Constraint Processing Offers Improved Expressiveness and Inference for Interactive Expert Systems

James Bowen

Cork Constraint Computation Centre
UCC, Cork, Ireland
Email: j.bowen@4c.ucc.ie

Abstract. Expert systems constitute one of the most successful application areas for Artificial Intelligence techniques; they have been deployed in many areas of industry and commerce. If-then rules are the core knowledge representation technology in currently deployed systems. However, if we replace rules by constraints, we get improved expressiveness in knowledge representation and richer inference.

1 Introduction

An expert system [5] is a program which mimics human problem-solvers, in several senses: it contains an explicit representation of the knowledge which is used by humans who are expert at solving tasks in some problem domain; its reasoning process mimics that of the human experts; it elicits data from its users in a fashion similar to that used by a human expert who questions his client during a consultation; it can explain its answers to its users in the same way as a human expert can explain his conclusions to his clients. Expert systems constitute one of the most successful application areas for Artificial Intelligence (AI) techniques - they have been assimilated into the mainstream where they are widely deployed, frequently in concert with non-AI software technologies [8].

In currently deployed expert systems, knowledge about the problem domain is represented in the form of if-then rules. There are two kinds of rules [5]: imperative rules, where the consequent of a rule specifies some operation(s) to be performed if the antecedent if satisfied; and declarative rules, where the consequent of a rule specifies some fact which is implied by the truth of the antecedent.

In this paper, it is argued that, if constraints are used instead of declarative rules, improved functionality is achieved: constraints provide a richer expressive medium than rules; constraints support a richer form of inference than rules.

The rest of this paper is organised as follows. First, a review of constraint-based reasoning is given, with particular emphasis on interactive processing. Then the notion of using constraints to build interactive expert systems is explored, with particular emphasis on mixed-initiative information acquisition during interactive processing. Following this, constraint programming is related, in a model-theoretic fashion, to Predicate Calculus, with particular emphasis on a treatment of constraint satisfaction as model completion. The approach is illustrated by means of an example expert system for selecting a laptop to meet a user's needs; in this discussion, emphasis is placed on the expressiveness of constraint-based knowledge representation. Then, it is explained that constraint propagation provides a richer form of inference than that supported by rule-based systems, because it supports *modus tollens* as well as *modus ponens*.

2 Constraint-based reasoning

In the literature, several different definitions are given for constraint networks, with varying degrees of formality. However, they may all be regarded as variations of the following theme:

> *Definition 1, Constraint Network:*
> A constraint network is a triple $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$. $\mathbf{D}$ is a finite set of $p > 0$ domains, the union of whose members forms a universe of discourse, $\mathcal{U}$. $\mathbf{X}$ is a finite tuple of $q > 0$ non-recurring parameters. $\mathbf{C}$ is a finite set of $r \geq q$ constraints. In each constraint $C_k(T_k) \in \mathbf{C}$, T_k is a sub-tuple of $\mathbf{X}$, of arity a_k; $C_k(T_k)$ is a relation, a subset of the a_k-ary Cartesian product $\mathcal{U}^{a_k}$. In $\mathbf{C}$ there are q unary constraints of the form $C_k(X_j) = D_i$, one for each parameter X_j in $\mathbf{X}$, restricting it to range over some domain $D_i \in \mathbf{D}$ which is called the *domain of* X_j.

The overall network constitutes an intensional specification of the simultaneous value assignments that can be assumed by the parameters. In other words, the network constitutes an intensional specification of a q-ary relation on $\mathcal{U}^q$, in which each q-tuple provides an ordered group of values, each value being an assignment for the corresponding parameter in $\mathbf{X}$. This implicitly specified relation is called the *intent* of the network:

> *Definition 2, The Intent of a Constraint Network:*
> The intent of a constraint network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$ is
> $$\Pi^{\mathbf{D},\mathbf{X},\mathbf{C}} = \overline{C_1}(\mathbf{X}) \cap ... \cap \overline{C_r}(\mathbf{X}),$$
> where, for each constraint $C_k(T_k) \in \mathbf{C}$, $\overline{C_k}(\mathbf{X})$ is its cylindrical extension [4] in $\mathcal{U}^q$.

Note that the definitions given above admit infinite domains, implying that the universe of discourse $\mathcal{U}$ may be infinite and that the constraint relations may be infinite, thereby making it possible that the intent of a network may be an infinite relation. In finite-domain networks, the domains and constraints are usually specified extensionally; in networks which contain infinite domains and relations, these domains and relations must, necessarily, be specified intensionally.

Many different forms of constraint satisfaction problem (CSP) have been distinguished. The forms of CSP encountered most frequently in the literature can be defined in terms of a network intent as follows:

Definition 3, The Decision CSP:
Given a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$, decide whether $\Pi^{\mathbf{D},\mathbf{X},\mathbf{C}}$ is non-empty.
Definition 4, The Exemplification CSP:
Given a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$: return nil if $\Pi^{\mathbf{D},\mathbf{X},\mathbf{C}}$ is empty; otherwise, return some tuple from $\Pi^{\mathbf{D},\mathbf{X},\mathbf{C}}$.
Definition 5, The Enumeration CSP:
Given a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$, return $\Pi^{\mathbf{D},\mathbf{X},\mathbf{C}}$.

The Exemplification CSP is the one most commonly addressed by algorithm researchers. Solving the Exemplification CSP is frequently used as a surrogate for solving the Decision CSP. The Enumeration CSP is rarely addressed; it is sometimes solved analytically (in the case of infinite-domain problems) or, in the case of finite-domain problems, by finding all solutions to the Exemplification CSP.

2.1 Interactivity

Most research on constraint processing has focus on autonomous problem-solving by machines. Many real-world applications, however, require interactive decision support rather than automated problem-solving. Consequently, recent constraints research has involved interactive processing. In [3] , for example, Exemplification CSPs are solved by an interactive version of the MAC algorithm, in which search moves are made by a user assigning values to network parameters, while the machine performs constraint propagation (arc-consistency) after each move by the user.

In [2], a more general form of interactive processing was introduced. This was based on a new form of CSP called the Specialization CSP:

Definition 6, The Specialization CSP:
Given a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$: return nil if $\Pi^{\mathbf{D},\mathbf{X},\mathbf{C}}$ is empty; otherwise, return (*i*) a set $\mathbf{A}$ of additional constraints such that $\Pi^{\mathbf{D},\mathbf{X},\mathbf{C} \cup \mathbf{A}}$ contains exactly one tuple and (*ii*) this tuple.

If the given network has an empty intent, the task is, as in the Exemplification CSP, to identify that fact. If the given network has a non-empty intent, the task includes, as in the Exemplification CSP, finding a group of consistent assignments for the network parameters. However, whereas solving the Exemplification CSP involves searching through the space of possible parameter assignments, solving the Specialization CSP involves searching through the space of possible constraints. When the Exemplification CSP is being solved interactively, the user directs the search by inputting parameter assignments (which, of course, are constraints of a restricted form). By contrast, when the Specialization CSP is solved interactively, the user can input $\underline{\text{arbitrary}}$ constraints.

3 Constraint-based Interactive Expert Systems

While not all expert systems interact with human users (process control expert systems, for example, interact with process sensors and actuators), most do. The general scenario is that the expert system should advise the user by determining the value for certain crucial characteristics of a situation affecting the user. This form of decision-support can be accomodated in constraint-based reasoning if we define a new form of CSP, as follows.

Definition 7, The Targeted Specialization CSP:
Given a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$ and a sub-tuple $\mathbf{T}$ of $\mathbf{X}$: return nil if $\Pi^{\mathbf{D},\mathbf{X},\mathbf{C}}$ is empty; otherwise return (i) a set $\mathbf{A}$ of additional constraints such that $\Pi_{\mathbf{T}}^{\mathbf{D},\mathbf{X},\mathbf{C}\cup\mathbf{A}}$ contains exactly one tuple and (ii) this tuple.

An expression of the form $\Pi_{\mathbf{T}}^{\mathbf{D},\mathbf{X},\mathbf{C}}$ denotes the *projection* of a network intent onto the sub-tuple of the network parameters that are in $\mathbf{T}$. This projection is a relation in which each tuple provides an ordered group of values which, if they are assumed by the corresponding paramters in $\mathbf{T}$, will satisfy the constraints. Thus, given a network with a non-empty intent, the task posed by the Targeted Specialization CSP is to find a group of consistent values for the $\underline{\text{targeted}}$ parameters. As in the case of the Specialization CSP, the task of interactively solving the Targeted Specialization CSP involves receiving information from the user and propagating it throughout the network. As in the Specialization CSP, the information from the user may be arbitrary constraints.

3.1 Knowledge Representation in a Constraint-based Expert System

Generic Domain Knowledge In a constraint-based expert system, the domain knowledge embedded in the system consists of the constraints $\mathbf{C}$ in a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$. The intent of this network will be non-empty – otherwise, the constraints would be just an inconsistent set of assertions, rather than a coherent body of expertise about a class of situations that affect the users of the expert system.

Instance-specific Knowledge Information about the problem instance affecting a particular user is represented by a set, $\mathbf{A}$, of additional constraints which are received from the user. A complete description of a problem instance consists, therefore, of $\mathbf{C} \cup \mathbf{A}$ – the generic domain knowledge plus the information about the specific problem instance.

The advice from the expert system to the user consists of the values for the target parameters, $\mathbf{T}$, that are admitted by the constraints in $\mathbf{C} \cup \mathbf{A}$ – that is, the advice consists of the single tuple in $\Pi_{\mathbf{T}}^{\mathbf{D},\mathbf{X},\mathbf{C}\cup\mathbf{A}}$.

3.2 Mixed-initiative acquisition of information

Mixed-initiative interaction is a desirable feature of intelligent programs. Achieving it is the subject of ongoing research – a workshop on the topic was organized at AAAI-99 [1].

In expert systems, supporting mixed-initiative interaction means enabling an appropriate balance between machine-generated questions and user-volunteered facts. Backward-chaining through declarative rules involves machine-driven acquisition of information about a specific problem instance: the system's hypotheses comprise the roots of a set of interlaced trees whose leaves represent possible data points; when considering a hypostheis, the system backward-chains from the root to its leaves, asking questions about these; when a root is found, all of whose leaf nodes receive positive answers from the user, *modus ponens* inference causes the expert system to derive the truth of the hypothesis corresponding to the root. Forward-chaining through declarative rules involves matching user-volunteered facts with leaf nodes and using *modus ponens* to derive a conclusion corresponding to some root. Mixed-initiative interaction with expert systems based on declarative rules involves some mixture of backward- and forward-chaining.

The interactive constraint-based systems which have been reported in the literature involve users volunteering information – in [3], for example, the user volunteers parameter assignments. This, and the fact that inference in constraint-based systems is called constraint propagation, may seem to imply that interaction in constraint-based systems must be user-driven. However, this is not the case. It is possible to use backward-chaining in constraint networks – given a target parameter whose value it must determine, the system can question the user about the immediately neighbouring parameters, or about their neighbours, and so on. However, the very richness of interconnectivity in constraint networks (including, for example, cyclical interdependence of parameters) means that it is much more difficult to decide which parameters to ask the user about – optimal question-generation in interactive constraint-based expert systems is the subject of ongoing research by the author of this paper. While optimal question-generation is still an open research topic, backward-chaining is already being done – a system which questions the user about the parameters neighbouring a target parameter will be illustrated below. Given that both backward- and forward-chaining is possible in interactive constraint-based systems, mixed-initiative interaction simply involves some mixture of the two – the system illustrated below supports mixed-initiative interaction.

4 Constraints and First-order Predicate Calculus

Depending on the nature of the symbols that can appear in their antecedents and consequents, declarative if-then rules are implication statements in either propositional or predicate calculus. Constraints offer a much richer expressiveness than rules because constraints can provide the expressive power of the full first-order Predicate Calculus (PC).

When a first-order language, $\mathcal{L}$, is used to discuss some universe of discourse, $\mathcal{U}$, the constant symbols of $\mathcal{L}$ denote elements in $\mathcal{U}$, the function and relation symbols of $\mathcal{L}$ denote functions and relations over $\mathcal{U}$, while the logic variables are quantified over $\mathcal{U}$. Note that a universe of discourse, being a set of entities, may be either a finite or an infinite set. If a universe of discourse is finite, the Predicate Calculus is not essential – the Propositional Calculus provides sufficient reasoning power.

However, there are other reasons for choosing a notation than just the power of its associated reasoning system – otherwise, there would have been no need for programming language developers to progress beyond machine code. In expert systems, facility of user-input and perspicuity of system-output are just as important as speed of system-internal inference. Thus, even in applications having finite universes of discourse, Predicate Calculus may be preferred to Propositional Calculus, simply because the greater expressive flexibility offered by the Predicate Calculus enables knowledge to be represented in a more naturalistic fashion – just as the father of a large family finds it easier to say that “”all my children have left school” than to say that “Al, Bob, Cait, Dora, ..., Xavier, Yuri and Zeev have left school”.

The discussion that follows will illustrate how, in applications having finite universes of discourse, constraint-based technology offers both programmers and users of expert systems the expressive flexibility of the full Predicate Calculus. It will also show how, in applications having infinite universes of discourse, constraint-based technology offers almost the same flexibility – the only limitation being that quantifiers must be relativized to finite subsets of an infinite universe of discourse[1].

4.1 Constraint Satisfaction as Model Completion

In [9], the current author first discussed the relationship between constraint satisfaction and finding models in Predicate Calculus – Mackworth, in [7], discussed a similar notion, the relationship between finite constraint satisfaction and finding models in the Propositional Calculus.

The predicate calculus approach can be described as follows. Consider some application domain for an expert system. The entities in this application domain comprise a universe of discourse $\mathcal{U}$. To discuss $\mathcal{U}$, a first-order language, $\mathcal{L}$, is defined, with an associated partial model, $\mathcal{M}_p$, which contains an interpretation, in terms of $\mathcal{U}$, for all function and predicate symbols of $\mathcal{L}$ and most, but not all, of its constant symbols.

This language $\mathcal{L}$ can then be used to specify constraint networks, networks in which the set of parameters will be the set of un-interpreted constant symbols of $\mathcal{L}$ and in which the set of constraints will be a set $\mathcal{S}$ of sentences of $\mathcal{L}$. Any sentence of $\mathcal{L}$, provided it references at least one of the un-interpreted constant symbols of $\mathcal{L}$, can be a constraint – even one containing arbitrarily nested quantification[2]. From this perspective, it can be seen that CSPs become

[1] Theoretically, this limitation is not necessary – quantifiers can range of over infinite universes of discourse. In practice, however, the constraint processing algorithms that have been developed so far can process only relativized quantifiers.

[2] Of course, constraint processing algorithms, in particular the type of consistency processing algorithm that is discussed later in this paper, will be more likely to

model-completion problems. For example, the Exemplification CSP becomes the task of computing, if one exists, a total model $\mathcal{M}$ of $\mathcal{L}$ such that $\mathcal{M} \supset \mathcal{M}_p$ and such that all the sentences in $\mathcal{S}$ are true under $\mathcal{M}$, or, if no such model exists, of returning the information that this is so. Similarly, if any model of $\mathcal{L}$ exists which subsumes $\mathcal{M}_p$ and entails $\mathcal{S}$, the Targeted Specialization CSP becomes the task of computing a set $\mathcal{S}_a$ of additional sentences such that every model $\mathcal{M}$, $\mathcal{M} \supset \mathcal{M}_p \wedge \mathcal{M} \models (\mathcal{S} \cup \mathcal{S}_a)$, has the same set of interpretations for the constant symbols in $\mathbf{T}$.

This approach to relating constraints and predicate calculus is quite different from the CLP paradigm [6]. In CLP, a constraint network is treated as a goal (a theorem to be proven); the clauses in a CLP program are not part of the network – they define the semantics of application-specific relation symbols in the query. Because a CLP network is treated as a query, network parameters correspond to existentially quantified logic variables while constraints are restricted to a very limited subset of PC utterances - arbitrary nesting of quantification, for example, is prohibited in CLP. A key advantage of the model-completion approach over CLP is that, since network parameters correspond to constant symbols (albeit symbols in search of an interpretation) rather than logic variables, the parameters can be referenced in multiple sentences; this is because usage of a constant symbol, unlike usage of a logic variable, is not restricted to the lexical scope of a single quantifier symbol within a single sentence. Thus, a parameter which is referenced in sentences that form part of a generic network for an application domain, can also be referenced in sentences input by the user to provide information about his specific problem instance. Thus, this approach is better than CLP at supporting user-interaction.

The greater expressiveness of the model-completion approach (the fact that, subject to the caveats given earlier, arbitrary sentences of $\mathcal{L}$ can be used as constraints) simply reinforces the benefits which are derived from the treatment of parameters as constant symbols. It does, however, mean that, provided the model completion approach is used, constraint-based reasoning offers greater expressiveness for knowledge representation than rule-based reasoning.

4.2 Example

To illustrate the approach, consider a very simple expert system for selecting, from a range of laptop computers, one which will best meet the needs of a user.

Language and Universe of Discourse The universe of discourse comprises the range of available laptops and the real numbers, the latter being present because the user's requirements (regarding CPU speed, RAM and disk space, and machine weight), as well as the corresponding laptop characteristics, and their prices, are numbers. Inspired by a laptop range manufactured by a certain well-known firm, our universe of discourse $\mathcal{U}$ is $\Re \cup \{$c810, c410, c210, c110$\}$,

achieve inferential completeness if either the universe of discourse is finite or all quantifiers are relativized to finite subsets of the universe.

where $\Re$ contains the real numbers while c810 etc. are laptops. Since $\mathcal{U}$ subsumes $\Re$, our language $\mathcal{L}$ contains symbols to discuss the members of $\Re$ - that is, $\mathcal{L}$ contains numerals (constant symbols interpreted to denote members of $\Re$) as well as predicate and function symbols, such as `>=`, `*`, etc., which are interpreted to denote standard relations and functions over $\Re$. Since $\mathcal{U}$ contains the laptops, $\mathcal{L}$ must also include some symbols to discuss these: a unary predicate symbol, `laptop`, whose extension is the set of laptops; some unary function symbols, `speed`, `ramCapacity`, `diskCapacity`, `weight` and `price`, which map from the laptops onto the numbers that are the obvious laptop characteristics; and constant symbols, such as `c110`, `c210`, etc., which are interpreted to denote the obvious laptops. The symbols listed above have their interpretations defined in the partial model $\mathcal{M}_p$ of $\mathcal{L}$. The symbols which are not interpreted in $\mathcal{M}_p$ are the constant symbols `minSpeed`, `minRAM`, `minDisk` and `maxWeight` (which, when they are finally interpreted, should denote the obvious user requirements), and `chosenModel` (which should denote the appropriate laptop).

Generic Domain Knowledge The generic domain knowledge for our expert system can now be expressed as a set of sentence in $\mathcal{L}$. For example, the need for the chosen laptop to satisfy the user's computing requirements could be expressed as a set of three ground atomic sentences

```
speed(chosenModel) >= minSpeed.
ramCapacity(chosenModel) >= minRAM.
diskCapacity(chosenModel) >= minDisk.
```

while the weight requirement could be expressed as

```
weight(chosenModel) =< maxWeight.
```

Suppose we wish to specify that the cheapest laptop which meets the user's computing requirements must be chosen. Given that the only money-related symbol we have is the function `price`, how do we represent the notion of "cheapest"? A laptop is "cheapest" if there is no other laptop whose price is less. Thus, our specification above can be expressed in $\mathcal{L}$ as:

```
not exists X :
  ( laptop(X) and
    price(X) =< price(chosenModel) and
    speed(X)>=minSpeed and
    ramCapacity(X)>=minRAM and
    diskCapacity(X)>=minDisk ).
```

This PC sentence can be paraphrased in English as "there should not exist any laptop which is cheaper than the chosen one and which also satisfies the computing requirements".

Note that the universe of discourse for this application is infinite – it subsumes $\Re$. However, the quantification in the constraint just given is relativized to a finite subset of the universe – to the set of laptops. This relativization means that the sentence is equivalent to a ground sentence, one in which all the available laptops are referenced by name. While this equivalance is what makes the sentence tractable[3] as a constraint, the freedom to use a quantifier in expressing the constraint makes for easier knowledge management – just like the father mentioned earlier, who often finds it easier to refer to his large family collectively instead of listing them individually by name.

In this simple application domain, a constraint using heavily nested quantification would be very contrived. However, if we extended the scope of the application to include, say, power supply systems around the world, then we might need a certain degree of quantifier nesting – in a constraint referring to worldwide rechargability, for example, one quantifier might range over the laptops while another ranged over the power supply systems used in various parts of the world.

Partial Model An implementation of the model-completion approach to constraint-based reasoning has been built by the author of this paper. An expert system for laptop selection built on top of this implementation consists of the above five sentences plus some statements which specify (a) the application-specific language $\mathcal{L}$ to which the sentences belong and (b) a partial model $\mathcal{M}_p$ for $\mathcal{L}$. It is assumed that every universe of discourse subsumes $\Re$, so the implementation provides a set of pre-defined symbols for discussing $\Re$ and a pre-defined interpretation for all these symbols. Thus, in specifying a universe, a language and a partial model for an application, all that needs to be done is to specify the application-specific material. In an expert system for laptop selection, the statements needed to provide the application-specific detail for $\mathcal{L}$ and $\mathcal{M}_p$ are as follows:

```
domain laptop =::= {c810,c410,c210,c110}.
function speed(laptop) -> number
    =::= { c810->1.2, c410->0.9, c210->0.6, c110->0.57 }.
function ramCapacity(laptop) -> integer
    =::= { c810->1024, c410->1024, c210->256, c110->256 }.
function diskCapacity(laptop) -> integer
    =::= { c810->68, c410->40, c210->20, c110->15 }.
function weight(laptop) -> number
    =::= { c810->3.2, c410->2.9, c210->3.1, c110->2.7 }.
function price(laptop) -> integer
    =::= { c810->3299, c410->2599, c210->1799, c110->1439 }.
chosenModel : laptop.
minSpeed : positive number.
minRAM : positive number.
minDisk : positive number.
maxWeight : positive number.
```

[3] To currently developed algorithms, at least.

These statements can be explained as follows. An application-specific universe of discourse consists of the union of $\Re$ with the application-specific domains. The example expert system needs only one such domain, containing the laptops; it is declared in the first statement above. This statement also introduces several symbols into $\mathcal{L}$: a unary predicate symbol, `laptop`, and four constant symbols, `c810`, `c410`, `c210` and `c110`. The statement implicitly defines the interpretation of these symbols: the constant symbols are interpreted as the obvious members of $\mathcal{U}$ while the unary predicate symbol is interpreted as denoting the set of laptops. Each of the subsequent five statements introduces an application-specific unary function symbol into $\mathcal{L}$ and defines its interpretation. Each of the final five statements introduces a constant symbol into $\mathcal{L}$. These symbols are not interpreted, although the space of possible interpretations for each symbol is restricted to a subset of $\mathcal{U}$; `chosenModel`, for example, must denote a laptop while `minSpeed` must denote a member of $\Re^+$.

Network The constraint network defined by the foregoing contains five constraints (one for each of the five sentences of generic domain knowledge specified above) and five parameters (one for each constant symbol that is un-interpreted in $\mathcal{M}_p$).

The parameters are: `chosenModel`, `minSpeed`, `minRAM`, `minDisk` and `maxWeight`. The domain of the parameter `chosenModel` is {c810, c410, c210, c110}, while each of the other parameters has the domain $\{X|X > 0\}$.

Four of the constraints are binary – those corresponding to the sentences of generic domain knowledge which specify that the chosen model must satisfy the user's computing and weight requirements. The fifth constraint, that corresponding to the sentence of generic domain knowledge which specifies that the cheapest satisfactory laptop must be chosen, involves five parameters.

The constraint relations corresponding to the sentences of generic domain knowledge can be computed from the interpretations, in $\mathcal{M}_p$, for the interpreted symbols in each sentence. For example, consider the sentence

```
speed(chosenModel) >= minSpeed.
```

This is a binary constraint – it contains two constant symbols, `chosenModel` and `minSpeed`, and, since neither of them is interpreted in $\mathcal{M}_p$, they are both parameters in the constraint network. The constraint relation which corresponds to this sentence can be computed from the pre-defined interpretation for the standard predicate symbol `>=` and the interpretation of the application-specific function symbol `speed`. Remember that the interpretation of the function symbol `speed` was defined above as follows:

```
function speed(laptop) -> number
=::= { c810->1.2, c410->0.9, c210->0.6, c110->0.57 }.
```

Thus, the sentence `speed(chosenModel) >= minSpeed` means that if `chosenModel` had the value c810, then 1.2 should be greater than or equal to the value of `minSpeed`; similarly, if `chosenModel` had the value c410, then 0.9 should be greater than or equal to the value of `minSpeed`; and so on. Thus, the sentence `speed(chosenModel) >= minSpeed` corresponds to the following constraint relation on the parameter pair ⟨`chosenModel`, `minSpeed`⟩:

$$\{\langle X,Y\rangle | (X = c810 \wedge Y \leq 1.20) \vee (X = c410 \wedge Y \leq 0.90) \vee (X = c210 \wedge Y \leq 0.60) \vee (X = c110 \wedge Y \leq 0.57)\}$$

How are quantified sentences handled? Instead of considering the quantified sentence which specifies that the cheapest satisfactory laptop must be chosen, we will consider the following shorter sentence, which specifies that the maximum possible amount of RAM is needed:

```
not exists X : (laptop(X) and ramCapacity(X)>minRAM).
```

Its meaning is computed as follows. First, the negation is moved inside the quantifier, so that the sentence becomes

```
all X : (laptop(X) implies ramCapacity(X) =< minRAM).
```

Then, because the `laptop` domain is finite, the effect of this sentence can be achieved by iterating over the domain – in essence, treating the sentence as if it were

```
ramCapacity(c810) =< minRAM and ramCapacity(c410) =< minRAM and
ramCapacity(c210) =< minRAM and ramCapacity(c110) =< minRAM.
```

Remember that the function symbol `ramCapacity` was defined earlier as follows:

```
function ramCapacity(laptop) -> number
    =::= { c810->1024, c410->1024, c210->256, c110->256 }.
```

This means that the constraint relation corresponding to the above sentence is the following unary constraint on the parameter `minRAM`: $\{X|X \geq 1024\}$.

At this stage, it may be appropriate to consider quantification over infinite domains. Consider the sentence:

```
not exists X : X > minRAM.
```

The system being described here can accept this sentence, although it will not be able to do anything useful with it. Recognizing that the quantifier is not relativized to a finite subset of the universe of discourse, it will not attempt to use iteration; instead, it will treat the above sentence as the following unary constraint on the parameter `minRAM`: $\{X|\neg\exists Y(Y > X)\}$. As we shall see below, the basic operation of the system involves arc consistency processing, using an approach in which term re-writing is used to process to infinite constraint relations. The basic difficulty with the above sentence lies in the fact that nothing in the set of term-rewriting rules used (so far) can do anything useful with the above intensional formula – therefore, the system cannot guarantee to achieve total arc consistency in a network containing such a constraint relation[4]. Possibly, however, one could imagine an augmented set of rules which could do more. Nevertheless, it will always be possible, given any set of re-writing rules, to devise a set of constraints which are beyond the inferential competence of the rules.

Constraint Processing The basic operation of the run-time system consists of applying (hyper-)arc consistency to the constraints in the network, constraints whose relations may be either finite (and represented extensionally) or infinite (and represented intensionally). Finite domains are pruned by removal of inconsistent values. An infinite domain is pruned in two stages: first, a conjunct is added to the intensional formula that specified the domain before pruning; then, term rewriting is used to simplify the extended intensional formula.

Consider, for example, the following constraint relation on the parameter pair $\langle$`chosenModel`, `minSpeed`$\rangle$:
$\{\langle X, Y\rangle | (X = c810 \wedge Y \leq 1.20) \vee (X = c410 \wedge Y \leq 0.90) \vee$
$(X = c210 \wedge Y \leq 0.60) \vee (X = c110 \wedge Y \leq 0.57)\}$
Initially, the domain of `chosenModel` is {c810, c410, c210, c110} and that of `minSpeed` is $\{X|X > 0\}$. When we use arc-consistency on the arcs of the above constraint, the domain of `chosenModel` is unchanged, but the intensional formula for the domain of `minSpeed` undergoes the following changes. First it becomes

$$\{X|X > 0 \wedge (X \leq 1.2 \vee X \leq 0.9 \vee X \leq 0.6 \vee X \leq 0.57)\}$$

in which the intensional formula has been expanded, to include a conjoined expression which captures the impact of the constraint on the set of possible values for `minSpeed`. Term rewriting then changes this formula, initially reducing it to

$$\{X|X > 0 \wedge (X \leq 1.2)\}$$

and then rewriting it further to

$$\{X|0 < X \leq 1.2\}.$$

When the constraints representing the generic domain knowledge have all been processed (as they would be before any user input were accepted), the domains of `minRAM`, `minDisk` and `maxWeight` would have been reduced to $\{X|0 < X =< 1024\}$, $\{X|0 < X =< 68\}$ and $\{X|X >= 2.7\}$, respectively.

Mixed-initiative Interaction The approach supports mixed-initiative interaction. A user interacting with this expert system can volunteer instance-specific

[4] Strictly speaking, of course, the system can achieve "'total" arc consistency, by propagating this intensional formula through all relevant parts of the network. In practice, of course, *useful* arc consistency involves producing simplified intensional formulae – for example, if a formula involves a contradiction, useful arc consistency would involve detecting this and inferring that the parameter whose domain is described by the formula is the empty set.

data by asserting appropriate sentences in $\mathcal{L}$ – each such sentence must, of course, refer to at least one of the un-interpreted symbols. For example, if the user knows that he needs a CPU speed of at least 0.7 gigahertz, he could assert

```
minSpeed = 0.7.
```

Similarly, if he wants the largest possible amount of RAM, he could assert

```
not exists X : (laptop(X) and ramCapacity(X)>minRAM).
```

Each such sentence[5] extends the constraint network defined in the expert system, by adding a new constraint. After each new constraint is asserted by the user, its arcs are entered into the queue of arcs maintained by the hyper-arc consistency algorithm. As usual in arc consistency algorithms, if the domain of any parameter is reduced by any arc of this new constraint, the other constraint arcs referencing this parameter are appended to the queue. The algorithm reaches quiescence when the queue becomes empty again, at which point the user can volunteer another assertion or retract one of his previous assertions – dependency records are maintained to facilitate such retractions. (The dependency records are also used to generate explanations when requested by the user.)

Suppose, for example, the user asserted `minSpeed=0.7`. Constraint propagation would result in the domain of `minSpeed` being restricted to $\{0.7\}$. Since this parameter is also referenced in `speed(chosenModel) >= minSpeed`, the arcs of this constraint would be appended to the queue. The effect of this is that the domain of `chosenModel` is reduced to $\{c810, c410\}$, because the other laptops are not fast enough. This change would then activate the other constraints which reference this parameter. By the time that quiescence is reached, the domain for `maxWeight` would have been reduced from $\{X|X >= 2.7\}$ to $\{X|X >= 2.9\}$. If the user were then to assert `not exists X : ( laptop(X) and ramCapacity(X) > minRAM )`, that would reduce the domain of `minRAM` from $\{X|0 < X =< 1024\}$ to $\{1024\}$ but would not reduce further the domain of any other parameter.

Instead of volunteering instance-specific data, the user could hand the initiative over to the machine by specifying that he wishes it to help him determine an appropriate value for `chosenModel`. In doing so, he would be creating a Targeted Specialization CSP in which $\mathbf{T}$, the set of targeted parameters, would be

[5] One reviewer asked whether a "real" user would be able to construct predicate calculus utterances like this one. Several comments must be made in response. First, the point being made at this stage in the paper is that the computational model being described in the paper is capable of accepting this kind of information from a user – if the user is capable of giving it. Second, some "real" users can use predicate calculus notation. Third, for those users who cannot use predicate calculus, an interface may be capable of translating from the user's preferred mode of expression into predicate calculus. The author is actually involved in a research project aimed at building an interface which will be capable of translating from natural language utterances into constraints expressed as PC sentences. Such translation is not a trivial task – nevertheless, the main point is that, regardless of whether all "real" users are capable of using PC, the computational model described here is capable of accepting PC sentences as input.

{`chosenModel`}. The machine would then ask the user for information about the other parameters in the network, each reply being treated as another constraint to be added to the network: a user's reply need not specify a value for the parameter which was the subject of the machine's question; it could, instead, be an arbitrary sentence involving the parameter – for example, a user asked about the required disk space could reply that he wants ten times as much disk space as RAM, by entering the sentence `minDisk = 10 * (minRAM/1000)`, division by 1000 being involved because RAM is specified in megabytes while disk space is specified in gigabytes.

5 Improved Inference

It has been shown how the use of constraints supports richer knowledge representation than that available to rule-based expert systems – constraints extend both the range of domain expertise that can be expressed and the range of instance-specific data that users of an interactive expert system can provide. However, it should also be pointed out that constraints also support richer inference.

Inference in rule-based expert systems is based on *modus ponens.* That is, we can have inferences of the form

$$(A \Rightarrow B, A) \quad \vdash \quad B.$$

Constraint propagation, however, subsumes both *modus ponens* and *modus tollens.* That is, it can also make inferences of the form

$$(A \Rightarrow B, \neg B) \quad \vdash \quad \neg A.$$

Suppose, for example, that we have an expert system in which there are two parameters, `length` and `width`, that must assume values from $\Re^{+}$. Suppose that one piece of domain knowledge is `length >= 1000 implies width >= 500`. Suppose that, while backward-chaining through some other piece of domain knowledge, the expert system asks the user for the value of `width` and receives the response that `width = 400`. A rule-based expert system would not be able to deduce from this reply, and from the domain knowledge just given, that `length` must be less than `1000`.

Now suppose that the expert system is constraint-based rather than rule-based. Initially, the domains of `length` and `width` would both be $\{X|X > 0\}$. According to the standard semantics of material implication in logic, the constraint relation corresponding to `length >= 1000 implies width >= 500` is the union of two infinite sets of tuples, represented by the two disjuncts in this intensional formula:

$$\{\langle X, Y\rangle | X < 1000 \vee (X \geq 1000 \wedge Y \geq 500)\}.$$

The first of these two sets corresponds to the case where the antecedent of the implication is not satisfied, the second to the case where it is. When the user replies that `width = 400`, this is treated as a unary constraint whose relation is $\{400\}$. Propagating this constraint would, first, reduce the domain of `width` from $\{X|X > 0\}$ to $\{400\}$ and then activate the constraint `length >= 1000 implies width >= 500`. The fact that the domain of `width` is now $\{400\}$ means that the second set in the union comprising the semantics of the constraint is irrelevant. Thus, arc consistency means that the first set can be projected onto the domain of `length`. Thus, the domain of `length` is reduced from $\{X|X > 0\}$ to $\{X|0 < X < 1000\}$. In other words, applying arc consistency to the constraint relation $\{\langle X, Y\rangle|X < 1000 \vee (X \geq 1000 \wedge Y \geq 500)\}$ on the parameters $\langle$ `length`, `width` $\rangle$, in a context where the domain of `width` is $\{400\}$, achieves the same effect as applying *modus tollens* to the following premises: `length >= 1000 implies width >= 500` and `width=400`.

6 Conclusions

The standard way of relating constraint-based reasoning to the Predicate Calculus (PC) is Constraint Logic Programming (CLP), in which constraints are integrated into a proof-theoretic approach to logic. This paper advocates an alternative, model-theoretic approach. In this approach, the task of solving a constraint satisfaction problem becomes that of completing a partial model for a first-order PC language. It has been shown that, in this context, constraint propagation algorithms provide an inference capability which subsumes the effects of both *modus ponens* and *modus tollens*. Thus, if constraints replace declarative rules in the construction of expert systems, two benefits follow: more expressive knowledge representation and more powerful inference.

7 References

1. AAAI, 1999, *Proc. AAAI-99 Workshop on Mixed-Initiative Intelligence.*
2. Bowen J, 2001, "The (Minimal) Specialization CSP: A basis for Generalized Interactive Constraint Processing", *Proc. CP-2001 Workshop on User-Interaction in Constraint Processing.*
3. Freuder E, Likitvivatanavong C and Wallace R, 2000, "A Case Study in Explanation and Implication", *Proc. CP-2000 Workshop on Analysis and Vizualization of Constraint Programs and Solvers.*
4. Friedman G and Leondes C, 1969, "Constraint Theory, Part I: Fundamentals", *IEEE Transactions on Systems Science and Cybernetics*, ssc-5, 1, 48-56.
5. Jackson P, 1999, *Introduction to Expert Systems*, 3 rd. Edition, Addison Wesley Longman.
6. Jaffar J and Lassez J, 1987, "Constraint Logic Programming", *Proc. POPL-87.*
7. Mackworth A, 1992, "The Logic of Constraint Satisfaction", *Artificial Intelligence*, 58, 3-20.
8. Rasmus D, 2000, "Knowledge Management Trends: The Role of Knowledge in E-Business", *PCAI Magazine*, 14(4), Special issue on Knowledge Management, Expert Systems and E-Business.
9. Bowen J and Bahler D, 1991, "Conditional Existence of Variables in Generalized Constraint Networks", *Proc. AAAI-91.*