Joint NCC & IIS Bull., Comp. Science, 12 (1999), 58–61

Integration of constraint programming methods with knowledge base technology*

Yury A. Zagorulko, Ivan G. Popov

An approach to integration of methods of constraint programming with various knowledge representation means, such as frames, semantic networks, and production rules, is considered. A knowledge representation language developed on the basis of this approach is presented. In contrast to other languages that also use the constraint programming technique, this language allows us to operate with imprecisely defined (subdefinite) values and to change the initial set of constraints. The language based on the proposed approach can be used for developing intelligent systems which require the combination of logical inference and computations over imprecise values.

Introduction

The paradigm of constraint programming has recently become very popular. The methods developed in this framework allow one to specify knowledge in the form of a set of constraints for the values of parameters of some objects. This approach is particularly convenient when the constraints can be represented by ordinary Boolean expressions. Since these methods allow one to solve a relatively narrow class of problems, they are usually used in combination with other tools rather than on their own. For example, methods of constraint programming are built into imperative languages [1], as well as into the languages for logic and functional programming [2, 3].

Application of this approach to AI languages appears to have a future. The methods of constraint programming, while usually used in numerical computations, nicely complement the traditional AI tools oriented at symbolic processing and declaration knowledge representation.

One shortcoming of constraint programming techniques restricting their range of application is that they deal with the static set of constraints. This means that only the values of the variables connected by constraints can change during computation, but the constraints themselves remain constant. However, in many problems of artificial intelligence, one does not need just to find values satisfying the conditions of the problem, but also to analyze all such solutions and modify the set of constraints during the computation.

While some attempts to control the set of constraints are being made (for example, through defining constraint hierarchies or assigning priorities to constraints [4, 5]), this ordering of constraints mostly helps one to use a set of constraints which is inconsistent, or it is a way to improve efficiency of a constraint-based system. Even though the set of constraints gets a certain structure, it is still static.

In this paper we consider a knowledge representation language which includes constraint programming techniques as well as some more conventional methods for representation and processing of knowledge and data. By the use of the object-oriented approach, this language allows us to manipulate objects and relations which naturally include constraints among their properties, as well as to modify dynamically the initial set of constraints. Another important feature of the language is the ability to manipulate objects that have slots with incomplete or subdefinite values [6]. The set of constraints for these slots ensures automatic refinement of their values. The method of subdefinite computational models [6, 7] is used to implement the mechanism of constraint satisfaction.

*Supported by the Russian Foundation for Basic Research under Grant N 96-01-01607.

1. The method of subdefinite computation models

The main idea of the method of subdefinite computation models is as follows: the variables of the problem to be solved are associated with subdefinite values (SD-values) which determine their domains of possible values (sets or intervals). These values are linked by constraints presented in the form of ordinary logical expressions. Interpretation of constraints with the help of a data-driven algorithm allows us to refine the subdefinite values that they connect. Thus, by defining a set of constraints (i.e. a computation model) we can obtain more or less precise values of the sought variables which satisfy the constraints.

Formally, *a constraint* is a boolean expression $C(v_1, ..., v_n)$ that is required to be true. The variables $v_1, ..., v_n$ linked by the constraint may get values of any subdefinite data types.

Each constraint must have *functional interpretations.* This means that the constraint can be represented by a set of functions called *interpretation functions.* Each of these functions allows us to calculate the value of one variable from the values of the other ones.

A *subdefinite computational model* (*SD-model*) is represented by a bipartite oriented graph (*subdefinite functional network* or *SD-network*) and a discipline of its processing, or *data-driven computations.* There are two types of vertices in an SD-network: variables and interpretation functions. The incoming edges of a function vertex connect it to the variables whose values are input arguments of the function. Outgoing edges of the function vertex point to variables which store the results produced by the function.

The principle of data-driven computations means that a change of the value of variable vertices causes execution of the function vertices; execution of the function vertices, in turn, may cause a change of the resulting value of variable vertices, and so on. If at least one variable gets an inconsistent value, the process will stop and the *SD-model* will be considered inconsistent.

During the process of interpretation, a new value of the variable is calculated and intersected with the old one. Since it is the result of this intersection that is assigned to the variable, its value can be only refined, i.e. the new value is a subset of the old one. It should be noted that the value of the variable is considered to be changed only if it is actually refined.

As it was shown in [7], for all data types containing only a finite set of SD-values, this algorithm terminates in a finite number of steps. In the case of infinite sets of SD-values (for instance, intervals of real values), the stopping criterion can be based on the preset threshold of computation accuracy ϵ. This threshold ϵ determines the maximum possible distance between two values for which they are still considered to be identical.

2. The knowledge representation language

Let us consider the main notions and facilities of the language based on an integrated knowledge representation model which unifies such means as frames, semantic networks, production rules, subdefinite computational models, and methods of constraint programming.

The base tool used to represent declarative knowledge in this language is a *semantic network* which consists of objects linked by binary relations.

An *object* can be any entity of the subject domain, defined by the knowledge engineer. Objects with the same properties are combined into one *class.* Classes may inherit properties of other classes (in this case, the former are called *subclasses*, and the latter are *superclasses*), with a possibility of multiple inheritance. Some properties of superclasses can be redefined in subclasses.

The properties of the class define the names and types of values of slots (attributes) of objects, possibly their default values, and the behavior of the object which is determined by the set of constraints.

The values of slots can be characters, strings, atoms, integer and real numbers, tuples and sets. An object may be the value of a slot, too. An important feature of objects is that their slots may be *subdefinite*, i.e., their values may be subsets of the domain of admissible values. Subdefinite values can be defined as intervals of values for numerical data types and as sets of possible values for other types.

The constraints associated with some object constitute its *local SD-network* and make it possible to automatically refine subdefinite values by means of the mechanism of constraint satisfaction. The method of subdefinite computation models is used to implement this mechanism.

Since the value of a slot can be an object, constraints can link the values of slots from several objects. The set of SD-networks of all objects presented in a semantic network constitutes *a global SD-network* which is activated for every modification of the objects; it ensures recalculation and modification of the values of slots of the related objects.

A *binary relation* is treated as a special object with two slots. We can define constraints for relations just as we do it for objects. We have selected binary relations as a particular class of relations because they have useful properties like reflexivity, symmetry, and transitivity which can be built into the system.

The language includes operations for creating objects and instances of binary relations (*new*), editing them (*edit*) and deleting them from the network (*delete*).

At the same timek, the global SD-network is modified with the edition of the semantic networ. So, the *new* operation creates objects and instances of binary relations and inserts them into the semantic network, and simultaneously adds constraints bound with the object or relation to the global SD-network. After objects or relations are deleted, all references to the deleted objects are replaced by the completely indefinite value '?' and all constraints related to the deleted objects and relations are removed from the global SD-network.

Thus, the global SD-network (the global set of constraints) can be modified during the application system operation as a consequence of both insertion of new objects and relations into the semantic network and edition of the existing ones. This takes place because the process of setting up new links among objects can lead to involving into the global SD-network new constraints which include slots of these objects.

It should be noted that the operations described above leads to the immediate activation and execution of the global SD-network until its stability is achieved.

Inference and information processing are specified in this language by means of describing properties of objects and relations and via a system of production rules over a semantic network, as well.

Production rules have a traditional format: a rule consists of a trigger condition and some actions to be performed when the condition is satisfied.

The trigger condition of a rule contains patterns for objects and relations, as well as predicates that define constraints for the context and the values of variables in the patterns.

The right-hand side of a rule may use operators that modify the semantic network, as well as other operators and functions. In particular, the language includes facilities for input/output and control of activation of production rules, which ensures high flexibility of control for the process of inference and information processing.

3. Controlling the set of constraints

Let us consider the manner of controlling the set of constraints.

Since local computation models are linked with objects and relations of the semantic network, the entire set of constraints is controlled through a small set of operations for creating, modifying, and deleting objects and relations. The order of changes is determined by production rules which are a traditional way of organizing inference in AI systems.

Since the modifications to the semantic network may produce inconsistencies in the global computation model, the language provides a means to perform a 'roll-back' to a previous state if such an inconsistency is found. To this end, it includes a mechanism of alternatives, which allows a knowledge engineer to mark the branching points determined by the specific subject domain and the application. If an inconsistency is found, the system returns to the nearest marked point and attempts to investigate another option. This method makes it possible to perform hypothesis checking and complete case analysis in a manner similar to backtracking.

This mechanism is implemented in the language by the statement of alternatives, which allows us to define each alternative as a sequence of valid statements.

Conclusion

The use of the object-oriented approach makes it possible to unify various means of knowledge representation such as frames, semantic networks, and production rules as well as methods of constraint programming in the framework of one language.

In contrast to other languages that also use the constraint programming technique, this language allows us to operate with subdefinite values and to change the initial set of constraints.

A user can bind to any class of objects a set of constraints defined on the values of their slots, and then these constraints will be used to automatically refine subdefinite values of the slots.

Since the constraints are linked with the objects and relations of the semantic network, they can be added to or deleted from the current set of constraints during the process of computation as a consequence of creation or deletion of these objects or relations.

The embedded mechanism of alternatives allows us to generate hypotheses and reject them if they lead to contradiction. With its help, for instance, we can find various choices of the exact values for the slots of a subdefinite object or try to link objects by relations in different ways. Additionally, this mechanism allows one to define a case analysis similar to backtracking.

The knowledge representation language based on the proposed approach can be used to create a broad class of intelligent systems which require the combination of logical inferences and computations over imprecise values. In addition, because the set of active constraints can be modified during the process of computation, this language can also be used for developing systems which can regard the dynamics of processes, in particular, systems for intelligent robot control.

References

[1] B.N. Freeman-Benson, A. Borning, *Integrating Constraints with an Object-Oriented Language*, Proc. of the Conf on Object-Oriented Programming, Marseille, 1992, 248–266.

[2] J. Cohenen, *Constraint Logic Programming Languages*, Comm. of the ACM, **33**, No 7, 1990, 52–68.

[3] M. Henz, G. Smolka, J. Wurtz, *Object-Oriented Concurrent Constraint Programming*, Oz. DFKI Research Report RR-93-16, 1993.

[4] B. Freeman-Benson, J. Malony, *Borning An Incremental Constraint Solver*, Comm. of the ACM, **33**, No 1, 1990, 54–63.

[5] A. Borning, B. Freeman-Benson, M. Willson, *Constraint Hierarchies, Lisp and Symbolyc Computation*, No 5, 1992, 223–270.

[6] A.S. Narin'yani, *Sub-definiteness and Basic Means of Knowledge Representation*, Computers and Artificial Intelligence, **2**, No 5, 1983, 443–452.

[7] V. Telerman, D. Ushakov, *Data Types in Subdefinite Models*, Jacques Calmet and others (eds.), Art. Intell. and Symbolic Mathematical Computation, Lect. Notes Comput. Sci., **1138**, 1996, 305–319.