

In A. Artiba and S.E. Elmaghraby (eds.),
The Planning and Scheduling of Production Systems: Methodologies and Applications,
London: Chapman and Hall, 1996.

Constraint Logic and Its Applications In Production: An Implementation Using the Galileo4 Language and System*

Dennis Bahler
bahler@ncsu.edu
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206 USA

and

James Bowen
jabowen@jim.ucc.ie
Department of Computer Science
National University of Ireland
Cork, Ireland

February 1996

Abstract

The flexible and efficient support of the process of product design and production requires combined expertise from many aspects of the product life-cycle, and the management and use of this knowledge in turn increasingly requires sophisticated computer-based advice systems. In this chapter we present a constraint-based approach to making the full First-Order Predicate Calculus available for knowledge representation purposes in intelligent networked colocation advice systems for concurrent engineering. Domain knowledge is represented as a theory written in a first-order logical language. The set of models of the language under which the theory is satisfied is the set of solutions of a constraint network. Solutions to the network are computed by a hierarchical and opportunistic constraint inference method. Galileo4, an implemented language and companion run-time system based on these ideas is discussed and its efficacy is demonstrated by presenting applications in which high problem-solving versatility is provided by concise and readable programs.

1 Introduction

Concurrent engineering (CE) is an approach to the organization and management of design, production, marketing, etc. in which, in order to avoid costly downstream redesign, aspects of the product life cycle such as manufacturability, testability, repairability, and marketing are taken into account as early in the design process as possible. Since most life-cycle costs are dictated in the

*This work was partially supported by the National Science Foundation under Grants DDM-8914200 and DDM-9215755 and by IBM Corp.

design phase, it is important that life-cycle information be available to the designer so that it can influence product design. Once a product has completed the design stage, it is too late to make significant changes in life-cycle costs. The U.S. National Science Foundation, for example, has estimated that 70% or more of a product's manufacturing cost is dictated by design decisions.

Currently many companies attempt to implement concurrent engineering at the detailed design phase by maintaining checklists or rulebooks that a designer must adhere to and then scheduling face-to-face meetings with the appropriate experts at critical project points. In our interaction with companies we have found several problems with this approach. First, the rulebooks or checklists are getting so large and complex that even the experts, let alone the designer, cannot help letting design deficiencies through to manufacture. Second, the need for physical meetings reduces time-to-market. One way to overcome these difficulties is to use **Networked Colocation**. In this approach, the team members supplement face-to-face meetings with electronic communication over a computer network.

The concept of networked colocation can be realized with varying degrees of sophistication. At its simplest, it may amount to no more than some combination of electronic mail and shared access to a CAD database. However, something much more sophisticated is needed if we are to succeed in CE. Product development teams often find themselves overwhelmed by the volume and variety of information that arises in CE. As the design evolves, the number of concerns grows quickly beyond what even a team of persons can successfully manage. We need software which can relieve some of this cognitive overload. What is needed is an **Intelligent Networked Colocation Advisor** (INCA) which not only relieves the logistic and scheduling difficulties but also reduces the problem complexity perceived by team members. These design advice tools are intended for use by and for multiple clients, including both computer systems and human users.

In our experience [3, 4, 5] constraint-based systems constitute a good basis for such tools; much of our effort has emphasized expressive competence in constraint-based languages in order to provide a convenient and sufficient representation of the kind of real-world problems encountered in concurrent engineering. Moreover, we believe that an approach to constraint processing which is strongly grounded in a formal logical model offers the greatest chance to gain insight and make progress. Accordingly, we view a network of interlocking constraints as a collection of sentences in First-Order Predicate Calculus (FOPC). Constraint processing then becomes exactly the process of semantic modeling, that is, finding an interpretation for the symbols appearing in the theory that renders the constraint sentences true according to some well defined measure of truth. In this way the notions of constraint logic and constraint network are mutually intertwined.

The remainder of this chapter is organized as follows. In Section 2, we briefly present an overview of Galileo4, the latest of a set of constraint programming languages and companion runtime systems we have developed for constructing INCAs, and illustrate the utility of this system for supporting a production task application in design for testability of printed wiring boards. In Section 3, we further illustrate the practical utility of the constraint-based approach by discussing another application, this time in the realm of selecting lowest-cost insertion robot equipment while fulfilling a set of user-supplied specifications. In Section 4, we compare our approach with other constraint-based techniques.

Beginning in Section 5, we elaborate in a more technical vein on the relationship between constraint networks and semantic models in logic. In this view the consistency of a constraint network corresponds to the satisfiability of a theory in logic, and we argue that a suitable computational approach to the full FOPC can be based on generalizing the consistency algorithms [21] that have been presented in the literature on constraint satisfaction problems (CSPs). In Section 6, we look at Galileo4 from a more formal perspective, using a progression of sample programs to illustrate the correspondence between constraint processing and semantic modeling. In Section 7, we discuss

human-machine interaction in Galileo4 in terms of a model of machine-interlocutor interaction.

Finally, in Section 8, we make some concluding remarks.

2 The Utility of Constraint Processing for Production

Galileo4 is a language and run-time system we have developed for the construction and use of INCAs. In this section, we will consider a Galileo4 program called KLAUS2, which is an INCA for printed wiring board design. In this section, we will consider only a few extracts from KLAUS2 (Figure 1). For a more complete presentation of KLAUS2, see [4].

A program in Galileo4 specifies a frame-based constraint network and comprises a set of declarations and definitions. The declaration statements include declarations of the parameters that exist in the network and the constraints that exist between these parameters. The definition statements include definitions for: partitions which divide the network into different regions that may be seen by different classes of user; application-specific domains, functions, and relations that are used in constraint declarations; and icons for application-specific domains, if it is intended to present certain regions of the network to users graphically rather than textually, a textual presentation being the default.

In a Galileo4 program, statements can be written in any order, although good software engineering practice dictates that some agreed standard ordering be used, in order to facilitate program maintenance. In Figure 1, we have taken advantage of this free-ordering by listing the statements in an order which facilitates discussion in this chapter; in KLAUS2 itself, however, the statements are ordered differently. Although the statements are numbered in Figure 1, this is purely for the purposes of the current exposition; the numbers are not part of the program text. The discussion that follows will be very dense, because of the need to illustrate many features of a richly expressive language in a very short space. Reviewers wanting a more leisurely overview of the language are encouraged to read [4]. Galileo4 is a research prototype which is available on an as-is basis. The current version runs on IBM PS/2's and compatible hardware.

Frame-based Product-Process-Organization Modeling: In a frame-based constraint network, frames can be used to represent a model of Product, Process, and Organization (PPO). Consider, for example, statement 1 of Figure 1 which declares the existence of a parameter called `the_test_facility` and specifies that it is of type, or domain, `test_facility`. The string delimited by apostrophes is a long synonym which specifies the name by which the parameter will be referenced in all output given to users of the program. From this, it can be seen that this parameter represents `the facility where the board will be tested`.

This parameter is a frame, as can be seen in statement 2, which defines the application-specific domain `test_facility` referenced in statement 1. The statement specifies that a test facility is represented by a frame containing three slots. For each slot, we specify its name, its long synonym and its domain. In output given by the system to users, the long synonyms of the slots in a frame are concatenated with the long synonyms of the frame-valued parameters.

The constituent slots of a `test_facility` frame, as specified in statement 2, have scalar application-specific domains. These are defined in statements 3, 4 and 5. Notice that, while scalar domains may be defined extensionally by listing each possible value (statement 3), they may also be defined intensionally by specifying an appropriate logical formula (statements 4 and 5). (Notice the use of long synonyms in statement 3; internally, the system will use values such as `erdsys`, but at run-time the user will see names like `Erdsys TX`.)

```

1  'the facility where the board will be tested'(the_test_facility) : test_facility.
2  domain test_facility ::=
    ('the equipment at'(name) : tester,
     'the maximum clock frequency testable at'(maxfreq) : frequency,
     'the maximum number of test points testable at'(maxtpoints) : counter).
3  domain tester ::= {'Dynamax Tester Mark I'(dynatest1),
                     'Dynamax Tester Mark II'(dynatest2), 'Erdsys TX'(erdsys)}.
4  domain frequency ::= {X : number(X) and 3 <= X <= 40}.
5  domain counter ::= {X : integer(X) and X >= 0}.
6  domain component ::= ('the type of'(type) : component_type,
                        'the number of leads on'(nleads) : counter, 'the power consumed by'(power) : positive number).
7  domain component_type ::= {processor,dsp,comm_controller,divider,
                             crystal,resistor,capacitor,diode}.
8  domain 'crystal oscillator'(osc_crystal) ::=
    {X : component(X) and X.type=crystal and
     exists(X.'the oscillation frequency of'(freq) : positive number)}.
9  domain 'frequency divider'(divider) ::= {X:component(X) and X.type=divider and ... }.
10 domain cpu ::= {X:component(X) and X.type=processor and ... }.
11 domain resistor ::= {X:component(X) and X.type=resistor and ... }.
12 all X : component(X) implies X.nleads in !lst_tpoints.
13 num_tpoints = sum(lst_tpoints).
14 num_tpoints <= the_test_facility.maxtpoints.
15 'the numbers of test points required by components'(lst_tpoints) : bag of counter.
16 'the total number of test points on the board'(num_tpoints) : counter.
17 not(exists X : tester(X) and cost(X) < cost(the_test_facility.name) and satisfactory(X)).
18 function 'the total cost per hour of'(cost) ::= {X -> Y : 1.8 * price(X)}.
19 function 'the direct cost per hour of usage of'(price) ::= datafile(qdb,TSTRCSTS).
20 relation 'capable of testing the board'(satisfactory) ::=
    {X : num_tpoints <= max_tpoints(X) and (all Y : osc_crystal(Y) implies Y.freq <= max_clock(X))}.
21 function 'the maximum number of nails usable by'(max_nails) ::= datafile(qdb,TSTRNALS).
22 function 'the clock limit for'(max_clock) ::= datafile(qdb,TSTRCLKS).
23 all X : component(X) and X.type in {processor,comm_controller,dsp}
    implies exists(X.'the oscillator for'(crystal) : osc_crystal) and
        exists(X.'the pullup resistor for'(pullup) : resistor).
24 all X : osc_crystal(X)
    implies X.freq <= the_test_facility.maxfreq else
        exists(X.ancillary_circuit : divider).
25 'the power consumptions of the components'(lst_powers) : bag of positive number.
26 'the total power consumed by the board'(total_power) : positive number.
27 all X : component(X) implies X.power in !lst_powers.
28 total_power = sum(lst_powers).
29 'the goal power consumption'(power_goal) : positive number.
30 'the maximum acceptable power consumption'(max_power) : positive number.
31 'the power consumption discrimination increment'(power_inc) : positive number.
32 total_power <= (power_goal to max_power step power_inc).
33 no_tpoints < 200.
34 hardness(no_tpoints < 200) = 0.9.
35 field 'the circuit designers perspective'(configuration) ::= {X : component(X)}.
36 field 'the perspective taken by test engineers'(testability) ::= {the_test_facility}.
37 permission({X:component(X)},configuration).

```

Figure 1: Extracts from KLAUS2, a Galileo4 program

Frame-based inheritance is illustrated in statements 6, 7, 8, 9, 10 and 11. Statement 6 defines the frame domain `component`; for simplicity, we give only three slots, although in KLAUS2 this frame has many more slots. Notice, from statement 7, that there can be several different types of component. By using inheritance, frames for these different types of component can be defined in terms of the generic component frame. In statement 8, for example, it is specified that an `osc_crystal` frame, which is used to represent a crystal oscillator, is a `component` frame which has the value `crystal` in its `type` slot and which also has an additional slot `freq` that represents the frequency at which the crystal oscillates. In statement 9, a `divider` frame, which is used to represent a frequency divider, is specified to be a `component` frame in which the `type` slot has the value `divider` and which has several other slots – these additional slots are not specified here, however. A `cpu` frame is defined similarly in statement 10, and a `resistor` frame in statement 11.

Using Constraints to Represent Life-Cycle Interdependencies: In a frame-based constraint network, constraints can be used to represent the life-cycle interdependencies that exist between parts of the PPO model. However, because of the large variety of life-cycle interdependencies that can exist in different product domains, the constraint language must be richly expressive. Knowledge representation research in Artificial Intelligence indicates that “for ... versatile [systems], the language in which declarative knowledge is represented must be at least as expressive as first-order predicate calculus” [26]. We consider the discovery that constraint networks could provide the full expressive power of the predicate calculus to be an important conceptual breakthrough to arise from our work. Thus, in Galileo4, any sentence in first-order predicate calculus is a well-formed constraint. (The language is not restricted to classical logic, as we will see below.)

Consider, for example, statement 12 in Figure 1. This is a universally quantified constraint which specifies that the number of test points required by each component on the board (which, for our purposes, we take to be equal to the number of leads on the component) must be recorded in a list. This list is referenced in statement 13, which states that the total number of test points in the circuit being designed is the sum of the numbers of test points required by the individual components. Statement 14 then expresses an important life-cycle interdependency between this total number of test-points and the intended test facility. It is interesting to note that, because of the natural language synonyms given in statements 1, 2 and 16, the constraint in statement 14 would be expressed to KLAUS2 users (in justifications, for example) as follows: `the total number of test points on the board must be less than or equal to the maximum number of test points testable at the facility where the board will be tested.`

The two parameters `lst_tpoints` and `num_tpoints` referenced in statements 12 and 13 are declared in statements 15 and 16. Statement 15 illustrates another feature of the language – its ability to handle set-valued parameters. Indeed, the constraint uses a special kind of set, a *bag* or multiset, that is, a set in which multiple copies of the same member are treated as distinct. This is needed in order to deal with the fact that different components can have the same number of leads.

While the life-cycle interdependency expressed in statement 12 required universal quantification, statement 17 shows a constraint which requires the other form of quantification in logic, that is, existential quantification. Statement 17 specifies that the cheapest tester available should be used, by specifying that there should not be any tester available which is capable of testing the board and which costs less than the chosen tester. This constraint is worth further discussion, because it illustrates usage of other features of Galileo4 that are available only because the language provides the full expressive power of the predicate calculus. This statement shows that while Galileo4 provides generic predicates such as `<` and generic functions such as `*`, it also allows application-specific functions and predicates to be used.

If we are to use application-specific functions and predicates, their meanings must be defined. In Galileo4, this can be done using either of the notions of extensional or intensional definition from set theory. Consider, for example, statement 18, which defines the meaning of the function `cost`. The meaning of a function is a set of mappings from inputs to outputs. In statement 18, the meaning of `cost` is defined intensionally; the total cost of using a tester is 1.8 times the direct cost of using the tester, which is denoted by the function symbol `price`. There is a finite number of possible testers, so the meaning of the `price` function is a finite set which can be defined extensionally.

Galileo4 allows extensional set definitions to be given either in the program text or in an external database file. Statement 19, for example shows that the meaning of the function `price` is defined by specifying that the set of pairs of values is in the database file TSTRCSTS. (We have found in our application experiments that tying function and predicate definitions to database files is a very natural way of linking INCA systems to corporate databases.)

As with a function, the meaning of a predicate is also a set, which can also be defined either extensionally or intensionally. In statement 20, we define the meaning of the predicate `satisfactory` by specifying an intensional formula which uses universal quantification and two application-specific function symbols whose meanings are defined in statements 21 and 22.

Non-Parametric Design: In parametric design, the overall architecture of the product and its life-cycle have already been determined and the task is merely one of establishing appropriate values for the parameters of this architecture. Concurrent engineering is not this simple. Parametric design must be accompanied by what is sometimes called componential design, in which the structure of the product and/or its life-cycle environment themselves are determined.

The belief is still widespread that constraint networks are incapable of addressing non-parametric design. However, another contribution of this research was the discovery that this limitation could be overcome by incorporating into constraint processing theory the notion of conditional existence from free logic [19]. This enables a constraint processing inference engine to deduce that, when certain conditions are true, additional parameters must be introduced into a constraint network. This was a fundamental discovery, since it enables a constraint-based CE system to reason about when to introduce new elements into a product or life-cycle architecture. For further information on the scientific basis for using free logic in constraint networks, see [2].

Consider, for example, statement 23. This is a universally quantified constraint which also uses the notion of conditional existence from free logic to specify that if parameter of domain `component` is used to represent a CPU, a communications controller or a digital signal processor device, then the parameter must have an extra slot to represent the oscillator which drives the device and must have a further slot to represent a pullup resistor. The `exists` tokens in this constraint are free logic existence specifiers, not existential quantifiers.

A more interesting usage of free logic appears in statement 24, which also uses modal logic. The `else` connective in this constraint comes from modal logic. The constraint specifies that, ideally, every crystal should oscillate at a frequency which does not exceed the maximum clock speed that is testable by the test facility. However, it then goes on to say that if this is not possible, then any crystal which oscillates at a faster frequency must have an ancillary divider circuit. Here, we see the constraint network extending itself by introducing a new parameter when a certain condition arises. This new parameter represents a new component, the necessity of whose existence has been inferred by the system.

Optimization: Statement 17 used existential quantification to require that the cheapest possible tester be used. Since there is a finite number of possible testers, this statement illustrated opti-

mization of a parameter which ranged over a discrete domain. Galileo4 also supports optimization of parameters which range over infinite domains, provided these domains are discretized into finite numbers of equivalence sets.

Consider, for example, statement 32. This specifies that the total power consumed by the board (see statements 25 through 28) should, ideally, not exceed the goal power consumption (statement 29), that it should certainly not exceed the maximum acceptable power consumption (statement 30), and that between those two values the optimization tolerance is equal to a value called the power consumption discrimination increment (statement 31).

Prioritization: By default, all constraints in a Galileo4 program are treated as being equally important, and are treated as hard constraints, that is, as constraints that must be satisfied. However, we can also specify that some constraints in a program are soft. The meaning of a constraint being soft is that the constraint should, if possible, be satisfied but, if there ever arises a situation in which violating the constraint would relieve an over-constrained situation, then it may be ignored. We can have as many soft constraints as we want in a Galileo4 program and can assign them different levels of hardness or priority. Constraint hardness is a number in $[0,1]$, with 1 being the default and 0 being the hardness of a constraint that has been disabled completely. Statement 34 is a second-order constraint which specifies that the hardness of the first-order constraint in statement 33 is 0.9.

Multiple Perspectives and Interfaces: Galileo4 enables constraint networks to be divided into (possibly overlapping) regions called **fields of view**. A field of view is that region within a constraint network that is currently important to a user interacting with the network. A field of view can be either global or local. The global field of view consists of the entire constraint network. A local field of view contains only a subnetwork. Each field of view contains all the parameters that are of interest to the user, as well as all constraints which reference these parameters.

We define a field of view by specifying the set of parameters which it contains. In Figure 1, for example, we define two of the fields of view that are provided by the KLAUS2 application. Statement 35 defines a **configuration** field of view, which will be seen by a circuit designer, and specifies that it contains the set of all parameters of domain **component**. Statement 36 defines a **testability** field of view and specifies that its set of parameters contains just one parameter, **the_test_facility**.

Different fields of view can be presented to their users through different styles of interface. Although the specification of these different types of interface is a simple matter in Galileo4, detailed discussion is beyond the scope of this necessarily brief presentation.

Specifications and Decisions: Galileo4 programs are interactive. A user can augment the set of constraints in the initial network that is specified in a program, by inputting additional constraints to represent his design decisions. Thus, for example, if a test engineer decides to use an Erdsys TX tester, he can indicate this decision by inputting the following equational constraint: **the_test_facility.name = erdsys**. (Note that the test engineer would not have to type this constraint — the desired decision can be input by using a mouse to select appropriate options in a series of pull-up menus. Furthermore, because of the system's use of long synonyms, the engineer would think that he was entering the following decision: **the equipment at the facility where the board will be tested = Erdsys TX**. The test engineer need never know about such “unfriendly” tokens as **the_test_facility.name** or **erdsys**.)

<input type="checkbox"/> Help	<input type="checkbox"/> File	<input type="checkbox"/> New	<input type="checkbox"/> Utilities	<input type="checkbox"/> Search
<input type="checkbox"/> Up	<input type="checkbox"/> Down	<input type="checkbox"/> Focus		<input type="checkbox"/> Toggle
<input type="checkbox"/> the equipment at the facility where the board will be tested				Erdsys TX
<input type="checkbox"/> the maximum clock frequency testable at the facility where the board will be tested				9.8
<input type="checkbox"/> the maximum number of test points testable at the facility where the board will be tested				200
>>>				
KLAUS2 - a PWB Design Advisor (Testability)				

Figure 2: The Galileo4 Scrollsheet Interface

Figure 2 shows the interface presented by KLAUS2 to the test engineer after he has selected the test equipment to be used for the project, and the system has inferred two of the attributes of this equipment via relational information represented as constraints. The largest window in this screen is a single-column spreadsheet, or “scrollsheet,” in which each cell occupies one or more lines. Various pull-down menus, as well as overlay windows for constraint violation detection and advice generation, also appear when appropriate.

Decisions like the above selection of a tester are parametric design decisions. Componential design decisions can be expressed by adding new parameters to the initial network that is defined by the program. Thus, for example, a circuit designer interacting with the KLAUS2 application can introduce new parameters to represent various parts of his evolving circuit. To introduce a CPU, for example, he can either introduce a parameter of domain `component` and specify that the `type` slot of this parameter has the value `processor`, or he can achieve exactly the same result, through frame-based inheritance, by introducing a parameter of domain `cpu`.

In Galileo4, we can specify which users of an application that supports multiple fields of view are allowed to introduce new parameters and what classes of parameters they are allowed to enter. Statement 37 of Figure 1, for example, specifies that users of the `configuration` field of view are allowed to introduce parameters of domain `component` or of any domain (such as `osc_crystal`, `divider`, `resistor` or `cpu`) that is a sub-class of the `component` domain.

One further point should be made on the representation of design decisions in Galileo4. Any syntactically well-formed Galileo4 constraint may be used to represent a design decision. We are not restricted to equations of the form seen above in the test engineer’s selection of test equipment. Any sentence, atomic, compound or quantified, in first-order predicate calculus, including modal and free logic as well as classical logic, can be used to represent a design agent’s decision.

Explanation: As well as specifying information by introducing new parameters and new constraints, the user of a Galileo4 program can ask for information. He can, for example, ask for the range of allowable values for any of the parameters in a network. He can also ask for justifications for these ranges — whenever the range of allowable values for a parameter is reduced by a constraint, the rationale for this reduction is noted by the run-time system as a dependency record which can be accessed later for explanation purposes.

Consider, for example, a scenario in which the circuit designer specifies that he wants the crystal to oscillate at 25 MHz. Before doing so, he could have asked KLAUS2 for the range of possible values. If he had done so, he would have been told that the frequency should ideally be in the range

```

domain robot ::= datafile(qdb,$ROBOTS.DBE$).
function price(robot) - > positive ::= datafile(qdb,$PRICES.DBE$).
function mag_capacity(robot) - > pos_int ::= datafile(qdb,$MAGCPTY.DBE$).
function fixtures(robot) - > pos_int ::= datafile(qdb,$FIXTURES.DBE$).
function throughput(robot) - > positive ::= datafile(qdb,$THRUPUT.DBE$).
constant tax_rate ::= 0.06.
positive(reqd_throughput).
pos_int(reqd_mag_capacity).
pos_int(reqd_fixtures).
robot(chosen_model).
positive(cost).
cost = price(chosen_model) * (1 + tax_rate).
fixtures(chosen_model) >= reqd_fixtures.
throughput(chosen_model) >= reqd_throughput.
mag_capacity(chosen_model) >= reqd_mag_capacity.
not(exists X : fixtures(X) >= reqd_fixtures and throughput(X) >= reqd_throughput and
      mag_capacity(X) >= reqd_mag_capacity and price(X) < price(chosen_model)).

```

Figure 3: A Galileo4 insertion robot selection application

[3,9.8] but that, as a last resort, any frequency in the range [3,40] could be used. If he had asked for a justification, the explanation would have referred to the fact that, in general, the only frequencies allowed are those in the range [3,40] (see statement 4 in Figure 1) but that the test engineer’s previous choice of an Erdsys TX tester and KLAUS2’s preference for avoiding the introduction of ancillary dividers (see the above discussion on statement 24 in the paragraph on Non-Parametric Design) means that the preferred range is [3,9.8] because the maximum frequency testable by the Erdsys TX is 9.8 MHz.

“What If” Design Reasoning: A user can always withdraw any constraint or parameter that he has added. Thus, by introducing and withdrawing constraints and parameters, the user can investigate “what if” scenarios.

3 A Second Application: Insertion Robot Selection

To demonstrate the practical utility of this constraint-based approach to logic, we present in Figure 3 a Galileo4 program for a type of application (manufacturing equipment selection) which is commonly implemented using declarative rule-based programming. In this case, the application involves selecting an insertion robot to provide required functionality at a minimum price. This practical application is a short program in Galileo4 and illustrates well the advantage of constraint-based programming: an equivalent declarative rule-based program would be longer and less perspicuous.

The problem of insertion robot selection is to select a robot model from among a set of available models, each with varying prices, insertion magazine capacities, number of fixtures accommodated, and throughput rates. A suitable robot should have throughput, fixture, and magazine capacity at least as great as those specified by the user of the Galileo4 program. The problem of optimal-cost robot selection is to select the suitable robot of lowest cost, factoring in the tax rate.

model_9520	model_9520	4499.99	model_9520	628
model_8581	model_8581	3100.99	model_8581	128
model_8582	model_8582	3500.99	model_8582	140
model_7001	model_7001	2859.99	model_7001	88
model_7002	model_7002	2959.99	model_7002	115
model_6001	model_6001	1999.99	model_6001	60
(a) ROBOTS.DBE		(b) PRICES.DBE		(c) MAGCPTY.DBE
model_9520	3	model_9520	20	
model_8581	6	model_8581	25	
model_8582	5	model_8582	25	
model_7001	6	model_7001	20	
model_7002	5	model_7002	20	
model_6001	1	model_6001	16	
(d) FIXTURES.DBE		(e) THRUPUT.DBE		

Figure 4: Contents of Relational Database Files

All of these aspects of the problem are compactly encoded as constraints in the Galileo4 program listed in Figure 3. The first five statements in this program illustrate an advantage of our approach that were not exercised by the printed wiring board example, namely the natural way in which programs based on this approach can be linked to relational databases. Application-specific domains, functions and predicates can be defined either extensionally or intensionally within a constraint-based program. Alternatively, as in this program, extensional definitions can be tied to the contents of external relational database files.

The first statement in this program declares that the extensional definition of the domain *robot* is the set of all robot models present in the relational file ROBOTS.DBE. Similarly, the next four statements declare that the extensional definitions of the four functions *price*, *mag_capacity*, *fixtures* and *throughput* are in the files PRICES.DBE, MAGCPTY.DBE, FIXTURES.DBE and THRUPUT.DBE, respectively. The sixth statement declares that $tax_rate \mapsto 0.06$.

Suppose that the contents of the files ROBOTS.DBE, PRICES.DBE, MAGCPTY.DBE, FIXTURES.DBE and THRUPUT.DBE are as shown in Figure 4.

There are five values required of the user in order to instantiate all the parameters in the program: *reqd_throughput*, *reqd_mag_capacity*, *reqd_fixtures*, *chosen_model*, and *cost*. These correspond, respectively, to the throughput (in parts per minute) that the user requires, the insertion magazine capacity needed, the number of fixtures required, the robot model most appropriate for his needs and the amount he will have to spend. There are constraints in the program to specify type restrictions for these symbols; another constraint specifies that *cost* is the price of the *chosen_model* plus six percent tax; still other constraints specify that the *chosen_model* must provide the required functionality, and the final constraint specifies that there should not be anything cheaper than the *chosen_model* which offers this functionality.

3.1 Problem-Solving Versatility: An Example Scenario

To show the versatility of this program and, in a broader context, to illustrate the problem-solving power provided by using constraint networks the way we do, we next present an example scenario which samples the wide variety of problems that the program can be used to solve.

Even this simple a program can be used to solve a wide variety of problems. The user is free to invoke forward-chaining by asserting (and later retracting) additional arbitrary constraints, the only restriction being that each such sentence must reference at least one parameter from the set $\{reqd_throughput, reqd_mag_capacity, reqd_fixtures, cost, chosen_model\}$. Alternatively, the user can invoke backward-chaining by asking the system to determine the value of any of the parameters in this set. The following example scenario illustrates some of this versatility.

Suppose there is a manufacturing operation, with a limited budget, which hopes to buy a robot which will satisfy two needs: a lot of magazine capacity and sufficient fixtures.

Immediately after invoking this program, the user specifies what he estimates to be the amount of magazine capacity needed, by interactively asserting $reqd_mag_capacity = 200$. The system reports that $chosen_model \mapsto model_9520$ and that $cost \mapsto 4769.9894$. If the user asks for a justification, the system will explain that these conclusions were caused by the requirements that

$$mag_capacity(chosen_model) \geq reqd_mag_capacity$$

and that

$$cost = price(chosen_model) * (1 + tax_rate).$$

But suppose that this is more than management wants to spend. The user retracts $reqd_mag_capacity = 200$ and asserts $cost \leq 4000$. If the user asks to be shown the current set of possible values for $reqd_mag_capacity$, he will be told that it is any positive integer up to 140, which is the magazine capacity of the `model_8582`, the robot with the largest magazine capacity whose cost does not exceed 4000.

Suppose that the user now decides to stop volunteering information and tells the system to determine the appropriate robot, asking whatever questions it sees fit along the way. Then, triggered by the specification $fixtures(chosen_model) \geq reqd_fixtures$, the system asks the user to specify a value for $reqd_fixtures$ and, in response, suppose the user asserts that $reqd_fixtures = 6$. The system then asks for the $reqd_throughput$; the user has no specific speed in mind but, choosing a ball-park minimum, he asserts that $reqd_throughput \geq 18$. The system then asks for the $reqd_mag_capacity$; the user, remembering the previous information about the set of possible values for this parameter, responds by asserting that $reqd_mag_capacity = 140$.

However, this causes a contradiction, because the maximum magazine capacity offered by any robot which can support six fixtures is 128. The system suggests that the user should retract one of the two assertions, $reqd_fixtures = 6$ or $reqd_mag_capacity = 140$.

However, suppose the user wants to know why he cannot have the `model_8582`. Therefore, before adopting any of the above suggestions, he asserts $chosen_model = model_8582$. In response, he is told that this contradicts the requirements that $fixtures(chosen_model) \geq reqd_fixtures$ and $reqd_fixtures = 6$.

Recognizing that he cannot get a robot which satisfies all his needs, the user decides that, for now, he will just buy a robot with a smaller insertion magazine. So, he retracts $chosen_model = model_8582$ and $reqd_mag_capacity = 140$, and asserts that $reqd_mag_capacity = 10$.

The set of possible values for $chosen_model$ which is allowed by the user's stated functionality requirements contains two robots, the `model_8581` and the `model_7001`. The system cannot decide for certain, however, that either of these robots is suitable, because the user has not been specific enough about $reqd_throughput$.

Suppose, however, that the user now chooses a ball-park maximum for *reqd_throughput* and asserts that *reqd_throughput* < 19. Both of the robots just mentioned provide the required functionality but, because of the constraint

$$\text{not}(\text{exists } X : \text{fixtures}(X) \geq \text{reqd_fixtures} \text{ and } \text{throughput}(X) \geq \text{reqd_throughput} \text{ and} \\ \text{mag_capacity}(X) \geq \text{reqd_mag_capacity} \text{ and } \text{price}(X) < \text{price}(\text{chosen_model})).$$

the system can choose the cheaper of the two robots, reporting that *chosen_model* \mapsto *model_7001*.

4 Comparative Discussion

Although it is less widely used than rule-based programming, constraint-based programming is not a new idea: the first constraint-based programming system was developed almost 30 years ago [29]. The major factor inhibiting a widespread application of constraint-based programming has been the highly specialized nature of the constraint-based programming systems available. For example, the first constraint-based system, Sketchpad [29] was oriented towards graphics while Thinglab [6] was developed for simulation applications. An early language which offered more generality was Constraints [28] but several important notions, including inequality, were not available in the language. Magritte [14] was restricted to algebraic relationships and did not support the use of arbitrary application-specific predicates, functions and domains.

More recently, there has been a surge of interest in the relationship between constraints and logic programming. Several languages based on Prolog have been developed, most notably CLP(\mathfrak{R}) [17]; Prolog III [8]; and CHIP [11]. In these languages, which are generically known as the CLP languages, unification in the Herbrand universe is supplemented with constraint processing of linear equations or inequalities over the numbers. CHIP also uses arc-consistency to process atomic constraints involving application-specific predicates defined over finite domains.

Galileo4 provides richer expressive power than any other constraint-based programming language because it allows the theory Γ used to specify a constraint network to contain arbitrary FOPC sentences (atomic, compound or quantified) about a many-sorted universe of discourse which includes, besides the real numbers \mathfrak{R} , any arbitrary application-specific sorts. Nearly all other constraint languages restrict the theory to ground sentences. For example, in the CLP languages, the theory used to specify a constraint network is the set of conjuncts in a compound goal presented to the top-level interpreter; although logic variables may appear in such a goal, these are implicitly subject to existential quantification, which means that, essentially, they are uninterpreted constant symbols, making the constraints ground sentences. One language which does seem to support quantified constraints is CONSUL [7], but it restricts the universe of discourse to the integers.

Expressiveness in knowledge representation, however, is only one side of the coin. Constraint-based languages also differ in the inferential competence and efficiency offered by their run-time systems. One of the earliest, and most commonly used, constraint propagation mechanisms is local propagation of known states, which is efficient but incomplete. The simplex-like algorithm used in the CLP languages is more complete (that is, can draw more inferences given the same premises) than local propagation of known states and considerable effort has been devoted to developing efficient and fast implementations. Until recently, there was little cross-fertilization between the literature on finite domain CSPs and that on constraint-based programming languages. The first attempt to amalgamate ideas from these two bodies of work seems to have been the CLP language CHIP [11], in which the application of simplex to constraints on the rationals was supplemented by the application of arc consistency to constraints on application-specific finite domains.

The CCP algorithm used in Galileo4 takes the idea of borrowing concepts from the finite domain CSP literature a step further, by integrating local propagation of known states with versions of

arc and path consistency that have been generalized to constraints of arbitrary arity on infinite domains. The inferential competence of the resulting algorithm is better in some respects, but worse in others, than the algorithms used in the CLP languages. For example, there are several classes of problem involving sets of simultaneous non-linear equations that can be handled by CCP [27], while the CLP languages can only handle non-linear constraints if they become linear during the propagation process. However, the CCP algorithm is limited to handling systems of simultaneous equations in which each equation involves only two unknowns or is reduced to a binary equation during propagation. Although there has been considerable research into constraint propagation algorithms for finite domain networks, and despite the fact that constraint processing has been around a long time, constraint propagation for infinite domain networks is still a relatively undeveloped field. We view CCP as being capable of further improvement; for example, an obvious topic to investigate is the amalgamation of simplex analysis with the battery of techniques currently used in CCP.

The query interface to CLP languages is basically the same as that provided by the early Prologs. That this interface is inadequate for constraint processing was recognized in [22] where a new interface was proposed. It is interesting to note that the “answer manipulation commands” which they propose would allow the same kind of non-monotonic editing of goals as the Galileo4 user can achieve by asserting/retracting assumptions.

5 The Relationship Between Constraint Networks and Constraint Logic

This chapter presents an attempt to make the full FOPC available in an INCA-construction language known as Galileo4. In the run-time system for this language, inference is based on a treatment of semantic entailment as constraint propagation. Very simply, constraint propagation is the communication of parameter values through the network to all constraints which refer to those parameters. This propagation process occurs omnidirectionally each time a parameter acquires either a new or revised value or a restriction of its domain of possible values. These new values may in turn stimulate further inference, and the process iterates until quiescence. To provide the theoretical basis for this approach, we review model theory and constraint processing and then we show how the satisfiability of a set of sentences in logic can be viewed as the consistency of a set of constraints in a constraint network.

5.1 A Review of Model Theory

The truth of a sentence in logic is based on the interpretation of the symbols in the sentence. Consider some first-order logic language $\mathcal{L} = \langle \mathcal{P}, \mathcal{F}, \mathcal{K} \rangle$, where \mathcal{P} is the vocabulary of predicate symbols, \mathcal{F} is the vocabulary of function symbols, and \mathcal{K} is the vocabulary of constant symbols. A theory Γ in \mathcal{L} is satisfiable iff there exists some model $\mathcal{M} = \langle \mathcal{U}, \mathcal{I} \rangle$ of the language \mathcal{L} under which every sentence in Γ is true, that is, iff there exists some \mathcal{M} such that $\mathcal{M} \models \Gamma$.¹

In a model $\mathcal{M} = \langle \mathcal{U}, \mathcal{I} \rangle$ for the language $\mathcal{L} = \langle \mathcal{P}, \mathcal{F}, \mathcal{K} \rangle$, \mathcal{U} is a universe of discourse while \mathcal{I} is an interpretation function for the constant, predicate, and function symbols of \mathcal{L} . For every $\kappa \in \mathcal{K}$, $\mathcal{I}(\kappa)$ is an element of \mathcal{U} . For every n -ary predicate symbol $p \in \mathcal{P}$, $\mathcal{I}(p)$ is an n -ary relation over \mathcal{U} . For every n -ary function symbol $f \in \mathcal{F}$, $\mathcal{I}(f)$ is an $(n + 1)$ -ary relation over \mathcal{U} . For a functional expression $f(a_1, \dots, a_n)$, where f is an n -ary function symbol and the terms $a_i, i = 1..n$, are either

¹The notation $\mathcal{M} \models \Gamma$ means that any interpretation of the symbols in \mathcal{M} satisfying \mathcal{M} will simultaneously satisfy Γ .

constant symbols or nested functional expressions, $\mathcal{I}(f(a_1, \dots, a_n))$ is an element of \mathcal{U} such that $\mathcal{I}(f)$ contains the $(n + 1)$ -ary tuple $\langle \mathcal{I}(a_1), \dots, \mathcal{I}(a_n), \mathcal{I}(f(a_1, \dots, a_n)) \rangle$.

Let $\mathcal{M} \models \gamma$, where γ is a sentence, mean that γ is true under the model \mathcal{M} and let $\mathcal{M}_{\{X \mapsto u\}}$ mean a model containing the extended interpretation function $\mathcal{I} \cup \{X \mapsto u\}$.² The rules for determining whether a sentence is true under a model are as follows:

- $\mathcal{M} \models p(a_1, \dots, a_n)$ iff $\langle \mathcal{I}(a_1), \dots, \mathcal{I}(a_n) \rangle$ is in $\mathcal{I}(p)$.
- $\mathcal{M} \models \neg A$ iff $\mathcal{M} \not\models A$.
- $\mathcal{M} \models A \wedge B$ iff $\mathcal{M} \models A$ and $\mathcal{M} \models B$.
- $\mathcal{M} \models A \vee B$ iff $\mathcal{M} \models A$ or $\mathcal{M} \models B$.
- $\mathcal{M} \models A \Rightarrow B$ iff $\mathcal{M} \not\models A$ or $\mathcal{M} \models B$.
- $\mathcal{M} \models A \Leftrightarrow B$ iff $(\mathcal{M} \not\models A \text{ and } \mathcal{M} \not\models B)$ or $(\mathcal{M} \models A \text{ and } \mathcal{M} \models B)$.
- $\mathcal{M} \models (\forall X)A$ iff $\mathcal{M}_{\{X \mapsto u\}} \models A$ for every $u \in \mathcal{U}$.
- $\mathcal{M} \models (\exists X)A$ iff $\mathcal{M}_{\{X \mapsto u\}} \models A$ for some $u \in \mathcal{U}$.

5.2 A Review of Constraint Processing

A constraint specifies some relationship that must be satisfied by the values, chosen from some domain, which are assumed by a group of parameters. A constraint network is a collection of constraints which are interlinked by virtue of having some parameters in common.

There has developed a considerable body of literature on the processing of constraint networks. This literature tends to fall into two broad classes. In the first kind of paper, which is where the term CSP is used, constraint networks are assumed to have finite domains [10, 13, 16, 21, 23, 25], the relationships imposed by constraints are expressed extensionally, and the CSPs are NP-complete. Although the CSPs in this literature can be solved by backtracking search, the exponential computational cost of such search algorithms has led to the development of preprocessing algorithms [12, 15, 21, 24], called consistency algorithms, which aim to reduce thrashing, that is, repeatedly exploring parameter assignments that cannot result in a solution, by *a priori* elimination of parameter to value mappings that cannot belong to any consistent valuation of the network parameters.

In the second class of literature on constraint processing, constraint networks are allowed, but not required, to have infinite domains, and the restrictions imposed by constraints can be expressed intensionally. In this literature, which includes that dealing with Constraint Logic Programming (CLP) [8, 11, 17], the problem-solving techniques include local propagation of known states [20] and a simplex-like version of gaussian elimination [17]. Consider, for example, the CLP(\mathfrak{R}) language, which is based on Prolog and in which constraint parameters can range over the real numbers \mathfrak{R} . In this language, a constraint network is treated as a conjunctive goal which is submitted to the top-level of a Prolog-like interpreter; for example, a goal such as

$$?- \text{Load} = \text{Area} * \text{Stress}, \text{Load} = 30, \text{Area} = 5.$$

is a network containing three constraints which intensionally specify restrictions on the values that can be assumed by the three parameters, Load, Area and Stress. When the CLP(\mathfrak{R}) interpreter responds with

²The notation $X \mapsto u$ indicates that symbol X is assigned the semantic interpretation u .

$$\text{Load} = 30 \quad \text{Area} = 5 \quad \text{Stress} = 6 \quad \text{***Yes}$$

it is giving an existence proof that there is a valuation for the parameters which satisfies all the constraints in the network.

In this second body of work, although constraint networks may have infinite domains, severe limits have been placed on the use of intensional specifications. Systems have restricted the types of logic connective that can be used and the situations in which they can be used. Furthermore, in most systems, only ground sentences may be used. In $\text{CLP}(\mathbb{R})$, for example, the free variables in a goal are treated as being implicitly subject to existential quantification, which means that effectively they are equivalent to uninterpreted constant symbols. Thus, the goal shown above can be regarded as a conjunction of three ground sentences, $\text{load} = \text{area} * \text{stress}$, $\text{load} = 30$, and $\text{area} = 5$, in which there are three uninterpreted constant symbols, load , area , and stress . The task of the interpreter is then to determine whether there is an interpretation for these constant symbols which satisfies all three ground sentences. Constraints in $\text{CLP}(\mathbb{R})$ cannot use quantification; it is not possible, for example, to submit goals such as

$$\begin{aligned} ?- \text{Load} &= \text{Area} * \text{Stress}, \text{Load} = 30, \\ &\text{Area} = 5, (\text{all } X : \text{Area} < X < \text{Stress} \text{ implies } X > \text{Load}/2). \end{aligned}$$

in which the last constraint is a universally quantified sentence. In the one language known to allow explicit quantification [7], the domain is restricted to the integers and it is unclear whether quantifiers may be arbitrarily nested.

Our work is directed towards removing these limitations and allowing arbitrary FOPC formulae to be used as constraints.

5.3 Definitions

The literature contains several definitions of constraint satisfaction, with varying degrees of formality. One consequence of the lack of formal definition of the concept is that many authors fail to distinguish between notions of decision, exemplification, and enumeration. In an effort to remove this ambiguity and to set the scene for a mapping between constraint processing and semantic modeling, we propose the following definitions.

Definition, Constraint Network:

A constraint network is a triple $\langle \mathcal{U}, \mathbf{X}, \mathbf{C} \rangle$ where \mathcal{U} is a universe of discourse, \mathbf{X} is a finite tuple of q non-recurring parameters, and \mathbf{C} is a finite set of r constraints. Each constraint $C_k(T_k) \in \mathbf{C}$ imposes a restriction on the allowable values for the a_k parameters in T_k , a sub-tuple of \mathbf{X} , by specifying that some subset of the a_k -ary Cartesian product \mathcal{U}^{a_k} contains all acceptable combinations of values for these parameters.

The overall network constitutes an intensional specification of a joint possibility distribution for the values of the parameters in the network. This joint possibility distribution, called the network *intent* [2], is a q -ary relation on \mathcal{U}^q , as defined below.

Definition, Projection:

Let R be a q -ary relation with indices $\langle x_1, x_2, \dots, x_q \rangle$, and let $T = \langle t_1, t_2, \dots, t_m \rangle$ be an m -ary subtuple of $\langle x_1, x_2, \dots, x_q \rangle$. The projection of R onto T , denoted $\text{proj}(R, T)$, is the largest set of m -tuples $\langle b_{t_1}, b_{t_2}, \dots, b_{t_m} \rangle$ such that there is some q -tuple $\langle c_{x_1}, c_{x_2}, \dots, c_{x_q} \rangle$ in R for which $b_{t_j} = c_{x_j}$, for all $j = 1, 2, \dots, m$.

Definition, Cylindrical Extension:

Let $\mathbf{X} = \langle x_1, x_2, \dots, x_q \rangle$ be a q -ary tuple of indices defining a Cartesian space \mathcal{U}^q , let $T = \langle t_1, t_2, \dots, t_m \rangle$ be a subtuple of $\langle x_1, x_2, \dots, x_q \rangle$, and let $C(T)$ be a relation on the space defined by T . The cylindrical extension of $C(T)$ into the space defined by \mathbf{X} , denoted $E(X)$, is the largest relation R on that space such that $proj(R, T) = C(T)$.

Definition, The Intent of a Constraint Network:

The intent of a constraint network $\langle \mathcal{U}, \mathbf{X}, \mathbf{C} \rangle$ is

$$\Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}} = E_1(\mathbf{X}) \cap \dots \cap E_r(\mathbf{X}),$$

where, for each constraint $C_k(T_k) \in \mathbf{C}$, $E_k(\mathbf{X})$ is its cylindrical extension into the Cartesian space defined by \mathbf{X} .

The network intent is a set of q -tuples, each tuple giving, for the q parameters in \mathbf{X} , a valuation which is acceptable to all the constraints in \mathbf{C} .

A constraint network is consistent if the network intent is not the empty set. Three forms of constraint satisfaction problem (CSP) can be distinguished:

Definition, The Decision CSP:

Given a network $\langle \mathcal{U}, \mathbf{X}, \mathbf{C} \rangle$, decide whether $\Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}}$ is non-empty.

Definition, The Exemplification CSP:

Given a network $\langle \mathcal{U}, \mathbf{X}, \mathbf{C} \rangle$, return some tuple from $\Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}}$, if $\Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}}$ is non-empty, or return nil otherwise.

Definition, The Enumeration CSP:

Given a network $\langle \mathcal{U}, \mathbf{X}, \mathbf{C} \rangle$, return $\Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}}$.

5.4 Networks and Models

A total interpretation function for a first-order language \mathcal{L} is a set of mappings, such that every predicate and function symbol of \mathcal{L} is mapped onto a relation of the appropriate arity over some universe of discourse \mathcal{U} , and such that every constant symbol of \mathcal{L} is mapped onto a member of \mathcal{U} . A partial interpretation function for \mathcal{L} is a set of mappings which does not contain a mapping for every symbol of \mathcal{L} ; at least one constant, predicate, or function symbol is not given a mapping.

Consider the class of problem in which, given a theory Γ written in a first-order language $\mathcal{L} = \langle \mathcal{P}, \mathcal{F}, \mathcal{K} \rangle$ and a partial interpretation function \mathcal{I}_p for \mathcal{L} in terms of a universe \mathcal{U} , one has to determine whether there is any total interpretation \mathcal{I} for \mathcal{L} such that $\mathcal{I}_p \subset \mathcal{I}$ and $\langle \mathcal{U}, \mathcal{I} \rangle \models \Gamma$. This, the class of modeling problems, can be divided into several subclasses.

Class 1 modeling problems are those where \mathcal{I}_p contains interpretations for all the predicate and function symbols of \mathcal{L} , where a finite subset of the constant symbols are uninterpreted, where every sentence in Γ references at least one of the uninterpreted constant symbols, and where each uninterpreted constant symbol is referenced by at least one sentence in Γ . Although there are several other classes, (for example, those modeling problems where some subset of the function and predicate symbols lack interpretations), Class 1 problems are those that are of interest here.

Class 1 problems can be further subdivided, as follows:

Definition, The Class 1 Decision Modeling Problem:

Given $\mathcal{L} = \langle \mathcal{P}, \mathcal{F}, \mathcal{K}' \cup \mathcal{K}'' \rangle$, \mathcal{U} , \mathcal{I}_p and Γ , where \mathcal{I}_p interprets, in terms of \mathcal{U} , all and only the symbols in $\mathcal{P} \cup \mathcal{F} \cup \mathcal{K}'$, \mathcal{K}'' is finite, each sentence in Γ references at least one symbol in \mathcal{K}'' , and each symbol in \mathcal{K}'' is referenced by at least one sentence $\gamma \in \Gamma$, decide whether there exists any \mathcal{J} such that \mathcal{J} interprets all symbols in \mathcal{K}'' and $\langle \mathcal{U}, \mathcal{I}_p \cup \mathcal{J} \rangle \models \Gamma$.

Definition, The Class 1 Exemplification Modeling Problem:

Given $\mathcal{L} = \langle \mathcal{P}, \mathcal{F}, \mathcal{K}' \cup \mathcal{K}'' \rangle$, \mathcal{U} , \mathcal{I}_p and Γ , as in the Class 1 Decision Modeling problem, return, if one exists, some \mathcal{J} such that \mathcal{J} interprets all symbols in \mathcal{K}'' and $\langle \mathcal{U}, \mathcal{I}_p \cup \mathcal{J} \rangle \models \Gamma$; if no such \mathcal{J} exists, return nil.

Definition, The Class 1 Enumeration Modeling Problem:

Given $\mathcal{L} = \langle \mathcal{P}, \mathcal{F}, \mathcal{K}' \cup \mathcal{K}'' \rangle$, \mathcal{U} , \mathcal{I}_p and Γ , as in the Class 1 Decision Modeling problem, return the set of all \mathcal{J} such that \mathcal{J} interprets all symbols in \mathcal{K}'' and $\langle \mathcal{U}, \mathcal{I}_p \cup \mathcal{J} \rangle \models \Gamma$.

These three forms of modeling problem can be shown to correspond to the three forms of CSP defined in Section 5.3. In what follows, let $\vec{A}\vec{B}$, where A and B are tuples of the same arity, be the mapping function from the components of A onto those of B in which each component of the tuple A is mapped onto the component in the corresponding position of tuple B ; for example, if $A = \langle x, y \rangle$ and $B = \langle 4, 2 \rangle$, then $\vec{A}\vec{B} = \{ x \mapsto 4, y \mapsto 2 \}$.

Theorem 1

Given $\mathcal{L} = \langle \mathcal{P}, \mathcal{F}, \mathcal{K}' \cup \mathcal{K}'' \rangle$, \mathcal{U} , \mathcal{I}_p and Γ , where \mathcal{I}_p interprets, in terms of \mathcal{U} , all and only the symbols in $\mathcal{P} \cup \mathcal{F} \cup \mathcal{K}'$, \mathcal{K}'' is finite, each sentence in Γ references at least one symbol in \mathcal{K}'' , and each symbol in \mathcal{K}'' is referenced by at least one sentence $\gamma \in \Gamma$. Let $\langle \mathcal{U}, \mathbf{X}, \mathbf{C} \rangle$ be a constraint network such that \mathbf{X} is the lexical ordering of the elements of \mathcal{K}'' and such that $|\mathbf{C}| = |\Gamma|$, with each sentence $\gamma \in \Gamma$ having a corresponding constraint $C(T) \in \mathbf{C}$ such that T is the lexical ordering of those elements of \mathcal{K}'' which appear in γ and $\langle \mathcal{U}, \mathcal{I}_p \cup \vec{T}t \rangle \models \gamma$ for all tuples $t \in C(T)$. Then $\langle \mathcal{U}, \mathcal{I}_p \cup \vec{\mathbf{X}}\tau \rangle \models \Gamma$ for all tuples $\tau \in \Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}}$.

The proof of this theorem appears in the appendix.

Thus, the set of all $\vec{\mathbf{X}}\tau$ where $\tau \in \Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}}$ is the set of all \mathcal{J} such that \mathcal{J} interprets all symbols in \mathcal{K}'' and $\langle \mathcal{U}, \mathcal{I}_p \cup \mathcal{J} \rangle \models \Gamma$. Thus, the Class 1 Decision Modeling Problem corresponds to the Decision CSP, the Class 1 Exemplification Modeling Problem corresponds to the Exemplification CSP and the Class 1 Enumeration Modeling Problem corresponds to the Enumeration CSP.

5.4.1 Example 1

Consider a language \mathcal{L} , a universe of discourse \mathcal{U} , and a partial interpretation \mathcal{I}_p for \mathcal{L} in terms of \mathcal{U} , as follows:

Language: $\mathcal{L} = \langle \{s, t\}, \emptyset, \{a, b, c\} \rangle$

Universe of discourse: $\mathcal{U} = \{1, 2, 3, 4, 5, 6, 7\}$

Partial Interpretation:

$$\begin{aligned} \mathcal{I}_p = \{ & s \mapsto \{5, 7\}, \\ & t \mapsto \{ \langle 1, 4 \rangle, \langle 2, 5 \rangle, \langle 3, 6 \rangle, \langle 4, 7 \rangle, \langle 5, 1 \rangle, \langle 5, 3 \rangle, \\ & \quad \langle 6, 2 \rangle, \langle 6, 3 \rangle, \langle 6, 7 \rangle, \langle 7, 1 \rangle, \langle 7, 3 \rangle \}, \\ & b \mapsto 7 \}. \end{aligned}$$

Suppose that, given the above situation, we need to determine the satisfiability of the following theory:

$$\Gamma = \{t(a, c), (\forall X)(s(X) \Rightarrow t(X, a)), t(c, b)\}.$$

The constraint network $\langle \mathcal{U}, \mathbf{X}, \mathbf{C} \rangle$ corresponding to this is a possibility distribution, such that Γ is satisfied, for interpretations of the remaining uninterpreted constant symbols of \mathcal{L} , in this case for a and c . The components of the network are:

$$\begin{aligned}
\mathcal{U} &= \{1, 2, 3, 4, 5, 6, 7\} \\
\mathbf{X} &= \langle a, c \rangle \\
\mathbf{C} &= \{C_1(a, c), C_2(a), C_3(c)\}.
\end{aligned}$$

Each sentence in the theory has a corresponding constraint in the network, the definition of which depends on whatever information is provided, by the partial interpretation \mathcal{I}_p , about the symbols appearing in the sentence. In this case, the tuples admitted by these constraints are as follows:

$$\begin{aligned}
C_1(a, c) &= \{\langle 1, 4 \rangle, \langle 2, 5 \rangle, \langle 3, 6 \rangle, \langle 4, 7 \rangle, \langle 5, 1 \rangle, \langle 5, 3 \rangle, \langle 6, 2 \rangle, \langle 6, 3 \rangle, \langle 6, 7 \rangle, \langle 7, 1 \rangle, \langle 7, 3 \rangle\} \\
C_2(a) &= \{1, 3\} \\
C_3(c) &= \{4, 6\}.
\end{aligned}$$

Consider, for example, $C_1(a, c)$, which corresponds to the sentence $t(a, c)$. There are two parameters in this constraint, corresponding to the two constant symbols in the sentence. The restriction imposed by the constraint is derived from the interpretation in \mathcal{I}_p for the predicate symbol t in the sentence. Consider $C_2(a)$, which corresponds to the sentence $(\forall X)(s(X) \Rightarrow t(X, a))$; the restriction imposed by this constraint is derived from the model-theoretic rules for universal quantification and implication and from the interpretations in \mathcal{I}_p for the predicate symbols s and t in the sentence. Consider $C_3(c)$, which corresponds to the sentence $t(c, b)$; although two constant symbols appear in the sentence, there is only one parameter in the constraint, because \mathcal{I}_p provides an interpretation for b .

Since the constraint $C_1(a, c)$ references all parameters in \mathbf{X} , the cylindrical extension of the constraint is just the constraint itself; $E_1(a, c) = C_1(a, c)$. That is,

$$\begin{aligned}
E_1(a, c) &= \{\langle 1, 4 \rangle, \langle 2, 5 \rangle, \langle 3, 6 \rangle, \langle 4, 7 \rangle, \langle 5, 1 \rangle, \langle 5, 3 \rangle, \langle 6, 2 \rangle, \\
&\quad \langle 6, 3 \rangle, \langle 6, 7 \rangle, \langle 7, 1 \rangle, \langle 7, 3 \rangle\}.
\end{aligned}$$

The constraint $C_2(a)$ admits two values for a so, since the cardinality of \mathcal{U} is 7, the cylindrical extension of this constraint in the Cartesian space defined by \mathbf{X} contains 14 pairs:

$$\begin{aligned}
E_2(a, c) &= \{\langle 1, 1 \rangle, \langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 1, 4 \rangle, \langle 1, 5 \rangle, \langle 1, 6 \rangle, \langle 1, 7 \rangle, \\
&\quad \langle 3, 1 \rangle, \langle 3, 2 \rangle, \langle 3, 3 \rangle, \langle 3, 4 \rangle, \langle 3, 5 \rangle, \langle 3, 6 \rangle, \langle 3, 7 \rangle\}.
\end{aligned}$$

Similarly, the cylindrical extension of $C_3(c)$ also contains 14 pairs:

$$\begin{aligned}
E_3(a, c) &= \{\langle 1, 4 \rangle, \langle 2, 4 \rangle, \langle 3, 4 \rangle, \langle 4, 4 \rangle, \langle 5, 4 \rangle, \langle 6, 4 \rangle, \langle 7, 4 \rangle, \\
&\quad \langle 1, 6 \rangle, \langle 2, 6 \rangle, \langle 3, 6 \rangle, \langle 4, 6 \rangle, \langle 5, 6 \rangle, \langle 6, 6 \rangle, \langle 7, 6 \rangle\}.
\end{aligned}$$

However, if we compute the intersection of these three cylindrical extensions, we find there is little overlap, the intent of the network containing only two pairs:

$$\Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}} = \{\langle 1, 4 \rangle, \langle 3, 6 \rangle\}.$$

The network intent constitutes a joint possibility distribution for a and c . By projecting this joint distribution onto the a and c dimensions, we can obtain marginal possibility distributions for the individual parameters. Thus, the marginal possibility distribution for a is $\Pi_a = \{1, 3\}$ and that for c is $\Pi_c = \{4, 6\}$.

The network intent being $\Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}} = \{\langle 1, 4 \rangle, \langle 3, 6 \rangle\}$ means that the theory Γ is satisfied under the following two models of the language \mathcal{L} :

$$\begin{aligned}
\mathcal{M}_1 &= \langle \mathcal{U}, \mathcal{I}_p \cup \{a \mapsto 1, c \mapsto 4\} \rangle; \\
\mathcal{M}_2 &= \langle \mathcal{U}, \mathcal{I}_p \cup \{a \mapsto 3, c \mapsto 6\} \rangle.
\end{aligned}$$

Augmenting a theory by asserting an additional sentence often reduces the number of possible models. Suppose we augment Γ by adding the assertion $t(b, c)$ to the theory. From $\mathcal{I}_p(t)$ and $\mathcal{I}_p(b)$, the corresponding constraint is $C_4(c) = \{1, 3\}$. Since this constraint admits neither of the two values for c that are allowed by the existing network, the new network intent will be the empty set. Thus, there is no model $\langle \mathcal{U}, \mathcal{I} \rangle, \mathcal{I}_p \subset \mathcal{I}$, of the language \mathcal{L} under which the following theory is satisfied:

$$\Gamma' = \{t(a, c), (\forall X)(s(X) \Rightarrow t(X, a)), t(c, b), t(b, c)\}.$$

Suppose, however, we retract $t(b, c)$ and assert $t(c, a)$. The resultant theory in this case,

$$\Gamma'' = \{t(a, c), (\forall X)(s(X) \Rightarrow t(X, a)), t(c, b), t(c, a)\}$$

is satisfiable. However, the new network intent would be a strict subset of $\Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}}$, namely $\Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}''} = \{\langle 3, 6 \rangle\}$. The fact that $\Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}''}$ admits only one pair means that unique values 3 and 6 can be inferred for a and c , respectively.

5.4.2 Example 2

In Example 1, the universe of discourse was finite and the predicate symbols had finite interpretations. The following example has an infinite universe and some of the predicate and function symbols have infinite sets as their interpretations:

Language: $\mathcal{L} = \{\textit{positive}, \textit{nonnegative}, =, \leq, h, j, k, g\},$
 $\{*, l, m\}, \mathcal{R} \cup \{a, b, c, d, e, f, g1, g2, g3, g4\}$

Universe of discourse: $\mathcal{U} = \mathfrak{R} \cup \{g1, g2, g3, g4\}$

Partial Interpretation:

$$\begin{aligned} \mathcal{I}_p = \mathcal{I}_{\mathcal{R}} \cup & \\ \{ \textit{positive} & \mapsto \mathfrak{R}^+ \\ \textit{nonnegative} & \mapsto \mathfrak{R}^{0+} \\ = & \mapsto \{\langle X, Y \rangle \mid X \in \mathcal{U} \wedge Y \in \mathcal{U} \wedge \text{EQUALS}(X, Y)\} \\ \leq & \mapsto \{\langle X, Y \rangle \mid X \in \mathfrak{R} \wedge Y \in \mathfrak{R} \wedge \text{LEQ}(X, Y)\} \\ h & \mapsto \{X \mid X \in \mathfrak{R}^+ \wedge \text{LEQ}(X, 10)\} \\ j & \mapsto \mathfrak{R}^+ \\ k & \mapsto \mathfrak{R}^- \\ g & \mapsto \{g1, g2, g3, g4\} \\ * & \mapsto \{\langle X, Y, Z \rangle \mid X \in \mathfrak{R} \wedge Y \in \mathfrak{R} \wedge Z \in \mathfrak{R} \wedge \text{EQUALS}(Z, \text{TIMES}(X, Y))\} \\ l & \mapsto \{\langle g1, 200 \rangle, \langle g2, 250 \rangle, \langle g3, 350 \rangle, \langle g4, 390 \rangle\} \\ m & \mapsto \{\langle g1, 230 \rangle, \langle g2, 190 \rangle, \langle g3, 240 \rangle, \langle g4, 245 \rangle\} \\ g1 & \mapsto g1 \\ g2 & \mapsto g2 \\ g3 & \mapsto g3 \\ g4 & \mapsto g4 \}. \end{aligned}$$

Theory: $\Gamma = \{\textit{positive}(a), h(b), \textit{positive}(c), g(e), d = a * f, j(f) \Rightarrow f \leq l(e), k(f) \Rightarrow f \leq m(e),$
 $(\forall X)(g(X) \Rightarrow m(X) \leq l(e)), (\exists X)(l(X) > l(e)), a = b * c\}.$

In the language \mathcal{L} , \mathcal{R} is the set of constant symbols composed from the characters $+$, $-$, $.$ and $0..9$ according to a grammar for real numeric strings. \mathcal{R} is distinguished from \mathfrak{R} , the set of real numbers. In this chapter, to distinguish between symbols of \mathcal{L} and entities of \mathcal{U} , we use typewriter font for the latter. Thus, $200 \in \mathcal{R}$ and $g1$ are constant symbols, while $200 \in \mathfrak{R}$ and $g1$ are in the universe of discourse. In the partial interpretation \mathcal{I}_p , $\mathcal{I}_{\mathcal{R}}$ is a bijection from the constant symbols in \mathcal{R} onto \mathcal{Q}_f , the set of finite-length rational numbers, $\mathcal{Q}_f \subset \mathcal{Q} \subset \mathfrak{R}$; thus $\mathcal{I}_{\mathcal{R}}$ contains mappings such as $200 \mapsto 200$.

The constraint network $\langle \mathcal{U}, \mathbf{X}, \mathbf{C} \rangle$ corresponding to this situation is a possibility distribution, such that Γ is satisfied, for interpretations of the remaining uninterpreted constant symbols of \mathcal{L} , that is, for a, b, c, d, e , and f . The components of this network are:

$$\begin{aligned} \mathcal{U} &= \mathfrak{R} \cup \{\mathbf{g1}, \mathbf{g2}, \mathbf{g3}, \mathbf{g4}\} \\ \mathbf{X} &= \langle a, b, c, d, e, f \rangle \\ \mathbf{C} &= \{C_1(a), C_2(b), C_3(c), C_4(e), C_5(a, d, f), C_6(e, f)\}, C_7(e, f), C_8(e), C_9(e), C_{10}(a, b, c)\}. \end{aligned}$$

Infinite cardinality means that the restrictions imposed by most of the constraints must be specified intensionally:

$$\begin{aligned} C_1(a) &= \mathfrak{R}^+ \\ C_2(b) &= \{X \mid X \in \mathfrak{R}^+ \wedge \text{LEQ}(X, 10)\} \\ C_3(c) &= \mathfrak{R}^+ \\ C_4(e) &= \{\mathbf{g1}, \mathbf{g2}, \mathbf{g3}, \mathbf{g4}\} \\ C_5(a, d, f) &= \{ \langle X, Y, Z \rangle \mid X \in \mathfrak{R} \wedge Y \in \mathcal{U} \wedge Z \in \mathfrak{R} \wedge \text{EQUALS}(Y, \text{TIMES}(X, Z)) \} \\ C_6(e, f) &= (\mathcal{U} \times (\mathcal{U} - \mathfrak{R}^+)) \cup \\ &\quad (\{\mathbf{g1}\} \times \{X \mid \text{LEQ}(X, 200)\}) \cup \\ &\quad (\{\mathbf{g2}\} \times \{X \mid \text{LEQ}(X, 250)\}) \cup \\ &\quad (\{\mathbf{g3}\} \times \{X \mid \text{LEQ}(X, 350)\}) \cup \\ &\quad (\{\mathbf{g4}\} \times \{X \mid \text{LEQ}(X, 390)\}) \\ C_7(e, f) &= (\mathcal{U} \times (\mathcal{U} - \mathfrak{R}^-)) \cup \\ &\quad (\{\mathbf{g1}\} \times \{X \mid \text{LEQ}(230, X)\}) \cup \\ &\quad (\{\mathbf{g2}\} \times \{X \mid \text{LEQ}(190, X)\}) \cup \\ &\quad (\{\mathbf{g3}\} \times \{X \mid \text{LEQ}(240, X)\}) \cup \\ &\quad (\{\mathbf{g4}\} \times \{X \mid \text{LEQ}(245, X)\}) \\ C_8(e) &= \{\mathbf{g2}, \mathbf{g3}, \mathbf{g4}\} \\ C_9(e) &= \{\mathbf{g1}, \mathbf{g2}, \mathbf{g3}\} \\ C_{10}(a, b, c) &= \{ \langle X, Y, Z \rangle \mid X \in \mathcal{U} \wedge Y \in \mathfrak{R} \wedge Z \in \mathfrak{R} \wedge \text{EQUALS}(X, \text{TIMES}(Y, Z)) \}. \end{aligned}$$

By augmenting the theory with arbitrary sentences from \mathcal{L} and observing the consequent effect on the joint and marginal possibility distributions, we can solve various problems. If we assert the three equations $b = 10$, $c = 20$, and $d = 5000$, we can determine f by observing the marginal possibility distribution which is $\Pi_f = \{25\}$. If, instead, we assert the equation $a = 10$ and the inequality $2600 \leq d$, we can observe that $\Pi_f = \{X \mid 260 \leq X \leq 350\}$, that $\Pi_d = \{X \mid 2600 \leq X \leq 3500\}$ and that $\Pi_e = \{\mathbf{g3}\}$.

5.5 Consistency Algorithms and Semantic Modeling

Although most consistency algorithms which have been published [15, 21, 24] all address only networks with finite domains and binary or unary constraints, the notions of node, arc and path consistency are not restricted to such networks [12]. They also apply to infinite-domain networks containing constraints of arbitrary arity.

Thus, the published consistency algorithms can be generalized to constraint networks corresponding to arbitrary Class 1 modeling problems. Consider the published arc consistency algorithms. These operate by storing the possibility distribution for each parameter extensionally, by treating each binary constraint as a pair of arcs and by eliminating from the possibility distribution for the parameter at the head of an arc any value that is not supported by some value in the possibility distribution for the parameter at the tail of the arc [21].

These algorithms can be generalized to constraints of arbitrary arity by treating each n -ary constraint as n hyper-arcs, each hyper-arc having one parameter at its head and a tuple containing the other $(n - 1)$ parameters at its tail. If any value in the possibility distribution for the parameter

at the head of a hyper-arc is not supported by some $(n - 1)$ -tuple of values for the other $(n - 1)$ parameters at the tail of the hyper-arc, the value can be removed from the possibility distribution.

Algorithms for arc consistency can be generalized to infinite domains by storing infinite possibility distributions intensionally. Removing unsupported values from the intensionally-defined possibility distribution for a parameter can be done by making the intensional formula suitably more specific. Whenever it is recognized that the possibility distribution has been restricted to a finite set of suitably small cardinality, the representation can be converted to an extensional format, if necessary.

The published path consistency algorithms can be generalized in a similar fashion. These algorithms operate by storing extensionally the joint possibility distribution corresponding to a constraint and by eliminating from the extension any tuple that is not supported by all other paths between the parameters in the constraint. The algorithms can be generalized to the infinite sets of tuples that are admitted by constraints in infinite-domain networks, by storing infinite joint possibility distributions intensionally. Removing unsupported tuples from an intensionally-defined joint possibility distribution can be done by making the intensional formula suitably more specific.

Thus, treating Class 1 modeling problems as CSPs of the generalized types defined in Section 5.3, and generalizing the published consistency algorithms appropriately, would appear to be an attractive computational companion to a representation employing the full FOPC. There is a problem, however, in that generalizing the consistency algorithms in this way means that the intensional formulae may become arbitrarily complex, which raises the specter of undecidability.³ Even the satisfiability of formulae that are theoretically decidable may be beyond the competence of whatever inference algorithm we embed in our system.

The way to escape from this problem is to side-step it. No inference algorithm for the full FOPC can be sound, terminating and complete. Soundness is a *sine qua non*⁴ and, to be practical, an algorithm has to terminate. However, if we take the right attitude to possibility distributions, incompleteness need not be a disaster. This suggests, therefore, a framework for a research effort directed at supporting the full FOPC as a programming language for knowledge-based applications. Applications are cast as Class 1 modeling problems, with salient parameters being treated as constant symbols in an FOPC theory which represents the currently available knowledge about the application area. Users of an application program interact with the program by augmenting the theory embedded in the program with assertions about their particular problem instance and by interpreting the computed marginal possibility distributions appropriately. The results of research leading to improved inferential competence can be incorporated into the run-time system for the language over time without altering application programs, in a fashion analogous to the way in which improvements to the run-time speed of a language interpreter have no impact on the correctness of programs in the language.

6 Galileo4 Revisited

As we have seen, Galileo4 is a programming language highly applicable to supporting production. On a more formal plane, Galileo4 is firmly rooted in the notion of constraint networks and on the relationship, discussed above, between such networks and possibility distributions for semantic

³A problem is said to be undecidable if it is not possible even in theory to devise an algorithm for solving it. A class of problems is undecidable if it contains at least one specific problem instance which is undecidable. For example, while many specific satisfiability problems in FOPC have algorithmic solutions, not all do, and so satisfiability in FOPC is undecidable.

⁴Unless unsound inferences are allowed in a principled way, by the introduction of non-monotonic common-sense assumptions.

models of theories written in the full FOPC. The run-time system for the language is a test-bed for various approaches to computing the intent of sets of well-formed FOPC formulae and to generalizing the consistency algorithms.

6.1 Galileo4 Programs

An application program in Galileo4 provides a declarative specification of a constraint network, analogous to the problem specifications in Example 1 and Example 2. That is, in general, an application program in Galileo4 specifies a first-order language $\mathcal{L} = \langle \mathcal{P}, \mathcal{F}, \mathcal{K} \rangle$, a theory Γ containing closed sentences from that language, a universe of discourse \mathcal{U} , and a partial interpretation function \mathcal{I}_p for \mathcal{L} . Of these, only the theory Γ must always be specified explicitly, because, in many situations, a default universe and a default partial interpretation provided by the run-time system are adequate, while the system can determine the language \mathcal{L} by computing the union of the constant symbols used in Γ with the vocabulary of constant symbols in the default language $\mathcal{L}_g = \langle \mathcal{P}_g, \mathcal{F}_g, \mathcal{K}_g \rangle$ provided by the run-time system. In \mathcal{L}_g , $\mathcal{K}_g = \mathcal{R}$ contains the real numeric strings, \mathcal{P}_g contains names of standard predicates ($=$, $=<$, etc.), and \mathcal{F}_g contains names of standard functions ($*$, $+$, etc.). The run-time system also provides a default universe of discourse $\mathcal{U}_g = \mathfrak{R}$ and an interpretation function \mathcal{I}_g for \mathcal{L}_g giving the standard interpretations in terms of \mathcal{U}_g . For example, the constant symbols in \mathcal{K}_g are mapped onto the intended finite-length rational numbers with interpretations such as $1 \mapsto 1$ and $2.1 \mapsto 2.1$.

The language $\mathcal{L} = \langle \mathcal{P}, \mathcal{F}, \mathcal{K} \rangle$ defined by an application program in Galileo4 has the following components: $\mathcal{P} = \mathcal{P}_g \cup \{\text{predicate symbols defined in the program}\}$; $\mathcal{F} = \mathcal{F}_g \cup \{\text{function symbols defined in the program}\}$; $\mathcal{K} = \mathcal{K}_g \cup \{\text{constant symbols used in the program}\}$. The universe of discourse \mathcal{U} defined by a Galileo4 program is the union of $\mathcal{U}_g = \mathfrak{R}$ with any application-specific domains that are defined in the program. The partial interpretation function \mathcal{I}_p defined by the program is the union of the set of mappings in \mathcal{I}_g with any application-specific interpretations that are provided in the program.

6.2 The Galileo4 Run-Time System

The Galileo4 run-time system attempts to compute the marginal possibility distribution Π_x , the projection of $\Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}}$ on the dimension x , for each parameter x in \mathbf{X} in the network $\langle \mathcal{U}, \mathbf{X}, \mathbf{C} \rangle$ that is defined by a Galileo4 program. In general, it is only possible to compute $\tilde{\Pi}_x$, an approximation to Π_x such that $\Pi_x \subseteq \tilde{\Pi}_x$. However, in many networks the run-time system can compute Π_x for all x in \mathbf{X} .

6.2.1 Propagation Algorithm

An important aspect of our ongoing research consists of the development of constraint propagation algorithms which compute better approximations to the marginal possibility distributions for parameters in infinite domain constraint networks. The propagation algorithm which is currently used to compute marginal possibility distributions is called Compound Constraint Propagation (CCP). Its top-level is shown in Figure 5.

Full details of the algorithm are beyond the scope of this chapter. However, as shown in Figure 5, CCP involves interleaved application of three inference techniques: LPKS, local propagation of known states [20]; GAC, a version of arc consistency [21], generalized, along the lines suggested in Section 5.5, to infinite domains and constraints of arbitrary arity; GPC, a form of path consistency [21], generalized to infinite domains and constraints of arbitrary arity, which is only applied to small portions of the network in certain very specific circumstances.

```

procedure CCP( $\Gamma$ );
localvar  $Q_{lp}, Q_{gac}, Q_{gpc}$ ;
begin
 $Q_{lp} \leftarrow \Gamma$ ;
 $Q_{gac} \leftarrow \emptyset$ ;
 $Q_{gpc} \leftarrow \emptyset$ ;
repeat
  LPKS( $Q_{lp}, Q_{gac}, Q_{gpc}$ );
  if  $Q_{gac} \neq \emptyset$ 
    then repeat
      GAC( $Q_{lp}, Q_{gac}, Q_{gpc}$ );
      if  $Q_{lp} = \emptyset \wedge Q_{gpc} \neq \emptyset$ 
        then GPC( $Q_{gac}, Q_{gpc}$ )
      until  $Q_{lp} \neq \emptyset \vee Q_{gac} = \emptyset$ 
until  $Q_{lp} = \emptyset$ 
end

```

Figure 5: Top Level of the CCP Algorithm

The basic idea behind the CCP algorithm is to use a set of complementary inference techniques, with the computationally cheapest technique being used whenever possible. Thus, all constraints are initially placed in Q_{lp} , the local propagation queue, so that local propagation of known states (LPKS), a relatively inexpensive technique is tried first. When LPKS is unable to draw any further inferences, the constraints at which LPKS reached a dead-end are placed on the generalized arc consistency queue Q_{gac} . If arc consistency results in restricting the domain of possible values for any parameter to a singleton set, then all constraints which reference this parameter are placed by GAC into Q_{lp} , GAC terminates and LPKS is re-invoked. If arc consistency fails to draw any conclusions, then GPC is invoked at those constraints where arc-consistency reached a dead-end. However, the GPC procedure only applies path consistency if certain conditions are satisfied, the details of which are beyond the scope of this chapter. Whenever path consistency succeeds in making any constraint more specific, arc consistency is re-invoked at the arcs involved in this constraint.

The various techniques are used to overcome each others' deficiencies. Thus, for example, arc consistency can push inference through inequality constraints, which is beyond the competence of local propagation of known states. Similarly, path consistency is used to overcome what is probably the best-known deficiency of local propagation of known states: its inability to penetrate so-called "constraint cycles" [20]. Path consistency is also used to overcome an important deficiency of arc consistency: its tendency to fall into infinite loops when processing parameters which range over (infinite subsets of) \mathfrak{R} [9].

Within GAC and GPC, conditional reductions [1] are used for manipulating the intensional formulae that arise during the application of arc and path consistency criteria to the sets which represent marginal and joint possibility distributions. A reduction is a rewrite rule which changes all subformulae of a certain form into a target form. A conditional reduction is a reduction with an associated guard condition; the reduction is applicable only if the guard condition is true. A reduction is *semantic* in that its use is based not upon the syntactic structure of the form to be

rewritten but upon its meaning; a guard condition is also semantic, in that it is decided by applying the model-theoretic rules for truth to the interpretations of the symbols which it references, rather than by the application of syntactic proof theory. Consider, for example, the following conditional reduction:

$$X < A \wedge X > B \xrightarrow{A \leq B} \perp$$

This can be applied to the intensional formula in the possibility distribution

$$\{X \mid X < 5 \wedge X > 6\}$$

because model theory indicates that the guard condition $5 \leq 6$ is true; as a result of applying the reduction, it is recognized that the possibility distribution is the empty set.

It is a subject of ongoing research to determine just what is the most appropriate collection of reductions for use in GAC and GPC. Nonetheless, the sets currently in use can be proved to allow no infinite reductions. This means that, although GAC and GPC cannot be guaranteed to find the minimal representation of an intensional specification, they are guaranteed to terminate.

6.2.2 Monotonicity, Soundness, and Completeness

The CCP algorithm is a monotonic filter. It operates by removing from the marginal possibility distribution for each parameter in the network those values which it is able to prove cannot appear in any tuple of the true intent $\Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}}$ of the network. The algorithm does not treat negation as failure or make any other non-monotonic assumptions. Thus, a candidate value v for a parameter x in \mathbf{X} is removed from $\tilde{\Pi}_x$ only when it can be proven that $v \notin \Pi_x$.

The CCP algorithm is sound: it can be shown, although the proof is beyond the scope of this chapter, that Π_x the true intensional marginal possibility distribution for a parameter x is guaranteed to be contained within the distribution $\tilde{\Pi}_x$ which the algorithm computes for x . If CCP were complete, the true intensional distribution for a parameter would be exactly that computed; that is, $\tilde{\Pi}_x = \Pi_x$, for all x in \mathbf{X} . But CCP is not guaranteed to be complete, so the intensional distribution for a parameter may sometimes be a proper subset of the computed distribution; that is, sometimes, for some x in \mathbf{X} , $\Pi_x \subset \tilde{\Pi}_x$.

By default, CCP is the only algorithm which is applied to a constraint network. However, the user may also, at his discretion, specify that backtracking search should be employed. Thus, although the proof is beyond the scope of this chapter, it can be shown that, in problems where each x in \mathbf{X} is subject to at least one positive literal whose predicate symbol has a finite interpretation, the Galileo4 run-time system can guarantee to compute not just $\tilde{\Pi}_x$ but Π_x for all x , albeit with exponential computational cost.

6.2.3 User Interaction

Galileo4 programs are interactive. The user is allowed to augment the theory defined in a program with additional sentences and, whenever he does so, the Galileo4 run-time system uses CCP to compute revised possibility distributions.

The system maintains a set of dependency records and, at any time, the user can request a justification for the current computed marginal possibility distribution $\tilde{\Pi}_x$ of any parameter x in the network.

Whenever, as a result of a sequence of user assertions, the system computes that $\tilde{\Pi}_x = \emptyset$ for any x , the system reports a violation, since $\tilde{\Pi}_x = \emptyset$ implies $\Pi_x = \emptyset$, which implies that $\Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}} = \emptyset$. If this happens, the user can recover from the situation by retracting one of his earlier assertions.

This ability to unpeel the consequences of a previous assertion is enabled by the same dependency records which enable the provision of justifications. By making assertions and retractions in any order he pleases, the user can explore various “what if” scenarios.

6.2.4 Backward- and Forward-chaining

The system supports both forward- and backward-chaining reasoning. The user can volunteer assertions to augment a theory, in which case the system will use CCP to forward-chain on the consequences of each assertion. Alternatively, the user can ask the system to determine the value of some parameter in the network, in which case the system will backward-chain through the sentences that correspond to the constraints in the network, asking the user for information about other parameters in the network.

During backward-chaining, whenever the system asks for information about a parameter, the user can input any well-formed assertion. At its simplest, this could be a ground equation giving the exact value for the parameter; e.g., the assertion $a = 2$ enables the system to infer the interpretation $a \mapsto 2$, since \mathcal{I}_g contains both the interpretation $2 \mapsto 2$, and the standard interpretation for equality. Alternatively, the user’s assertion could be any well-formed sentence (atomic, compound or quantified) in the first-order language \mathcal{L} . In particular, since the equality predicate need not be used; the user can provide whatever information he has about the parameter, even if he does not know the parameter’s value.

The fact that inference in Galileo4 is based on the computation of possibility distributions for parameters means that the system’s inferential competence degrades gracefully whenever insufficient information is obtained during backward-chaining. Thus, if the user invokes backward-chaining by asking the system to determine the value of some parameter x , but he is then unable to specify values for other parameters related to x , the system may not be able to infer a singleton value for x ; however, the system may still be able to compute a more restricted possibility distribution for x from whatever lower-grade information the user is able to provide about the other parameters.

6.2.5 Mixed-Initiative Interaction

The ability to accept any well-formed sentence during backward-chaining has an important added benefit: it supports mixed-initiative interaction. Whenever the system, during backward-chaining, asks for information about one parameter, the user can, if he sees fit, input an assertion that does not even reference the parameter which the system asked about. Since the system forward-chains on the consequences of each assertion before asking further questions, an experienced user who follows the system’s line of reasoning can short-circuit the process by providing information that the system has not asked for but which the user knows is relevant.

The user can determine the system’s line of reasoning in the usual way, by asking the system to justify itself whenever it asks a question during backward-chaining. To provide a justification, the system merely outputs the sentences which link the parameter about which information is being sought to the parameter whose value the user asked the system to determine.

6.3 Simple Example Programs

In Section 3, we will provide a practical application program written in Galileo4. Initially, however, in this section, we build on the discussion in Section 5.4 by providing programs that correspond to the abstract problems discussed there.

```

relation s ::= {5,7}.
relation t ::= { (1,4), (2,5), (3,6), (4,7), (5,1),
                (5,3), (6,2), (6,3), (6,7), (7,1), (7,3) }.
constant b ::= 7.
t(a,c).
all X : s(X) implies t(X,a).
t(c,b).

```

Figure 6: Galileo4 Program 1

6.3.1 Program 1

The Galileo4 program corresponding to Example 1 is shown in Figure 6.

The last three statements in this program define the theory given in Example 1. The first three statements in the program, in conjunction with the set of constant symbols used in the theory, define the first-order language $\mathcal{L} = \langle \mathcal{P}, \mathcal{F}, \mathcal{K} \rangle$ whose components are as follows: $\mathcal{P} = \mathcal{P}_g \cup \{s, t\}$; $\mathcal{F} = \mathcal{F}_g$; $\mathcal{K} = \mathcal{K}_g \cup \{a, b, c\}$. Since the program defines no application-specific domains, the universe of discourse is just the default; that is $\mathcal{U} = \mathcal{U}_g = \mathfrak{R}$. As well as contributing to the definition of \mathcal{L} , the first three statements in the program also specify that the partial interpretation function \mathcal{I}_p is the union of the default partial interpretation \mathcal{I}_g with the set of mappings for s , t and b given in the problem statement of Example 1.⁵

If the Galileo4 run-time system were given this program, it would report that $\tilde{\Pi}_a$ was $\{1, 3\}$ and that $\tilde{\Pi}_c$ was $\{4, 6\}$. As in Example 1, if the user were to assert $t(b, c)$ when interacting with this program, the run-time system would report a constraint violation, since now $\tilde{\Pi}_c$ would be \emptyset . If the user then retracted $t(b, c)$ and replaced it by $t(c, a)$, the system would compute that $\tilde{\Pi}_a$ was $\{3\}$ and that $\tilde{\Pi}_c$ was $\{6\}$ and would report that $a \mapsto 3$ and $c \mapsto 6$.

If instead of volunteering assertions, the user were to invoke backward-chaining by asking the system to determine the value of a , the system would respond by asking the user for the value of c . If, in response, the user were to assert that $c = 4$, the system would infer that $c \mapsto 4$ and $a \mapsto 1$; if, however, the user were to assert that $c > 5$, the system would infer that $c \mapsto 6$ and $a \mapsto 3$.

6.3.2 Program 2

A Galileo4 program corresponding to Example 2 is shown in Figure 7.

The first statement in the program defines an application-specific domain. The universe of discourse \mathcal{U} is the union of \mathcal{U}_g with all application-specific domains, so

$$\mathcal{U} = \mathfrak{R} \cup \{g1, g2, g3, g4\}.$$

⁵Actually, because of the default language \mathcal{L}_g defined by the Galileo4 run-time system, there are two differences between the situation defined by this program and that defined in Example 1. First, the language \mathcal{L} defined by this program is a superset of the language in Example 1; second, the partial interpretation function defined by this program is a superset of the function defined in Example 1. Despite these differences, however, the constraint networks of Example 1 and Program 1 are identical, for three reasons: first, the theories are identical; second, these theories contain the same set of uninterpreted constant symbols; third, the partial interpretation functions provide the same mappings for all symbols in the theory that they do interpret.

```

domain g ::= {g1,g2,g3,g4}.
relation j ::= {X : positive(X)}.
relation k ::= {X : negative(X)}.
relation h ::= {X : 0 < X =< 10}.
function l(g) -> positive
    ::= {(g1,200),(g2,250),(g3,350),(g4,390)}.
function m ::= {(g1,230),(g2,190),(g3,240),(g4,245)}.
positive(a).
h(b).
positive(c).
g(e).
d = a * f.
j(f) implies f =< l(e).
k(f) implies f =< m(e).
all X : g(X) implies m(X) =< l(e).
exists X : l(X) > l(e).
a = b * c.

```

Figure 7: Galileo4 Program 2

As well as introducing an entity to the universe of discourse, each member of an application-specific domain also introduces a constant symbol to the first-order language and introduces to the partial interpretation a mapping from the constant symbol to the entity. Thus, as well as introducing the four entities $g1$, $g2$, $g3$ and $g4$ to \mathcal{U} , the first statement in this program also introduces the four symbols $g1$, $g2$, $g3$ and $g4$ to the constant symbol vocabulary of \mathcal{L} and the four mappings $g1 \mapsto g1$, $g2 \mapsto g2$, $g3 \mapsto g3$ and $g4 \mapsto g4$ to \mathcal{I}_p .

A domain declaration also introduces the domain name into the first-order language, as a predicate symbol, and specifies that the set of domain contents constitute the interpretation for this symbol. Thus, the first statement in this program also introduces the following mapping to \mathcal{I}_p

$$g \mapsto \{g1, g2, g3, g4\}.$$

The next five statements define application-specific function and predicate symbols; both extensional and intensional specifications are used.

The last ten statements specify the following theory:

$$\Gamma = \{ \begin{array}{l} \text{positive}(a), h(b), \text{positive}(c), g(e), \\ d = a * f, j(f) \Rightarrow f \leq l(e), \\ k(f) \Rightarrow f \leq m(e), \\ (\forall X)(g(X) \Rightarrow m(X) \leq l(e)), \\ (\exists X)(g(X) \wedge l(X) > l(e)), \\ a = b * c \end{array} \}.$$

Note that, as shown in the definition for the function l , definitions for predicate and function symbols optionally can include type information about the arguments (and, for functions, about the value returned). This type information facilitates processing the theory specified by a program. Thus, for example, the type information that the function l takes arguments of type g enables the system, when implementing the statement

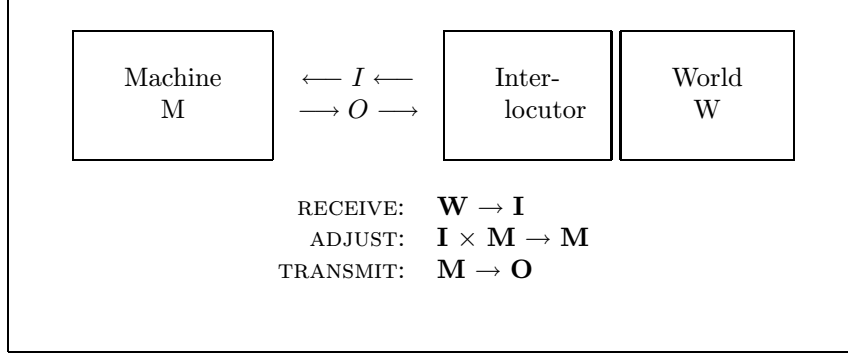


Figure 8: Machine-interlocutor interaction

exists $X : l(X) > l(e)$

to introduce a relativizing predicate:

$$(\exists X)(g(X) \wedge l(X) > l(e)).$$

This relativizing predicate enables the system to treat the quantified variable as ranging over the finite set $\{g1, g2, g3, g4\}$, instead of over the infinite set \mathcal{U} .

As a consequence of the domain, function and predicate symbol definitions, and of the constant symbols used in the theory, the language \mathcal{L} and the partial interpretation \mathcal{I}_p defined by this program are as follows:

$$\begin{aligned}
 \mathcal{L} = & \langle \mathcal{P}_g \cup \{g, j, k, h\}, \mathcal{F}_g \cup \{l, m\}, \mathcal{K}_g \cup \{a, b, c, d, e, f, g1, g2, g3, g4\} \rangle \\
 \mathcal{I}_p = & \mathcal{I}_g \cup \{ \\
 & g \mapsto \{g1, g2, g3, g4\}, j \mapsto \mathfrak{R}^+, \\
 & k \mapsto \mathfrak{R}^-, h \mapsto \{X \mid X \in \mathfrak{R}^+ \wedge \text{LEQ}(X, 10)\}, \\
 & l \mapsto \{\langle g1, 200 \rangle, \langle g2, 250 \rangle, \langle g3, 350 \rangle, \langle g4, 390 \rangle\}, \\
 & m \mapsto \{\langle g1, 230 \rangle, \langle g2, 190 \rangle, \langle g3, 240 \rangle, \langle g4, 245 \rangle\}, \\
 & g1 \mapsto g1, g2 \mapsto g2, g3 \mapsto g3, g4 \mapsto g4 \quad \}.
 \end{aligned}$$

As with Program 1, a user interacting with this program can solve a variety of problems by making various assertions and observing the consequent marginal possibility distributions, or by invoking backward-chaining to determine the value or restricted possibility distribution for some parameter.

7 Human-Machine Interaction in Galileo4

Human-machine interaction in Galileo4 can be described in terms of the view of machine-interlocutor interaction model in Figure 8. In this view, the universe of discourse (set of all possible values for all parameters) defined in a Galileo4 application program constitutes the World. User inputs to the machine contain descriptions about the state of the World, expressed as sentences in the first-order language \mathcal{L} which is defined in the Galileo4 application.

In our constraint-based approach to logic, the machine state \mathbf{M} at any time is characterized by a set of possible conceptualizations of the World. A conceptualization of the World comprises the universe of discourse \mathcal{U} and those interrelationships between the entities in \mathcal{U} that the machine finds interesting. There may be many other interrelationships in which the machine is uninterested. Indeed, since, as a first-order language, \mathcal{L} has finite predicate and function vocabularies, this has

to be the case when \mathcal{U} is infinite: the number of possible q -ary relations between the entities of \mathcal{U} is $2^{|\mathcal{U}|^q}$.

If the machine has a total model $\mathcal{M} = \langle \mathcal{U}, \mathcal{I} \rangle$ for the language \mathcal{L} , (that is, if \mathcal{I} provides a mapping for each symbol in the union of the symbol vocabularies in \mathcal{L}) then the machine knows everything worth knowing about (that is, it has a total conceptualization of) the World.

Giving inputs to the machine about the state of the World has a purpose only if the machine does not know all the interrelationships that exist between the entities of \mathcal{U} , that is, if the machine does not have a total model of the language \mathcal{L} or if its model is not an accurate depiction of the state of the World.

When the machine does not have a total model for \mathcal{L} , the state of the machine (the set of possible conceptualizations of the World) is the set of possible models for \mathcal{L} . That is, if at some time t the machine's partial model of \mathcal{L} is $\mathcal{M} = \langle \mathcal{U}, \mathcal{I}_t \rangle$ then the machine state M_t is the set of all total models $\mathcal{M} = \langle \mathcal{U}, \mathcal{I} \rangle$ for \mathcal{L} , $\mathcal{I}_t \subset \mathcal{I}$, which are consistent with the information about the state of the World that the machine has been given. Thus, if at time t the information that the machine has been given about the state of the World is a set Γ_t of sentences, then the state M_t of the machine is the set of all total models $\langle \mathcal{U}, \mathcal{I} \rangle$, $\mathcal{I}_t \subset \mathcal{I}$, which are consistent with the truth of all sentences in Γ_t .

Since the state of the World, or the interlocutor's perception of it, may change, the information about the World which the machine is given may not be monotonic. One input to the machine may say that a certain sentence γ of \mathcal{L} should be taken as true and a later input may say that γ should no longer be taken as true. This means that the inputs to the machine are not sentences of \mathcal{L} ; rather, they are *about* sentences of \mathcal{L} . The two inputs just mentioned could not be just γ and $\neg\gamma$; that would lead to a contradiction in the theory that the machine has about the World. Rather, the two inputs must be meta-level sentences, which might be expressed imperatively as ASSERT γ and RETRACT γ or declaratively as $\Gamma_{t+1} = \Gamma_t \cup \{\gamma\}$ and $\Gamma_{t+2} = \Gamma_{t+1} - \{\gamma\} = \Gamma_t$.

This form of machine-interlocutor interaction is what happens when a user interacts with a Galileo4 application program. The program defines the language \mathcal{L} , the universe of discourse \mathcal{U} and an initial partial interpretation \mathcal{I}_p . The program also primes the machine with an initial set Γ of \mathcal{L} sentences about the World.

8 Summary

In this chapter, we have introduced a generalized notion of constraint processing. We have shown that this generalized notion corresponds to a class of modeling problem in logic. We have then used this correspondence as the theoretical basis for a computational approach to supporting the full first-order predicate calculus as a knowledge representation language. In this approach, constraint propagation is used to treat inference as semantic entailment rather than syntactic theorem-proving.

We have introduced Galileo4, a new constraint programming language based on these ideas and shown how short constraint-based application programs can have high problem-solving versatility.

We have discussed a new algorithm for constraint propagation which generalizes and integrates several of the best existing techniques. The algorithm interleaves local propagation of known states with forms of arc and path consistency which have been generalized to infinite domains and constraints of arbitrary arity. In the case of infinite domains, semantic reductions are used for entailment. However, we regard this algorithm as only the first in a sequence of progressively more competent and efficient algorithms. We regard Galileo4 as a test-bed for further developments in this area. Although no terminating constraint propagation algorithm can ever be both sound and complete for the full first-order predicate calculus, an incomplete algorithm is adequate for some

applications, as we have illustrated by means of examples from the realms of design for testability and equipment selection.

We have shown how the man-machine interface presented by Galileo4 is a special case of Nilsson's view of machine-world interaction. We believe that the lexical economy offered by the full first-order predicate calculus and the problem-solving versatility offered by a constraint-based approach to inference mean that the generalized form of constraint-based programming that we have introduced in this chapter is destined to become the declarative paradigm of choice.

However, this constraint-based approach to supporting the full first-order predicate calculus is still relatively immature and much remains to be done. Notable open problems include: the development of constraint propagation algorithms which are more efficient and which compute better approximations to the true intent of infinite domain constraint networks; extending the object-level language to include modal operators; supporting application-specific sets of reductions and the use of results such as the Knuth-Bendix theorem [18] to compute complete reduction sets from the application-specific sets proposed by application programmers.

Several other related issues also present themselves. One involves the extension of the constraint-based approach to problems having non-scalar universes of discourse, where constant symbols can be interpreted to sets or sequences; work here might include looking at issues of non-monotonic inference. Other work might be directed at using second-order and meta-level notions to ease the programming of large-scale applications. The specification of partial interpretations can be eased by allowing the interpretations of predicate symbols to be specified as the reflexive, symmetric, or transitive closures of interpretations for other symbols. The specification of a theory in a program, or of augmentations to the theory by a user, can be eased by using meta-level notions to allow these theories and augmentations to be specified intensionally rather than extensionally; in addition, using such meta-level notions in a constraint-based approach to logic makes it possible to treat (the declarative aspects of) frame-based programming as a special case of theory specification.

References

- [1] Bledsoe, W. (1977) Non-resolution Theorem Proving. *Artificial Intelligence*, **9**, 1-35.
- [2] Bowen, J. and Bahler, D. (1991) Conditional Existence of Variables in Generalized Constraint Networks. *Proc. 9th. Nat. Conf. on Artificial Intelligence (AAAI-91)*, 215-220.
- [3] Bowen, J. and Bahler, D. (1991) Supporting Cooperation between Multiple Perspectives in a Constraint-based approach to Concurrent Engineering. *Journal of Design & Manufacturing*, **1**, 89-105.
- [4] Bowen, J. and Bahler, D. (1992) Frames, quantification, perspectives and negotiation in constraint networks for life-cycle engineering. *Int. Journal of AI in Engineering*, **7**, 199-226.
- [5] Bowen, J. and Bahler, D. (1993) Constraint-Based Software for Concurrent Engineering. *IEEE Computer*, Special Issue on Computer Support for Concurrent Engineering, **26** (1), 66-68.
- [6] Borning, A. (1981) The Programming Language Aspects of ThingLab, A Constraint-Oriented Simulation Laboratory. *ACM Trans. on Prog. Lang. and Sys.*, **3** (4), 353-387.
- [7] Chronaki, C. and Baldwin, D. (1990) A Front End for CONSUL. Technical Report 319, Department of Computer Science, University of Rochester.
- [8] Colmerauer, A. (1990) An Introduction to Prolog III. *Commun. ACM*, **28** (4), 412-418.
- [9] Davis, E. (1987) Constraint Propagation with Interval Labels. *Artificial Intelligence*, **32**, 281-331.
- [10] Dechter, R. (1990) Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition. *Artificial Intelligence*, **41**, 273-312.

- [11] Dincbas, M., van Hentenryck, P., Simonis, H., Aggoun, A., Graf, T. and Berthier, F. (1988) The Constraint Logic Programming Language CHIP. *Proc. Fifth Gen. Comp. Sys.* '88.
- [12] Faltings, B. (1994) Arc-consistency for continuous variables. *Artificial Intelligence*, **65** (2), 363-376.
- [13] Freuder, E. (1982) A Sufficient Condition for Backtrack-free Search. *Journal of the ACM*, **29** (1), 24-32.
- [14] Gosling, J. (1983) Algebraic Constraints. Technical Report CS-83-132, Carnegie-Mellon University, Pittsburgh.
- [15] Han, C.-C. and Lee, C.-H. (1988) Comments on Mohr and Henderson's Path Consistency Algorithm. *Artificial Intelligence*, **36**, 125-130.
- [16] Haralick, R. and Elliot, G. (1980) Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, **14**, 263-313.
- [17] Jaffar, J., Michaylov, S., Stuckey, P., and Yap, R. (1992) The CLP(\Re) Language and System. *ACM Trans. on Prog. Lang. and Systems*, **14** (3), 339-395.
- [18] Knuth, D. and Bendix, P. (1970) Simple Word Problems in Universal Algebras. in *Computational Problems in Abstract Algebra*, (ed. J. Leech), Pergamon Press, New York.
- [19] Lambert, K. and van Fraassen, B. (1972) *Derivation and Counterexample: An Introduction to Philosophical Logic*, Dickenson Publishing Company, Enrico, CA.
- [20] Leler, W. (1988) *Constraint Programming Languages*, Addison-Wesley, Reading, MA.
- [21] Mackworth, A. (1977) Consistency in Networks of Relations. *Artificial Intelligence*, **8**, 99-118.
- [22] Maher, M. and Stuckey, P. (1989) Expanding Query Power in Constraint Programming Languages. *Proc. North American Conf. on Logic Programming*, 20-36.
- [23] Mittal, S. and Falkenhainer, B. (1990) Dynamic Constraint Satisfaction Problems. *Proc. 8th. Nat. Conf. on Artificial Intelligence (AAAI-90)*, 25-32.
- [24] Mohr, R. and Henderson, T. (1986) Arc and Path Consistency Revisited. *Artificial Intelligence*, **28**, 225-233.
- [25] Nadel, B. (1989) Constraint Satisfaction Algorithms. *Computational Intelligence*, **5** (4), 188-224.
- [26] Nilsson, N. (1991) Logic and artificial intelligence. *Artificial Intelligence*, **47**, 31-56.
- [27] Paramasivam, M. (1991) An Axiom Based Approach to Constraint Satisfaction. M.S. Thesis, Computer Science Department, North Carolina State University, Raleigh.
- [28] Sussman, G. and Steele, G. (1980) Constraints: A Language for Expressing Almost-Hierarchical Descriptions. *Artificial Intelligence*, **14**, 1-39.
- [29] Sutherland, I. (1963) SKETCHPAD: A Man-Machine Graphical Communication System. *Proc. Spring Joint Comp. Conf.*, IFIPS.

A Proof of Theorem 1

The proof of this theorem depends on the following lemma:

Lemma

Given $\mathcal{L} = \langle \mathcal{P}, \mathcal{F}, \mathcal{K}' \cup \mathcal{K}'' \rangle$, \mathcal{U} , \mathcal{I}_p and γ , where \mathcal{I}_p interprets, in terms of \mathcal{U} , all and only the symbols in $\mathcal{P} \cup \mathcal{F} \cup \mathcal{K}'$, \mathcal{K}'' is finite, and the sentence γ references at least one symbol in \mathcal{K}'' . Let \mathbf{X} be the lexical ordering of the elements of \mathcal{K}'' and let $C(T)$ be a constraint such that T is the lexical ordering of those elements of \mathcal{K}'' which appear in γ and such that $\langle \mathcal{U}, \mathcal{I}_p \cup \vec{T}t \rangle \models \gamma$ for all tuples $t \in C(T)$. Then $\langle \mathcal{U}, \mathcal{I}_p \cup \vec{\mathbf{X}}\tau \rangle \models \gamma$ for all tuples $\tau \in E(\mathbf{X})$ where $E(\mathbf{X})$ is the cylindrical extension of $C(T)$ into the space defined by \mathbf{X} .

Proof

Take any tuple $\tau \in E(\mathbf{X})$. The projection of the singleton set $\{\tau\}$ on T will be a singleton set containing some tuple τ' ; that is, $proj(\{\tau\}, T) = \{\tau'\}$. Then, by the definition of cylindrical extension, $\tau' \in C(T)$. The mapping function $\vec{\mathbf{X}}\tau$ provides a mapping for each constant symbol in \mathcal{K}'' . However, the truth of γ depends only on the interpretation of those members of \mathcal{K}'' which appear in γ . Thus, the truth of γ depends only on the mappings provided by $\vec{T}\tau'$, which is a subset of $\vec{\mathbf{X}}\tau$. Since $\tau' \in C(T)$, $\langle \mathcal{U}, \mathcal{I}_p \cup \vec{T}\tau' \rangle \models \gamma$. Since $\langle \mathcal{U}, \mathcal{I}_p \cup \vec{T}\tau' \rangle \models \gamma$, $\langle \mathcal{U}, \mathcal{I}_p \cup \vec{\mathbf{X}}\tau \rangle \models \gamma$, because the additional mappings in $(\vec{\mathbf{X}}\tau - \vec{T}\tau')$ have no impact on the truth of γ . Since τ is any tuple in $E(\mathbf{X})$, $\langle \mathcal{U}, \mathcal{I}_p \cup \vec{\mathbf{X}}\tau \rangle \models \gamma$ for any tuple $\tau \in E(\mathbf{X})$. Q.E.D.

Proof of Theorem 1

Since $\langle \mathcal{U}, \mathcal{I}_p \cup \vec{T}_1 t \rangle \models \gamma_1$ for all tuples $t \in C_1(T_1)$ then, by the lemma, $\langle \mathcal{U}, \mathcal{I}_p \cup \vec{\mathbf{X}}\tau \rangle \models \gamma_1$ for all tuples $\tau \in E_1(\mathbf{X})$, the cylindrical extension of $C_1(T_1)$. Similarly, $\langle \mathcal{U}, \mathcal{I}_p \cup \vec{\mathbf{X}}\tau \rangle \models \gamma_2$ for all tuples $\tau \in E_2(\mathbf{X})$, the cylindrical extension of $C_2(T_2)$. Therefore, $\langle \mathcal{U}, \mathcal{I}_p \cup \vec{\mathbf{X}}\tau \rangle \models \{\gamma_1, \gamma_2\}$ for all tuples $\tau \in E_1(\mathbf{X}) \cap E_2(\mathbf{X})$. Similarly, $\langle \mathcal{U}, \mathcal{I}_p \cup \vec{\mathbf{X}}\tau \rangle \models \{\gamma_1, \gamma_2, \dots, \gamma_r\}$ for all tuples $\tau \in E_1(\mathbf{X}) \cap E_2(\mathbf{X}) \cap \dots \cap E_r(\mathbf{X})$. That is, $\langle \mathcal{U}, \mathcal{I}_p \cup \vec{\mathbf{X}}\tau \rangle \models \Gamma$ for all tuples $\tau \in \Pi_{\mathcal{U}, \mathbf{X}, \mathbf{C}}$. Q.E.D.