Reasoning Precisely with Vague Concepts*

Nita Goyal and Yoav Shoham
Robotics Laboratory, Computer Science Department
Stanford University
Stanford, CA 94305
{nita,shoham}@cs.stanford.edu

Abstract

Many knowledge-based systems need to represent vague concepts. Although the practical approach of representing vague concepts as precise intervals over numbers is well-accepted in AI, there is no systematic method to delimit the boundaries of intervals, only *ad hoc* methods. We present a framework to reason precisely with vague concepts based on the observation that the vague concepts and their interval-boundaries are constrained by the underlying domain knowledge. The framework is comprised of a constraint language to represent logical constraints on vague concepts, as well as numerical constraints on the interval-boundaries; a query language to request information about the interval boundaries; and a computational mechanism to answer the queries. A key step in answering queries is preprocessing the constraints by extracting the numerical constraints from the logical constraints and combining them with the given numerical constraints.

1 Introduction

The input to an AI system embedded in a real-world environment is often numerical whereas the reasoning is done with abstract symbols. Many abstract symbols embody vague concepts over continuous numerical ranges. To quote Davis, "In some respects, the concepts of commonsense knowledge are *vague*,... Many categories of common sense have no well-marked boundary lines; there are clear examples and clear nonexamples, but in between lies an uncertain region that we cannot categorize, even in principle." [Davis 1990]. For example, there is no minimum precise body temperature that a doctor considers high and there is no maximum number of hairs that a person might have and still be considered bald.

The representation of such vagueness poses a problem. "From a theoretical point of view, this vagueness is extremely difficult to deal with, and no really satisfactory solutions have been proposed." [Davis 1990]. Some of the approaches that try to address this theoretical difficulty are fuzzy logic [Zadeh 1983] and vague predicates [Parikh 1983]. However, it is commonly accepted in AI that, though inadequate theoretically, in practice it is often adequate to assume that a vague concept *is* precise and that there is indeed a well-defined boundary. In fact, most system builders who encounter the vagueness problem [Hayes-Roth *et al.* 1989; Shahar, Tu & Musen 1992] adopt a similar approach of representing a vague concept as an interval over the range of numbers. This practical approach is illustrated by an example from [Davis 1990]: "Suppose that "bald" did refer to some specific number of hairs on the head, only we do not know which number. We know that a man with twenty thousand hairs on his head is not bald, and that a man with three hairs on his head is bald, but somewhere in between we are doubtful." This precise representation of the vague concept bald is still useful for reasoning.

Despite the pervasiveness of the vagueness problem, and the pervasiveness of the practical approach of representing vague concepts as intervals, there has been no effort in AI to provide a systematic account of this practical approach. We propose a framework for representing and reasoning with vague concepts as intervals that has the advantages of (1) improving our understanding of the issues involved in the practical approach, and (2) replacing the *ad hoc* approach used by system designers to delimit the interval-boundaries.

The framework is based on the observation that vague concepts and their interval-boundaries (also referred to as *thresholds*) are constrained by the underlying domain knowledge that must be used to reason about the thresholds. We motivate the components of the framework by extending Davis' baldness example.

Example 1: "Anyone with 3 or fewer hairs is bald and anyone with 20000 or more hairs is not bald"
"All old people are bald" (note that "old" itself is a vague concept that we will assume has a well-defined boundary)
"Anyone who is 50 years or younger is not old whereas anyone over 80 is old"

*This research has been supported by grant AFOSR-89-0326.

"All presidents of companies are old"
"Tom's age is 70 years, he has 500 hairs and is the president of a company"
"Jim's age is 75 years and he has 800 hairs"
"Sam's age is 45 and he has 650 hairs"

Is Tom bald? Logical reasoning tells us that since Tom is president of a company, he is old and therefore bald. Note that here we used only the logical relations between the concepts *president*, old and bald, where old and bald are vague concepts but *president* is not.

Is Jim bald? We can reason that since Tom is old, the oldness threshold[1] can be at most 70. Since Jim's age is 75 which is over the oldness threshold, he must be old and therefore bald. Note that here we needed numerical reasoning with Tom and Jim's ages and oldness threshold, as well as logical reasoning that since Jim is old he must be bald.

We can ask if the baldness threshold is necessarily more than 800? Since Jim is bald and has 800 hairs, the baldness threshold must be at least 800. Therefore, the answer to the query is *yes* and hence anyone with less than 800 hairs is bald. Here we needed numerical reasoning about Jim's hairs and the baldness threshold.

Is Sam bald? Since anyone with less than 800 hairs is bald, and Sam has only 650 hairs, he must be bald. Here we needed numerical reasoning with number of hairs on Sam's head and the baldness threshold. ∎

As illustrated by this example, we need to *represent* both logical relations between symbolic concepts and numerical relations on thresholds. Also, logical as well as numerical *reasoning* is required to answer the interesting queries. Hence, the proposed framework facilitates this representation and supports queries about the thresholds.

Framework: The framework is comprised of three main parts – a constraint language to express domain knowledge, a query language to query the domain knowledge and a computational mechanism to answer the queries.

The first part of the framework is a *constraint language* that captures the domain knowledge. The language enables the expression of logical constraints on the vague concepts as well as numerical constraints on the thresholds of these concepts. Explicit representation of the thresholds is important to represent the numerical constraints and as we shall see, to ask queries.

The second part of the framework is a *query language* that extracts relevant information about the thresholds implied by the domain knowledge. In particular, the queries enable us to delimit the thresholds based on the information provided in the domain knowledge[2]. This is exactly what a system designer needs to define intervals for a vague concept that are consistent with the domain knowledge. For example, the answer to the query "what is the minimum permissible value for the baldness threshold?" provides the designer with useful information to define the interval for bald.

The third part of the framework is a *computational mechanism* to answer the queries in the query language using the domain knowledge expressed in the constraint language.

In Sections 2 and 3, we introduce particular constraint and query languages. In Section 4, we describe a computational mechanism to answer queries for these languages. It includes a sound and complete algorithm, a discussion of the complexity, and experimental results illustrating the applicability of the framework. Our experiments were carried out in the domain of medical diagnosis where numerical measurements of parameters such as blood pressure and heart rate are abstracted to vague concepts such as high and low blood pressure and used for the diagnosis of the patient's condition. In this paper, for the sake of clarity and understanding, we stick to the more everyday example of bald people.

2 Constraint Language

To express the domain knowledge, the constraint language must have an explicit representation of thresholds, a language to express numerical constraints and a language to express the logical constraints. We present such a language here, chosen for its familiarity as well as to strike a tradeoff between expressivity and efficiency of answering queries.

The vague predicates in the logical language are distinguished from the other predicates. We refer to the vague predicates, which must all be unary, as *interval-predicates* and to all other predicates as *noninterval-predicates*. The set of *interval-predicates* is denoted by $\mathcal{IP}$, and the set of *noninterval-predicates* by $\mathcal{NIP}$. With every $\mathsf{P} \in \mathcal{IP}$ we associate two *threshold terms* P^- and P^+, called the *lower* and *upper thresholds* of P, respectively. The set of all threshold terms is denoted by $\mathcal{T}$ ($\mathcal{T} = \{\mathsf{P}^-, \mathsf{P}^+ \mid \mathsf{P} \in \mathcal{IP}\}$). The interval-predicates will be interpreted in a special way to reflect our intuition about the vague predicates: P will be interpreted as the interval $[\mathsf{P}^-,\mathsf{P}^+]$ over $\Re$, the set of real numbers. We will refer to this interpretation as the *predicate-as-interval assumption.*

1. **Numerical Constraints:** The language of numerical constraints is that of linear arithmetic inequalities where the threshold terms in $\mathcal{T}$ are the variables of the inequalities. A numerical constraint must be reducible to the form $(a_1x_1 + \ldots + a_nx_n)$ *rel* b, where $a_1, \ldots, a_n, b \in \Re$, $x_1, \ldots, x_n \in \mathcal{T}$, and $rel \in \{\leq, \geq, <, >, =\}$. We denote the set of numerical constraints by NC.

[1] By oldness threshold we mean that age such that everyone of higher age is old whereas everyone of lower age is not old. The baldness threshold is defined analogously.

[2] Note that it is not necessary to assign specific values to the thresholds to answer any queries, although this assignment is made much easier in our framework.

2. **Logical Constraints:** These are definite Horn clauses without function symbols (also called Datalog sentences in the deductive database literature [Ullman 1988]). The predicates of these logical constraints are interval-predicates $\mathcal{IP}$ as well as noninterval-predicates $\mathcal{NIP}$. We denote the set of logical constraints by LC.

The constraints in Example 1 are represented in the language as follows. We extend the example to include another constraint that all rich VPs become presidents of companies.

Example 2:

$$\mathcal{IP} = \{\mathsf{bald}, \mathsf{old}, \mathsf{rich}\}$$

$$\mathcal{NIP} = \{age, hairs, pres, money, was\_VP\}$$

$$NC = \{\mathsf{bald}^- = 0,\ 3 \leq \mathsf{bald}^+ \leq 20000,\ \mathsf{old}^+ = \infty,\ 50 \leq \mathsf{old}^- \leq 80,\ 0.1 \leq \mathsf{rich}^- \leq 1,\ \mathsf{rich}^+ = \infty\ \} \cup \{\mathsf{P}^- \leq \mathsf{P}^+ \mid \mathsf{P} \in \mathcal{IP}\}$$

The unit for bald is number of hairs, for old is age in years, and for rich is money in millions of dollars.

$$LC = \{pres(x) \leftarrow was\_VP(x) \wedge money(x,y) \wedge \mathsf{rich}(y)$$
$$\mathsf{bald}(z) \leftarrow \mathsf{old}(y) \wedge age(x,y) \wedge hairs(x,z)$$
$$\mathsf{old}(y) \leftarrow pres(x) \wedge age(x,y)$$
$$age(\mathtt{Tom}, 70),\ hairs(\mathtt{Tom}, 500),\ was\_VP(\mathtt{Tom}),$$
$$money(\mathtt{Tom}, 6),\ age(\mathtt{Jim}, 75),\ hairs(\mathtt{Jim}, 800)$$
$$age(\mathtt{Sam}, 45),\ hairs(\mathtt{Sam}, 650)\}\ \blacksquare$$

There are other languages that combine quantitative and qualitative constraints. For instance, Williams' qualitative algebra [Williams 1988] expresses operations on reals and signs of reals, but is not concerned with logical constraints. Similarly, [Meiri 1991] and [Kautz & Ladkin 1991] present frameworks for expressing and processing both quantitative and qualitative temporal constraints. Their language limits the constraints, whether numerical or logical, to be binary whereas our language does not. On the other hand, their language can express disjunctive relations between intervals which our language does not.

Most closely related to our language are languages for *constraint logic programming* (CLP) in the style of Lassez *et al.* [Jaffar & Lassez 1987]. CLP considers general Horn theories, as opposed to our limited Datalog theories. However, CLP does not allow numerical constraints in the head of a clause. In our language the interval-predicates can occur in the head which, if represented in CLP, would correspond to numerical constraints occurring in the head.

3 Query Language

The purpose of the query language is to enable a user to extract information about the thresholds that is implied by the domain constraints. It is a useful tool for a system designer to find the threshold values allowed by the constraints. The kind of queries supported are informally described below. Here $\mathsf{P}_1^{th}, \ldots, \mathsf{P}_n^{th} \in \mathcal{T}$, $\mathsf{a}_1, \ldots, \mathsf{a}_n \in \Re$, $rel_1, \ldots, rel_n \in \{\leq, \geq, <, >, =\}$, and $i \in \{1, \ldots, n\}$.

1. Is it *necessarily* the case that $(\mathsf{P}_1^{th}\ rel_1\ \mathsf{a}_1) \wedge \ldots \wedge (\mathsf{P}_n^{th}\ rel_n\ \mathsf{a}_n)$?
2. Is it *possibly* the case that $(\mathsf{P}_1^{th}\ rel_1\ \mathsf{a}_1) \wedge \ldots \wedge (\mathsf{P}_n^{th}\ rel_n\ \mathsf{a}_n)$?
3. What is the *minimum* value that P_i^{th} can take?
4. What is the *maximum* value that P_i^{th} can take?

Many queries may be derived using the above primitives. For example, the query "P(a) ?" can be cast as "Is it necessarily the case that $(\mathsf{P}^- \leq \mathsf{a}) \wedge (\mathsf{P}^+ \geq \mathsf{a})$?". If the answer is *yes* then P(a) is *true*, otherwise it is unknown. If the answer to "Is it possibly the case that $(\mathsf{P}^- \leq \mathsf{a}) \wedge (\mathsf{P}^+ \geq \mathsf{a})$?" is *no* then P(a) is *false*, otherwise it is unknown.

In addition, it is possible to request a specific assignment of values to the thresholds that satisfies the constraints. For Example 2, assigning the value 2000 to the baldness threshold is consistent with the given constraints. We will indicate briefly in Section 4.3 how this assignment is made. This procedure is particularly useful for a system designer who assigns specific numbers to the thresholds in the design stage of the system.

4 Computational Mechanism for Answering Queries

The final component of our framework is the computational mechanism responsible for answering queries on constraints. A key step in answering queries is preprocessing the constraints by extracting the numerical constraints from the logical constraints and combining them with the given numerical constraints. The preprocessing is a two-step procedure: first, using the predicate-as-interval assumption, the procedure extracts the numerical information on the interval-predicates from the logical constraints LC. This derived information is in the form of disjunctions of numerical constraints. Next, the procedure combines these disjunctive constraints with the given numerical constraints NC. We describe the procedure to extract numerical information from LC in Section 4.1. In Section 4.2 we prove that the procedure is sound and complete and also discuss the complexity issues. In Section 4.3 we describe how to combine the numerical information from LC with NC.

4.1 Extracting Numerical Information from Logical Constraints

The algorithm *Symb_to_Numeric* described in this section takes as input the logical constraints LC and the sets of interval and noninterval-predicates $\mathcal{IP}$ and $\mathcal{NIP}$, and returns a set of numerical constraints *quant_LC*. This process of conversion from logical to numerical constraints preserves the information about the thresholds of interval-predicates but discards the information on the noninterval-predicates. A formal discussion is deferred to Section 4.2.

Function $Symb\_to\_Numeric(LC, \mathcal{IP}, \mathcal{NIP})$: $quant\_LC$
  $S \leftarrow \emptyset$;
  for every clause $c \in LC$ such that $head(c) \in \mathcal{IP}$ **do**
    $S_c \leftarrow Expand(c, LC, \mathcal{IP}, \mathcal{NIP})$;
    $S \leftarrow S \cup S_c$; /* S has no $\mathcal{NIP}$ predicates */
  endfor
  $quant\_LC \leftarrow \emptyset$;
  for every clause $c \in S$ **do**
    $quant\_LC \leftarrow quant\_LC \cup Convert\_LC\_to\_NC(c)$;
  return($quant\_LC$)
endfunction

Figure 1: Numerical Information from Logical Constraints

The algorithm *Symb_to_Numeric* is described in Figure 1. Starting with all those clauses in LC that have interval-predicates at the *head*, we *expand* their bodies using other clauses in LC until all noninterval-predicates are eliminated from the body. *Expand* is very similar to SLD resolution [Lloyd 1987] but with two differences: (1) only noninterval-predicates are expanded (2) all possible expansions are computed. Thus, each clause in set S of Figure 1 has only interval-predicates. Using the predicate-as-interval assumption, we convert the resultant clauses to numerical constraints as described by function *Convert_LC_to_NC* in Figure 2. This function works by fragmenting each clause into subclauses such that each subclause has at most one variable and no two subclauses have the same variable[3]. For example, the clause $\mathsf{P(a)} \leftarrow \mathsf{Q}(x) \wedge \mathsf{R}(x) \wedge \mathsf{S(b)}$ is a disjunction of three subclauses: "$\mathsf{P(a)}$", "$\leftarrow \mathsf{Q}(x) \wedge \mathsf{R}(x)$" and "$\leftarrow \mathsf{S(b)}$". In general, each subclause thus obtained will be one of the six basic types described in Figure 2. Each type of subclause is converted to numerical constraint by using the predicate-as-interval assumption, and by interpreting the connectives $\neg, \vee, \wedge$ as complement, union and intersection of intervals, respectively.

An application of the algorithm on Example 2 is illuminating:

Example 3: The first step in the procedure is to locate clauses with interval-predicates at the head in LC and expand them until all noninterval-predicates are eliminated. Here there are two such clauses with old and bald at the head. On expansion, we obtain set S:

$\mathsf{bald(500)} \leftarrow \mathsf{old(70)}$ $\mathsf{old(70)} \leftarrow \mathsf{rich(6)}$
$\mathsf{bald(800)} \leftarrow \mathsf{old(75)}$ $\mathsf{bald(650)} \leftarrow \mathsf{old(45)}$

On applying the function *Convert_LC_to_NC*, each of these clauses fragments into subclauses of the first two types: "$\mathsf{P(a)}$" and "$\leftarrow \mathsf{P(a)}$". On conversion we obtain the set $quant\_LC$[4]:

$quant\_LC =$
$\{(\mathsf{bald}^- \leq 500 \leq \mathsf{bald}^+) \vee (70 < \mathsf{old}^-) \vee (\mathsf{old}^+ < 70)$
$(\mathsf{old}^- \leq 70 \leq \mathsf{old}^+) \vee (6 < \mathsf{rich}^-) \vee (\mathsf{rich}^+ < 6)$
$(\mathsf{bald}^- \leq 800 \leq \mathsf{bald}^+) \vee (75 < \mathsf{old}^-) \vee (\mathsf{old}^+ < 75)$
$(\mathsf{bald}^- \leq 650 \leq \mathsf{bald}^+) \vee (45 < \mathsf{old}^-) \vee (\mathsf{old}^+ < 45)\}$ ■

[3]Note that this is always possible because all interval-predicates are unary.

[4]Note that each of the 4 clauses obtained here will actually split into 2 clauses.

Function $Convert\_LC\_to\_NC(lc)$: nc
  $nc \leftarrow \emptyset$;
  $lc\_subclauses \leftarrow Make\_Subclauses(lc)$; /* Every subclause of lc with a constant or the same variable */
  for every subclause $subcl \in lc\_subclauses$ **do**
    Case $subcl$ **of**: /* a is a constant */
      "$\mathsf{P(a)}$": $subcl' \leftarrow (\mathsf{P}^- \leq \mathsf{a} \leq \mathsf{P}^+)$
      "$\leftarrow \mathsf{P(a)}$": $subcl' \leftarrow (\mathsf{a} < \mathsf{P}^-) \vee (\mathsf{a} > \mathsf{P}^+)$
      "$\mathsf{P}(x)$": $subcl' \leftarrow (\mathsf{P}^- = -\infty) \wedge (\mathsf{P}^+ = +\infty)$
      "$\leftarrow \mathsf{P}(x)$": $subcl' \leftarrow \mathsf{P}^- > \mathsf{P}^+$
      "$\leftarrow \mathsf{P}_1(x), \ldots, \mathsf{P}_n(x)$":
        $subcl' \leftarrow \vee_{i=1}^n \vee_{j=1}^n (\mathsf{P}_i^- > \mathsf{P}_j^+)$
      "$\mathsf{P}(x) \leftarrow \mathsf{Q}_1(x), \ldots, \mathsf{Q}_n(x)$":
        $subcl' \leftarrow (\vee_{i=1}^n \mathsf{P}^- \leq \mathsf{Q}_i^-) \wedge (\vee_{i=1}^n \mathsf{P}^+ \geq \mathsf{Q}_i^+)$;
    $nc \leftarrow nc \vee subcl'$
  endfor
  return(nc)
endfunction

Figure 2: Conversion from Logical to Linear Arithmetic Constraint

4.2 Formal results on conservation of numerical information

We establish formally that no numerical information is lost in the conversion performed by algorithm *Symb_to_Numeric*. We begin by defining the models of LC that are faithful to the predicate-as-interval assumption; we call these the *standard models*. Specifically, in all standard models $M = (D, \mu)$ over a domain D, the interpretation function μ will have to map interval-predicates to intervals over the reals. In the following, $\Re$ denotes the set of real numbers.

Definition 1: Given a set of logical constraints LC, the set of interval-predicates $\mathcal{IP}$, and the set of threshold terms $\mathcal{T}$, a *standard model of LC w.r.t.* $\mathcal{IP}$ is a model $M = (D, \mu)$ such that $M \models LC$, and for every $\mathsf{P} \in \mathcal{IP}$ there exist $\mathsf{P}^-, \mathsf{P}^+ \in \mathcal{T}$ and it is the case that $\mu(\mathsf{P}^-), \mu(\mathsf{P}^+) \in \Re$ and $\mu(\mathsf{P}) = \{x \mid \mu(\mathsf{P}^-) \leq x \leq \mu(\mathsf{P}^+),\ x \in \Re\}$. ■

Definition 2: Given LC, $\mathcal{IP}$ and $\mathcal{T}$ as above, a *numerical submodel of LC w.r.t.* $\mathcal{IP}$ is a model $M = (\Re, \mu)$ such that there is some standard model $M' = (D, \mu')$ of LC w.r.t. $\mathcal{IP}$, and μ is the restriction of μ' to terms in $\mathcal{T}$. ■

The following theorem establishes that the algorithm *symb_to_numeric* is sound and complete w.r.t. the numerical information (complete proof in [Goyal 1993]).

Theorem 4: (Soundness and Completeness) *The class of numerical submodels of LC w.r.t.* $\mathcal{IP}$ *is identical to the class of models of* $quant\_LC$.

Proof: *(Sketch for Completeness)* An arbitrary model M of $quant\_LC$ is extended to a standard model M' of

LC such that its numerical submodel is exactly M. M' is constructed by first building a dependency graph of predicates in LC and then by defining the interpretation of the predicates in $\mathcal{NIP}$ in the topological order of the graph. The intuition is that when a clause in LC is used to build the interpretation of the predicate in the head from the predicates in the body, the body predicates would have been already interpreted because of the order of interpretation. The equivalence of the numerical submodel of M' and the model M is proved through mathematical induction on the topological order of predicates. ∎

In the worst case, the space and time complexity of computing *quant_LC* is exponential in the size of LC. This is not surprising, since in the worst case, *quant_LC* is of exponential size. However, we have identified syntactic restrictions on the constraint language for which we can avoid such exponential blowup. In practice, the performance of the algorithm has been found to be quite acceptable for the following reasons. First, we observe that the algorithm is exponential only in the size of the non-ground constraints. Typically, the number of non-ground constraints is small compared to the number of ground literals. Second, this algorithm is invoked only once for all the queries on a given set of constraints; hence, the cost is amortized over all the queries. Thus, the overall performance of the system is not severely affected despite the apparent intractability. A more detailed discussion of the complexity issues may be found in [Goyal 1993].

4.3 Combining with Numerical Constraints

The constraints in the set *quant_LC*, obtained by converting the logical constraints to numerical constraints, are disjunctive. These constraints must be combined with the set of given numerical constraints NC to answer the queries. In principle, we can convert the set *quant_LC* to disjunctive normal form (DNF) and add the constraints NC to each disjunct. The disjunction thus obtained is referred to as *output_C*. However, in practice, we leave *quant_LC* in its conjunctive normal form to save space, and generate the disjuncts of *output_C* one by one through backtracking. Furthermore, to make the process more efficient, we first reduce the size of the set *quant_LC* using the constraints NC. We elaborate on these below and also discuss how existing methods are applicable to answer queries on a single disjunct of *output_C*.

Pruning *quant_LC*: We have developed a procedure that uses the constraints in NC to reduce the size of the set *quant_LC* significantly. This procedure, called *reduce*, uses the upper and lower bounds of all thresholds implied by the constraints in NC to prune *quant_LC* in two ways. First, if a disjunct of a constraint in *quant_LC* is already satisfied by the bounds, then that constraint can be deleted from *quant_LC*. In Example 3, the lower and upper bounds for old^- are 50 and 80 respectively, hence $(\mathsf{old}^- > 45)$ is already satisfied. Second, if a disjunct is inconsistent with the bounds, then that disjunct can be deleted. For instance, $(\mathsf{old}^- > 82)$ is inconsistent with the bounds for old^-.

The experimental results confirm that the procedure *reduce* reduces the size of *quant_LC* significantly. In Example 3, *quant_LC* has 8 constraints with 3 disjuncts each that should give rise to 3^8 disjuncts (in DNF) in the worst case. Applying procedure *reduce* eliminates all but 1 disjunct, that is:

$$output\_C = NC \cup \{(\mathsf{old}^- \leq 70),\ (\mathsf{bald}^+ \geq 800)\}$$

When the procedure was applied to the medical diagnosis domain, in the first application *quant_LC* had 12 constraints with 3 disjuncts each, giving rise to 3^{12} disjuncts in the worst case. Procedure *reduce* eliminated all but 2 disjuncts. In a second medical application, *quant_LC* had 416 constraints with 2 or 3 disjuncts per constraint that would have given rise to at least 2^{416} disjuncts. Procedure *reduce* eliminated all but 2592 disjuncts.

Generating disjuncts of *output_C*: Once the set *quant_LC* has been pruned, queries are answered by generating the remaining disjuncts of *output_C* one at a time through backtracking. We avoid generating redundant disjuncts in *output_C* by recognizing the presence of common disjuncts in the constraints of *quant_LC*. For instance, in the second medical application, only 184 disjuncts had to be generated out of the 2592 that were possible.

In practice, most queries do not require backtracking even over all possible distinct disjuncts that are generated. For instance, a query whether a constraint is *possibly* true or not, has to find any one disjunct over which the constraint is satisfied. Furthermore, even for queries where all disjuncts have to be checked, an approximate answer can be obtained by computing only on a few disjuncts. For instance, a query to find the minimum value of a threshold can return the minimum over only a few disjuncts. This approximate answer is still useful since it supplies a lower bound on the threshold, even though not the tightest lower bound. Thus, this procedure gives a useful approximate answer any time that an answer is required, and the approximation gets closer to the optimal as the allowed time increases. The experimental results on answering queries are available in [Goyal 1993].

We can even assign a specific value to the thresholds using heuristic criteria. For Example 3, the baldness threshold (bald^+) could be 10400 which is halfway between its bounds of 800 and 20000, and satisfies all the given constraints. When a large amount of ground data is available, clustering techniques are utilized to assign a specific value.

Answering for each disjunct: We have discussed previously how the set *quant_LC* is pruned *a priori* to eliminate redundant disjuncts and how the disjuncts

of *output_C* are generated. Each disjunct thus generated is a set of linear arithmetic constraints. We now discuss how any query is answered on a single disjunct. The queries for maximum and minimum values of thresholds (queries 3 and 4 in Section 3) require the computation of lower and upper bounds of thresholds. Queries for checking a constraint for consistency (queries 1 and 2 in Section 3) require a consistency check on a set of constraints. Thus, any existing method for computing bounds and checking consistency of linear arithmetic constraints can be used. If NC has only simple order relations or bounded differences, we can use an efficient $\mathcal{O}(n^3)$ procedure (where n is the number of variables) from [Davis 1987] or [Meiri 1991]. Sacks' *bounder* [Sacks 1990] is applicable but more useful for nonlinear constraints. For more general linear constraints, we have to use a linear programming method that is still tractable $\mathcal{O}(n^{3.5}L)$ (L is size of input) [Karmarkar 1984]. Lassez's work on canonical form of generalized linear constraints [Huynh *et al.* 1990] has potential applications, though the advantage of a canonical form would be offset by the cost of maintaining the canonical form because we backtrack on disjunctive constraints.

5 Conclusions

We have provided a systematic account of the practical approach of representing vague concepts as precise intervals over numbers. Based on the observation that the vague concepts and their interval-boundaries are constrained by the underlying domain knowledge, we motivated and proposed a framework to reason precisely with vague concepts. The framework is comprised of a constraint language to represent the domain knowledge; a query language to request information about the interval boundaries; and a computational mechanism to answer the queries.

We described the constraint and query languages and a computational mechanism to answer queries. A key step in answering queries is preprocessing the constraints by extracting the numerical constraints from the logical constraints and combining them with the given numerical constraints. We proved this algorithm to be sound and complete and also discussed the complexity issues. Some experimental results of applying this framework to a medical domain were discussed.

The main contribution of our work is in providing a systematic framework to understand the common though *ad hoc* approach of representing vague predicates as intervals. This work is particularly applicable to a knowledge base during its development stage where the vague concepts over numbers need to be defined precisely.

Acknowledgements We would like to thank Surajit Chaudhuri, Ashish Gupta, Alon Levy, Pandu Nayak, Moshe Tennenholtz, Becky Thomas and the anonymous reviewers.

References

Davis, E. 1987. Constraint Propagation with Interval Labels. *Artificial Intelligence* 32(3):281–331.

Davis, E. 1990. *Representations of Commonsense Knowledge*. Morgan Kaufmann Publishers, 19–20.

Goyal, N. 1993. A Framework for Reasoning Precisely with Vague Concepts. Ph.D. diss. (in preparation), Dept. of Computer Science, Stanford University.

Hayes-Roth, B.; Washington, R.; Hewett, R.; Hewett, M.; and Seiver, A. 1989. Intelligent Monitoring and Control. In *Proc. of Eleventh International Joint Conference on Artificial Intelligence*, 43–249.

Huynh, T.; Joskowicz, L.; Lassez, C.; and Lassez, J-L. 1990. Reasoning about Linear Constraints using Parametric Queries. In *Proc. of Tenth FST TCS*, Bangalore, India.

Jaffar, J.; and Lassez, J-L. 1987. Constraint Logic Programming. In *Proc. of 14th ACM Symposium on Principles of Programming languages*, 111–119.

Karmarkar, N. 1984. A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica* 4:373–395.

Kautz, H.A.; and Ladkin, P.B. 1991. Integrating Metric and Qualitative Temporal Reasoning. In *Proc. of Ninth National Conference on Artificial Intelligence*, 241–246.

Lloyd, J.W. 1987. *Foundations of Logic Programming, 2nd. ed.*. Springer-Verlag.

Meiri, I. 1991. Combining Qualitative and Quantitative Constraints in Temporal Reasoning. In *Proc. of Ninth National Conference on Artificial Intelligence*, 260–267.

Parikh, R. 1983. The problem of Vague Predicates. In Cohen and Wartofsky (eds.), *Language, Logic, and Method.* Reidel Publishers, 241–261.

Sacks, E. 1990. Hierarchical Reasoning about Inequalities. In *Readings in Qualitative Reasoning about Physical Systems*, eds. D.S. Weld and J. de Kleer. Morgan Kaufmann Publishers, 344–350.

Shahar, Y.; Tu, S.W.; and Musen, M.A. 1992. Knowledge Acquisition for Temporal-Abstraction Mechanisms. *Knowledge Acquisition* 4:217–236.

Ullman, J.D. 1988. *Principles of Database and Knowledge-Base Systems, Vol. 1*. Computer Science Press.

Williams, B.C. 1988. MINIMA: A Symbolic Approach to Qualitative Algebraic Reasoning. In *Proc. of Seventh National Conference on Artificial Intelligence*, 264–269.

Zadeh, L.A. 1983. Commonsense and Fuzzy Logic. In *The Knowledge Frontier: Essays in the Representation of Knowledge*, eds. N. Cercone and G. McCalla. New York: Springer-Verlag, 103–136.