Characterization of Type Hierarchies with Open Specification

Stephen J. Hegner

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

Abstract. Type hierarchies which arise in applications are often described incompletely; this missing information may be handled in a variety of ways. In this work, such incomplete hierarchies are viewed as *open specifications*; that is, descriptions which are sets of constraints. The actual hierarchy is then any structure satisfying these constraints. For such specifications, two forms of characterization are provided. The first is algebraic and utilizes a generalization of weak partial lattices; it provides a structure-based characterization in which optimality is characterized via an initial construction. The second is logical, an inference-based representation, in which models are characterized as products of models of propositional-based specifications.

1. Introduction

Type hierarchies play a central rôle in the foundations of database and knowledge-base systems; consequently, a vast literature surrounding them has developed. Frameworks such as description logics [3] and formal concept analysis [10], as well as formal models for object-oriented database systems themselves [2, 18, 19], have evolved in response to the need for a comprehensive theoretical foundation. The work reported in this paper lays the foundation for extending the ideas of these formalisms to contexts in which information about the type hierarchy is *open*, in the sense that specification is via a set of constraints rather than via a single instance, so there may be none, one, or many instances which satisfy those constraints.

1.1 Background: the relationship between subsumption, suprema and infima Regardless of the specific formalism, the major underlying notion in a type hierarchy is that of subsumption. Type τ_1 *is subsumed* by type τ_2, (or, equivalently, τ_2 *subsumes* τ_1), written $\tau_1 \sqsubseteq \tau_2$, just in case every object of type τ_1 is also an object of type τ_2. Writing $\mathfrak{O}(\tau)$ to denote the collection of objects of type τ, this subsumption is expressed as $\mathfrak{O}(\tau_1) \subseteq \mathfrak{O}(\tau_2)$.

Subsumption is often visualized using lattice like diagrams, such as that of Fig. 1, which depicts information about a simple hierarchy for univer-

sity people. The symbol $\bot$ represents the empty type, with $\top$ the universal type of all such people. A line from a higher object to a lower one indicates subsumption; e.g., $\mathsf{Grad} \sqsubseteq \mathsf{Student}$.

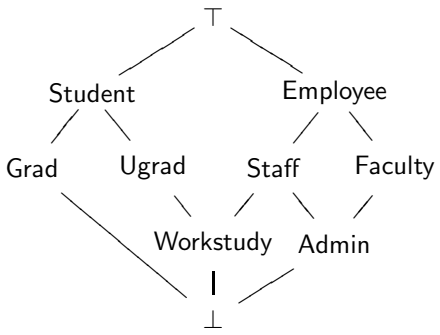


Fig. 1: Visualization of a simple type hierarchy

It is tempting to make further use of the lattice-like properties of the infimum and supremum operations on the ordering induced by type subsumption. For example, suppose that it is known that Mary is both an undergraduate (Mary $\in \mathfrak{O}(\mathsf{Ugrad})$) and that she is a member of the staff (Mary $\in \mathfrak{O}(\mathsf{Staff})$). Since $\mathsf{inf}(\{\mathsf{Ugrad}, \mathsf{Staff}\}) = \mathsf{Workstudy}$ in this hierarchy, one might conclude that she is a workstudy (Mary $\in \mathfrak{O}(\mathsf{Workstudy})$). More generally, from the above hierarchy, one might conclude that $\mathfrak{O}(\mathsf{Ugrad}) \cap \mathfrak{O}(\mathsf{Staff}) = \mathfrak{O}(\mathsf{Workstudy})$. In [16], this has been called the *natural meet semantics*. However, it is not appropriate to presume this semantics universally; rather such a decision must be based upon further information. Regardless of whether or not it is named and represented explicitly, there is a type τ which embodies precisely those people who are both undergraduates and staff members, i.e., $\mathfrak{O}(\tau) = \mathfrak{O}(\mathsf{Ugrad}) \cap \mathfrak{O}(\mathsf{Staff})$. If this type τ is represented explicitly in the hierarchy, then it must have the same objects as Workstudy. However, it is quite possible that the hierarchy of Fig. 1 is just a partial representation of the total state of affairs, with τ not explicitly represented. For example, undergraduate students could be hired to do grounds maintenance, which might not be classified as workstudy. The point is that, from the subsumption relationships, it can only be concluded that $\mathsf{Workstudy} \sqsubseteq \tau$; to conclude that $\mathsf{Workstudy} = \tau$, further information is needed.

The ideas described above are central to an operation known as *(conjunctive) type unification*, which may be described as follows. Given that it is known that an object a is of both type τ_1 and of type τ_2, determine the most specific type τ for which a is of type τ. Such unification is of fundamental importance in parsing of natural language, particularly within formalisms such as HPSG [17], which employ type hierarchies to classify linguistic objects. Indeed, in parsers built using systems designed to support such parsing, which include ACQUILEX [4], CUF [7], and TFS [20], the underlying database is effectively a large type hierarchy, which embodies both the lexicon and the grammar, without recourse to traditional phrase-structure grammars. Even in systems such as ALE [5], which does incorporate an underlying context-free grammar, type unification plays a central rôle.

There is a dual question, which looks at infima rather than suprema. Again referring to Fig. 1, suppose that it is known that Mary is a student (Mary $\in$ Student). The fact that $\mathsf{Student} = \mathsf{sup}(\{\mathsf{Grad}, \mathsf{Ugrad}\})$ in the hierarchy suggests that Mary must be either an undergraduate or a graduate student. How-

ever, there could easily be another class of student, such as nondegree students, which is not represented explicitly in the hierarchy. The conclusion that $\mathfrak{O}(\mathsf{Student}) = \mathfrak{O}(\mathsf{Grad}) \cup \mathfrak{O}(\mathsf{Ugrad})$ is not automatic; it depends upon further information. In [16], this property is called the *natural join semantics*. Although not as widely used in constraint-based problem solving as conjunctive unification, the corresponding operation of *disjunctive unification* has seen use in the management of solution search [8].

1.2 Focus of this work The work reported in this paper was motivated by an interest in coming to grips, in a formal manner, with the semantics of the infimum and supremum operations in type hierarchies, particularly in applications involving unification. More often than not, in the current literature, such hierarchies are described in terms of subsumption only, and while infimum (and often supremum as well) is used in the computational process, the semantics of these operators is not described explicitly. The reader is thus left to reverse engineer the framework in order to determine these important details.

The starting point of the work presented here is that all constraints on the hierarchy are to be expressed via formulas of the forms $(\bigsqcap\{\tau_1, \tau_2, .., \tau_n\} = \tau)$ and $(\bigsqcup\{\tau_1, \tau_2, .., \tau_n\} = \tau)$, with $\bigsqcap$ denoting a generalized meet operation, and $\bigsqcup$ a generalized join operation. The semantics of these operations are $(\bigcap\{\mathfrak{O}(\tau_1), \mathfrak{O}(\tau_2), .., \mathfrak{O}(\tau_n)\} = \mathfrak{O}(\tau))$ and $(\bigcup\{\mathfrak{O}(\tau_1), \mathfrak{O}(\tau_2), .., \mathfrak{O}(\tau_n)\} = \mathfrak{O}(\tau))$, respectively, with $\mathfrak{O}(\omega)$ denoting the set of objects of type ω. It is important to note that operations of extended arity are required. It is quite possible for $\bigsqcap\{\tau_1, \tau_2, \tau_3\}$ to be defined without any stated definition for $\bigsqcap\{\tau_1, \tau_2\}$, $\bigsqcap\{\tau_1, \tau_3\}$, or $\bigsqcap\{\tau_2, \tau_3\}$. The subsumption $\tau_1 \sqsubseteq \tau_2$ is expressed via the constraint $(\bigsqcap\{\tau_1, \tau_2\} = \tau_1)$. A description based upon constraints of this form is called an *open specification* because it does not state the complete properties of a single hierarchy; rather, it provides constraints which any *model* hierarchy must obey. There may be none, one, or several such models for a given set of constraints.

In this work, there is no explicit notion of attribute; thus, it differs fundamentally from work in description logics and formal concept analysis. This decision, however, was not made due to any lack of belief in the importance of these concepts. Clearly, a meaningful notion of attribute, and its behavior under order-induced operations, is central to any comprehensive theory of type hierarchies. At the same time, it seems clear that any attempt to introduce attributes into the work at this initial stage would only complicate matters and cloud development of the fundamental issues. A useful formalism of simple attribute-free types under open specification must precede the development of a more complex formalism which incorporates attributes. The focus here is upon how pure types behave in the presences of open specification. Once this is understood, the results should be combined with the more general notions embodied in other formalisms.

1.3 Content and organization of this paper The work reported here is based, to a substantial degree, on the papers [13], [16], and [15]. Nonetheless, the method of presentation, and even some of the results, particularly with respect to algebraic characterization, are entirely new.

The primary emphasis of this paper is the *characterization* of type hierarchies under open specifications; that is, the development of mathematical principles necessary to model and compute upon such hierarchies. Two major avenues of characterization are presented and contrasted. In Section 3, an algebraic characterization, based upon a generalized form of bounded partial lattice, and couched in a general framework which makes use of universal constructions, is provided. In Section 4, a logical characterization, based upon propositional logic, is given. Each of these approaches has its strengths and weaknesses, as are exposed in the presentation.

An additional topic of great importance in this area is that of computational complexity and algorithms for the manipulation of and inference on such hierarchies. Unfortunately, space limitations preclude a thorough treatment. However, the most important results are summarized in Section 5.

Finally, Section 2 provides a common backdrop of basic definitions and results, in support of both of the characterizations, while Section 6 provides some conclusions and further directions.

1.4 Prerequisites and notation Of course, it is presumed that the reader has an appreciation for the central rôle of type hierarchies in computer science in general and database systems in particular. Beyond that, a knowledge of standard propositional logic is expected. It is also assumed that the reader has a basic knowledge of the theory of partial orders and lattices; the necessary background may be obtained from standard references such as [12] and [6]. To minimize the possibility of confusion with the corresponding logical symbols $\wedge$ and $\vee$, the order relation and the lattice-like operations of join, and meet in such structures will always be denoted with the squared symbols $\sqsubseteq$, $\sqcup$, and $\sqcap$, respectively. A *bounded lattice* is denoted as $\mathbf{L} = (L, \sqcup, \sqcap, \top, \bot)$, with $\top$ the greatest element and $\bot$ the least element. In particular, the boldface symbol (e.g., $\mathbf{L}$) denotes the entire structure, while the normal symbol L denotes just the underlying set. This font convention will also be used with other algebraic structures (e.g., the generalized bounded weak partial lattices of Section 3).

A very small amount of category theory background will prove helpful. Specifically, *isomorphisms* are always characterized as morphisms which have both left and right inverses. In addition, some familiarity with the ideas of free and initial objects would prove helpful, although full definitions are always given. A suitable reference is [1].

The following specific notation, which may be a bit less than standard, is used. If f is a partial function, then $f(s)\downarrow$ (resp. $f(s)\uparrow$) means that f is defined (resp. undefined) on argument s. If A is a set, then $\mathbf{2}^A$ (resp. $\mathbf{2}^A_f$) denotes the set of all subsets (resp. finite subsets) of A, while $\mathsf{Card}(A)$ denotes the cardinality of A.

2. Constraints and Models

In this section, the formal foundations for the description of type hierarchies via open specification are presented. The presentation follows most closely that of

[15]. However, some of the ideas, particularly those related to morphisms and completions, are new and were developed in support of the algebraic characterization of the next section.

2.1 Elementary positive constraints A *clean set* is any finite set P which does not contain either of the special symbols $\bot$ and $\top$. Define $\mathsf{Aug}(P) = P \cup \{\bot, \top\}$. The elements $\bot$ and $\top$ are called the *extreme types*; the elements of P are called the *base types*. As shall soon be formalized, $\top$ (resp. $\bot$) represents the greatest (resp. least) type in the hierarchy.

The most fundamental class of constraint is the *elementary positive constraint*, of which there are two basic forms.[1] An *elementary positive meet constraint* has the form $(\bigsqcap\{\tau_1, \tau_2, .., \tau_n\} = \tau)$, with the τ_i's and τ members of $\mathsf{Aug}(P)$. The set of all such constraints over P is denoted $\mathsf{ElemConstr}^+_\sqcap$. Dually, an *elementary positive join constraint* has the form $(\bigsqcup\{\tau_1, \tau_2, .., \tau_n\} = \tau)$, with the set of all such constraints denoted by $\mathsf{ElemConstr}^+_\sqcup$. Combining these two classes, $\mathsf{ElemConstr}^+(P)$ denotes $\mathsf{ElemConstr}^+_\sqcap(P) \cup \mathsf{ElemConstr}^+_\sqcup(P)$.

2.2 Interpretation of types Let P be a clean set of types. An *interpretation* over P is a pair $S = (\mathfrak{U}, \mathfrak{O})$, in which $\mathfrak{U}$ is a finite nonempty set, called the *universe of objects*, and $\mathfrak{O} : \mathsf{Aug}(P) \to \mathbf{2}^{\mathfrak{U}}$ is a function which associates a subset of $\mathfrak{U}$ to each type in $\mathsf{Aug}(P)$, subject to the conditions that $\mathfrak{O}(\top) = \mathfrak{U}$ and $\mathfrak{O}(\bot) = \emptyset$. Think of $\mathfrak{O}(\tau)$ as the set of all objects of type τ.

An interpretation $S = (\mathfrak{U}, \mathfrak{O})$ is to be viewed as the specification for a unique, complete type hierarchy over P, in which all infima and suprema have their natural semantics. Specifically, define $\mathsf{Lat}(S)$ to be the smallest set of subsets of $\mathfrak{U}$ which contains every member of $\{\mathfrak{O}(\tau) \mid \tau \in \mathsf{Aug}(P)\}$, and which is closed under union and intersection. It is easy to see that $\mathsf{Lat}(S)$ admits the structure of a finite, bounded distributive lattice, with union as join, intersection as meet, $\mathfrak{U}$ as top element and $\emptyset$ as least element. This lattice is denoted $\mathbf{Lat}(S) = (\mathsf{Lat}(S), \cup, \cap, \mathfrak{U}, \emptyset)$.

The *size* of an interpretation $S = (\mathfrak{U}, \mathfrak{O})$ is the cardinality of $\mathfrak{U}$; an interpretation of size m is often called an m-element interpretation. In particular, a *one-element interpretation* is of the form $S = (\{a\}, \mathfrak{O})$.

Let $S = (\mathfrak{U}, \mathfrak{O})$ be an interpretation for P. For each nonempty subset $\mathfrak{V} \subseteq \mathfrak{U}$, define the $\mathfrak{V}$-*projection* of S to be $S_{|\mathfrak{V}} = (\mathfrak{V}, \mathfrak{O}_{|\mathfrak{V}})$, with $\mathfrak{O}_{|\mathfrak{V}} : \mathsf{Aug}(P) \to \mathbf{2}^{\mathfrak{V}}$ given by $X \mapsto \mathfrak{O}(X) \cap \mathfrak{V}$. Conversely, let $S_i = (\mathfrak{U}_i, \mathfrak{O}_i)$ be interpretations for P for $i = 1, 2$. Assume further that $\mathfrak{U}_1 \cap \mathfrak{U}_2 = \emptyset$. The *product interpretation* $S_1 \times S_2$ is given by $(\mathfrak{U}_1 \cup \mathfrak{U}_2, \mathfrak{O}_1 \times \mathfrak{O}_2)$, with $\mathfrak{O}_1 \times \mathfrak{O}_2 : \mathsf{Aug}(P) \to \mathbf{2}^{\mathfrak{U}_1 \cup \mathfrak{U}_2}$ given by $X \mapsto \mathfrak{O}_1(X) \cup \mathfrak{O}_2(X)$. This definition extends easily to any finite number of interpretations. Note that, for any interpretation $S = (\mathfrak{U}, \mathfrak{O})$, $\prod_{a \in \mathfrak{U}} S_{|\{a\}}$ is just S, up to a renaming of $\mathfrak{O}$.

[1] The definition here differs slightly from that found in [15]; in which equality is replaced by subsumption. This is of no major consequence, since the two formalisms are equivalent in expressive power. See 2.7 for a further discussion.

2.3 Satisfaction of constraints The constraint $(\bigsqcap\{\tau_1, \tau_2, .., \tau_n\} = \tau)$ is *satisfied* by the interpretation $S = (\mathfrak{U}, \mathfrak{O})$ if $\mathfrak{O}(\tau_i) \cap \mathfrak{O}(\tau_2) \cap .. \cap \mathfrak{O}(\tau_n) = \mathfrak{O}(\tau)$. Similarly, $(\bigsqcup\{\tau_1, \tau_2, .., \tau_n\} = \tau)$ is *satisfied* by S if $\mathfrak{O}(\tau_i) \cup \mathfrak{O}(\tau_2) \cup .. \cup \mathfrak{O}(\tau_n) = \mathfrak{O}(\tau)$. In general, if $\varphi \in \mathsf{ElemConstr}^+(P)$ and S is an interpretation, then $M \models \varphi$ denotes that φ is satisfied by S; in this case, S is called a *model* of φ. If $\Phi \subseteq \mathsf{ElemConstr}^+(P)$, $S \models \Phi$ holds iff $S \models \varphi$ for each $\varphi \in \Phi$. The set of all S for which $S \models \varphi$ (resp. $S \models \Phi$) holds is denoted $\mathsf{Mod}_P(\varphi)$ (resp. $\mathsf{Mod}_P(\Phi)$). Two sets Φ_1 and Φ_2 of constraints over P are *equivalent* if $\mathsf{Mod}_P(\Phi_1) = \mathsf{Mod}_P(\Phi_2)$.

Following standard notation from mathematical logic, if $\varphi \in \mathsf{ElemConstr}^+(P)$ (resp. $\Psi \subseteq \mathsf{ElemConstr}^+(P)$), and $\Phi \subseteq \mathsf{ElemConstr}^+(P)$, then $\Phi \models \varphi$ (resp. $\Phi \models \Psi$) holds just in case $\mathsf{Mod}_P(\Phi) \subseteq \mathsf{Mod}_P(\varphi)$ (resp. $\mathsf{Mod}_P(\Phi) \subseteq \mathsf{Mod}_P(\Psi)$). Although this assigns double duty to the symbol $\models$, no confusion can result, since the nature of the first argument (interpretation or constraint) identifies the usage unambiguously.

2.4 Abbreviations and variants In addition to these elementary constraints, there are a few notational variants which are important enough to warrant their own notation. First of all, $(\tau_1 \sqsubseteq \tau_2)$ is an abbreviation for $(\bigsqcap\{\tau_1, \tau_2\} = \tau_1)$. Second, $(\tau_1 = \tau_2)$ is an abbreviation for $(\bigsqcap\{\tau_1\} = \tau_2)$.

In addition to these definitions, the notational variant of infix (as opposed to prefix) notation is allowed for all forms of constraints. Thus, for example, $(\tau_1 \sqcap \tau_2 \sqcap \tau_3 = \tau)$ is a perfectly acceptable (and obvious) abbreviation for $(\bigsqcap\{\tau_1, \tau_2, \tau_3\} = \tau)$. This notation in no way implies that meets for subsets (e.g., $\tau_1 \sqcap \tau_2$) are defined.

2.5 Open specification Let P be a clean set of types. An *elementary positive open specification* is a pair (P, Φ) in which $\Phi \subseteq \mathsf{ElemConstr}^+(P)$.

2.6 Examples In each of the cases below, let (Q, Ω) be the elementary positive open specification with $Q = \{\tau_a, \tau_b, \tau_c\}$ and $\Omega = \{(\tau_a \sqcup \tau_c = \top), (\tau_c \sqsubseteq \tau_b)\}$.

(a) Let $\mathfrak{V}_1 = \mathfrak{V}_2 = \mathfrak{V}_3 = \mathfrak{V}_4 = \{\mathfrak{a}\}$, and define $\mathfrak{O}_1 : \mathsf{Aug}(Q) \to \mathbf{2}^{\mathfrak{V}_1}$ by $\tau_a \mapsto \{\mathfrak{a}\}$, $\tau_b \mapsto \emptyset$, $\tau_c \mapsto \emptyset$; $\mathfrak{O}_2 : \mathsf{Aug}(Q) \to \mathbf{2}^{\mathfrak{V}_2}$ by $\tau_a \mapsto \{\mathfrak{a}\}$, $\tau_b \mapsto \{\mathfrak{a}\}$, $\tau_c \mapsto \emptyset$; $\mathfrak{O}_3 : \mathsf{Aug}(Q) \to \mathbf{2}^{\mathfrak{V}_3}$ by $\tau_a \mapsto \emptyset$, $\tau_b \mapsto \{\mathfrak{a}\}$, $\tau_c \mapsto \{\mathfrak{a}\}$; $\mathfrak{O}_4 : \mathsf{Aug}(Q) \to \mathbf{2}^{\mathfrak{V}_4}$ by $\tau_a \mapsto \{\mathfrak{a}\}$, $\tau_b \mapsto \{\mathfrak{a}\}$, $\tau_c \mapsto \{\mathfrak{a}\}$. Then $\mathfrak{S}_i = (\mathfrak{V}_i, \mathfrak{O}_i)$ for $i = 1 \ldots 4$ are models for (Q, Ω). It is not difficult to see that they are the only one-element models, up to renaming of elements.

(b) Let $\mathfrak{V}_5 = \{\mathfrak{a}, \mathfrak{b}\}$, and define $\mathfrak{O}_5 : \mathsf{Aug}(Q) \to \mathbf{2}^{\mathfrak{V}_5}$ by $\tau_a \mapsto \{\mathfrak{a}\}$, $\tau_b \mapsto \{\mathfrak{b}\}$, $\tau_c \mapsto \{\mathfrak{b}\}$. Then $\mathfrak{S}_5 = (\mathfrak{V}_5, \mathfrak{O}_5)$ is a model for (Q, Ω). This model also satisfies positive constraints which are not embodied in Ω, including $(a \sqcap c = \bot)$ and $(b = c)$.

(c) Let $\mathfrak{V}_6 = \{\mathfrak{a}, \mathfrak{b}, \mathfrak{c}\}$, and define define $\mathfrak{O}_6 : Q \to \mathbf{2}^{\mathfrak{V}_6}$ by $\tau_a \mapsto \{\mathfrak{a}, \mathfrak{b}\}$, $\tau_b \mapsto \{\mathfrak{b}, \mathfrak{c}\}$, $\tau_c \mapsto \{\mathfrak{c}\}$. $\mathfrak{S}_5 = (\mathfrak{V}_5, \mathfrak{O}_5)$ is also a model for (Q, Ω). It still satisfies a positive constraint not embodied in Ω, namely $(a \sqcap c = \bot)$.

(d) Let $\mathfrak{V}_7 = \{\mathfrak{a}, \mathfrak{b}, \mathfrak{c}, \mathfrak{d}, \mathfrak{e}\}$, and define $\mathfrak{O}_7 : Q \to \mathbf{2}^{\mathfrak{V}_7}$ by $\tau_a \mapsto \{\mathfrak{a}, \mathfrak{b}, \mathfrak{d}, \mathfrak{e}\}$, $\tau_b \mapsto \{\mathfrak{b}, \mathfrak{c}, \mathfrak{d}\}$, $\tau_c \mapsto \{\mathfrak{c}, \mathfrak{d}\}$. $\mathfrak{S}_7 = (\mathfrak{V}_7, \mathfrak{O}_7)$ is a model for (Q, Ω). While this model

does not satisfy any positive constraints not embodied in Ω, as do $\mathfrak{S}_4$ and $\mathfrak{S}_5$ above, it is redundant in that the elements $\mathfrak{a}$ and $\mathfrak{e}$ are not distinguishable.

(e) Let $\mathfrak{V}_8 = \{\mathfrak{a}, \mathfrak{b}, \mathfrak{c}, \mathfrak{d}\}$, and define $\mathfrak{O}_8 : Q \to \mathbf{2}^{\mathfrak{V}_8}$ by $\tau_a \mapsto \{\mathfrak{a}, \mathfrak{b}, \mathfrak{d}\}$, $\tau_b \mapsto \{\mathfrak{b}, \mathfrak{c}, \mathfrak{d}\}$, $\tau_c \mapsto \{\mathfrak{c}, \mathfrak{d}\}$. $\mathfrak{S}_8 = (\mathfrak{V}_8, \mathfrak{O}_8)$ is a model for (Q, Ω). This model is "optimal" in the sense that it neither enforces unnecessary constraints, as do $\mathfrak{S}_4$ and $\mathfrak{S}_5$, nor does it introduce superfluous elements, as does $\mathfrak{S}_6$. See 3.13 for a further discussion of this idea.

2.7 Subsumption-based constraints In [15], the basic forms of the constraints are taken to be $(\bigsqcap\{\tau_1, \tau_2, , \ldots \tau_n\} \sqsubseteq \tau)$ and $(\tau \sqsubseteq \bigsqcup\{\tau_1, \tau_2, \ldots, \tau_n\})$; i.e., equality is replaced by subsumption. This choice was made because such *subsumption constraints* are more suitable to the computational algorithms of that paper. On the other hand, the *equality constraints* used in this paper are more natural and intuitive when characterization — particularly algebraic characterization — is the principal topic. These two representations are effectively equivalent. The equality constraint $(\bigsqcap\{\tau_1, \tau_2, , \ldots \tau_n\} = \tau)$ is equivalent to the subsumption constraint $(\bigsqcap\{\tau_1, \tau_2, , \ldots \tau_n\} \sqsubseteq \tau)$ together with the set $\{(\tau \sqsubseteq \tau_1), (\tau \sqsubseteq \tau_2), \ldots, (\tau \sqsubseteq \tau_n)\}$. To represent the subsumption constraint $(\bigsqcap\{\tau_1, \tau_2, , \ldots \tau_n\} \sqsubseteq \tau)$ using equality constraints, use $(\bigsqcap\{\tau_1, \tau_2, , \ldots \tau_n\} = \sigma)$ together with $(\bigsqcap\{\sigma, \tau\} = \sigma)$. Here σ is a new type symbol not used previously. The join constraints are represented similarly.

3. Algebraic Characterization

While an open specification is not a lattice, it is nonetheless apparent that the constraints look very much like constraints on a lattice. A major difference is that, with an open specification, there is no guarantee that $\tau_1 \sqcap \tau_2$ and $\tau_1 \sqcup \tau_2$ will be defined for an arbitrary pair $\{\tau_1, \tau_2\} \subseteq P$. Thus, a natural question to ask is whether there is some sort of algebraic characterization of open specifications and their models, using a lattice-like notion with partial operations. In this section, this question is answered in the affirmative.

The author's initial report on open specification [13] contains a rather detailed development of their algebraic properties. While these results remain valid, simpler and more concise characterizations, which have not been previously published, have been discovered since the appearance of [13]. In this section, many of these newer results are presented.

3.1 Generalized Bounded Weak Partial Lattices In [12, Ch. I, Sec. 5], Grätzer describes two notions, the *partial lattice* and the *weak partial lattice*. Roughly speaking, a partial lattice may be characterized as a subset of a lattice under the induced operations, while a weak partial lattice is a set with partial lattice-like operations. Grätzer also provides an example of a weak partial lattice which cannot be embedded in a partial lattice.

Neither of these concepts is directly applicable to the modelling of open specifications. Most importantly, the definitions given in [12] have their meet and join operations restricted to pairs of elements, while the constraints described here

allow these operations to take finite subsets as arguments. This is an essential difference, as it is quite possible to have $\bigsqcap\{\tau_1, \tau_2, \ldots, \tau_n\}$ and/or $\bigsqcup\{\tau_1, \tau_2, \ldots, \tau_n\}$ defined without having the these operations defined on any proper subset of $\{\tau_1, \tau_2, \ldots, \tau_n\}$ of cardinality greater than one. On the other hand, it is quite possible to extend the definitions of [12] to this more general context. The appropriate one to generalize for the context at hand is the weak partial lattice. The definition given here, which originally appeared in [13, 1.2.3], generalizes that of [12] to meet and join operations with more than two arguments, and adds universal bounds as well.

A *generalized bounded weak partial lattice* (*GBWPL*) is a five-tuple $\mathbf{L} = (L, \bigsqcup, \bigsqcap, \top, \bot)$ in which the following nine conditions are satisfied.

(gbwpl:1) L is a set (the *underlying set*).

(gbwpl:2) $\bigsqcup : \mathbf{2}_f^L \to L$ is a partial operation, called the *generalized join*.

(gbwpl:3) $\bigsqcap : \mathbf{2}_f^L \to L$ is a partial operation, called the *generalized meet*.

(gbwpl:4) $\bot, \top \in L$ with $\bot \neq \top$.

The operations $\bigsqcup$ and $\bigsqcap$ are subject to the following conditions.

(gbwpl:5) $\bigsqcup \emptyset = \bot$ and $\bigsqcap \emptyset = \top$.

(gbwpl:6) For all $a \in L$, $\bigsqcup\{a\} = a$, $\bigsqcap\{a\} = a$, $\bigsqcup\{a, \bot\} = a$, $\bigsqcap\{a, \top\} = a$.

(gbwpl:7) If $S_1, S_2, \ldots, S_n$ are finite subsets of L with $\bigsqcup S_i\downarrow$ for all i, $1 \leq i \leq n$, then $\bigsqcup(\bigcup_{i=1}^n S_i)\downarrow$ iff $\bigsqcup\{\bigsqcup S_1, \bigsqcup S_2, \ldots \bigsqcup S_n\}\downarrow$, and $\bigsqcup\{\bigsqcup S_1, \bigsqcup S_2, \ldots \bigsqcup S_n\} = \bigsqcup(\bigcup_{i=1}^n S_i)$ in this case.

(gbwpl:8) If $S_1, S_2, \ldots, S_n$ are finite subsets of L with $\bigsqcap S_i\downarrow$ for all i, $1 \leq i \leq n$, then $\bigsqcap(\bigcup_{i=1}^n S_i)\downarrow$ iff $\bigsqcap\{\bigsqcap S_1, \bigsqcap S_2, \ldots \bigsqcap S_n\}\downarrow$, and $\bigsqcap\{\bigsqcap S_1, \bigsqcap S_2, \ldots \bigsqcap S_n\} = \bigsqcap(\bigcup_{i=1}^n S_i)$ in this case. The constraints of (gbwpl:7) and (gbwpl:8) are called the *generalized associativity laws*.

(gbwpl:9) If S is a finite subset of L with $\bigsqcup S\downarrow$, then for all $a \in S$, $\bigsqcap\{a, \bigsqcup S\}\downarrow$ with $\bigsqcap\{a, \bigsqcup S\} = a$. Dually, for all $a \in S$, $\bigsqcup\{a, \bigsqcap S\}\downarrow$ with $\bigsqcup\{a, \bigsqcap S\} = a$. These are called the *generalized absorption identities*.

In (gbwpl:5), the condition $\bigsqcup\{a\} = a$ is a restatement of the idempotency of the join, which becomes $a \vee a = a$ in a lattice. The condition $\bigsqcup\{a, \bot\} = a$ states that $\bot$ is the least element. The other two new conditions are dual. In (gbwpl:9), The condition $\bigsqcap\{a, \bigsqcup S\} = a$ for $a \in S$ and $\bigsqcup S\downarrow$ is a generalization of the absorption identity $a \wedge (a \vee b) = a$ of an ordinary lattice [12, Cond. (L4), p. 5]. The other condition is dual.

Let $\mathbf{L_1} = (L, \bigsqcup, \bigsqcap, \top, \bot)$ and $\mathbf{L_2} = (L, \bigsqcup, \bigsqcap, \top, \bot)$ be GBWPL's. A *morphism* $f : \mathbf{L_1} \to \mathbf{L_2}$ is a function $f : L_1 \to L_2$ subject to the following constraints.

(gbwpl:mor1) For a finite $S \subseteq L_1$, if $\bigsqcup S\downarrow$, then $\bigsqcup(f(S))\downarrow$ and $f(\bigsqcup S) = \bigsqcup(f(S))$. Dually, if $\bigsqcap S\downarrow$, then $\bigsqcap(f(S))\downarrow$ and $f(\bigsqcap S) = \bigsqcap(f(S))$.

(gbwplmor:mor2) $f(\bot) = \bot$ and $f(\top) = \top$.

It is easy to see that $f : \mathbf{L}_1 \to \mathbf{L}_2$ is an isomorphism iff the underlying function is a bijection which preserves and reflects $\top$, $\bot$, and all meets and joins.

Any bounded lattice may be viewed as a GBWPL in which all of the operations are total; that is, for any finite $S \subseteq L$, $\bigsqcap S$ and $\bigsqcup S$ are defined. Conversely,

any GBWPL for which all such operations are total is a bounded lattice; this is easily verified from the axioms.

Given a GBWPL $\mathbf{L} = (L, \bigsqcup, \bigsqcap, \top, \bot)$ and a subset $M \subseteq L$ containing $\{\top, \bot\}$, the *restriction* of $\mathbf{L}$ to M is the GBWPL $\mathbf{L}_{|M} = (M, \bigsqcup, \bigsqcap, \top, \bot)$. The operations of $\mathbf{L}_{|M}$ are just those of $\mathbf{L}$, restricted to M. Note that this means that all components of an operation must lie in M. For example, if $\bigsqcap\{\tau_1, \tau_2, \ldots, \tau_n\} = \tau$ holds in $\mathbf{L}$, then for it to apply to $\mathbf{L}_{|M}$, the entire subset $\{\tau_1, \tau_2, \ldots, \tau_n, \tau\}$ must lie in M. It is easy to see that such a restriction of a GBWPL is always a GBWPL; the conditions (gbwpl:5)-(gbwpl:9) never generate any new elements, other than $\top$ and $\bot$, so it is never necessary to add any additional elements to M to maintain the GBWPL properties.

Finally, the notion of the product $\prod_{i\in I}$ of a family $\{\mathbf{L}_i \mid i \in I\}$ of GBWPL's is well defined, with the obvious coordinatewise operations.

3.2 Notational convention For the rest of this section, unless noted otherwise, fix (P, Φ) to be an elementary positive open specification.

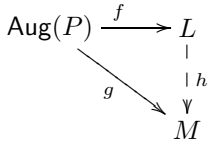
3.3 Representations of open specifications A *GBWPL representation* of (P, Φ) is a pair $(\mathbf{L}, f)$ in which $\mathbf{L}$ is a GBWPL and $f : \mathsf{Aug}(P) \to L$ is a function which satisfies the following rules.

(rep:1) $f(\top) = \top$ and $f(\bot) = \bot$.

(rep:2) For $\tau_1, \tau_2, \ldots, \tau_n, \tau \in \mathsf{Aug}(P)$, if $(\bigsqcap\{\tau_1, \tau_2, \ldots, \tau_n\} = \tau) \in \Phi$, then $(\bigsqcap\{f(\tau_1), f(\tau_2), \ldots, f(\tau_n)\} = f(\tau))$ holds in $\mathbf{L}$.

(rep:3) For $\tau_1, \tau_2, \ldots, \tau_n, \tau \in \mathsf{Aug}(P)$, if $(\bigsqcup\{\tau_1, \tau_2, \ldots, \tau_n\} = \tau) \in \Phi$, then $(\bigsqcup\{f(\tau_1), f(\tau_2), \ldots, f(\tau_n)\} = f(\tau))$ holds in $\mathbf{L}$.

A *morphism* $h : (\mathbf{L}, f) \to (\mathbf{M}, g)$ of GBWPL representations is a GBWPL morphism $h : \mathbf{L} \to \mathbf{M}$ with the property that that the diagram to the right commutes. It is easy to see that h is an isomorphism of representations iff it is an isomorphism of the underlying GBWPL's.



A representation $(\mathbf{L}, f)$ is *surjective* precisely in the case that f is a surjective function. It is always possible to derive a surjective representation from an arbitrary one; just replace $\mathbf{L}$ with $\mathbf{L}_{|f(\mathsf{Aug}(P))}$.

Finally, if $\{(\mathbf{L}_i, f_i) \mid i \in I\}$ is a family of representations of (P, Φ), so too is $(\prod_{i\in I} \mathbf{L}_i, \prod_{i\in I} f_i)$, with $\prod_{i\in I} f_i$ sending τ to the tuple whose i^{th} entry is $f(\tau)$.

Perhaps the most fundamental example of a representation of an open specification is one which arises from a model of the specification. The following proposition is immediate.

3.4 Examples The examples of 2.6 may easily be transformed to GBWPL representations. Indeed, it is easy to see that, for any model $\mathfrak{S}_i = (\mathfrak{R}_i, \mathfrak{O}_i)$ of (Q, Ω), $(\mathsf{Lat}(\mathfrak{S}_i), \mathfrak{f}_i)$ is a GBWPL representation. Here $\mathfrak{f}_i : \mathsf{Aug}(Q) \to \mathsf{Lat}(\mathfrak{S}_i)$ is the function which is identical to $\mathfrak{O}_i$, save that its codomain is restricted to $\mathsf{Lat}(\mathfrak{S}_i)$. Of course, these representations are in fact bounded lattices, with all meets and joins defined.

To obtain a "best" GBWPL representation of (Q, Ω), define $\mathfrak{L}_8 = (\mathfrak{l}_8, \bigsqcup, \bigsqcap, \top, \bot)$ to be the GBWPL in which $\mathfrak{l}_8 = Q \cup \{\bot, \top\}$; the constraints of this GBWPL are those which include $(\tau_a \sqcup \tau_c = \top)$ and $(\tau_c \sqsubseteq \tau_b)$, together with those constraints implied by these under the rules given in 3.1. Note that the closure of this set of constraints includes, in particular, $(\tau_a \sqcup \tau_b = \top)$. Define $\mathsf{f}_8 : \mathsf{Aug}(Q) \to \mathfrak{l}_8$ to be the natural identity embedding $\tau_x \mapsto \tau_x$. Then $(\mathfrak{L}_8, \mathsf{f}_8)$ is a GBWPL representation for (Q, Ω). This representation is best in the sense that it includes neither unnecessary constraints nor superfluous information. The precise meaning of "best" is developed in 3.5-3.7 below.

It might seem that one could always obtain a best GBWPL representation of (P, Φ) as a GBWPL whose elements are precisely the members of $\mathsf{Aug}(P)$, and whose constraints are a natural closure of the obvious translation of Φ. However, this is not always the case, since Φ may force the collapsing of certain elements. For example, let $R = \{\kappa_i \mid 1 \leq i \leq 6\}$, and let $\Gamma = \{(\bigvee\{\kappa_1, \kappa_2\} = \kappa_5), (\bigvee\{\kappa_2, \kappa_3\} = \kappa_5), (\bigvee\{\kappa_1, \kappa_3\} = \kappa_4), (\bigwedge\{\kappa_2, \kappa_3\} = \kappa_6), (\bigwedge\{\kappa_1, \kappa_6\} = \bot), ((\kappa_5 \leq \kappa_4)\}$. Then, as shown in [16, 5.5], any model $(\mathfrak{U}, \mathfrak{O})$ of (R, Γ) has the property that $\mathfrak{O}(\kappa_4) = \mathfrak{O}(\kappa_5)$. This means that, when constructing a GBWPL representation from R, the elements $\{\kappa_4, \kappa_5\}$ must be collapsed into a single equivalence class. This idea is expanded below; in particular in the development of the canonical representation 3.6.

3.5 Initial representations In view of the preceding examples, there are many representations of an open specification. The next task is to characterize formally the notion of a best representation, as illustrated by the final two examples above. Such a representation must be optimal in the sense that it contains just the right amount of information, without adding superfluous elements, and without imposing unnecessary constraints. In the world of arrows, the standard way to characterize such entities is via a free construction [1, Chap. 7]; or, equivalently, via an initial object.

Formally, an *initial GBWPL representation* for (P, Φ) is a GBWPL representation $(\mathbf{L}, \eta)$ with the property that, for any other GBWPL representation $(\mathbf{M}, g)$, there is a unique GBWPL morphism $h : (\mathbf{L}, \eta) \to (\mathbf{M}, g)$. Such a representation is unique, up to isomorphism [1, 7.3].

3.6 Canonical representations The notion of an initial representation is abstract, and says nothing about how to construct one. The canonical representation, described next, provides a concrete realization of an initial representation.

Assume that (P, Φ) has at least one representation. Define the equivalence relation $\equiv_{(P,\Phi)}$ on $\mathsf{Aug}(P)$ by $\tau_1 \equiv_{(P,\Phi)} \tau_2$ iff $f(\tau_1) = f(\tau_2)$ for all representations $(\mathbf{L}, f)$ of (P, Φ). For $\tau \in \mathsf{Aug}(P)$, $[\tau]_{\equiv_{(P,\Phi)}}$ denotes the equivalence class of τ in $\equiv_{(P,\Phi)}$. When no confusion can result, the subscript will be dropped; i.e., $[\tau]$. Further, for $A \subseteq \mathsf{Aug}(P)$, $[A]$ denotes $\{[\tau] \mid \tau \in A\}$. For $\varphi \in \Phi$, let $[\varphi]$ denote the constraint on $[\mathsf{Aug}(P)]$ obtained by replacing all members of $\mathsf{Aug}(P)$ with their respective equivalence classes. For example, $(\bigsqcap\{\tau_1, \tau_2, \ldots, \tau_n\} = \tau)$ becomes $(\bigsqcap\{[\tau_1], [\tau_2] \ldots, [\tau_n]\} = [\tau])$. Define $[\Phi] = \{[\varphi] \mid \varphi \in \Phi\}$.

Next, define $\mathsf{GBWPL}(P, \Phi) = ([\mathsf{Aug}(P)] \bigsqcup, \bigsqcap, [\top], [\bot])$. The constraint $(\bigsqcap\{[\tau_1], [\tau_2] \ldots, [\tau_n]\} = [\tau])$ holds in $\mathsf{GBWPL}(P, \Phi)$ iff for every GBWPL rep-

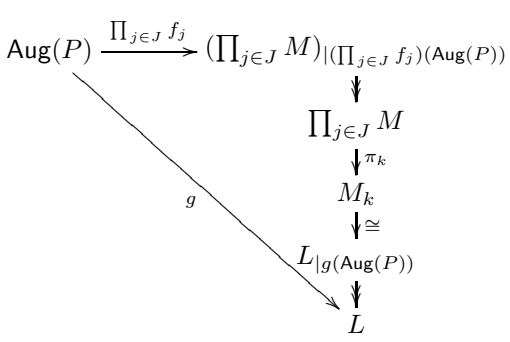
resentation $(\mathbf{L}, f)$ of (P, Φ), $(\bigsqcap\{f(\tau_1), f(\tau_2) \ldots, f(\tau_n)\} = f(\tau))$ holds in $\mathbf{L}$. Similarly, the constraint $(\bigsqcup\{[\tau_1], [\tau_2] \ldots, [\tau_n]\} = [\tau])$ holds in $\mathsf{GBWPL}(P, \Phi)$ iff for every GBWPL representation $(\mathbf{L}, f)$ of (P, Φ), $(\bigsqcup\{f(\tau_1), f(\tau_2) \ldots, f(\tau_n)\} = f(\tau))$ holds in $\mathbf{L}$.

The pair $\mathsf{CanRep}(P, \Phi) = (\mathsf{GBWPL}(P, \Phi), [-]_{\equiv_{(P,\Phi)}})$, with $[-]_{\equiv_{(P,\Phi)}} : \mathsf{Aug}(P) \to [\mathsf{Aug}(P)]$ given by $\tau \mapsto [\tau]$, is called the *canonical representation* of (P, Φ). Of course, it remains to be shown that $\mathsf{GBWPL}(P, \Phi) = ([\mathsf{Aug}(P)]\bigsqcup, \bigsqcap, [\top], [\bot])$ is indeed a GBWPL; this is the next topic.

3.7 Proposition *If (P, Φ) has a GBWPL representation, then it has an initial GBWPL representation, which is given by $\mathsf{CanRep}(P, \Phi)$.*

Proof Assume that (P, Φ) has a GBWPL representation. Begin with the class A of all GBWPL representations of (P, Φ), and pare this collection down in two ways. First of all, limit it to just the surjective representations; in view of the discussion at the end of 3.3, each original element of A will give rise to such a surjective relative. Next, pare the resulting collection down by choosing just one representative of each isomorphism class. Note that this resulting collection B is not only a set, but a finite one, since P is finite.

Denote B explicitly as $\{(\mathbf{M}_j, f_j) \mid j \in J\}$, and form the product $\prod B = (\prod_{j\in J} M_j, \prod_{j\in J} f_j)$. The diagram to the right shows that $(\prod B)_{|(\prod_{j\in J} f_j)(\mathsf{Aug}(P))}$ is an initial GWBPL representation for (P, Φ). Let $(\mathbf{L}, g)$ be any GBWPL representation of (P, Φ) whatever. The first and fourth vertical morphisms are just the natural injections. The second morphism π_k is the obvious projection; and the third morphism $\cong$ is the isomorphism between $(\mathbf{L}_{|g(\mathsf{Aug}(P))}, g)$ and its representative $(\mathbf{M}_k, f_k)$ in B. Finally, it is immediate from the way in which $\mathsf{CanRep}(P, \Phi)$ has been defined that $\prod B$ is isomorphic to $\mathsf{CanRep}(P, \Phi)$. □



3.8 Examples $\mathfrak{L}_8$ with the natural embedding of 3.4 is already a canonical representation of (Q, Ω). To obtain the canonical representation of (R, Γ) of 3.4, the types κ_4 and κ_5 must be identified in the equivalence relation $\equiv_{(R,\Gamma)}$.

As illustrated by the conversions in 3.4 of the example models in 2.6, every model $S = (\mathfrak{U}, \mathfrak{O})$ of (P, Φ) gives rise to a GBWPL representation of same which in fact a bounded lattice; namely, $\mathbf{Lat}(S)$. This leads naturally to the converse question, which asks whether an arbitrary representation (L, f) in which L is a bounded lattice identifies a model. The answer is a qualified yes. Two additional conditions must be met are, first, the lattice must be distributive, and, second, the mapping f must be dense in $\mathbf{L}$, so that no superfluous elements arise. The details follow.

3.9 Dense mappings and completions Let $\mathbf{L} = (L, \sqcup, \sqcap, \top, \bot)$ be a bounded lattice, let A be any set, and let $f : A \to L$ be a function. f is said to *dense* in $\mathbf{L}$ if the smallest sublattice of $\mathbf{L}$ containing $f(A)$ is $\mathbf{L}$ itself.

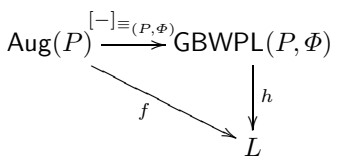
A GBWPL representation $(\mathbf{L}, f)$ for (P, Φ) in which $\mathbf{L}$ is a lattice is said to be *dense* just in case $f : \mathsf{Aug}(P) \to L$ is dense in $\mathbf{L}$. By construction, the representation $(\mathbf{Lat}(L), \mathfrak{O})$ associated with an interpretation $S = (\mathfrak{U}, \mathfrak{O})$ is dense. Similarly, if $h : (\mathbf{L}_1, f_1) \to (\mathbf{L}_2, f_2)$ is a morphism of GBWPL's, with $\mathbf{L}_2$ furthermore a lattice, then h is said to be *dense* if $h(L_1)$ dense in $\mathbf{L}_2$.

A representation $(\mathbf{L}, f)$ of (P, Φ) in which $\mathbf{L}$ is a bounded distributive lattice and f is dense is called a *completion* of (P, Φ). Note in particular that for every model $S = (\mathfrak{U}, \mathfrak{O})$ of (P, Φ), the representation $(\mathbf{Lat}(S), \mathfrak{O})$ is a completion. The following proposition shows that, up to isomorphism, all completions arise in this fashion.

3.10 Proposition *Let $(\mathbf{L}, f)$ be a completion of (P, Φ). Then there is a model $S = (\mathfrak{U}, \mathfrak{O})$ for (P, Φ) with the property that $(\mathbf{Lat}(S), \mathfrak{O})$ is isomorphic to $(\mathbf{L}, f)$. In particular, $\mathbf{L}$ must be finite.*

Proof In view of the characterization theorem of Birkhoff and Stone [12, Ch. II, Sec. 1, Thm. 19], $\mathbf{L}$ may be taken to be a ring of sets over a (finite) set A. Define $\mathfrak{U} = A$, and take $\mathfrak{O}$ to be identical to f, save that the codomain is extended to be all of $\mathbf{2}^{\mathfrak{U}}$. It is immediate that $S = (\mathfrak{U}, \mathfrak{O})$ is a model for (P, Φ). Furthermore, since f is dense, it follows immediately from the definition of $\mathbf{Lat}(S)$ that $\mathsf{Lat}(S)$ and L are identical. Thus, the pair $(\mathfrak{U}, \mathfrak{O})$ is the desired model. □

3.11 Theorem: algebraic characterization of models *There is a natural bijective correspondence between completions $(\mathbf{L}, f)$ of (P, Φ) and dense GBWPL morphisms h from $(\mathsf{GBWPL}(P, \Phi), [-]_{\equiv_{(P,\Phi)}})$ to distributive lattices, as illustrated in the diagram to the right. Thus, every $S \in \mathsf{Mod}(\Phi)$ is represented, up to isomorphism, by a dense GBWPL morphism from $\mathsf{CanRep}(P, \Phi)$ to a distributive lattice.*



Proof The proof follows immediately from the fact that the canonical representation $\mathsf{CanRep}(P, \Phi)$ is in fact initial, as shown in 3.7. □

3.12 The central rôle of distributivity In view of the above theorem, as well as the very definition of completion in 2.5, it is clear that any completion of an open specification **must** be distributive. If a diagram such as that of Fig. 1 (which is easily shown not to be distributive) is presented as a description of the hierarchy under consideration, then it is certain that some information is missing. Either natural semantics do not hold for meet or for join (in which case the diagram is a partial description of an open specification), or else there are some types which are equivalent (in which case the actual hierarchy would take the form of a distributive quotient of the one presented).

3.13 Initial and canonical models In general, a satisfiable open specification (P, Φ) will have many models. All of these models must satisfy every constraint in Φ, but some may satisfy other constraints as well. It turns out that there is a unique (up to isomorphism) model, called the *canonical model*, which satisfies as few constraints as possible; in fact, it satisfies only those constraints which are a consequence of Φ.

Space limitations do not permit presentation of the full construction here; rather, the reader is referred to [16, 3.3-3.8] for details. However, a brief discussion is in order. A set A of subsets of P is called a *crown* of P if whenever $C_1, C_2 \in A$ and $C_1 \subseteq C_2$, then $C_1 = C_2$. $\mathsf{Crown}(P)$ denotes the set of all crowns of P, while $\mathsf{Crown}_\top(P)$ denotes $\mathsf{Crown}(P) \cup \{\top\}$. The set $\mathsf{Crown}_\top(P)$ has the structure of a bounded distributive lattice, with $\top$ the upper bound and $\{\emptyset\}$ the least element. The join operation is union followed by removal of elements which are proper subsets of others, while the meet operation is pairwise intersection.

In general, the canonical model is a quotient of this lattice of crowns. The key point to be observed here is that this lattice can be *very* large. If Φ does not contain any constraints, then the canonical model is the lattice of crowns itself. Note that this is a super-exponential construction. In general, there is no way to represent it explicitly when P contains more than a few elements. The bottom line is that any explicit representation of the canonical model is impossible for all but the most trivial of situations.

3.14 Hierarchies with complements In some applications, it is desired that types in the hierarchy have complements. It is possible to extend the results given here to such hierarchies; the key idea is to replace bounded distributive lattices with Boolean algebras in the characterization. The rôle of GBWPL's does not change. Some results along these lines are presented in [13].

3.15 Limitations Despite the elegance of this algebraic characterization, there are some inherent limitations. First of all, only positive, equality constraints are readily characterizable. It is possible to check for satisfaction of other types of constraints, such as type inequality constraints (e.g., $(\tau_1 \neq \tau_2)$ indirectly, but there is no direct construction of canonical models which satisfy such constraints.

Another major limitation is that the characterization is not computational; that is, there are no obvious tools of any efficiency for computing whether models exist, or for answering queries about existing models. Therefore, it is important to have an alternate characterization without these weaknesses. The logical characterization, presented in the next section, is a step in this direction.

4. Logical Characterization

In this section, a logical characterization of open specifications is developed. The approach consists of two distinct branches. First, a syntax for formulas, as well as a semantics, for constraints over a clean system P of types is developed from the elementary constraints introduced in Section 2. Second, a mapping from such formulas to a special propositional logic, denoted $\mathcal{L}_P$, is introduced. It is shown

that the two logics are equivalent for *one-element models*; that is, models for which the underlying universe contains only one element.

The advantage of tying the constraint logic to propositional logic is that there exists a wealth of knowledge on the theoretical and practical aspects of solving satisfiability and model identification problems within propositional logic. Thus, with this approach, computational tools are ready at hand.

Unfortunately, one element models are not adequate to model all situations. Therefore, additional results are needed which show how to combine one-element models (*qua* propositional models) to obtain general models of sets of constraints. Much of this section is devoted to the presentation of such results.

Some of the basic ideas in this section are based upon the initial part of [15]. However, the presentation given here emphasizes logical characterization in a general form, while the emphasis of that of [15] is upon laying the foundations for the development of efficient inference algorithms on certain classes of constraints. (See 5.3).

4.1 Notational convention Throughout this section, P denotes a clean set of types.

4.2 General constraints and their semantics The set $\mathsf{UnresConstr}(P)$ of *unrestricted constraints over* P is built up from $\mathsf{ElemConstr}^+(P)$ using the usual logical connectives $\wedge$, $\vee$, $\neg$. The semantics of such constraints is the obvious one; if S is an interpretation over P and $\varphi, \varphi_1, \varphi_2 \in \mathsf{UnresConstr}(P)$, then $S \models (\varphi_1 \wedge \varphi_2)$ iff $S \models \varphi_1$ and $S \models \varphi_2$; $S \models (\varphi_1 \vee \varphi_2)$ iff $S \models \varphi_2$ or $S \models \varphi_2$; $S \models (\neg\varphi)$ iff $S \not\models \varphi$.

The *positive (unrestricted) constraints*, denoted $\mathsf{UnresConstr}^+(P)$, is the subset of $\mathsf{UnresConstr}(P)$ which is constructed using only the connectives $\wedge$ and $\vee$; without negation. Given an arbitrary set $\Phi \subseteq \mathsf{UnresConstr}(P)$, the subset $\Phi \cap \mathsf{UnresConstr}^+(P)$ is denoted Φ^+.

The definition of open specification is also extended to this more general context. Specifically, an *open specification* (resp. *positive open specification*) is a pair (P, Φ) in which P is a clean set of types and $\Phi \subseteq \mathsf{UnresConstr}(P)$ (resp. $\Phi \subseteq \mathsf{UnresConstr}^+(P)$.

4.3 The propositional logic of one-element models Define the propositional logic $\mathcal{L}_P$ to have as proposition symbols the set $\{\mathfrak{r}_\tau \mid \tau \in \mathsf{Aug}(P)\}$. Also, within this context, **false** and **true** will be used to denote special propositions which always have the values false and true, respectively. For $\varphi \in \mathsf{ElemConstr}^+(P)$, associate a formula $\mathfrak{F}(\varphi)$ according to Table 1.

$\mathfrak{F}$ is extended to formulas in $\mathsf{UnresConstr}(P)$ in the obvious manner: $\mathfrak{F}((\varphi_1 \wedge \varphi_2)) = (\mathfrak{F}(\varphi_1) \wedge \mathfrak{F}(\varphi_2))$; $\mathfrak{F}((\varphi_1 \vee \varphi_2)) = (\mathfrak{F}(\varphi_1) \vee \mathfrak{F}(\varphi_2))$; $\mathfrak{F}((\neg\varphi)) = (\neg\mathfrak{F}(\varphi_1))$. $\mathfrak{F}$ may furthermore be extended to one-element interpretations. Let $S = (\{a\}, \mathfrak{O})$ be any one-element interpretation for P. Define the interpretation (i.e., truth assignment) $\mathfrak{F}(S)$ in $\mathcal{L}_P$ to be true on the propositions in the set $\{\mathfrak{r}_\tau \in \mathsf{Aug}(P) \mid \mathfrak{O}(\tau) = \{a\}\}$, and false otherwise.

Using this translation to propositional logic, it is possible to characterize the one-element models of an arbitrary family of constraints in a simple fashion. The proof of the following is immediate from the definition of $\mathfrak{F}$.

Constraint φ	Associated Logical Formula $\mathfrak{F}(\varphi)$
$(\bigsqcap\{\tau_1, \tau_2, .., \tau_n\} = \tau)$	$(\mathfrak{r}_{\tau_1} \wedge \mathfrak{r}_{\tau_2} \wedge \ldots \wedge \mathfrak{r}_{\tau_n}) \Leftrightarrow \mathfrak{r}_\tau$
$(\bigsqcup\{\tau_1, \tau_2, .., \tau_n\}) = \tau$	$(\mathfrak{r}_{\tau_1} \vee \mathfrak{r}_{\tau_2} \vee \ldots \vee \mathfrak{r}_{\tau_n}) \Leftrightarrow \mathfrak{r}_\tau$

Table 1. Translation from constraints to propositional logic.

4.4 Theorem — propositional models as models of open specifications *Let (P, Φ) be an open specification. A one-element interpretation $S = (\{a\}, \mathfrak{D})$ for P is a model of (P, Φ) iff $\mathfrak{F}(S)$ is a model of $\mathfrak{F}(\Phi)$.* □

In order to use the above result in a more general context, it is necessary to understand how one-element models combine to form arbitrary models. The next result shows that, at least for positive constraints, there is a direct componentwise decomposition.

4.5 Proposition *Let (P, Φ) be a positive open specification, and let $(\mathfrak{U}, \mathfrak{D})$ be an interpretation for P. Then S is a model for (P, Φ) iff for each $\varphi \in \Phi$ and every $a \in \mathfrak{U}$, $S_{|\{a\}} \models \varphi$.*

Proof For $\Phi \subseteq \mathsf{ElemConstr}^+(P)$, the result is immediate, since the basic set operations involved ($\cup$, $\cap$, and $=$) are all defined pointwise. For $\varphi = \varphi_1 \wedge \varphi_2 \wedge \ldots \varphi_n \in \mathsf{UnresConstr}^+(P)$, with each $\varphi_i \in \mathsf{ElemConstr}^+(P)$, it suffices to note that $M \models \varphi$ iff $M \models \varphi_i$ for each i, and then to replace Φ with $(\Phi \cup \{\varphi_1, \varphi_2, \ldots, \varphi_n\}) \setminus \{\varphi\}$. Thus, the results holds whenever $\Phi \subseteq \mathsf{UnresConstr}^+(P)$. Next, suppose that $\varphi \in \mathsf{UnresConstr}(P)$ is of the form $(\varphi_1 \vee \varphi_2 \vee \ldots \varphi_n)$ for $\varphi_1, \varphi_2, \ldots, \varphi_n \in \mathsf{UnresConstr}^+(P)$, By definition, there must be some i,, $1 \leq i \leq n$, for which $S \models \varphi_i$. The above result then applies. The most general case now follows easily by induction on the structure of the formula. □

4.6 Some issues with negative constraints Unfortunately, the above characterization does not extend to contexts with negation in the constraints. For example, consider an open specification which includes the three constraints $(\tau_1 \neq \tau_2)$, $(\tau_1 \neq \tau_3)$, and $(\tau_2 \neq \tau_3)$. Any model of these constraints must be based upon a universe with at least two distinct elements, since three distinct types are required. In general, a collection of constraints which mandates at least n distinct elements requires a universe of at least n elements. The following two paragraphs establish some more general results in the direction of model size.

4.7 Conjunctive normal form for constraints An $\mathsf{ElemConstr}(P)$-*clause* is an element of $\mathsf{UnresConstr}(P)$ of the form $\varphi_1 \vee \varphi_2 \vee \ldots \vee \varphi_n$, with $\varphi_i \in \mathsf{ElemConstr}(P)$ for $1 \leq i \leq n$. In this context, each $\varphi_i \in \mathsf{ElemConstr}(P)$ is called a $\mathsf{ElemConstr}(P)$-*literal*. A formula $\varphi \in \mathsf{UnresConstr}(P)$ is said to be in $\mathsf{UnresConstr}(P)$-*CNF* (or just *CNF*, if the context is clear) if it is the conjunction of $\mathsf{ElemConstr}(P)$-literals. It is easy to see that any constraint in $\mathsf{UnresConstr}(P)$

may be converted to an equivalent one which is in CNF. The procedure is analgous to that used for ordinary propositional formulas. This leads to the following results.

4.8 Theorem – characterization of model size *Let (P, Φ) be an open specification with each member of Φ in CNF. If (P, Φ) is satisfiable, then it has a model of cardinality no greater than the total number of* $\mathsf{ElemConstr}(P)$*-clauses in all formulas of Φ.*

Proof Assume that (P, Φ) is satisfiable, and let $S = (\mathfrak{U}, \mathfrak{D})$ be a model. Without loss of generality, Φ may be taken to be a single constraint ψ in CNF. If it is not of that form, just replace it with the conjunction of its members. Now, let $\varphi = \ell_1 \vee \ell_2 \vee \ldots \vee \ell_n$ be a conjunct of ψ. By assumption, φ is an $\mathsf{ElemConstr}(P)$-clause. Furthermore, it must be the case that $S \models \varphi$, so that $S \models \ell_i$ for some i. If $\ell_i \in \mathsf{ElemConstr}^+(P)$, then every one-element projection $S_{|\{a\}}$ is a model of ℓ_i, in view of 4.5. On the other hand, if $\ell_i \in \mathsf{ElemConstr}^-(P)$, then $\neg\ell_i$ must fail to be a model for some one-element projection $S_{|\{a\}}$, again in view of 4.5. In any case, there is a element $a_\varphi \in \mathfrak{U}$ with the property that $S_{|a_\varphi} \models \varphi$. Define $A = \{a_\varphi \mid \varphi \text{ is a conjunct of } \psi\}$. Then, $S_{|A} \models \psi$, and $\mathsf{Card}(A)$ is, by construction, no larger than the number of clauses in Φ. □

This bound on model size also provides a simple way of establishing decidability of satisfaction of open specification.

4.9 Corollary – decidability *The question of whether or not an arbitrary open specification (P, Φ) is satisfiable is decidable.*

Proof First, convert Φ to CNF. Then, compute the bound on model size n stipulated by the above theorem. Up to isomorphism, there can be only a finite number of models of size n or less, which can be tested, in turn. □

5. Complexity and Algorithms

The major focus of this paper is characterization. However, it is important to give a flavor for the computational implications of adopting an open-specification modelling strategy. Therefore, a brief overview of the most important aspects are provided here.

5.1 Important problems on open specifications Let $\mathbf{\Omega} = (P, \Phi)$ be an open specification.

Satisfiability: Does $\mathbf{\Omega}$ have a model? In other words, does there exist an interpretation S such that $S \models \Phi$?

Query solution: Given $\varphi \in \mathsf{Constraints}(P)$, does $\Phi \models \varphi$ hold?

It is important to point out that query solution is a particularly important problem, in light of the combinatorial explosion which can easily result when attempting to construct a canonical model.

5.2 Theorem — complexity results *Let* $\mathbf{S} = (P, \Phi)$ *be an open specification.*

(a) *The satisfiability problem for* $\mathbf{\Omega}$ *is NP-complete.*

(b) *The query solution problem for* $\mathbf{\Omega}$ *is co-NP-complete.*

These results remain valid even under the following circumstances:

(i) *The context is limited to positive constraints.*

(i) *The context is limited to positive equality constraints.*

(iii) *The context is limited to constraints with fanout at most three (that is, constraints which involve the operations* $\bigsqcap$ *and* $\bigsqcup$ *on at most three elements).*

The size of an instance is the length of the associated formula. □

Sketch of Proof (a) Given the result 4.4 of the previous section, it should be no surprise that these problems are closely related to the satisfaction problem for propositional formulas, which is known to be NP-compete [11, Sec. 2.6]. However, a formal proof of the NP-completeness of satisfiability for open specifications must present a reduction in the opposite direction, showing that satisfiability for propositional formulas can be reduced to satisfiability for open specifications. The proof is quite nontrivial; complete details may be found in [13, Sec. 2].
(b) This is immediate from (a), since $\Phi \models \varphi$ iff $\Phi \cup \{\neg\varphi\}$ is unsatisfiable.

5.3 Efficient inference over a class of constraints Despite the intractability results described above, it is possible to develop efficient inference algorithms over a useful subset of constraints. Results in this direction are reported in [15]. These results are based upon the fundamental observation that inference on Horn clauses is computable in linear time [9], [14].

6. Conclusions and Further Directions

Two distinct forms of characterization of openly specified type hierarchies have been provided. The *algebraic* characterization provides insight into the structure of such hierarchies, and in particular provides a canonical representation, in the form of a generalized bounded weak partial lattice, which recaptures in algebraic form a structure which embodies exactly those constraints which the specification mandates. The *logical* characterization provides a representation, in terms of propositional logic, which provides the basis for effective inference on the properties of such hierarchies.

For open specification to become a useful tool, it is clear that substantial further advances must be made in the tractability of inference on such structures. Classes of constraints which are at the same time useful in modelling those hierarchies which arise in practice and amenable to tractable inference algorithms must be identified. Although significant first steps in this direction appear in [15], significant further steps remain to be taken. Finally, in many situation there is further structure associated with the hierarchy. The computational results must be extended to representations which involve attributes, such as description logics [3] and formal concept analysis [10], as well as associated methods, as are commonly found in the object-oriented database context [2, 18, 19].

References

[1] J. Adámek, H. Herrlich, and G. Strecker. *Abstract and Concrete Categories*. Wiley-Interscience, 1990.

[2] C. Beeri. A formal approach to object-oriented databases. *Data and Knowledge Engineering*, 5:353–382, 1990.

[3] A. Borgida. Description logics in data management. *IEEE Trans. Knowledge Data Engrg.*, 7:671–682, 1995.

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

[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] B. A. Davey and H. A. Priestly. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[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] B. Ganter and R. Wille. *Formal Concept Analysis*. Springer-Verlag, 1999.

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

[12] G. Grätzer. *General Lattice Theory*. Birkhäuser Verlag, second edition, 1998.

[13] 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 R1.2B*, pages 29–120. DYANA, 1994.

[14] 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.

[15] S. J. Hegner. Efficient inference algorithms for databases of type hierarchies with open specification. Submitted for publication, 2001.

[16] S. J. Hegner. Computational and structural aspects of openly specified type hierarchies. In M. Moortgat, editor, *Logical Aspects of Computational Linguistics, Third International Conference, LACL '98 Grenoble, France, December 1998, Selected Papers*, in press, 2001.

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

[18] K.-D. Schewe and B. Thalheim. Fundamental concepts of object oriented databases. *Acta Cybernetica*, 11:49–84, 1993.

[19] J. Van den Bussche. *Formal Aspects of Object Identity in Database Manipulation*. PhD thesis, University of Antwerp, 1993.

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