Press, 1994.

[23] B. Selman and H. Kautz. Domain-independent extensions to gsat: Solving large structured satisfiability problems. In *Proc. Thirteenth IJCAI*, pages 290–295, 1993.

[24] R. Zajac. Notes on the Typed Feature System, Version 4, January 1991. Technical report, Universität Stuttgart, Institut für Informatik, Project Polygloss, 1991.