November 1990

The Common Order-Theoretic Structure of Version Spaces and ATMS's

Carl A. Gunter
University of Pennsylvania

Teow-Hin Ngair
University of Pennsylvania

Prakash Panangaden
McGill University

Devika Subramanian
University of Cornell

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-86.

The Common Order-Theoretic Structure of Version Spaces and ATMS's

Abstract

This paper exposes the common order-theoretic properties of the structures manipulated by the version space algorithm [Mit78]and the assumption-based truth maintenance systems (ATMS) [dk86a,dk86b] by recasting them in the framework of convex spaces. Our analysis of version spaces in this framework reveals necessary and sufficient conditions for ensuring the preservation of an essential finite representability property in version space merging. This analysis is used to formulate several sufficient conditions for when a language will allow version spaces to be represented by finite sets of concepts (even when the universe of concepts may be infinite). We provide a new convex space based formulation of computation performs by an ATMS which extends the expressiveness of disjunctions in the systems. This approach obviates the need for hyper-resolution in dealing with disjunction and results in simpler label-update algorithms.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-86.

The Common Order-Theoretic Structure Of Version Spaces And ATMS's

MS-CIS-90-86
LOGIC & COMPUTATION 28

Carl A. Gunter
Teow-Hin Ngair
Prakash Panangaden
Devika Subramanian

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

November 1990

The Common Order-theoretic Structure of Version Spaces and ATMS's*

Carl A. Gunter
Univ. of Pennsylvania
gunter@cis.upenn.edu

Teow-Hin Ngair
Univ. of Pennsylvania
ngair@saul.cis.upenn.edu

Prakash Panangaden
McGill University
prakash@opus.cs.mcgill.ca

Devika Subramanian
Cornell University
devika@cs.cornell.edu

September 9, 1991

Abstract

This paper exposes the common order-theoretic properties of the structures manipulated by the version space algorithm [Mit78] and the assumption-based truth maintenance system (ATMS) [dK86a, dK86b] by recasting them in the framework of convex spaces. Our analysis of version spaces in this framework reveals necessary and sufficient conditions for ensuring the preservation of an essential finite representability property in version space merging. This analysis is used to formulate several sufficient conditions for when a language will allow version spaces to be represented by finite sets of concepts (even when the universe of concepts may be infinite). We provide a new convex space based formulation of computations performed by an ATMS which extends the expressiveness of disjunctions in the system. This approach obviates the need for hyper-resolution in dealing with disjunctions and results in simpler label-update algorithms. The semantics of the label-update algorithms is established using the order-theoretic characterization of the algorithms.

1 Introduction

This paper arose out of the observation that the version space algorithm and the ATMS label-update algorithms operate on very similar structures. The version space algorithm learns concept descriptions from examples. Central to this algorithm is the notion of all concept descriptions consistent with a given set of positive and negative examples. The assumption-based truth maintenance system for recording dependencies during reasoning maintains labels for a proposition which encode all environments in which that proposition is true.

This gives rise to two questions: one, what is the precise nature of the relationship between the structures employed by these algorithms taken from two different problem areas, and second: are the *computations* performed by the two algorithms related, and what special properties of these structures do they depend on? Our aim is to find a common mathematical basis in order to determine applicability conditions for the algorithms, and to cast the computations done in a form that reveals new, more efficient implementations.

We first show that the version space of a concept is a special case of a convex space. We re-express the computation performed by the version space merging operation in terms of lattice operations on convex spaces. The mathematics of these structures is then brought to bear on the question of ensuring that finite representability is preserved under the merging operation. We derive a necessary and sufficient condition, called the $\mathcal{MW}$ property which identifies the class of concept languages for which version spaces arising from finitely many observations are finitely representable. Mitchell [Mit78] calls such concept languages admissible. The $\mathcal{MW}$ property is the first condition on admissibility that captures both finite and infinite concept languages. We also show that extended version spaces described in Mitchell's thesis can be treated within the same mathematical framework, and we provide a rigorous account for the construction of these spaces.

We then recast the label-update algorithms in the ATMS also as lattice operations on a convex space. This help us to establish a simple semantics for both the basic and extended ATMS algorithms. An important result is a new

*An extendend abstract of this paper appears in: **Ninth National Conference on Artificial Intelligence**, Anaheim, CA, July 1991, pp. 500–505.

algorithm for computing labels that handles complex disjunctions such as *choose*($\{A, B, C\}$, $\{D, E, F\}$), which stands for either A, B, and C are true, or D, E, and F are true. This algorithm is a natural extension of de Kleer's original ATMS algorithm and does not require hyper-resolution rules to compute minimal, consistent, sound and complete labels. We also identify several easy-to-detect conditions under which label updates can be done efficiently.

The paper is structured as follows. In Section 2 we introduce the mathematics of ordered spaces and closure operators on them. In Section 3 we present the basic theory of convex spaces: finite representability and the $\mathcal{MW}$ properties, the isomorphism between convex spaces and their boundary set representations, as well as algorithms for computing meets and joins of convex spaces and their boundaries. The version space and extended version space are formulated in terms of convex spaces in Section 4, and we present three admissibility results for concept languages. In Section 5, we perform a similar analysis of the ATMS algorithm and show that the label computation performed by the disjunction-free ATMS is akin to the boundary set updates of the version space algorithm. We extend the class of disjunctions expressible within the ATMS and use a new closure operator to derive a new label-update algorithm which is more efficient in general, and does not rely on hyper-resolution rules. We then show that these ATMS algorithms have a simple and consistent semantics.

2 Mathematical Background

In this section we develop some of the basic mathematical theory of ordered spaces and closure operators on them. The intuition that is captured by order structures is a qualitative notion of *information* content. Roughly speaking, the orders that we use express the notion that one item is more informative than another. Closure operators are certain special functions that describe operations that *increase* the information content of the items that they manipulate. Thus the common order-theoretic mathematical structures of the systems that we analyze correspond to common intuitions about how these systems represent and manipulate information.

The viewpoint that one can understand the mathematics of data items, and programs that manipulate them, in terms of ordered structures is primarily due to Dana Scott [Sco76, Sco82] in his work on programming language semantics. It is in [Sco82], that one sees the basic intuition relating the order structure to information content spelled out precisely. A rich subject, known as *domain theory* has arisen from his work; see for example the recent review article by Gunter and Scott [GS90]. A rather more daunting but very informative reference is [GHK*80]. Our treatment is intended to be, for the most part, self contained.

Closure operators first appeared in the work of Moore in his study of lattices. Scott's original paper on data types already contains a discussion of closure operators though they are used by him for a completely different purpose. The idea that processes that increase information content can be modeled by closure operators appears in [JPP89] and [SRP91] in the context of constraint programming.

Order Structures.

We begin by recapitulating some basic definitions. A *partially ordered set* (or "poset" for short) is a set P together with a binary relation $\preceq$ which is transitive, reflexive and anti-symmetric. We assume that the reader is familiar with the concepts, *least upper bound* also called "join" or "sup", written as $\vee$ in infix form or $\bigvee$ in prefix form and *greatest lower bound*, written as $\wedge$ or $\bigwedge$ and also called "meet" or "inf". A *lattice* is a poset in which every finite subset has a least upper bound and a greatest lower bound together with some equations that describe how meets and joins interact. The classical example of a lattice is the *powerset* of a set X, i.e. the collection of all subsets of X ordered by inclusion; usually written $\mathcal{P}(X)$. A *sublattice*, S, of a lattice L, is a lattice that has as its underlying set a subset of L and the the meet and join operations are those of L restricted to the subset. It is important to keep in mind the distinction between the sublattices and subsets of lattices that happen to be lattices. In the case of a sublattice we mean that when we compute the meet or join of two elements in the sublattice we get the same result as when we compute the meet or join in the original lattice. This need not be true in a subset that happens to be a lattice. We will see examples of both situations.

As is usual in the study of algebraic structures, an important role is played by structure preserving functions. A function f, from a poset $(P, \preceq)$ to another poset $(P', \preceq')$ is said to be monotone (or monotonic) if whenever $x \preceq y$ then $f(x) \preceq' f(y)$. Intuitively one thinks of the elements of an ordered space as being ordered by information content. Functions then stand for operations that produce some output given some input information. It is clear that for most reasonable notions of transforming information, better input information should result in better output

information. The collection of monotone functions forms a lattice in its own right. The ordering is given by $f \preceq g$ if, for all x in L, $f(x) \preceq g(x)$. It is of course possible to define other orderings on functions but this one is the most widely used and is commonly called the *extensional* ordering.

If one has a function f from L to itself, then an x in L such that $f(x) = x$ is called a *fixed point* of f. The following theorem is an elementary special case of much more general theorems, but is all that we shall need in this paper.

Theorem 1 *Any monotone function f from a finite lattice L, with a least element $\perp$, to itself has a least fixed point.*

Proof: Consider the set $S = \{\perp, f(\perp), f(f(\perp)), \ldots\}$. Since L is finite there are only finitely many distinct elements in this set. Since f is monotone and $\perp$ is the least element we have the following ordering among the members of S, $\perp \preceq f(\perp) \preceq f(f(\perp)) \preceq \ldots$; in other words for any member a of S we have $a \preceq f(a)$. Let x be the largest element of S. Now, by the definition of S, $f(x)$ is in S but $f(x)$ is larger than x, on the other hand, x is the largest element of S thus $f(x) = x$. Suppose that u is another fixed point of f. Now $\perp \preceq u$, by definition of $\perp$. Using the monotonicity of f, we have $f(\perp) \preceq f(u) = u$, and by applying f repeatedly we see that all the elements of S are less than u. Thus, in particular, x is less than u, i.e. x is the least fixed point of f. □

The iterative process used in the proof of this theorem is the idea behind techniques to search for least fixed points.

Closure Operations on Lattices.

A very important class of monotone functions are *closure operators*. They occur throughout this paper to represent processes that correspond to adding information or "completing" a collection in some way. Therefore, it will be convenient to develop some general properties of closure operations.

Definition: Given a lattice L and a monotone function $f : L \longrightarrow L$, f is called a *closure operation* if it satisfies the following properties:

C1. $x \preceq f(x)$ for any x, in L, and

C2. $f(f(x)) = f(x)$. □

Note that if x is in L, then C2 says that $f(x)$ is a fixed point of f. Intuitively, C1 says that f increases information and C2 says that adding the same information twice is no different from adding it once. We frequently say that f is a closure operator *on* L.

There are many familiar examples of closure operations. For instance, consider a language $\mathcal{L}$ of propositions over a fixed alphabet of propositional atoms. Let $\models$ stand for the usual notion of entailment between propositions. Consider the lattice $\mathcal{P}(\mathcal{L})$. Now consider the function $T : \mathcal{P}(\mathcal{L}) \longrightarrow \mathcal{P}(\mathcal{L})$ defined by $T(S) = \{\phi \in \mathcal{L} \mid u \models \phi, u \subseteq S\}$. Thus $T(S)$ adds to a set all the propositions that are entailed by it. It is easy to see that this is a closure operator. This is, in fact, a very typical closure operator. Other examples from mathematics are the subgroup generated by a subset of a group, the span of a set of elements in a vector space and the closure of a set in a topological space.

Another example of a closure operator that will be of interest later, is defined as follows:

Definition: Given any poset P, a subset $C \subseteq P$ is said to be *downward closed* if $p \in C$ and $p' \preceq p$ implies $p' \in C$. Furthermore, given any subset $S \subseteq P$, the *downward closure* of S is the set $\downarrow S = \{p \in P \mid \exists p' \in S, p \preceq p'\}$. □

The operation $\downarrow$ is a closure operation on subsets of P. Note that a set C is downward closed if, and only if, it is a fixed point of this operation, that is $C = \downarrow C$.

Indeed, it is a general fact that the image of a closure operation (*i.e.* $\mathcal{CL}(f)$) is exactly its set of fixed points. Moreover a closure is uniquely determined by its set of fixed points. In our subsequent discussion we will often be more interested in the fixed points of a closure operation than in the closure operation itself. In fact, one reason for our interest in closure operations is the set of properties that the fixed points of such an operator possess. In particular, we can use the following Lemma. For any poset P, let $\uparrow x$ for the set $\{u \in L \mid x \preceq u\}$.

Lemma 2 *Suppose that f is a closure operator on a lattice L.*

1. *Then f is uniquely determined by its image according to the formula*

$$f(x) = \mathit{min}\,(\uparrow x \cap \mathcal{CL}(f))$$

   where $\mathit{min}(S)$ means the smallest element of S.

2. *If X is a subset of $\mathcal{CL}(f)$, then $\bigwedge X$ is also in $\mathcal{CL}(f)$.*

3. *$(\mathcal{CL}(f); \wedge^f, \vee^f)$ forms a lattice where:*

$$\begin{aligned} \textstyle\bigwedge^f X &= \textstyle\bigwedge X \\ \textstyle\bigvee^f X &= f(\textstyle\bigvee X) \end{aligned}$$

for each $X \subseteq \mathcal{CL}(f)$.

Proof: 1. Note that $f(x)$ is in the set $\uparrow x \cap \mathcal{CL}(f)$. According to C1, $x \preceq f(x)$, i.e. $f(x) \in \uparrow(x)$, whereas by C2, $f(x)$ is a fixed point of f, in other words, $f(x) \in \mathcal{CL}(f)$. Now suppose that u is any other element of $\uparrow x \cap \mathcal{CL}(f)$. In other words, $x \preceq u$ and u is a fixed point of f. By monotonicity of f, $f(x) \preceq f(u)$; but, u is a fixed point of f so $f(x) \preceq u$. Thus $f(x)$ is indeed the least element of the set.

2. First, we have $\bigwedge X \preceq f(\bigwedge X)$, since f is a closure operator. Next, by monotonicity of f, $f(\bigwedge X) \preceq u$ for any u in X. Thus, by definition of greatest lower bound, $f(\bigwedge X) \preceq \bigwedge\{f(u) \mid u \in X\}$. The elements of X are all fixed points of f so $\{f(u) \mid u \in X\} = \bigwedge X$. In short we have $f(\bigwedge X) = \bigwedge X$.

3. The result for meets is immediate from (2) above. The second follows immediately from the fact that $f(\bigvee X)$ is the least fixed point of f above $\bigvee X$. We omit the checking of the various equations satisfied by meets and joins in lattices. □

Although the set of fixed points of a closure operator forms a lattice in its own right, *it may not be a sublattice* of the original lattice on which the closure operator is defined. So some care must be take about the difference between the operations $\wedge^f$ and $\vee^f$ on the closed sets versus the meet and join operations on the original lattice. It is easy to see that the operations $\wedge^f$ and $\vee^f$ satisfy the associative, commutative and absorptive properties. For the rest of this paper, we will omit the superscripts in the join and meet operations when the context makes obvious the closure operation that we are discussing.

Many of the lattices that we consider below are powerset lattices of some set. In other words, we have some set X and we consider the set of subsets of X ordered by inclusion, i.e $\mathcal{P}(X)$. A closure operator on this lattice takes a subset of X to what is called a *closed subset* of X. What the phrase "closed subset" means varies as one considers different closure operators. All the general theory that we have established for closure operators applies here. The closed subsets form a lattice as we have already noted in the previous lemma but in the special case of a subset lattice the meets and joins can be defined in terms of union and intersection. We have the following special instance of Lemma 2

Lemma 3 *If* $f : \mathcal{P}(P) \longrightarrow \mathcal{P}(P)$ *is a closure operation, then for any subset* $S \subseteq \mathcal{CL}(f)$,

1. $f(\bigcap S) = \bigcap S$,
2. $f(\bigcup S)$ = *smallest element in* $\mathcal{CL}(f)$ *containing* $\bigcup S$. □

The following lemma will be pivotal in our subsequent discussion of closed sets.

Lemma 4 *If* $f : \mathcal{P}(P) \longrightarrow \mathcal{P}(P)$ *is a closure operation* S *is a subset of* $\mathcal{CL}(f)$, *such that* $f(\bigcup S) = \bigcup S$, *then*

1. $\bigcup S = \bigvee S$,
2. $(\bigvee S) \wedge C' = \bigvee_{C \in S} (C \wedge C'), \quad \forall C' \in \mathcal{CL}(f)$.

Proof: 1. Immediate from the definition of join in $\mathcal{CL}(f)$.

2. Let $X = (\bigvee S) \wedge C'$. By 1. above, we have $X = (\bigcup S) \wedge C' = (\bigcup S) \cap C'$. By distributivity of $\cap$ and $\cup$, we get $X = \bigcup_{C \in S}(C \cap C')$. But $X = f(X)$, so by 1. again, we get $X = \bigvee_{C \in S}(C \wedge C')$. □

If we are working with closed sets, then the third part of Lemma 2 says that we can substitute meet for intersection. Moreover, if the result of a union is closed, then Lemma 4 says that we can substitute join for union and attain a limited form of distributivity. We will see later that it is desirable to substitute meet and join for intersection and union respectively in order to tap useful properties of the lattice operations. However, this requires that every union operation produce a closed set. One way of achieving this is to restrict the possible combinations of the operands for the union operation so that the results are always closed, another is to restrict the universe of closed sets so that every union of closed sets is a closed set. In the discussion below, we will see that the theory of extended version spaces falls into the former category and the theory of ATMS's falls into the latter.

Common Fixed Points of Closure Operations.

The next crucial property that we wish to establish is how one finds least fixed points for closure operators. Now any closure operator is a monotone function and thus has a least fixed point if the lattice has a least element. Closure operators enjoy a special property that is not shared by arbitrary monotone functions. Any *family* of closure operators has a least *common* fixed point. Furthermore, this least common fixed point can be calculated by iterating all the closure operators in the family sufficiently often. The next theorem makes this precise.

Theorem 5 *Suppose that $\{f_i \mid i \in I\}$ is a family of closure operators on L indexed by some arbitrary set I. Suppose that L is finite and has a least element $\perp$. Let σ be an infinite sequence of members of I such that every element of I appears in every suffix of σ. Let $\sigma[n]$ be the nth element of σ. Consider the set*

$$S = \{\perp, f_{\sigma[1]}(\perp), f_{\sigma[2]}(f_{\sigma[1]}(\perp)), \ldots\}.$$

The least upper bound of S, call it x, is the least common fixed point of all the closure operators $\{f_i \mid i \in I\}$.

Proof: We need a convenient notation for sequences of f_is composed together. We write σ_n for the length n prefix of σ. We write f_{σ_n} for the composition $f_{\sigma[n]} \circ f_{\sigma[n-1]} \circ \cdots \circ f_{\sigma[1]}$. First, note that the elements are written in increasing order because all the f_i are closure operators. Second, because L is finite, the least upper bound is attained at some finite stage, i.e. for some n the least upper bound of S is $f_{\sigma[n]}(f_{\sigma[n-1]}(\ldots(f_{\sigma[1]}(\perp))))$ or, using our notation, f_{σ_n} Consider any of the closure operators, say f_i. We claim that $f_i(x)$ is less than x. To see this consider any m greater than n such that $\sigma[m] = i$, we know there must be such an m from the assumption on σ. Now we have $x = f_{\sigma_n}(\perp) \preceq f_{\sigma_m}(\perp)$, the later inequality follows from the assumption that all the f_i are closure operators. But $f_i(x) = f_i \circ f_{\sigma_n}(\perp) \preceq f_i \circ \ldots \circ f_{\sigma_n}(\perp) = f_{\sigma_m}(\perp)$. The inequality again follows from the fact that closure operators increase their arguments. But notice that $f_{\sigma_m}(\perp)$ is in the set S and that x is the least upper bound of S, so $f_i(x) \preceq f_{\sigma_m}(\perp) \preceq x$. Since f_i is a closure operator we must have $x \preceq f_i(x)$, in other words, $x = f_i(x)$. Thus, x is a fixed point of any of the closure operators in our set. Suppose that u is any other common fixed point, we have, by an easy induction on m, that for all m, $f_{\sigma_m}(\perp) \preceq u$; using the monotonicity of the f_i and the fact that $\perp \preceq u$. Thus, u is an upper bound for S and, since x is the least upper bound, $x \preceq u$. □

The upshot of this theorem is that if we wish to find the least common fixed point of a set of closure operators we need only apply each one often enough in succession, not necessarily in any systematic order, and we will find the fixed point. The appearance of the infinite sequence σ in the proof is only to formalize the notion of "often enough" it need not cause alarm to those readers worried about the effectiveness of the process we have described.

Φ Operators.

A closure operator which will be the focus of our attention when we discuss assumption based truth maintenance systems later is defined as follows:

Definition: Let $(P, \preceq)$ be any lattice. Given any subset $T \in \mathcal{P}(P)$, the operation Φ_T is defined on downward closed sets $C \in \mathcal{P}(P)$ by:

$$\Phi_T(C) = \{p \in P \mid \forall t \in T, p \wedge t \in C\}. \ \square$$

It is not difficult to check that $\Phi_T(C)$ is downward closed and Φ_T is itself a closure operation. To understand the Φ_T operation more "intuitively" it is helpful to think in terms of downward closed sets generated by elements of P. The *principal ideal generated by* $p \in P$ is the set $\downarrow\{p\}$ of elements of P that are below p. Given any downward closed $A \subseteq P$, let us say that the set $\{p \in A \mid \exists p' \in T, \text{ s.t. } p \preceq p'\}$ is the *restriction* of A by T. So, one can think of $\Phi_T(C)$ as the expansion of C to include those elements of P such that the restrictions of their principal ideals by T are subsets of C. Some basic properties of Φ are:

Lemma 6 *1. $\Phi_A \circ \Phi_B = \Phi_T$ where $T = \{t \in P \mid \exists t_A \in A, \exists t_B \in B, t = t_A \wedge t_B\}$.*

2. $\Phi_A \circ \Phi_B = \Phi_B \circ \Phi_A$.

Proof: 1. Given any $A, B \in \mathcal{P}(P)$ and any downward closed C, then for all $p \in P$, we have $p \in \Phi_A \circ \Phi_B(C)$ if and only if for all $t_A \in A, t_B \in B$ we have $(p \wedge t_A) \wedge t_B \in C$, which means $p \wedge (t_A \wedge t_B) \in C$ which means that $p \in \Phi_T(C)$.

2. This follows immediately from 1 and the commutativity of meets. □

Theorem 7 *Given any $\Phi_{T_1}, \ldots, \Phi_{T_n}$ and any downward closed C, $F = \Phi_{T_1} \circ \cdots \circ \Phi_{T_n}(C)$ is the least* common *fixed point of every $\Phi_{T_i}, 1 \leq i \leq n$, above C.*

Proof: Given any $\Phi_{T_i}, 1 \leq i \leq n$ and any C, by using the commutative and idempotent properties of $\circ$, we have:

$$\begin{aligned} & \Phi_{T_i}(F) \\ = \ & \Phi_{T_i} \circ \Phi_{T_1} \circ \cdots \circ \Phi_{T_n}(C) \\ = \ & \Phi_{T_1} \circ \cdots \circ \Phi_{T_i} \circ \Phi_{T_i} \circ \cdots \circ \Phi_{T_n}(C) \\ = \ & \Phi_{T_1} \circ \cdots \circ \Phi_{T_i} \circ \cdots \circ \Phi_{T_n}(C) \\ = \ & F. \end{aligned}$$

So F is a fixed point of every Φ_{T_i} and F contains C. Furthermore, given any $Y \in \mathcal{P}(P)$ with the same properties, since $C \subseteq Y$, we have $\Phi_{T_n}(C) \subseteq \Phi_{T_n}(Y) = Y$. Hence, by applying $\Phi_{T_{n-1}}, \ldots, \Phi_{T_1}$ successively to the two sides

of the inequality, we get $F = \Phi_{T_1} \circ \cdots \circ \Phi_{T_n}(C) \subseteq Y$. Therefore, F is the least common fixed point of every $\Phi_{T_i}, 1 \leq i \leq n$, above C. □

Observe that $\Phi_{T_1} \circ \cdots \circ \Phi_{T_n}$ is itself a closure operation. However, it is not true in general for the composition of closure operations to be a closure operation because idempotence usually fails.

3 Convex Spaces

In this section we develop some of the basic theory of convex spaces that we will need later. In mathematics, convex spaces are most familiar in the context of geometry where, for example, a subset C of the real plane is said to be convex if the line connecting any pair of points of C lies entirely within C. However, our interest in this paper is in convex spaces in a poset. A subset C contained in a poset P is said to be a *convex space* if, whenever $p_1 \preceq p \preceq p_2$ and $p_1, p_2 \in C$, then $p \in C$. In this section we focus on analyzing the purely order-theoretic characteristics of convex spaces. To this end, we begin by noting that much of the theory of closure operators discussed in the previous section is applicable to convex spaces. To see why, let $c : \mathcal{P}(P) \longrightarrow \mathcal{P}(P)$ be the function defined by

$$c(C) = \{p \in P \mid \exists p_1, p_2 \in C,\ s.t.\ p_1 \preceq p \preceq p_2\}$$

The map c is called the *convex closure* and it is easily shown to be a closure operation whose fixed points are exactly the convex spaces of P. In particular, it follows from Lemma 2 that convex spaces form a complete lattice under the ordering defined by set inclusion.

Boundary sets and finite representability.

The key characteristic of convex spaces that underlies their usefulness in the applications we shall discuss is the simple fact that they may sometimes be represented by their upper and lower fringes. In a computational setting this may mean that a small set can be used to represent a large one. Since computers can only hold finite sets of data, if this set is to be represented directly as something like a list, then it must be finite; however, nothing prevents the larger set from being infinite—and, in practice, it often *will be* infinite. We now develop some results about convex spaces which can be finitely represented with their fringes.

Definition: Given any $A, B \in \mathcal{P}(P)$, define $\mathcal{B}(A, B) = \{p \in P \mid \exists p_1 \in A, \exists p_2 \in B,\ s.t.\ p_1 \preceq p \preceq p_2\}$. An *interval* (in a poset) is a set of the form $\mathcal{B}(\{p_1\}, \{p_2\})$. □

Lemma 8 *For all $A, B \in \mathcal{P}(P)$, $\mathcal{B}(A, B)$ is a convex space.*

Proof: We know that $c(\mathcal{B}(A, B)) \supseteq \mathcal{B}(A, B)$. To show equality, we observe that for all $p \in c(\mathcal{B}(A, B))$, there exist $p_1, p_2 \in \mathcal{B}(A, B)$ such that $p_1 \preceq p \preceq p_2$. But $p_1, p_2 \in \mathcal{B}(A, B)$ implies that there exist $p_1' \in A$ and $p_2' \in B$ such that $p_1' \preceq p_1$ and $p_2 \preceq p_2'$. Hence, $p \in \mathcal{B}(A, B)$. □

Given any $C \in \mathcal{P}(P)$, define:[1]

- $\mathcal{MIN}(C) = \{p \in C \mid \forall p' \in C, p' \preceq p \Rightarrow p' = p\}$,
- $\mathcal{MAX}(C) = \{p \in C \mid \forall p' \in C, p' \succeq p \Rightarrow p' = p\}$.

$\mathcal{MIN}(C)$ and $\mathcal{MAX}(C)$ are called the *boundary sets* of C. A subset $C \in \mathcal{P}(P)$ is *representable by boundary sets* if $C = \{p \in P \mid \exists s \in \mathcal{MIN}(C), \exists g \in \mathcal{MAX}(C),\ s.t.\ s \preceq p \preceq g\}$. Furthermore, if $\mathcal{MIN}(C)$ and $\mathcal{MAX}(C)$ are both finite, then C is said to be *finitely representable*. Observe that any subset that is representable by boundary sets is a convex space. An example of a finitely representable convex space C is shown in figure 1, where $\mathcal{MIN}(C) = \{q_1, q_2\}$ and $\mathcal{MAX}(C) = \{p_1, p_2, p_3\}$. Each of the quadrilaterals represents the interval bounded by two concepts (where the pairs of elements p_1, q_2 and p_3, q_1 are unrelated); their union represents the convex space C. Each of the quadrilateral could have infinitely many elements.

Lemma 9 *If $C = \mathcal{B}(X, Y)$ for some finite $X, Y \in \mathcal{P}(P)$, then C is finitely representable.*

Proof: Let $A = \{p \in \mathcal{MIN}(X) \mid \exists p' \in Y,\ s.t.\ p \preceq p'\}, B = \{p \in \mathcal{MAX}(Y) \mid \exists p' \in X,\ s.t.\ p' \preceq p\}$. By the definition of $\mathcal{B}(X, Y)$, we have $\mathcal{MIN}(C) = A, \mathcal{MAX}(C) = B$ and both are finite. Furthermore, for all $p \in C$, there exist $p_1 \in X, p_2 \in Y$ such that $p_1 \preceq p \preceq p_2$. Since X and Y are finite, hence bounded by A and B, this implies that there exist $p_1' \in A, p_2' \in B$ such that $p_1' \preceq p_1 \preceq p \preceq p_2 \preceq p_2'$, *i.e.* C is representable by the boundary sets A and B. Hence, C is finitely representable. □

[1] It should be noted that many of our notations and definitions related to convex spaces and version spaces are adapted from the works of Tom Mitchell [Mit78] and Hyam Hirsh [Hir90]

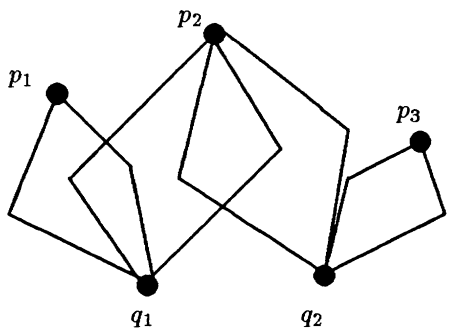

Figure 1: A finitely representable convex space

For a poset P, let $F(P)$ be the set of finitely representable convex spaces of P ordered by *superset* inclusion, that is, $C \preceq C'$ iff $C \subseteq C'$. Elements of $F(P)$ can be represented as a pairs of finite sets since any such element can be recovered as the convex closure of its upper and lower boundaries. In fact, one can characterize $F(P)$ more abstractly simply as such a set of pairs under an appropriate ordering. To see this, we introduce some order-theoretic notation taken from work on the study of what are called *powerdomains* in the semantics of programming languages (for example, see [Gun93] and the references there). Suppose U and V are finite subsets of P. We define three binary relations as follows:

- $U \preceq^\sharp V$ iff for every $y \in V$ there is a $x \in U$ such that $x \preceq y$,
- $U \preceq^\flat V$ iff for every $x \in U$ there is a $y \in V$ such that $x \preceq y$,
- $U \preceq^\natural V$ iff $U \preceq^\sharp V$ and $U \preceq^\flat V$

In a poset P, an *anti-chain* A is a subset of P with the property that, whenever $p, q \in A$ and $p \preceq q$, then $p = q$.

Definition: Let P be a poset. A *fringe* is a pair (S, G) such that S, G are finite anti-chains of P and $S \preceq^\natural G$. The poset $G(P)$ is the set of fringes under the ordering:

$$(S, G) \preceq (S', G') \text{ iff } S \preceq^\sharp S' \text{ and } G' \preceq^\flat G$$

□

We leave for the reader the demonstration that fringes do form a poset under this ordering. The following result shows that finitely representable convex spaces can indeed be viewed as pairs of finite sets:

Theorem 10 *For any poset P, the poset $F(P)$ of finitely representable convex spaces is isomorphic to the poset $G(P)$ of fringes.*

Proof: Define maps $f : F(P) \longrightarrow G(P)$ and $g : G(P) \longrightarrow F(P)$ as follows:

$$f : C \mapsto (\mathcal{MIN}(C), \mathcal{MAX}(C))$$
$$g : (S, G) \mapsto \mathcal{B}(S, G)$$

It is easy to check that these maps are well-defined (in particular, that $(\mathcal{MIN}(C), \mathcal{MAX}(C))$ *is*, in fact, a fringe). Let us first check that f is monotone. Suppose that $C \supseteq C'$ and suppose $p' \in \mathcal{MIN}(C')$. Since C is finitely representable, there is some $p \in \mathcal{MIN}(C)$ with $p \preceq p'$. Hence we may conclude that $\mathcal{MIN}(C) \preceq^\natural \mathcal{MIN}(C')$. On the other hand, if $p' \in \mathcal{MAX}(C')$, then, by finite representability of C, there is some $p \in \mathcal{MAX}(C)$ with $p' \preceq p$. Hence $\mathcal{MAX}(C') \preceq^\flat \mathcal{MAX}(C)$. Thus $f(C) \preceq f(C')$.

Next we check that g is monotone. Suppose that $(S, G) \preceq (S', G')$ and $p' \in \mathcal{B}(S', G')$. Then $p'_1 \preceq p' \preceq p'_2$ for some $p'_1 \in S'$ and $p'_2 \in G'$. Now, $S \preceq^\sharp S'$ means that there is some $p_1 \in G$ such that $p_1 \preceq p'_1$. On the other hand, $G' \preceq^\flat G$ means that there is some $p_2 \in G$ such that $p'_2 \preceq p_2$. Hence $p \in \mathcal{B}(S, G)$ and we may conclude that $g(S, G) \preceq g(S', G')$.

To complete the proof the f, g define an isomorphism, we must show that the compositions $f \circ g$ and $g \circ f$ yield the identity. To see that $(f \circ g)(S, G) = (S, G)$ we must demonstrate that $\mathcal{MIN}(\mathcal{B}(S, G)) = S$ and $\mathcal{MAX}(\mathcal{B}(S, G)) = G$. For the first of these, suppose that p is a minimal element of $\mathcal{B}(S, G)$. Since $p \in \mathcal{B}(S, G)$, there is some $p' \in S$ with $p' \preceq p$. Since p is minimal, we must have $p = p'$. On the other hand, if $p \in S$ then $p \in \mathcal{B}(S, G)$ so there is minimal element of $\mathcal{B}(S, G)$ below p. But minimal elements of $\mathcal{B}(S, G)$ are in S and related elements of S must be equal since S is an anti-chain. Hence $p \in \mathcal{MIN}(\mathcal{B}(S, G))$. The proof that $\mathcal{MAX}(\mathcal{B}(S, G)) = G$ is similar. To show that $(g \circ f)(C) = C$ we must prove that $\mathcal{B}(\mathcal{MIN}(C), \mathcal{MAX}(C)) = C$. But this follows immediately from the assumption that C is finitely representable. □

Ensuring finite representability.

Our primary interest is in the joins and meets of finitely representable convex spaces. It is easy to see that the *join* of two such spaces is again finitely representable. However, the *meet* of finitely representable spaces *may not be*

finitely representable! It is only under special circumstances that this will be the case. What we want to know is whether, for a given poset P, the poset $F(P) \cong G(P)$ is a sublattice of the convex spaces of P. We now present a criterion which insures that finite representability is preserved under intersections. Another such criterion will be presented in the next section.

In a lattice, a pair of elements x, y has a *least* upper bound $x \vee y$ and a *greatest* lower bound $x \wedge y$. But for given elements x, y of an arbitrary poset P there may be no such distinguished upper and lower bounds. On the other hand, if P is finite, then there *will be* a set u such that

1. $x, y \leq z$ for each $z \in u$
2. if $x, y \leq z'$, then there is a $z \in u$ such that $z \leq z'$.

Namely, u is the set of *minimal* upper bounds of x, y. Similarly, there is a set v of *maximal* lower bounds of x, y. The sets u and v may be viewed as a kind of *quasi*-join and *quasi*-meet of x, y. Now, in an infinite poset P, there is no guarantee that, for a given pair $x, y \in P$, a set u having properties 1,2 above exists (we leave the search for a counterexample to the reader). Hence, in infinite posets, the existence of such a form of quasi-join and quasi-meet is a special property of the poset. This concept is familiar, for example, in the study of the mathematical semantics of programming languages where it is related to what is usually called *property* $\mathcal{M}$ [Smy83, GJ88]. In the current context, our goal is to show that property $\mathcal{M}$ and its dual (which one might call "property $\mathcal{W}$") are related to the problem of the *admissibility* of concept spaces that use the version space algorithm. To this end, we begin with a rigorous definition of quasi-join and quasi-meet:

Definition: Let P be a poset with $p_1, p_2 \in P$ and $S, G \subseteq P$. We define the *quasi-join of* p_1 *and* p_2 *relative to* G as the set

$$\tilde{\bigvee}(p_1, p_2, G) = \mathcal{MIN}(\{p \in P \mid p_1 \preceq p, p_2 \preceq p, \text{ and } \forall g \in G, p \preceq g\})$$

and the the *quasi-meet of* p_1 *and* p_2 *relative to* S as the set

$$\tilde{\bigwedge}(p_1, p_2, S) = \mathcal{MAX}(\{p \in P \mid p_1 \succeq p, p_2 \succeq p, \text{ and } \forall s \in S, p \succeq s\})$$

□

A graphical representation of the definition is shown in figure 2 where the quasi-join and quasi-meet of p_1, p_2 are $\{g_1, \ldots, g_m\}$ and $\{s_1, \ldots, s_n\}$ respectively. In some contexts, the set $\tilde{\bigvee}(p_1, p_2, \phi)$ is called the "most specialized generalizations" of p_1, p_2 and $\tilde{\bigwedge}(p_1, p_2, \phi)$ is called "most general specializations" of p_1, p_2.

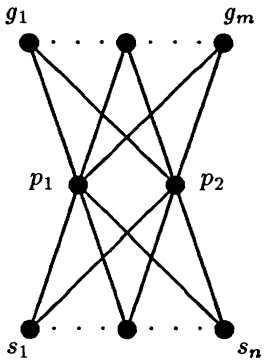

Figure 2: The $\mathcal{MW}$ property

Property $\mathcal{MW}$ asserts that the quasi-join and quasi-meet of a pair of elements are finite and "cover" the elements from above and below respectively:

Definition: A poset P is said to have the $\mathcal{MW}$ property if it is finitely representable and for all $p_1, p_2 \in P$:

M1. $\tilde{\bigvee}(p_1, p_2, \phi)$ and $\tilde{\bigwedge}(p_1, p_2, \phi)$ are finite,

M2. $\forall p, p \succeq p_1, p \succeq p_2 \Rightarrow \exists p' \in \tilde{\bigvee}(p_1, p_2, \phi)$, *s.t.* $p \succeq p'$,

M3. $\forall p, p \preceq p_1, p \preceq p_2 \Rightarrow \exists p' \in \tilde{\bigwedge}(p_1, p_2, \phi)$, *s.t.* $p \preceq p'$. □

It is important to realize that the first condition does not imply the next two. For example, one could have an infinite descending chain of elements above both p_1 and p_2 with no minimal element. One should also note that the $\mathcal{MW}$ property implies $\tilde{\bigvee}(p_1, p_2, G)$ and $\tilde{\bigwedge}(p_1, p_2, S)$ are finite for all subsets $S, G \subseteq P$.

Join and meet algorithms of convex spaces.

In this section we show how one can decide join and meet operations on convex spaces by operations on their boundary sets. The first step is to show the desired result for finitely representable convex spaces:

Theorem 11 *Let P be a poset and suppose that C_1, C_2 are finitely representable convex spaces of P. Then*

1. $C_1 \vee^c C_2$ *is finitely representable and*

2. *if* P *has property* $\mathcal{MW}$*, then* $C_1 \wedge^c C_2$ *is finitely representable.*

Proof: Suppose that C_1, C_2 have finite boundary sets S_1, G_1 and S_2, G_2 respectively, define:

$$S_{1\cap 2} = \mathcal{MIN}(\{s \in \tilde{\bigvee}(s_1, s_2, \{g_1, g_2\}) \mid s_1 \in S_1, s_2 \in S_2, g_1 \in G_1, g_2 \in G_2\})$$

$$G_{1\cap 2} = \mathcal{MAX}(\{g \in \tilde{\bigwedge}(g_1, g_2, \{s_1, s_2\}) \mid g_1 \in G_1, g_2 \in G_2, s_1 \in S_1, s_2 \in S_2\})$$

$$S_{1\cup 2} = \mathcal{MIN}(S_1 \cup S_2)$$

$$G_{1\cup 2} = \mathcal{MAX}(G_1 \cup G_2)$$

For any $s_1 \in S_1, s_2 \in S_2, g_1 \in G_1, g_2 \in G_2$, $\tilde{\bigvee}(s_1, s_2, \{g_1, g_2\})$ is finite by the $\mathcal{MW}$ property of P. Since S_1, S_2, G_1, G_2 are finite, there are only finitely many distinct s_1, s_2, g_1, g_2, hence, $S_{1\cap 2}$ is finite. Similarly, $G_{1\cap 2}$ is finite. Moreover. $S_{1\cup 2}$ and $G_{1\cup 2}$ are finite, because $S_1 \cup S_2$ and $G_1 \cup G_2$ are finite. Hence, $\mathcal{B}(S_{1\cap 2}, G_{1\cap 2}), \mathcal{B}(S_{1\cup 2}, G_{1\cup 2})$ are finitely representable.

$C_1 \wedge C_2$: To show $C_1 \wedge C_2 = \mathcal{B}(S_{1\cap 2}, G_{1\cap 2})$:

($\subseteq$) If $p \in C_1 \wedge C_2$, then there exist $s_1 \in S_1$, $s_2 \in S_2$, $g_1 \in G_1$, $g_2 \in G_2$, such that $s_1 \preceq p \preceq g_1$, $s_2 \preceq p \preceq g_2$. Since P has the $\mathcal{MW}$ property, there exists $s \in \tilde{\bigvee}(s_1, s_2, \{g_1, g_2\})$ such that $s \preceq p$. Hence, there exists $s' \in S_{1\cap 2}$ such that $s' \preceq s \preceq p$, by the definition of $S_{1\cap 2}$. Similarly, there exists $g' \in G_{1\cap 2}$ such that $p \preceq g'$, which implies that $p \in \mathcal{B}(S_{1\cap 2}, G_{1\cap 2})$

($\supseteq$) If $p \in S_{1\cap 2}$, then there exist $s_1 \in S_1$, $s_2 \in S_2$, $g_1 \in G_1$, $g_2 \in G_2$, such that $p \in \tilde{\bigvee}(s_1, s_2, \{g_1, g_2\})$, which implies that $s_1 \preceq p \preceq g_1$ and $s_2 \preceq p \preceq g_2$. Hence, $p \in C_1 \wedge C_2$. So, $S_{1\cap 2} \subseteq C_1 \wedge C_2$. Similarly, $G_{1\cap 2} \subseteq C_1 \wedge C_2$. So $\mathcal{B}(S_{1\cap 2}, G_{1\cap 2}) \subseteq C_1 \wedge C_2$.

$C_1 \vee C_2$: To show $C_1 \vee C_2 = \mathcal{B}(S_{1\cup 2}, G_{1\cup 2})$:

($\subseteq$) If $p \in C_1 \vee C_2$, then there exist $s, g \in C_1 \cup C_2$ such that $s \preceq p \preceq g$. By definition of $S_{1\cup 2}$ and $G_{1\cup 2}$, there exist $s' \in S_{1\cup 2}, g' \in G_{1\cup 2}$ such that $s' \preceq s \preceq p \preceq g \preceq g'$. Hence, $p \in \mathcal{B}(S_{1\cup 2}, G_{1\cup 2})$.

($\supseteq$) If $p \in S_{1\cup 2}$, then $p \in S_1 \cup S_2$ which implies $p \in C_1 \vee C_2$. Hence, $S_{1\cup 2} \subseteq C_1 \vee C_2$. Similarly, $G_{1\cup 2} \subseteq C_1 \vee C_2$. Therefore, $\mathcal{B}(S_{1\cup 2}, G_{1\cup 2}) \subseteq C_1 \vee C_2$. □

Corollary 12 *If* P *has the* $\mathcal{MW}$ *property, then the finitely representable convex spaces form a sublattice in* $(\mathcal{CL}(c), \wedge^c, \vee^c)$. □

Since the poset of finitely representable convex spaces is isomorphic to the poset of fringes, it is now possible to express the join and meet operations on such spaces entirely in terms of fringes. The following result can be obtained directly from the proofs of Theorem 10 and Theorem 11.

Theorem 13 *If* P *has the* $\mathcal{MW}$ *property, then the space* $G(P)$ *of fringes is a lattice where, for every pair of fringes* (S_1, G_1) *and* (S_2, G_2)*, we have*

$$(S_1, G_1) \vee (S_2, G_2) = (S_{1\cup 2}, G_{1\cup 2})$$
$$(S_1, G_1) \wedge (S_2, G_2) = (S_{1\cap 2}, G_{1\cap 2})$$ □

The theorem shows that finitely representable convex spaces can be represented and manipulated indirectly through operations on pairs of finite sets. In the next theorem, we show that the converse of the second part of Theorem 11 is true.

Theorem 14 *If* P *is a finitely representable poset and the intersection of any two finitely representable convex spaces in* P *is finitely representable, then* P *has property* $\mathcal{MW}$.

Proof: Let $S = \mathcal{MIN}(P)$ and $G = \mathcal{MAX}(P)$. Then, for any $p_1, p_2 \in P$,

$$\tilde{\bigvee}(p_1, p_2, \emptyset) = \mathcal{MIN}(\mathcal{B}(\{p_1\}, G) \cap \mathcal{B}(\{p_2\}, G))$$
$$\tilde{\bigwedge}(p_1, p_2, \emptyset) = \mathcal{MAX}(\mathcal{B}(S, \{p_1\}) \cap \mathcal{B}(S, \{p_2\}))$$

It is easy to see that these sets have the desired properties. □

Corollary 15 *A finitely representable poset* P *has the* $\mathcal{MW}$ *property if and only if the intersection of every two finitely representable convex spaces is finitely representable.* □

Hence the $\mathcal{MW}$ property is the basic order-theoretic condition which insures that we are able to manipulate convex spaces in a finitely representable way. Of course, to apply the results that we have developed in this subsection in a real computational setting, it is also essential that we know how to calculate the quasi-join and quasi-meet operations with acceptable efficiency.

4 Version Spaces

The version space algorithm which was introduced by Mitchell [Mit78] can be formulated using the ideas of the

previous section. We have two principal goals. The first of these is to characterize the order-theoretic conditions under which the version space representation is legitimate. Since it is not the case that every concept space supports the version space learning technique, it is desirable to provide some simple conditions which will certify, for a given concept space, that the algorithm is sound. Such conditions have been proposed in several discussions of version spaces including the original work [Mit78] and a more recent textbook account [GN87]. However, the admissibility conditions which have been given are *sufficient* conditions which are too weak to support many of the examples of concept spaces for which the version space algorithm is sound (and, indeed, efficient).

Our second goal is to isolate the essential order-theoretic content of the version space algorithm. We will later employ the same techniques to develop new algorithms in the context of ATMS's.

Concepts consistent with observations.

For our purposes a *concept space* is a set of sets P with the property that $\phi \in P$ and

$$UP = \{x \mid x \in p \text{ for some } p \in P\} \in P.$$

The elements of UP are called the *instances* and the elements of P are called *concepts*. A concept space is partially ordered by set inclusion $\subseteq$. If $p \subseteq q$ then we say that p is *more specific* than q or we say that q is *more general* than p. We define an operation $\mathcal{K} : \mathcal{P}(UP) \times \mathcal{P}(UP) \longrightarrow \mathcal{P}(P)$ on a pair of sets of instances as follows:

$$\mathcal{K}(\Gamma, \Delta) = \{p \in P \mid \Gamma \subseteq p \subseteq \overline{\Delta}\}$$

where $\Gamma, \Delta \subseteq UP$ and $\overline{\Delta}$ is the complement of Δ in UP. Here Γ represents the "positive" instances and Δ the "negative" instances.

Example: (Adapted from [Mit78].) Let $I = (0,1) \times (0,1)$ be the open unit rectangle in the two-dimensional real plane. A *real interval* is defined to be a set of real numbers having one of the following forms:

$$[l,u] = \{x \mid l \leq x \leq u\}$$
$$(l,u) = \{x \mid l < x < u\}$$
$$[l,u) = \{x \mid l \leq x < u\}$$
$$(l,u] = \{x \mid l < x \leq u\}$$

Note that any interval with $l > u$, is equal to the empty set. We define the *rectangular* concept space R as the set of subsets $p \subseteq I$ such that p is the product of a pair of intervals.

Although many of the version spaces of R are uncountably infinite, it can be shown that each such version space is a finitely representable convex subset of R. □

Definition: A subset C of P is called a *version space* if there exist $\Gamma, \Delta \subseteq UP$, such that $C = \mathcal{K}(\Gamma, \Delta)$. The set of version spaces over a concept space P is denoted by $\mathcal{VS}_P$ (where the subscript is omitted when P is obvious from the context). □

The two subsets P and ϕ are version spaces, which arise respectively when $\Gamma = \Delta = \phi$ and $\Gamma \cap \Delta \neq \phi$. Now, given a $C \in \mathcal{P}(P)$, we define an operation $d : \mathcal{P}(P) \longrightarrow \mathcal{P}(P)$ as follows:

$$d(C) = \{p \in P \mid \bigcap C \subseteq p \subseteq \bigcup C\}$$

It is easy to check that d is a closure operation. Indeed, we have the following:

Theorem 16 *For any concept space P, a subset $C \subseteq P$ is a version space if, and only if, it is a fixed point of d.*

Proof: ($\Rightarrow$) By C1, $d(C) \supseteq C$. To show that the inclusion is indeed an equality, we observe that for all $p \in d(C)$, we have $\bigcap C \subseteq p \subseteq \bigcup C$. However, since C is a version space, there exist $\Gamma, \Delta \subseteq UP$, such that for all $p' \in C$, we have $\Gamma \subseteq p' \subseteq \overline{\Delta}$. Hence, $\Gamma \subseteq \bigcap C$ and $\bigcup C \subseteq \overline{\Delta}$. Therefore, $\Gamma \subseteq p \subseteq \overline{\Delta}$, *i.e.* $p \in C$. So $d(C) \subseteq C$.

($\Leftarrow$) If $d(C) = C$, we have $C = \{p \in P \mid \bigcap C \subseteq p \subseteq \bigcup C\}$. So by assigning $\Gamma = \bigcap C, \Delta = \overline{\bigcup C}$, we have $C = \mathcal{K}(\Gamma, \Delta)$, *i.e.* C is a version space. □

The next lemma shows how to calculate meets and joins in the lattice defined by the closure operation d.

Lemma 17 *Let $S = \{C_i \mid \forall i \in I\}$, where $C_i = \mathcal{K}(\Gamma_i, \Delta_i)$, be an indexed family of elements in $\mathcal{VS}$, then:*

1. $\bigwedge^d S = \mathcal{K}(\bigcup\{\Gamma_i \mid i \in I\}, \bigcup\{\Delta_i \mid i \in I\})$,
2. $\bigwedge^d S = \mathcal{K}(\bigcup\{\cap C_i \mid i \in I\}, \overline{\bigcap\{\cup C_i \mid i \in I\}})$,
3. $\bigvee^d S = \mathcal{K}(\bigcap\{\cap C_i \mid i \in I\}, \overline{\bigcup\{\cup C_i \mid i \in I\}})$.

Proof: 1. By definition, $\bigwedge^d S = \bigcap S$. Hence,

$$\begin{aligned} & p \in \bigwedge^d S \\ \iff & p \in \bigcap S \\ \iff & \forall i \in I, \Gamma_i \subseteq p \subseteq \overline{\Delta_i} \\ \iff & \bigcup\{\Gamma_i \mid i \in I\} \subseteq p \subseteq \overline{\bigcup\{\Delta_i \mid i \in I\}} \\ \iff & p \in \mathcal{K}(\bigcup\{\Gamma_i \mid i \in I\}, \bigcup\{\Delta_i \mid i \in I\}). \end{aligned}$$

2. By the argument given in Theorem 16, we have $\forall i \in I, C_i = \mathcal{K}(\cap C_i, \overline{\cup C_i})$. Hence, by 1 above, we get $\bigwedge^d S = \mathcal{K}(\bigcup\{\cap C_i \mid i \in I\}, \overline{\bigcup\{\cup C_i \mid i \in I\}}) = \mathcal{K}(\bigcup\{\cap C_i \mid i \in I\}, \overline{\bigcap\{\cup C_i \mid i \in I\}})$.

3. By definition, $\bigvee^d S = d(\bigcup S)$. Hence,

$$\begin{aligned} & p \in \bigvee^d S \\ \iff & \bigcap(\cup S) \subseteq p \subseteq \bigcup(\cup S) \\ \iff & \bigcap\{\cap C_i \mid i \in I\} \subseteq p \subseteq \bigcup\{\cup C_i \mid i \in I\} \\ \iff & p \in \mathcal{K}(\bigcap\{\cap C_i \mid i \in I\}, \overline{\bigcup\{\cup C_i \mid i \in I\}}). \ \square \end{aligned}$$

The next lemma shows that the version spaces are fixed points of the closure operation c (*i.e.* they are convex spaces) and relates the meets and joins in the two lattices defined by the fixed points of the closure operations c and d.

Lemma 18 *1. Every $C \in \mathcal{VS}$ is a fixed point of the operation c,*

2. *If S is a subset of $\mathcal{VS}$, then $\bigwedge^d S = \bigwedge^c S$ and $\bigvee^d S \supseteq \bigvee^c S$.*

Proof: 1. $\forall p \in c(C)$, there exist $p_1, p_2 \in C$, such that $p_1 \subseteq p \subseteq p_2$. However, $\bigcap C \subseteq p_1$, and $p_2 \subseteq \bigcup C$, implies $\bigcap C \subseteq p \subseteq \bigcup C$, *i.e.* $p \in d(C) = C$. Therefore, by C1, we have $c(C) = C$.

2. By definition, $\bigwedge^d S = \bigcap S = \bigwedge^c S$. By C1, we have $\bigvee^d S \supseteq \bigcup S$. So by C2, we get $c(\bigvee^d S) \supseteq \bigvee^c S$. Hence by 1 above, we have $\bigvee^d S \supseteq \bigvee^c S$. $\square$

In general, there are convex spaces which are not version spaces. For example, in the rectangular concept space R described earlier, the convex space determined by the zero area "rectangles"

$[1/4, 1/4] \times [1/4, 1/4]$
$[3/4, 3/4] \times [3/4, 3/4]$

together with the set I is not a version space since the image of this convex space under the map d is $\mathcal{K}(\phi, I)$. On the other hand, since version spaces are convex spaces and their meet operations are identical, several results that are true for convex spaces are also true for version spaces. For instance, it is immediate that the $\mathcal{MW}$ property implies that the meet of every two finitely representable version spaces is finitely representable. Furthermore, observe that given any $s, g \in P$, the convex space $\mathcal{B}(\{s\}, \{g\})$ is also a version space, because it is equal to

$\mathcal{K}(\{s\}, \overline{\{g\}})$.

Therefore, the proof of Theorem 14 still works if we substitute version spaces for convex spaces. Hence, we have the following:

Theorem 19 *A concept space P has the $\mathcal{MW}$ property if and only if the meet of every two finitely representable version spaces is finitely representable.* $\square$

In a concept learning system using the version space representation, the new version space after the addition of some new observations is the same as the intersection (merging) of the current version space with the version space representing the new observations. Thus, the $\mathcal{MW}$ property is a necessary and sufficient condition for ensuring the preservation of finite representability in version space merging.

Admissibility.

As mentioned earlier, we seek a condition on concept spaces which will certify that version spaces can be represented with their boundary sets. We say that concept spaces having this property are *admissible*. More precisely:

Definition: A concept space P is said to be *admissible*, if $\mathcal{K}(\Gamma, \Delta)$ is finitely representable whenever $\Gamma \cup \Delta \subseteq UP$ finite. $\square$

We now demonstrate several conditions which imply (or are equivalent to) the admissibility of a concept space.

Definition: A concept space P is said to have *property $\mathcal{G}$* if, for all $x \in UP$, $\mathcal{K}(\{x\}, \phi)$ and $\mathcal{K}(\phi, \{x\})$ are finitely representable. $\square$

The following lemma allows us to check the admissibility of a pattern language by verifying that if observations are all positive or negative, then the version space is finitely representable.

Lemma 20 *A poset P is admissible if and only if for all non-empty finite $\Gamma, \Delta \subseteq UP$, both $\mathcal{K}(\Gamma, \phi)$ and $\mathcal{K}(\phi, \Delta)$ are finitely representable.*

Proof: ($\Rightarrow$) Immediate.

($\Leftarrow$) Given $\Gamma, \Delta \subseteq UP$, with $\Gamma \cup \Delta$ finite and nonempty. If either Γ or Δ is an empty set, then $\mathcal{K}(\Gamma, \Delta)$ is finitely representable by our assumption. Otherwise, let $C = \mathcal{K}(\Gamma, \Delta), A = \mathcal{K}(\Gamma, \phi), B = \mathcal{K}(\phi, \Delta)$. Note that $C = A \wedge B$. We want to show that $C = \mathcal{B}(\mathcal{MIN}(A), \mathcal{MAX}(B))$:

($\subseteq$) If $p \in C$, then there exist $s_1, g_1 \in A$ and $s_2, g_2 \in B$ such that $s_1 \subseteq p \subseteq g_1$ and $s_2 \subseteq p \subseteq g_2$. Hence, $s_1 \subseteq p \subseteq g_2$, *i.e.* $p \in \mathcal{B}(\mathcal{MIN}(A), \mathcal{MAX}(B))$.

($\supseteq$) If $p \in \mathcal{B}(\mathcal{MIN}(A), \mathcal{MAX}(B))$, then there exist $s \in \mathcal{MIN}(A)$ and $g \in \mathcal{MAX}(B)$ such that $s \subseteq p \subseteq g$. Since $s \subseteq g$, we have $\Gamma \subseteq g$. Hence, by definition of A, we have $g \in A$, which implies that $p \in A$. Similarly, $p \in B$. Hence $p \in C$.

Since both $\mathcal{MIN}(A)$ and $\mathcal{MAX}(B)$ are finite by Lemma 9, we conclude that C is finitely representable. □

The next theorem allows one to check for admissibility by checking finite representability in some special cases.

Theorem 21 *If a concept space P has the $\mathcal{MW}$ and $\mathcal{G}$ properties, then P is admissible.*

Proof: Observe that for all finite $\Gamma, \Delta \subseteq UP$, the version space $\mathcal{K}(\Gamma, \Delta)$ can be constructed by the finite $\wedge$ of version spaces, each of a single observation. The result is then immediate from Theorem 11. □

To appreciate the point of having a condition for verifying admissibility, we consider again the earlier rectangular concept space R, where each concept p is either the product of a pair of intervals (rectangles) or p is the empty set ϕ. The set of observations is a subset of P of the form: $[x, x] \times [y, y]$ (points). Given a set of positive and negative observations, the learning task is to find the set of concepts in P that are consistent with the observations.

For each positive observation $[x, x] \times [y, y]$, its version space is the convex closure of $\{I\}$ and $\{[x, x] \times [y, y]\}$. For each negative observation $[x, x] \times [y, y]$, its version space is the convex closure of $\{(x, 1) \times (0, 1), (0, x) \times (0, 1), (0, 1) \times (y, 1), (0, 1) \times (0, y)\}$ and $\{\phi\}$. Hence, P has property $\mathcal{G}$.

Given a pair of rectangles $p_1 = [lx, ux] \times [ly, uy]$, $p_2 = [lx', ux'] \times [ly', uy']$, we have:

$$\begin{aligned} \tilde{\bigvee}(p_1, p_2, \phi) &= \{[\min(lx, lx'), \max(ux, ux')] \times \\ &\qquad [\min(ly, ly'), \max(uy, uy')]\}, \\ \tilde{\bigwedge}(p_1, p_2, \phi) &= \{[\max(lx, lx'), \min(ux, ux')] \times \\ &\qquad [\max(ly, ly'), \min(uy, uy')]\}. \end{aligned}$$

Besides being finite, they also satisfy the second and third conditions of the $\mathcal{MW}$ property. Similar results can be derived for different combinations in the types of rectangles. Hence, P has $\mathcal{MW}$ property and it is therefore admissible. Note that the version spaces of P typically include infinite (and even uncountable) chains, so an admissibility condition which precludes such properties in the concept space will fail to cover this example.

Note that to apply Theorem 21, it is necessary for P to have the property $\mathcal{G}$, so that the finite representability property can be propagated to version spaces having more than one observation. To see how essential this is, consider the following variation on the rectangular concept space R. Let us expand our collection of concepts to include any subset of the unit rectangle I which is convex in the usual geometric sense. Our concept space now does not support the version space algorithm because it fails to satisfy property $\mathcal{G}$. For example, if $(1/2, 1/2)$ is observed to be a negative instance, then there is an uncountable collection of most general concepts consistent with this observation. Hence the version space is not finitely representable.

Another admissibility criterion is related to the concept of interval refinement. In the following, we introduce the notion of parafiniteness for a pattern language which can be used to determine the admissibility of the language.

Definition: A given P is said to be *parafinite* if for all $p, q \in P$ with $p \supseteq q$, and for all x with $x \in p$ but $x \notin q$, we have:

1. there exists $\{p_i\}$ where $1 \leq i \leq m$ for some finite m such that $p \supseteq p_i \supseteq q$, $x \notin p_i$ and for all p' with $p \supseteq p' \supseteq q, x \notin p'$, there exists i such that $p_i \supseteq p'$,
2. there exists $\{q_j\}$ where $1 \leq j \leq n$ for some finite n such that $p \supseteq q_j \supseteq q$, $x \in q_j$ and for all q' with $p \supseteq q' \supseteq q, x \in q'$, there exists j such that $q_j \subseteq q'$. □

The interpretation of parafiniteness is that, given any interval in the poset of concepts and any instance of the more general concept, we can find a finite cover that refines the original space to account for the results of this new trial instance. A graphical illustration of the definition is shown in figure 3, where $\{q_1, q_2, q_3\}$ is the finite cover refining the interval formed by p and q when a trial is found to be a positive and $\{p_1, p_2\}$ is the finite cover refining the interval when the trial is found to be negative. The quadrilaterals represent intervals in the poset of concepts.

Theorem 22 *A given P is parafinite if and only if the meet of every finitely representable convex space with any finitely observable version space is finitely representable.*

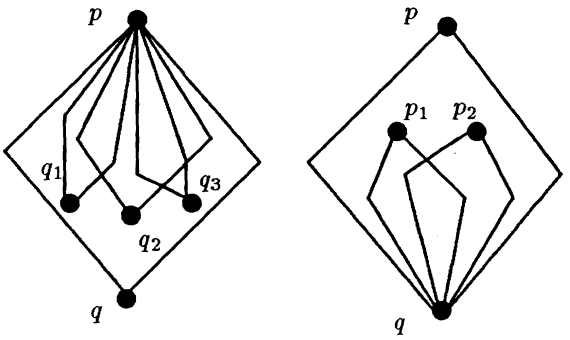

Figure 3: The parafinite property

Proof: ($\Leftarrow$) For all $p, q \in P$, with $p \supseteq q$, and for all x with $x \in p$ but $x \notin q$, we have $C = \mathcal{B}(\{q\}, \{p\})$ is a finitely representable convex space and $A = \mathcal{K}(\phi, \{x\}), B = \mathcal{K}(\{x\}, \phi)$ are finite observation version spaces. Let $C_1 = A \wedge C$ and $C_2 = B \wedge C$. By assumption C_1, C_2 are finitely representable, therefore we can find m, n, G, S where $G = \{p_i \mid 1 \leq i \leq m\}$ and $S = \{q_j \mid 1 \leq j \leq n\}$ such that $C_1 = \mathcal{B}(\{q\}, G)$ and $C_2 = \mathcal{B}(S, \{p\})$. Hence, $p \supseteq p_i \supseteq q$ and $p \supseteq q_j \supseteq q$. Furthermore, for all p' with $p \supseteq p' \supseteq q, x \notin p'$, we have by definition that $p' \in C_1$, which implies that there exists i such that $p_i \supseteq p'$. Similar results hold for C_2 and q_j by duality. Hence P is parafinite.

($\Rightarrow$) Given any finitely representable convex space $A = \mathcal{B}(X, Y)$ and a finite observation version space $B = \mathcal{K}(\Gamma, \Delta)$, with $X, Y \subseteq P$ and $\Gamma, \Delta \subseteq UP$ are finite, we want to show that $C = A \wedge B$ is finitely representable. Since $B = \bigwedge_{\gamma \in \Gamma} \mathcal{K}(\{\gamma\}, \phi) \wedge \bigwedge_{\delta \in \Delta} \mathcal{K}(\phi, \{\delta\})$, by associativity of $\wedge$, it suffices to show that C is finitely representable for those cases where B is a single observation version space. First, we assume that $B = \mathcal{K}(\{\gamma\}, \phi)$. Observe that $q' \in C$ if and only if there exist $q \in X, p \in Y$ such that $q \subseteq q' \subseteq p$ and $\gamma \in q'$. Since P is parafinite, implies that there exist finite m and q_j, where $1 \leq j \leq m$, such that for any q', we have $q \subseteq q' \subseteq p$ and $\gamma \in q'$ if and only if there exists j such that $q_j \subseteq q'$, *i.e.* $q' \in \mathcal{B}(\{p\}, \{q_j \mid 1 \leq j \leq n\}) \subseteq C$. Since X, Y are finite, we have only finite distinct pairs of such p and q. Therefore C is the union of finite finitely representable convex spaces. Furthermore, C is by definition a convex space, therefore C is a join of finitely representable convex spaces by the definition of join. Hence, by Theorem 11, we conclude that C is finitely representable. Similarly, we can show that C is finitely representable for the case where $B = \mathcal{K}(\phi, \{\delta\})$. □

Corollary 23 *If a concept language is parafinite then it is admissible.* □

The above results indicate that it may be desirable to look for the parafiniteness property in a concept description language. In particular, it constitutes part of the sufficient conditions for ensuring the admissibility of a language. Furthermore, it is conceivable that in some applications, domain knowledge may be used to infer an initial (or intermediate) finitely representable convex space which is further refined when additional observations are collected. In these cases, the parafiniteness property provides the necessary and sufficient conditions for the final concept space to be finitely representable.

Extended version spaces.

In many real applications, the training data for constructing the version space of a concept may be erroneous or the concept itself may be a disjunction of several version spaces, therefore it is essential to work with a more general notion of version spaces. There are at least two approaches to this problem. One approach, suggested by Hirsh [Hir90], is to generalize from the notion of a version space as a collection of concepts consistent with positive and negative training data to a notion of "abstract" version space having the needed mathematical properties and potentially arising from other sources of information such as domain knowledge. In particular, the finitely representable convex spaces are an ideal candidate for such a abstract theory. The structures Hirsh considers are slightly more general than this, but the basic results about version space merging are similar to those we have discussed above (for example we adopted the notation in our proof of Theorem 11 from Theorem 8.6 of [Hir90]).

In this section we retreat to an analysis of a class of finitely representable convex spaces which was used in the implementation of the Meta-DENDRAL project [BM78].

Definition: Given a finite set of observations (Γ, Δ), the *extended version spaces* of (Γ, Δ) are:

$$V_{s,g} = \bigcup \{\mathcal{K}(\gamma, \delta) \mid \gamma \subseteq \Gamma, \delta \subseteq \Delta, |\gamma| = s, |\delta| = g\}$$

where $0 \leq s \leq |\Gamma|, 0 \leq g \leq |\Delta|$. □

Lemma 24 *$V_{s,g}$ is a convex space.*

Proof: By definition of c, we have $c(V_{s,g}) \supseteq V_{s,g}$. To show equality, we observe that for all $p \in c(V_{s,g})$, there exist $p_1 \in \mathcal{K}(\gamma_1, \delta_1)$ and $p_2 \in \mathcal{K}(\gamma_2, \delta_2)$ such that $p_1 \subseteq$

$p \subseteq p_2$, where $\gamma_1, \gamma_2 \subseteq \Gamma$ and $\delta_1, \delta_2 \subseteq \Delta$, with $|\gamma_1| = |\gamma_2| = s, |\delta_1| = |\delta_2| = g$. Since $\gamma_1 \subseteq p_1$ and $p_2 \subseteq \delta_2$, we have $p \in \mathcal{K}(\gamma_1, \delta_2)$. Hence, $p \in V_{s,g}$ which implies $c(V_{s,g}) = V_{s,g}$. □

The last lemma establishes the basis for the formal treatment of extended version spaces using the theory of convex spaces. In fact, the convexity is the main reason for why extended version spaces can be represented by boundary sets and computable using similar operations that we have established for the version spaces. Note that several results are immediate from the lemma:

Corollary 25 *Any extended version space is a finite join of version spaces:*

$$V_{s,g} = \bigvee \{\mathcal{K}(\gamma, \delta) \mid \gamma \subseteq \Gamma, \delta \subseteq \Delta, |\gamma| = s, |\delta| = g\}$$

where $0 \leq s \leq |\Gamma|, 0 \leq g \leq |\Delta|$. □

Corollary 26 *If a concept space P has the $\mathcal{MW}$ and $\mathcal{G}$ properties, and $V_{s,g}$ is an extended version space with either $s > 0$ or $g > 0$, then $V_{s,g}$ is finitely representable.* □

With the facts that we have established in Lemma 4, we can show that the union operations which occur in the manipulation of extended version spaces can usually be replaced by the join operations of convex spaces. Therefore, the algorithms for computing the meet and join of convex spaces can be applied directly to the extended version spaces without any modification. Using only the lattice operations, the following theorem allows us to generate the extended version spaces incrementally and without referencing any of the previous observations.

Theorem 27 *If the extended version space induced by a set Γ of positive instances and a set Δ of negative instances is $V_{s,g}$, then:*

1. *given a new positive observation x, the extended version space of $(\Gamma \cup \{x\}, \Delta)$ is:*

$$V'_{s,g} = (V_{s-1,g} \wedge \mathcal{K}(\{x\}, \phi)) \vee V_{s,g},$$

2. *given a new negative observation x, the extended version space of $(\Gamma, \Delta \cup \{x\})$ is:*

$$V'_{s,g} = (V_{s,g-1} \wedge \mathcal{K}(\phi, \{x\})) \vee V_{s,g}.$$

where we assume that:

$$V_{s,g} = \begin{cases} \phi & \text{if } s < 0 \text{ or } s > |\gamma| \text{ (for 1.)} \\ \phi & \text{if } g < 0 \text{ or } g > |\delta| \text{ (for 2.)} \end{cases}$$

Proof: For boundary cases, we have the following desired results:

1. $V'_{|\Gamma|+1,g} = V_{|\Gamma|,g} \wedge \mathcal{K}(\{x\}, \phi)$ and $V'_{0,g} = V_{0,g}$.
2. $V'_{s,|\Delta|+1} = V_{s,|\Delta|} \wedge \mathcal{K}(\phi, \{x\})$ and $V'_{s,0} = V_{s,0}$.

For non-boundary cases, we have:

1. $V'_{s,g}$

$= \bigcup\{\mathcal{K}(\gamma, \delta) \mid \gamma \subseteq \Gamma \cup \{x\}, \delta \subseteq \Delta, |\gamma| = s, |\delta| = g\}$

$= \bigcup\{\mathcal{K}(\gamma, \delta) \mid \gamma \subseteq \Gamma, \delta \subseteq \Delta, |\gamma| = s, |\delta| = g\} \cup \bigcup\{\mathcal{K}(\gamma \cup \{x\}, \delta) \mid \gamma \subseteq \Gamma, \delta \subseteq \Delta, |\gamma| = s - 1, |\delta| = g\}$

$= V_{s,g} \cup \bigcup\{\mathcal{K}(\gamma, \delta) \wedge \mathcal{K}(\{x\}, \phi) \mid \gamma \subseteq \Gamma, \delta \subseteq \Delta, |\gamma| = s - 1, |\delta| = g\}$ – by Lemma 17

$= V_{s,g} \cup (V_{s-1,g} \wedge \mathcal{K}(\{x\}, \phi))$ – by Lemma 4

$= (V_{s-1,g} \wedge \mathcal{K}(\{x\}, \phi)) \vee V_{s,g}$ – because $V'_{s,g}$ is a convex space and by noting that, for a closure operation f, we have $f(A \cup B) = f(f(A) \cup f(B))$.

2. Similar proof as 1. □

The following shows that it is possible to compute the entire extended version space $V_{s,g}$ from $V_{s,0}$ and $V_{0,g}$ using the lattice operations:

Lemma 28 *For all $0 \leq s' \leq s \leq |\Gamma|$ and $0 \leq g' \leq g \leq |\Delta|$,*

1. $V_{s,g} \subseteq V_{s',g'}$,
2. $V_{s,g} = V_{s,g'} \wedge V_{s',g}$,
3. $V_{s,g} = V_{s,0} \wedge V_{0,g}$.

Proof: 1. If $p \in V_{s,g}$, then there exist $\gamma \subseteq \Gamma, \delta \subseteq \Delta$, with $|\gamma| = s, |\delta| = g$, such that $\gamma \subseteq p \subseteq \overline{\delta}$. However, there exist $\gamma' \subseteq \gamma, \delta' \subseteq \delta$, with $|\gamma'| = s', |\delta'| = g'$. So $\gamma' \subseteq p \subseteq \overline{\delta'}$, *i.e.* $p \in V_{s',g'}$.

2. ($\subseteq$) From 1., we have $V_{s,g} \subseteq V_{s,g'}, V_{s,g} \subseteq V_{s',g}$. This implies that $V_{s,g} \subseteq V_{s,g'} \wedge V_{s',g}$ because $V_{s,g}$ is a convex space.

   ($\supseteq$) If $p \in V_{s,g'} \wedge V_{s',g}$, then there exist $\gamma_1, \gamma_2 \subseteq \Gamma$ and $\delta_1, \delta_2 \subseteq \Delta$, with $|\gamma_1| = s, |\gamma_2| = s', |\delta_1| = g', |\delta_2| = g$, such that $\gamma_1 \subseteq p \subseteq \overline{\delta_1}$ and $\gamma_2 \subseteq p \subseteq \overline{\delta_2}$. Hence, we have $\gamma_1 \subseteq p \subseteq \overline{\delta_2}$, *i.e.* $p \in V_{s,g}$.

3. By assigning $s' = g' = 0$ in 2. above.

The following theorem allows us to independently calculate several sets of extend version spaces and later combine them without referencing any of the previous observations.

Theorem 29 *If a finite set of observation* (Γ, Δ) *is partitioned into* (Γ_1, Δ_1) *and* (Γ_2, Δ_2), *then:*

1. $V_{s,0} = \bigvee_{0 \le k \le s} (V^1_{s-k,0} \wedge V^2_{k,0})$,
2. $V_{0,g} = \bigvee_{0 \le k \le g} (V^1_{0,g-k} \wedge V^2_{0,k})$,

where the various extended version spaces are distinguished by superscripts.

Proof: 1. $V_{s,0}$

$= \bigcup\{\mathcal{K}(\gamma, \phi) \mid \gamma \subseteq \Gamma, |\gamma| = s\}$

$= \bigcup\{\mathcal{K}(\gamma_1, \phi) \wedge \mathcal{K}(\gamma_2, \phi) \mid \gamma_1 \subseteq \Gamma_1, \gamma_2 \in \Gamma_2, |\gamma_1| + |\gamma_2| = s\}$

$= \bigcup_{0 \le k \le s} \left(\bigcup_{|\gamma_1| = s-k} \left(\bigcup_{|\gamma_2| = k} \{\mathcal{K}(\gamma_1, \phi) \wedge \mathcal{K}(\gamma_2, \phi)\} \right) \right)$

$= \bigcup_{0 \le k \le s} \left(\bigcup_{|\gamma_1| = s-k} \{\mathcal{K}(\gamma_1, \phi) \wedge V^2_{k,0}\} \right)$ – by Lemma 4

$= \bigcup_{0 \le k \le s} \{V^1_{s-k,0} \wedge V^2_{k,0}\})$ – by Lemma 4

$= \bigvee_{0 \le k \le s} (V^1_{s-k,0} \wedge V^2_{k,0})$ – by Lemma 4

2. Similar to the proof of 1. □

5 Assumption-based TMS's

In this section, we examine Johan de Kleer's formulation of truth maintenance systems (TMS's) known as *assumption-based* TMS's [dK86a, dK86b]. An ATMS works in conjunction with a problem solver recording all conclusions drawn by the solver together with the assumptions they depend on. The job of an ATMS is to efficiently recalculate the status of beliefs in a solver when the premises that underlie them are changed. Each conclusion derived by the problem solver is represented as a node in the ATMS. The derivation of a node is also obtained from the solver and recorded in the ATMS. Central to the ATMS is the notion of an assumption: each node has associated with it all minimal sets of assumptions that would make the node true. The ATMS recalculates these sets as new formulas are acquired from the problem solver.

The assumption sets form a finite lattices under set inclusion. We show that the sets of assumptions that make a node true, form a convex space. Hence such sets of assumptions can be represented by boundary sets. The existence of the common inconsistent sets of assumptions makes it possible to simplify this representation so that only the upper boundary (greatest) elements are required. These constitute the *label* of a node.

Our goal in this section is to formalize the working of an ATMS and show how the calculations of label sets can be formulated as computations on the boundaries of convex spaces. As with our analysis of version spaces, the convex space reformulation of the ATMS leads to the discovery of several new results. In particular, it allows us to describe the semantics of the label computations in both the basic and extended ATMS in a uniform framework. Furthermore, it leads to the development of new general algorithms for label computations.

To eliminate many implementation related details of an ATMS, we propose the following specification of an ATMS which focuses on the label calculation.

Definition: An ATMS is characterized by the following inputs and outputs.

Input: a finite set of propositional formulas $\mathcal{F}$, and a finite set of propositional literals $\mathcal{A}$, called *assumptions*. Subsets of $\mathcal{A}$ are called *environments*, the power set of $\mathcal{A}$ is called the *environment lattice* P. Propositional literals that appear in $\mathcal{F}$ and $\mathcal{A}$ are the *nodes* of the ATMS.

Output: for each X, where X can be a propositional atom in $\mathcal{F}$, an assumption in $\mathcal{A}$, or $\perp$ (a special proposition that represents falsity), the ATMS computes V_X which is a set of subsets of $\mathcal{A}$ where

Semantics: using the short hand $\mathcal{F} \cup p$ for the set $\mathcal{F} \cup \{\Rightarrow x \mid x \in p\}$, the output of the ATMS, V_X, must satisfy the following (where $X \neq \perp$):

$V_\perp = \{p \in P \mid \mathcal{F} \cup p$ is inconsistent$\}$,
$V_X = \{p \in P \mid \mathcal{F} \cup p$ is consistent, and $\mathcal{F} \cup p \models X\}$.

Algorithm: a procedure that takes the input of the ATMS and manipulates the values of V_X's so as to satisfy the semantical requirement above. □

In the ATMS literature, the boundary set representation of V_X is called the label of X, and is the data structure that is manipulated. Our approach in this paper, however, is to describe the principles behind the algorithms of ATMS by first discussing the operations on the set V_X. This approach is more intuitive and it offers additional insights into the functionality of the ATMS. In terms of algorithms, as long as we work with V_X's that are convex spaces, and the operations on the V_X's are restricted to meets and joins, we can apply the isomorphism theorem 13 to obtain equivalent algorithms that operate on boundary representations. Thus the boundary representation that is normally used in the ATMS can be considered as an optimization technique as it was in the case of the Version Space.

In this paper, we will discuss two specializations of an ATMS, namely the basic ATMS and the extended ATMS described in [dK86a, dK86b].

Basic ATMS.

A basic ATMS can be specified as the following specialization of an ATMS:

Definition: A *basic* ATMS is an ATMS where the input formulas $\mathcal{F}$ are propositional Horn clauses $X_1, \ldots, X_n \Rightarrow Y$ also called *justifications*, and the assumption set $\mathcal{A}$ contains positive literals only. □

The consequent of a justification with no antecedents is called a *premise*. A premise can be an assumption. Every justification must have a consequent. If a Horn clause justification $\neg X_1 \wedge \cdots \wedge \neg X_n$ has no positive literals, it is rewritten as $X_1, \ldots, X_n \Rightarrow \perp$, with the special consequent $\perp$.

In our convex space reformulation, we let UP be equal to $\mathcal{A}$ and P be the environment lattice. However, the partial order $\preceq$ on P is defined to be the reverse of set inclusion, i.e., set containment, *i.e.* $\preceq \equiv \supseteq$. This partial order captures the notion of the generality of an environment. Since the set of assumptions is finite, the environment lattice is also finite and therefore convex subsets of P can be represented by their finite boundaries.

Definition: Given an ATMS with input formulas $\mathcal{F}$, the set of values for the V_X's (where X is either a propositional atom in $\mathcal{F}$, or an assumption, or the proposition $\perp$) are called *states* of the ATMS. An *initial* ATMS state is a state such that for each propositional atom $X \in \mathcal{F}$,

1. V_X is the downward closure of $\{\{X\}\}$, if X is an assumption but not a premise.
2. V_X is P, if X is a premise.
3. V_X is the empty set for every other X. □

Observe that every V_X is downward closed in an initial ATMS. In fact, for the following discussion, we will only be concerned with those states of ATMS where each V_X is either downward closed or convex. As described in section 2, the downward closed sets form a lattice and in particular, a sublattice of the convex spaces. Therefore, the following are well defined:

Definition: For any justification $\psi (\equiv X_1, \ldots, X_n \Rightarrow Y) \in \mathcal{F}$, we say that ψ is *applicable* to the ATMS if in the current state of the ATMS,

$$V_{X_1} \wedge \cdots \wedge V_{X_n} \not\subseteq V_Y.$$

The *application* of ψ to the current state of the ATMS results in the modification of V_Y as follows:

$$V_Y = V_Y \vee (V_{X_1} \wedge \cdots \wedge V_{X_n}). \square$$

Note that for downward closed sets, the meet and join operations are equivalent to the set intersection and union operations respectively. The above definitions can be extended to cover a sequence of justifications as follows:

Definition: Let $(\psi_1, \ldots, \psi_s)$ be a sequence of justifications. The sequence is said to be *applicable* to the ATMS if ψ_1 is applicable and each $\psi_i, 1 < i \leq s$ is applicable after the sequential application of $\psi_1, \ldots, \psi_{i-1}$ to the ATMS. The *application* of the sequence to the ATMS is defined to be the sequential application of $\psi_1, \ldots, \psi_s$. An applicable sequence of justifications is said to be *complete* with respect to $\mathcal{F}$ if, after the application of the sequence, no justification in $\mathcal{F}$ is applicable. □

With the above definitions, we can now proceed to discuss the basic ATMS algorithm. The algorithm can be viewed as a process of modifying the values of V_X's from an initial ATMS:

Definition: Given a basic ATMS with input $\mathcal{F}$, the *basic ATMS algorithm* is defined to be the selection and application of a complete applicable sequence of justifications with respect to $\mathcal{F}$ on the corresponding initial ATMS, followed by the removal of every environment in $V_\perp$ from every consistent V_X. □

The following are some simple observations that one can derive from the above definitions:

1. To implement the basic ATMS algorithm, we only need the procedure that compares sets and the procedures that compute the meet and join of convex spaces.
2. After the application of a sequence, V_X will remain downward closed. Hence, V_X will be convex after the removal of the environments that are in $V_\perp$.
3. The application of a justification strictly increases the size of a single V_X in the ATMS. Other V_X's remain unchanged.
4. Since the size of the environment lattice and the number of propositional atoms are both finite, the previous observation allows us to conclude that there is no infinite applicable sequence of justifications; so the basic ATMS algorithm terminates.
5. By repeating the search for an applicable justification in $\mathcal{F}$, we can always derive a finite complete applicable sequence of justifications. In particular, given any applicable sequence of an ATMS, we can extend the sequence to a finite and complete one. This is a consequence of Theorem 5 in Section 2, because the modification process is a closure operator.

In analyzing the basic ATMS algorithm, it is convenient to first ignore $\perp$ and consider only those Horn sentences in $\mathcal{F}$ that have exactly one positive literal. An important fact about such a propositional system is that it has a minimal model. We will denote the minimal model of $\mathcal{F}$ by $M_\mathcal{F}$. Another fact is that if we define $T_\mathcal{F}(M) = \{y \mid (x_1, \ldots, x_n \Rightarrow y) \in \mathcal{F} \text{ and } \{x_1, \ldots, x_n\} \subseteq M\} \bigcup M$, then $M_\mathcal{F} = \bigcup_{i \geq 0} (T_\mathcal{F})^i(\phi) = (T_\mathcal{F})^s(\phi)$ for some non-negative integer s [FMH90]. The following relates the initial ATMS to the operation $T_\mathcal{F}$:

Lemma 30 *1. The initial ATMS corresponds directly to $T_{\mathcal{F} \cup p}(\phi)$, i.e. $p \in V_X$ in the initial ATMS if, and only if, $X \in T_{\mathcal{F} \cup p}(\phi)$.*

2. Given $D = (T_{\mathcal{F} \cup p})^i(\phi)$ where $i > 0$, the only justifications that contribute additional items to $(T_{\mathcal{F} \cup p})(D)$ are those that have non-empty antecedents, i.e. the premise justifications and those justifications in p.

Proof: 1. Observe that $X \in T_{\mathcal{F} \cup p}(\phi)$ if, and only if, X is a premise or an assumption in p, *i.e.* $X \in p$. Similarly, $p \in V_X$ in the initial ATMS if, and only if, X is a premise or X is an assumption and $X \in p$.

2. This is obvious since the empty set is a subset of $(T_{\mathcal{F} \cup p})^{i-1}(\phi)$, therefore everything that is derivable from justifications with empty antecedents is already present in D. □

An observation from the above lemma is that if we are given the initial ATMS, then we can obtain $M_{\mathcal{F} \cup p}$ by computing $(T_{\mathcal{F}'})^{s-1}(C_p)$ where $\mathcal{F}'$ is $\mathcal{F}$ with all premise justifications deleted and $C_p = \{x \mid p \in V_x \text{ in the initial ATMS}\}$. Therefore, with the initialization that we made to the ATMS, we can assume that $\mathcal{F}$ contains no premise justifications and every justification has non-empty antecedents.

Consider an ATMS with an input set of justifications $\mathcal{F}$ not containing $\perp$. The basic ATMS algorithm is reduced to the application of any complete applicable sequence of justifications to the initial ATMS. To justify our definition, we need to show that every complete applicable sequence derives the same results. This, however, is a direct consequence of theorem 5 because every application of a justification can be considered as a closure operation on the cross product of all V_X's. More generally, as stated in the following theorem, we can define a semantics of our algorithm which is independent of the choice of the sequence.

Theorem 31 *If $\mathcal{S} \equiv (\psi_1, \ldots, \psi_s)$ is a complete applicable sequence of justifications of an initial ATMS, then at the end of the application of $\mathcal{S}$, we have:*

$$p \in V_X \Leftrightarrow \mathcal{F} \cup p \models X.$$

Proof: Since there exists a minimum model $M_{\mathcal{F} \cup p}$ for $\mathcal{F} \cup p$, it suffices to show that $p \in V_X \Leftrightarrow X \in M_{\mathcal{F} \cup p}$.

($\Rightarrow$) We will prove this by induction on the number of applications of justifications:

Basis From lemma 30, we know that the setup of the initial ATMS is such that if $p \in V_X$, we have $X \in M_{\mathcal{F} \cup p}$.

Step The only way that $p \in P$ can be added to V_y for some node y after the application of a justification ψ is when $\psi \equiv x_1, \ldots, x_n \Rightarrow y$ and $p \in V_{x_i}, 1 \leq i \leq n$. By induction, we have for all i that $x_i \in M_{\mathcal{F} \cup p}$. Since $\psi \in \mathcal{F}$, we have $y \in M_{\mathcal{F} \cup p}$.

($\Leftarrow$) We want to show that if $X \in M_{\mathcal{F} \cup p}$, then $p \in V_X$ at the end of the application of $\mathcal{S}$. As indicated in the observation after lemma 30, it is sufficient

to show that if $x \in (T_{\mathcal{F}})^{s-1}(C_p)$, where $C_p = \{x \mid p \in V_x$ in the initial ATMS$\}$, then $p \in V_x$.

Assume that this is not true, then at the end of the basic ATMS algorithm, there exists $y \in (T_{\mathcal{F}})^{s-1}(C_p)$ such that $p \notin V_y$. Note that lemma 30 says that $x \in (T_{\mathcal{F}})^0(C_p)$ implies that $p \in V_x$. Therefore the minimal i, such that there exists $y \in (T_{\mathcal{F}})^i(C_p)$ and $p \notin V_y$, is non-zero. However, by definition of T, $y \in (T_{\mathcal{F}})^i(C_p)$ implies that there exists $\psi \equiv x_1, \ldots, x_n \Rightarrow y$ such that $x_i \in (T_{\mathcal{F}})^{i-1}(C_p), 1 \leq i \leq n$. But the minimality of i implies that $p \in V_{x_i}$ for all $i, 1 \leq i \leq n$. Therefore, $p \in V_{x_1} \wedge \cdots \wedge V_{x_n}$ and $p \notin V_y$, *i.e.* $V_{x_1} \wedge \cdots \wedge V_{x_n} \not\subseteq V_y$. This contradicts the assumption that we have applied a *complete* applicable sequence. □

Note that since $\mathcal{F}$ does not contain $\bot$, for all environments $p \in P$, $\mathcal{F} \cup p$ is always consistent.

Incrementality and $\bot$.

What about the incrementality of the basic ATMS algorithm? When an additional justification ψ is added to $\mathcal{F}$, a complete applicable sequence $(\psi_1, \ldots, \psi_s)$ with respect to $\mathcal{F}$ may no longer be complete with respect to $\mathcal{F} \cup \{\psi\}$ even though it is still applicable. However, one can always extend $(\psi_1, \ldots, \psi_s)$ to obtain a complete applicable sequence with respect to $\mathcal{F} \cup \{\psi\}$. Observe that the results of applying the latter sequence is equivalent to applying a complete applicable sequence with respect to $\mathcal{F} \cup \{\psi\}$ on the ATMS state S, where S is the ATMS state that results from applying the sequence $(\psi_1, \ldots, \psi_s)$ on the initial ATMS. Therefore, the incremental basic ATMS algorithm can be viewed as the application of a complete applicable sequence with respect to $\mathcal{F} \cup \{\psi\}$ on the correct ATMS state with respect to $\mathcal{F}$.

The addition of a node into an ATMS does not pose any problems since the additional node, with the proper initialization if the node is an assumption or premise, will not be in the antecedent or consequence of any formula in $\mathcal{F}$. Hence, any complete applicable sequence will remain complete and applicable, and none of the existing V_X's will be affected by the addition of the new node. One could also make a non-premise node X into a premise (assumption) incrementally, by changing the value of V_X to P (downward closure of $\{\{X\}\}$) and follow it by the application of a complete applicable sequence to the resulting ATMS state.

We are now ready to address the issue of the node $\bot$. Consider an ATMS containing $\bot$. Each justification of the form $(x_1, \ldots, x_n \Rightarrow \bot)$ stands for the fact that $x_1 \wedge \cdots \wedge x_n$ is inconsistent. Let $\mathcal{F}_\bot$ be the set of justifications of the form $(x_1, \ldots, x_n \Rightarrow \bot)$, and let $\mathcal{F}' = \mathcal{F} - \mathcal{F}_\bot$. We only want to consider models of $\mathcal{F}' \cup p$ that also satisfy $\mathcal{F}_\bot$. If there exists at least one such model, we said that $\mathcal{F} \cup p$ is *consistent*. We also say that the environment p is consistent with respect to $\mathcal{F}$. If we treat $\bot$ in the manner as any other node in the basic ATMS algorithm, we have $p \in V_\bot$ if, and only if, there exists $(x_1, \ldots, x_n \Rightarrow \bot) \in \mathcal{F}$ such that $p \in V_{x_i}$ where $1 \leq i \leq n$. From theorem 31, this means $x_1 \wedge \cdots \wedge x_n$ holds, *i.e.* $\mathcal{F} \cup p$ is inconsistent. This establishes the exact correspondence between the environments in $V_\bot$ and the environments that are inconsistent with respect to $\mathcal{F}$.

Lemma 32 *If an ATMS containing* $\bot$ *is given the input* $\mathcal{F}$*, then after the basic ATMS algorithm, we have:*

$p \in V_\bot \Leftrightarrow \mathcal{F} \cup p$ *is inconsistent.* □

With the introduction of $\bot$ in an ATMS, the basic ATMS algorithm performs the step of removing any $p \in V_X$ that is also present in $V_\bot$, *i.e.* V_X is left with only consistent environments that, together with the input formulas, imply X. Hence, the basic ATMS algorithm satisfies the semantical requirement of an ATMS:

Corollary 33 . *If an ATMS is given the input* $\mathcal{F}$ *(which contain* $\bot$*), then after the basic ATMS algorithm, we have for every* $X \not\equiv \bot$*:*

$V_\bot = \{p \in P \mid \mathcal{F} \cup p$ *is inconsistent*$\}$,
$V_X = \{p \in P \mid \mathcal{F} \cup p$ *is consistent, and* $\mathcal{F} \cup p \models X\}$. □

Since $V_\bot$ increases monotonically, the results of the algorithm will not be affected if one removes new inconsistent environments from V_X $(X \not\equiv \bot)$ after each application of a justification, instead of removing them after applying a complete set of justifications as shown in our basic ATMS algorithm. de Kleer's basic ATMS algorithm does remove inconsistent environments in this incremental fashion.

Boundary representation.

In an actual implementation of an ATMS, we usually resort to boundary representation of V_X's for reasons of computational and storage efficiencies. Instead of providing the users of an ATMS with V_X, the ATMS provides $\mathcal{MAX}(V_X)$ denoted by L_X. Furthermore, for the initial ATMS, L_X is $\{\{X\}\}$ or $\{\phi\}$ when X is an assumption or a premise respectively. Otherwise $L_X = \{\}$. In the ATMS literature, one usually refers to L_X as the *label*

of X. Hence, such ATMS is said to use the *label representation*. This representation is sufficient because an environment p is in V_X for $X \neq \bot$ if there does not exist $p' \in L_\bot$ such that $p \preceq p'$ and there exists $p'' \in L_X$ such that $p \preceq p''$.[2] Therefore, to encode the information of V_X, we need only to know the labels of X and $\bot$. This nice property is captured by the following definition:

Definition: A convex space C is *representable by label* if $C = \{p \in P \mid (\forall p' \in L_\bot, \text{ p } \not\preceq p')$ and $(\exists p'' \in \mathcal{MAX}(C), p \preceq p'')\}$. The set $\mathcal{MAX}(C)$ is called the *label* of C. □

In order to make the label representation of an ATMS useful, we also need to be able to run the basic ATMS algorithm using only the labels, without any reference to the V_X's. The following shows that this can be achieved.

One can easily verify that convex spaces representable by label are downward closed sets because they also form a sublattice of the convex spaces of P, *i.e.* the meet and join operations of convex spaces representable by label are still representable by label. In particular, given two convex spaces X and Y that are representable by label, if L_X and L_Y are their labels and, S and T are $\mathcal{MAX}(X \wedge Y)$ and $\mathcal{MAX}(X \vee Y)$ respectively, then by specializing the definitions of $G_{1 \cap 2}$ and $G_{1 \cup 2}$ in the proof of Theorem 11, we obtain the following formulas for computing the labels:

$$S = \mathcal{MAX}(\{p_X \cup p_Y \mid p_x \in L_X, p_y \in L_Y, \text{ s.t. } \neg\exists p' \in L_\bot, p_X \cup p_Y \preceq p'\}) \quad (1)$$

$$T = \mathcal{MAX}(L_X \cup L_Y) \quad (2)$$

For the operations on downward closed sets, the formulas are still the same except that one need not bother with $L_\bot$ in computing S. Therefore, one can compute the meet and the join of the convex spaces in the basic ATMS algorithm in terms of their labels alone. Furthermore, the subset comparison of any two convex spaces in the algorithm can be easily decided by examing their labels. Since the procedures that compare sets and the procedures that compute the meet and join of convex spaces are the only procedures required by the basic ATMS algorithm, and we have shown that each of these procedures has an equivalent procedure that operates only on the labels, we have:

Theorem 34 *In a basic ATMS that uses label representation, the basic ATMS algorithm can be transformed to one that operates on labels only.* □

[2]This is not equivalent to there exists $p_1 \in L_\bot$ and $p_2 \in L_X$ such that $p_1 \prec p \preceq p_2$, because the latter fails when $V_\bot$ is empty.

Extended ATMS.

In [dK86b], de Kleer extended the basic ATMS to allow additional input called *primitive disjunctions* written as *choose*$(C_1, ..., C_n)$, where each $C_i, 1 \leq i \leq n$ is a distinct assumption.[3] The interpretation of the primitive disjunction is that at least one of the C_i's must be true in the ATMS. Primitive disjunctions can be used to encode negated assumptions, hence, all propositional expressions can be encoded using justifications and primitive disjunctions only. With the addition of primitive disjunctions, the basic ATMS algorithm for computing labels is no longer sufficient. To see why, consider the following example from [dK86b]. Suppose $\mathcal{F}$ is $\{A \Rightarrow a; B \Rightarrow b; C \Rightarrow c; c, a \Rightarrow \bot; c, b \Rightarrow \bot\}$. The label for the proposition $\bot$ is $\{\{A, C\}, \{B, C\}\}$. Adding *choose*$(\{A\}, \{B\})$ causes this label to change to $\{\{C\}\}$ because one of A or B holds in the new ATMS state. The basic ATMS algorithm fails to make this correction because it handles Horn clause justifications only, and our choose statement is non-Horn. To solve this problem, de Kleer corrected the labels computed by the basic ATMS algorithm using two hyper-resolution rules, one for the $\bot$ node and one for the others.

In this paper, we extend the expressive power of the choose operation to allow the encoding of complex disjunctions which can have sets of assumptions as their arguments, *i.e.* DNF (disjunctive normal form) formula of assumptions. For instance, we may have *choose*$(\{A, B, C\}, \{D, E, F\})$ and the interpretation is that either A, B and C are true or D, E and F are true. This allows greater flexibility in the encoding of knowledge in an ATMS.

In the following, we reformulate the problem of label calculation in the extended ATMS using the convex spaces. In particular, we describe a general algorithm for computing the correct labels that depends only on the meet and join operations of convex spaces. Note that the only difference between the basic and the extended ATMS is that in addition to justifications, the extended ATMS allows the input formulas $\mathcal{F}$ to contain disjunctions (we also refer to these formulas as DNF formulas or choose statements), *i.e.*

Definition: An *extended* ATMS is an ATMS where the input formulas $\mathcal{F}$ are propositional Horn clauses and DNF formulas that contains assumptions only, and $\mathcal{A}$ contains positive literals only. □

[3]De Kleer also introduces *ignore* to hide information from the problem solver. This is an added feature which does not affect the calculation of labels and hence will be ignored in this paper.

Our approach is to extend the basic ATMS algorithm in the previous section by adding a procedure to handle disjunctions. Instead of using propositional inference to correct labels, we first examine how the introduction of disjunctions changes the set of environments where a proposition holds, and use this to derive the label update algorithm for the extended ATMS.

The introduction of disjunctions into an ATMS causes the set of inconsistent environments in $V_\perp$ to expand to include any environment p with the property that *every* superset of p that satisfies at least one disjunct in every choose statement will also derive $\perp$. We now show how the set $V_\perp$ changes for our running example. Using justifications alone, we determine that the environments $\{A,C\}$, $\{B,C\}$ and $\{A,B,C\}$ are inconsistent. Thus $\mathcal{MAX}(V_\perp)$, which is the label for the $\perp$ node, is $\{\{A,C\},\{B,C\}\}$. Now we consider the effects of *choose*$(\{A\},\{B\})$. The set of inconsistent environments expands to include $\{C\}$, because every superset of $\{C\}$ that satisfies our choose statement is also inconsistent. The new value of $V_\perp$ is $\{\{C\},\{A,C\},\{B,C\},\{A,B,C\}\}$. Therefore the new label for $\perp$ is $\{\{C\}\}$.

Similarly, the set of consistent environments deriving a node X, *i.e.* V_X, is expanded to include any consistent environment p with the property that every superset of p which satisfies every choose statement also derives X. We will see in the following that these effects can be achieved by the applications of the operation Φ as defined in section 2. First, we need to introduce some notations:

- $C_X = V_X \cup V_\perp = V_X \vee V_\perp$,
- *choose*$(t^i_1, ..., t^i_{n_i})$ stands for the i_{th} disjunction, where $1 \leq i \leq n$.

Consider a node $X \neq \perp$ and a basic ATMS that is correct with respect to some formulas $\mathcal{F}$. When the i'_{th} disjunction is added to to $\mathcal{F}$, we know that at least one of the $t^{i'}_j$ (interpreted as a conjunction), $1 \leq j \leq n_{i'}$ needs to be true, hence, we need those consistent environments p such that given any choice of $t^{i'}_j$, $p \wedge t^{i'}_j \in V_X$ or $p \wedge t^{i'}_j \in V_\perp$. This will capture every consistent environment p that when expanded to agree with the i'_{th} disjunction will derive X or become inconsistent. Note that the p's that get expanded into inconsistent environments can be easily filtered out because they will now be present in the new $V_\perp$. Therefore, we want to expand V_X to be $\{p \in P \mid \forall j, 1 \leq j \leq n_{i'}, p \wedge t^{i'}_j \in C_X, p \notin V_\perp\}$ which is $\Phi_{T_{i'}}(C_X) - V_\perp$, where $T_{i'} = \{t^{i'}_j \mid 1 \leq j \leq n_{i'}\}$. Taking all the disjunctions into consideration, we expand V_X by each Φ_{T_i} until it becomes a (least) fixed point of every Φ_{T_i}. Therefore, as a consequence of the Theorem 7, we arrive at the following definition:

Definition: Given an extended ATMS with input $\mathcal{F}$, if the set of disjunctions in $\mathcal{F}$ are represented by $T_i, 1 \leq i \leq n$, then the *extended ATMS algorithm* is defined to be the following sequence of steps:

1. Apply the basic ATMS algorithm to calculate V_X using only the justifications in $\mathcal{F}$,
2. $C_X \leftarrow V_X \cup V_\perp$,
3. $V_\perp \leftarrow \Phi_{T_n} \circ \cdots \circ \Phi_{T_1}(V_\perp)$,
4. $C_X \leftarrow \Phi_{T_n} \circ \cdots \circ \Phi_{T_1}(C_X)$,
5. $V_X \leftarrow C_X - V_\perp$. □

In the following, we show that the above algorithm satisfies the semantical requirement of an ATMS, *i.e.*

Theorem 35 *In an ATMS with the input $\mathcal{F}$ which contains the set of DNF formulas $\mathcal{D} = \{\theta_1, \ldots, \theta_m\}$, at the termination of the extended ATMS algorithm given above, we have for all node $X \not\equiv \perp$:*

$$p \in V_\perp \Leftrightarrow \mathcal{F} \cup p \text{ is inconsistent,}$$
$$p \in V_X \Leftrightarrow \mathcal{F} \cup p \text{ is consistent, and } \mathcal{F} \cup p \models X.$$

Proof: To make references easier, we will assume V'_X, $V'_\perp$ and C'_X to be the values before the application of the Φ operations, *i.e.* those defined in steps 1 and 2.

¿From lemma 32 and corollary 33, we know that at the end of step 2 in the extended ATMS algorithm, we have $V'_\perp$ containing all inconsistent environments with respect to $(\mathcal{F} - \mathcal{D})$, and $p \in C'_X \Leftrightarrow (\mathcal{F} - \mathcal{D}) \cup p \models X$. If $V_\perp$ and C_X remain correct with respect to these semantics, then the resulting V_X at the end of step 5 will be correct.

We will prove the theorem by induction on the number of DNF formulas in $\mathcal{F}$. Using the results of the basic ATMS as basis, it suffices to show that the addition of a single DNF formula θ to $\mathcal{F}$ will produce the correct results.

First we will show that at the end of step 3, $V_\perp$ contains all the inconsistent environments with respect to $\mathcal{F} \cup \theta$. Observe that $\mathcal{F} \cup \theta \cup p$ is inconsistent if, and only if, $\mathcal{F} \cup p \cup t$ is inconsistent for all $t \in \theta$. By induction, we have $\mathcal{F} \cup p \cup t$ is inconsistent if, and only if, $p \wedge t \in V'_\perp$.

Therefore, $\mathcal{F} \cup p \cup t$ is inconsistent if, and only if, $p \in \Phi_T(V'_\perp)$, where T is the set of conjuncts in θ.

Next, we need to show that $p \in C_X \Leftrightarrow \mathcal{F} \cup p \models X$ at the end of step 4. Let T be the set of conjuncts in θ. We have by definition of Φ that $p \in C_X$ implies that $p \wedge t \in C'_X$ for all $t \in T$. By induction, we have $\mathcal{F} \cup (p \cup t) \models X$. Hence, $(\mathcal{F} \cup \theta) \cup p \models X$.

Conversely, given any $p \in P$ such that $(\mathcal{F} \cup \theta) \cup p \models X$. Observe that for every $t \in T$, we have $(\mathcal{F} \cup t) \cup p \models f$ for all $f \in (\mathcal{F} \cup \theta) \cup p$. Hence, $\mathcal{F} \cup (t \cup p) \models X$. By induction, we have $p \wedge t \in C'_X$ for all $t \in T$, *i.e.* $p \in C_X$. □

To make the extended ATMS algorithm meaningful, we still need to find a way to compute Φ. This can be achieved by the following:

Theorem 36 *If P is an environment lattice, then given any downward closed C with $\mathcal{MAX}(C) = \{s_1, ..., s_m\}$, and any $T = \{t_1, ..., t_n\}$,:*

$$\Phi_T(C) = \bigwedge_{1 \le j \le n} \bigvee_{1 \le i \le m} E^i_j$$

where $E^i_j = \{p \in P \mid p \wedge t_j \preceq s_i\}$. Furthermore, E^i_j is a downward closed set with a unique upper bound $e^i_j = s_i - t_j$.

Proof: For the first part of the theorem, we know that $\Phi_T(C) = \{p \in P \mid \forall t_j \in T, p \wedge t_j \in C\}$ and $p \wedge t_j \in C$ if and only if there exists $i, 1 \le i \le m$ such that $p \wedge t_j \preceq s_i$, which means $p \in \bigcup_i E^i_j = \bigvee_i E^i_j$. Therefore, $\Phi_T(C) = \bigwedge_j \bigvee_i E^i_j$.

The downward closed property of E^i_j is obvious from its definition. Furthermore, if we rewrite the definition in terms of set operations: $E^i_j = \{p \in P \mid p \cup t_j \supseteq s_i\}$, then $s_i - t_j$ is obviously the largest element (smallest subset) in E^i_j, *i.e.* $e^i_j = s_i - t_j$. □

Again, in an actual implementation of an extended ATMS, one may wish to improve the efficiency of the algorithm by exploiting the label representation. Since $\Phi_T(C) = \{p \in P \mid \forall t_i \in T, p \wedge t_i \in C\} = (E^1_1 \vee \cdots \vee E^m_1) \wedge \cdots \wedge (E^1_n \vee \cdots \vee E^m_n)$, and each of the terms E^i_j can be represented by the boundary set $\{e^i_j\}$, we can apply the formula (1) and (2) to calculate $\mathcal{MAX}(\Phi_T(C))$ – with the deletion of inconsistent environments suppressed. Hence, as a consequence of the Theorem 34, we have:

Corollary 37 *In an extended ATMS that uses label representation, the extended ATMS algorithm can be transformed to one that operates on labels only.*

In an incremental extended ATMS where recalculation of labels is needed for each new input, the addition of a disjunction involves applying the corresponding Φ operation to every label in the ATMS. However, the addition of a justification involves runing the incremental basic ATMS algorithm, followed by applying the Φ operation on each affected node for every disjunction previously input to the ATMS.

What is still lacking in the extended ATMS are the negative literals. However, given any propositional formulas with negative literals, we can always map them to an equivalent set of extended ATMS inputs without negative literals. Such a mapping was described in [dK86b]. This mapping can be rigorously formalized and shown to be correct. We leave it as an exercise for interested readers.

Examples of label calculations.

We now present some examples drawn from [dK86b] to show the calculation of label sets using the formulas derived in the previous section. We adopt the same notation as in [dK86a], upper case letters are assumptions nodes, lower case letters are derived nodes. In addition, we use the following:

- $s_1, s_2, ..., s_m$ are the environments in $\mathcal{MAX}(C_X)$, *i.e.* the $\mathcal{MAX}$ of the union of the old label set of X and $\mathcal{MAX}(V_\perp)$,
- $t_1, t_2, ..., t_n$ are the environments in the disjunction under consideration,
- $e^i_j = s_i - t_j$, for $1 \le i \le m$ and $1 \le j \le n$,
- $Y_j = \bigvee_{1 \le i \le m} E^i_j = \mathcal{MAX}(\bigcup_{1 \le i \le m} \{e^i_j\})$.

1. $A \Rightarrow a$,
$B \Rightarrow b$,
$C \Rightarrow c$,
$choose(\{A\}, \{B\})$,
$c, a \Rightarrow \perp$,
$c, b \Rightarrow \perp$.

The label of $\perp$ before considering the disjunction is $\{\{A, C\}, \{B, C\}\}$. Hence, the calculation of the label of $\perp$ is as follows:

$$s_1 = \{A, C\}, \quad s_2 = \{B, C\}$$
$$t_1 = \{A\}, \quad t_2 = \{B\}$$

e_j^i	s_1	s_2	Y_j
t_1	$\{C\}$	$\{B,C\}$	$\{C\}$
t_2	$\{A,C\}$	$\{C\}$	$\{C\}$
$L_\perp$			$\{C\}$

where $L_\perp$ is the new label set of the node $\perp$.

2. *choose*$(\{\overline{A}\},\{B\},\{C\})$,
   choose$(\{A\},\{\overline{A}\})$,
   $A,\overline{A} \Rightarrow \perp$,
   $B \Rightarrow b$,
   $C \Rightarrow c$,
   $b \Rightarrow c$,
   $c \Rightarrow d$.

The label of d before considering the disjunctions is $\{\{B\}, \{C\}\}$ and $L_\perp$ is $\{\{A,\overline{A}\}\}$, hence, $C_d = \{\{B\}, \{C\}, \{A,\overline{A}\}\}$. Since we have two disjunctions, we have the options of either applying two successive Φ's each representing one disjunction, or combine the two disjunctions and apply a single Φ operation. In this example, we adopt the latter method to illustrate the flexibility of the convex space approach. Note that the two disjunctions are equivalent to *choose*$(\{\overline{A}\}, \{A,B\}, \{A,C\})$. Hence, the calculation of label of d is as follows:

$$s_1 = \{B\}, \quad s_2 = \{C\} \quad s_3 = \{A,\overline{A}\}$$
$$t_1 = \{\overline{A}\}, \quad t_2 = \{A,B\} \quad t_3 = \{A,C\}$$

e_j^i	s_1	s_2	s_3	Y_j
t_1	$\{B\}$	$\{C\}$	$\{A\}$	$\{\{A\},\{B\},\{C\}\}$
t_2	$\{\}$	$\{C\}$	$\{\overline{A}\}$	$\{\{\}\}$
t_3	$\{B\}$	$\{\}$	$\{\overline{A}\}$	$\{\{\}\}$
L_d				$\{\{A\},\{B\},\{C\}\}$

where L_d is the new label set of the node d.

New label-update algorithm.

The convex space reformulation of the label update computations in the extended ATMS reveals new opportunities for efficient implementations. The key calculation is that of $\mathcal{MAX}(\Phi_T(C))$. From the previous section, we know that for any downward closed C, we have:

$$\Phi_T(C) = \bigwedge_j \bigvee_i E_j^i \tag{3}$$

$$\mathcal{MAX}(\Phi_T(C)) = \bigwedge_j \mathcal{MAX}(\bigcup_i \{e_j^i\}) \tag{4}$$

If we let Σ denote the set of all functions with domain $\{1,...,n\}$ and codomain $\{1,...,m\}$. We can rewrite formula (3) as:

$$\Phi_T(C) = \bigvee_{\sigma \in \Sigma} \bigwedge_j E_j^{\sigma(j)} \tag{5}$$

Observe that we always get better efficiency by using formula (3) since it requires fewer operations than formula (5). However, under certain circumstances, we may be able to ignore a big proportion of σ's in calculating $\Phi_T(C)$, thus making the use of formula (5) reasonable. In particular, as we will see later, de Kleer's hyper-resolution rules exploit such situations.

In this section, we discuss possible ways of optimizing the calculation of $\mathcal{MAX}(\Phi_T(C))$. We identify conditions on T and C that simplify this computation. In the naive implementation of the algorithm using the formula above, we first calculate the matrix of e_j^i's for $1 \leq j \leq n$ and $1 \leq i \leq m$. The columns in this matrix correspond to the s_i's which are elements of $\mathcal{MAX}(C)$: the current label of a node, the rows correspond to the t_j's which are taken from a choose statement. The entry in cell (j,i) is e_j^i which is $s_i - t_j$. We compute the union of all elements in each column to obtain $Y_j = \mathcal{MAX}(\bigcup_i \{e_j^i\})$. Then the new label of interest is $\bigwedge_j Y_j$.

Here are two special cases that allow us to compute the label sets without filling in all $n * m$ entries in the e matrix. We will fill entries in the e matrix row by row.

1. If there exist i' and j' such that $s_{i'} \succeq t_{j'}$, we have $s_{i'} - t_{j'} = \{\}$ which implies that $\bigvee_i E_{j'}^i = \{\{\}\}$. Hence we may ignore the column $j = j'$ in the calculation of formula (3) if $\{\}$ is ever generated in that column. We will call this condition the *rowdone* condition.

2. If there exists a j' such that for all i, we have $s_i - t_{j'} = s_i$, then $\bigvee_i E_{j'}^i = C$, hence, $\bigwedge_j \bigvee_i E_j^i \subseteq C$. But the right hand side of the equation is $\Phi_T(C)$ which is always larger than C, therefore, $\Phi_T(C) = C$, and no further entries in the e matrix are required. This is the *matrixdone* condition.

Sometimes, the application of Φ_T on C does not produce many new environments. In such situation, the algorithm can be further improved by first ignoring computations that will only contribute to environments that are already in C. The partial results can then be unioned with

C to obtain the desired answer. The following describes a condition when one can ignore some of the e_j^i:

3. If there exist i' and j' such that $s_{i'} - t_{j'} = s_{i'}$, *i.e.* $e_{j'}^{i'} = s_{i'}$, then $E_{j'}^{i'} \subseteq C$. Therefore, $E_{j'}^{i'}$ will not contribute to anything new in the computation of $\Phi_T(C)$ and thus can be ignored. This is the *emptycell* condition.

This optimization reduces the size of the unions that need to be taken for the calculation of Y_j.

We can use the special cases described above to improve on the naive implementation of the label calculation algorithm. The optimized algorithm takes two arguments: the label set S of a given node, and the set T of conjuncts in DNF. It fills in the e matrix one row at a time. For each column i, it examines s_i in S computing $e_j^i = s_i - t_j$ when the emptycell condition doesn't hold, until the row is exhausted or skipping over rows for which the rowdone condition is true. If at any time the matrixdone condition is satisfied, the algorithm returns the set S. Otherwise, temporary unions $Y_j = \mathcal{MAX}(\bigcup_i \{e_j^i\})$ are computed for each row. The final result is $\bigwedge_j Y_j \vee S$ where the meet operation is taken over j's that fail the rowdone condition.

procedure $\Phi(S,T)$
Initialize e[j,i] and Y[j] to nil, $1 \leq j \leq n$ and $1 \leq i \leq m$;
rowdone $\leftarrow$ false; matrixdone $\leftarrow$ false;
Result $\leftarrow$ nil;
repeat for each t_j in T
  repeat for each s_i in S
    if not emptycell(i,j)
    then $e[j,i] \leftarrow \{s_i - t_j\}$;
  until rowdone
until matrixdone
if matrixdone then return Result = S;
Y[j] = $\mathcal{MAX}(\bigcup_i e[j,i])$, $1 \leq j \leq n$ and not rowdone(j);
return Result = $\bigwedge_j Y[j] \vee S$, not rowdone(j).
end-procedure Φ

If we know that t_j's are pairwise disjoint, *i.e.* $t_{j_1} \vee t_{j_2} = \{\}$, for $1 \leq j_1 \neq j_2 \leq m$, then one can further prune away some unnecessary operations. Consider the formula (5) in the following:

4. For any fixed i, say i', and any j_1, j_2, we have $e_{j_1}^{i'} \wedge e_{j_2}^{i'} = (s_{i'} - t_{j_1}) \wedge (s_{i'} - t_{j_2}) = s_{i'} - (t_{j_1} \cap t_{j_2}) = s_{i'}$, *i.e.* $E_{j_1}^{i'} \wedge E_{j_2}^{i'} \subseteq C$. Hence, only for cases where σ is an one to one function will $\bigwedge_j E_j^{\sigma(j)}$ contribute new environments.

5. For any fixed i, say i', if there exist j_1, j_2 such that $t_{j_1}, t_{j_2} \succeq s_{i'}$, then given any set $A = \{E_j^{\sigma(j)} \mid 1 \leq j \leq n\}$ with σ one to one, then at least one of $E_{j_1}^{\sigma(j_1)}, E_{j_2}^{\sigma(j_2)}$ is not in A. Without loss of generality, assume $E_{j_1}^{\sigma(j_1)}$ is not in A. If there exists j_3 such that $\sigma(j_3) = i'$, then since t_{j_1} and t_{j_3} are disjoint, we have $e_{j_3}^{i'} \preceq t_{j_1}$. Consider $p = e_{j_1}^{\sigma(j_1)} \wedge e_{j_3}^{i'}$, since $e_{j_1}^{\sigma(j_1)} = s_{\sigma(j_1)} - t_{j_1}$, we have $p \preceq t_{j_1} \cup (s_{\sigma(j_1)} - t_{j_1}) = s_{\sigma(j_1)}$. Hence, $\bigwedge_{1 \leq j \leq n} e_j^{\sigma(j)} \preceq s_{\sigma(j_1)}$, *i.e.* $\bigwedge A \subseteq C$. Therefore, we can ignore any A that contains $E_j^{i'}$ for any j, *i.e.* $E_j^{i'}$ for $1 \leq j \leq n$ can be ignored.

In the special case where arguments of the disjunction are singleton and disjoint, we conclude that we need only consider $\bigwedge_{1 \leq j \leq n} E_j^{\sigma(j)}$ which satisfies the following:

$$\left.\begin{array}{ll} 4: & \sigma \text{ is one to one,} \\ 3: & \forall j, s_{\sigma(j)} - t_j \neq s_{\sigma(j)}, \\ 5: & \forall j', j' \neq j, s_{\sigma(j')} - t_j = s_{\sigma(j)}. \end{array}\right\} \quad (6)$$

Therefore, for each $\bigwedge_j E_j^{\sigma(j)}$ in formula (5), only those σ that satisfies (6) can contribute new environments. For any of these σ, we need only to add the environments $\bigwedge_j e_j^{\sigma(j)} = \bigcup_j (s_{\sigma(j)} - t_j)$ to the label set. This reduces to exactly the hyper-resolution rules discussed in [dK86b] for $C = V_\perp$ and $C = C_X$ which are the rules for ensuring *consistency* and *completeness* respectively.

Observation 5 leads us to the following optimized algorithm for calculating Φ when the given disjunction is primitive. T is a clause of length n whose terms are literals, and S is the label set for some node.

procedure Primitive-$\Phi(S,T)$
Initialize e[j,i] and Y[j] to nil, $1 \leq j \leq n$ and $1 \leq i \leq m$;
rowdone $\leftarrow$ false; matrixdone $\leftarrow$ false;
Result $\leftarrow$ nil;
for each s in S
  if exactly one literal b of s occurs as $t_j \in T$
  then e[j,i] $\leftarrow \{s - b\}$;
end
if matrixdone then return Result = S;
Y[j] = $\mathcal{MAX}(\bigcup_i e[j,i])$, $1 \leq j \leq n$ and not rowdone(j);
return Result = $\bigwedge_j Y[j] \vee S$, not rowdone(j).
end-procedure Primitive-Φ

There are several advantages to using this new algorithm over de Kleer's hyper-resolution approach. The absence of resolution contributes substantially to the per-

formance of our algorithm in comparison to a hyper-resolution based approach. This advantage is similar to the way ATMS's are an improvement over the earlier Truth Maintenance Systems [Doy79], because the labeling eliminates the need to re-evaluate some computations multiple times during backtracking. In our case, the redundant computation involved in a hyper-resolution based ATMS is the pattern matching required by the last condition in formula (6). This is because different σ's may share the same values for some subset of j's, hence the pattern matcher may run on every one of these σ's even though all of them may fail for the same reason.

Another advantage is the easier encoding and potential improvement in efficiency because a formula in disjunctive normal form can be asserted as a single *choose* statement. Furthermore, the new label-update algorithm allows the flexibility for combining any set of disjunctions into a single disjunction. For instance, we can combine *choose*$(\{A\},\{B\})$, *choose*$(\{B\},\{C\})$ and *choose*$(\{A\},\{C\})$ into *choose*$(\{A,B\},\{B,C\},\{A,C\})$. Hence, a single application of the Φ operation, instead of three, suffices to compute the new label of a node.

6 Conclusions

We have formulated a theory of convex spaces of partially ordered sets which includes algorithms for basic operations on finitely representable convex spaces in the presence of a simple assumption on the partial order. Using this theory, we can also describe conditions that could ensure the admissibility of the version space representation of a concept description language.

We then show how the convex spaces can be used to describe the label-update algorithm in de Kleer's basic assumption-based truth maintenance systems. This idea suggests a new approach to the label-update algorithm for the extended ATMS. Our approach generalizes the extended ATMS *choose* operations to allow the use of disjunctions such as *choose*$(\{A,B,C\},\{D,E,F\})$. This provides additional flexibility in expressing constraints and also contributes to the efficiency of label updating. Our new label-update algorithm does not require the introduction of any form of hyper-resolution rule. Instead, we use an approach which is similar to that employed in the version space algorithms to recalculate labels. This simplifies the description of how labels are updated and makes the extended ATMS label-update algorithm more consistent with the algorithm used for the basic ATMS's.

The convex space treatment of ATMS provides more than just a new algorithm for the ATMS. We also show that the approach leads to better understanding of the logical foundations for both the basic and the extended ATMS. In the paper [Nga91], we also show that if negation is introduced into the ATMS architecture, one can apply the Φ operation to calculate the prime implicates [RdK87, KT90] of any set of DNF formulas. This algorithm has been implemented on top of our ATMS implementation.

It is especially our hope that the abstraction of the convex space algorithms which we have discussed will lead to new insights in other areas which do, or could, employ similar structures for the representation of knowledge.

Acknowledgements

Gunter's work was supported in part by NSF grant CCR-8912778 and by a Young Investigator Award from the Office of Naval Research. Ngair's work was supported by the Institute of Systems Science, Singapore. Panangaden's work was supported by NSF grant CCR-8818979 and by NSERC. Subramanian's work was supported by NSF grant IRI-8902721.

References

[BM78] B. G. Buchanan and T. M. Mitchell. Model-directed learning of production rules. In D. A. Watterman and F. Hayes-Roth, editors, *Pattern-Directed Inference Systems*, Academic Press, 1978.

[dK86a] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, **28**:127–162, 1986.

[dK86b] Johan de Kleer. Extending the ATMS. *Artificial Intelligence*, **28**:163–196, 1986.

[Doy79] Jon Doyle. A truth maintenance system. *Artificial Intelligence*, **12**:231–272, 1979.

[FMH90] Yasushi Fujiwara, Yumiko Mizushima, and Shinichi Honiden. On logical foundations of the ATMS. *Proc. of ECAI-90 Workshop on Truth Maintenance Systems*, 1990.

[GHK*80] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*. Springer, 1980.

[GJ88] C. A. Gunter and A. Jung. Coherence and consistency in domains (extended outline). In Y. Gurevich, editor, *Logic in Computer Science*, pages 309–319, IEEE Computer Society, July 1988.

[GN87] M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, 1987.

[GS90] C. A. Gunter and D. S. Scott. Semantic domains. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 633–674, North Holland, 1990.

[Gun93] C. A. Gunter. The mixed powerdomain. *Theoretical Computer Science*, **102**, 1993. In press for **TCS**; available as University of Pennsylvania, Department of Computer and Information Science Technical Report 90-75.

[Hir90] Haym Hirsh. *Incremental Version Space Merging: A General Framework for Concept Learning*. Kluwer Academic Publishers, 1990.

[JPP89] Radha Jagadeesan, Prakash Panangaden, and Keshav Pingali. A fully abstract semantics for a functional language with logic variables. *Proceedings of the Third Annual IEEE Symposium on Logic in Computer Science*, 1989.

[KT90] Alex Kean and George Tsiknis. An incremental method for generating prime implicants/implicates. *J. Symbolic Computation*, 185–206, 1990.

[Mit78] Tom Mitchell. *Version Space: An approach to Concept Leaning*. PhD thesis, Stanford University, 1978.

[Nga91] Teow-Hin Ngair. *A Study on ATMS*. Technical Report Forthcoming, University of Pennsylvania, 1991.

[RdK87] Raymond Reiter and Johan de Kleer. Foundations of assumption-based truth maintence systems: preliminary report. *Proc. of AAAI-87*, 183–188, 1987.

[Sco76] D. S. Scott. Data types as lattices. *SIAM Journal of Computing*, **5**:522–587, 1976.

[Sco82] D. S. Scott. Domains for denotational semantics. In M. Nielsen and E. M. Schmidt, editors, *International Colloquium on Automata, Languages and Programs*, pages 577–613, *Lecture Notes in Computer Science vol. 140*, Springer, 1982.

[Smy83] M. Smyth. The largest cartesian closed category of domains. *Theoretical Computer Science*, **27**:109–119, 1983.

[SRP91] Vijay Saraswat, Martin Rinard, and Prakash Panangaden. Semantic foundations of concurrent constraint programming. *17th POPL*, 1991.