Constraint-Based Knowledge Representation for Configuration Systems

Article · March 1996

Source: CiteSeer

CITATIONS
24

READS
369

2 authors, including:

Boi Faltings
École Polytechnique Fédérale de Lausanne
441 PUBLICATIONS **9,489** CITATIONS

SEE PROFILE

Constraint-based knowledge representation for configuration systems

Technical Report No. TR-94/59

Boi Faltings Rainer Weigel

Département d'Informatique
Laboratoire d'Intelligence Artificielle
Ecole Polytechnique Fédérale de Lausanne (EPFL)
IN-Ecublens, CH-1015 Lausanne
Switzerland

E-mail : { Faltings / Weigel } @di.epfl.ch

August, 1994

Abstract

Configuration is a design activity where the set of available components and their allowed combinations are known a priori. Configuration systems of today are typically rule-based systems. Maintenance and verification of rule-based system has shown to be extremely difficult because the knowledge relating to a single entity is spreading over several rules and modifications to rules must be checked for interactions with other rules. Since the configuration knowledge of a product evolves during the whole life-cycle modifications and extension of the knowledge must be facilitated in order to build productive systems. We show, how using a constraint-based approach for the configuration task makes this problem simpler.

1 Representing knowledge

By *knowledge*, we mean general truths which relate small sets of symbols. Examples of knowledge are:

> All humans are mortal.
> The speed of an object is equal to the ratio of distance travelled per time unit.

Since the time of the Greek philosophers, knowledge has been expressed in terms of syllogisms, i.e. *rules* of the form:

> IF <condition> THEN <conclusion>

as in:

> IF human(x) THEN mortal(x).

Modern science, in particular physics, has found it more convenient to represent knowledge in the form of equations or more generally *constraints*, such as:

$$v = s/t$$

which can be used both to calculate the v from s and t, but also t from v and s or s from v and t. In fact, in modern physics there is only one single law, minimize the energy, which is to be satisfied in the presence of a large set of constraints.

Constraints in *discrete* domains can be expressed as compatibility relations between variable-value pairs, stating that certain combinations are allowed or not. For example, a certain kind of motor may only admit gears of type Gear-21.

In the following, we examine the advantages of constraints as a basis for knowledge representation and reasoning as opposed to classical formalisms such as rules and algorithms. We will show that ensuring consistency of a rule-based system is a almost impossible in a fast changing world.

2 Configuration task

The general configuration task can be defined as follows:
Given

- a set of predefined components,
- the knowledge of how components can be connected,
- the customer requirements for a specific configuration

The goal is to find the sets of components fullfilling the user-requirements and respecting all the compatibility constraints. This configuration task can be expressed as the problem of assigning values to variables subject to the constraints, where the variables are classes of component types and the values are specific instances of these types.

3 Rule-based configuration systems

3.1 Inference by deduction

When knowledge is expressed as rules, new facts can be obtained by a logical inference called deduction. Given a rule of the form A $\Rightarrow$ B and the observation, that A is true, one can immediatley conclude B. The global control of a rule-based system is either forward or backward chaining. Using forward chaining, we start from the conditions, which we know to be true, towards the conclusions we want to establish. For backward chaining, we move from the conclusions we want to be true towards the facts (the conditions of the rules), which must be true in order to establish the conclusions. The deduction process is logically sound and the only reason for an overall incorrect or conflicting reasoning process lies in the incorrectness of the rules. Ensuring correctness of a rule-based system in a fast changing world is still a big problem and it was one of the major obstacles to the success of rule-based expert systems. The reason for this is that deductive rules always amount to a context-dependent knowledge representation, as the correctness of each rule depends on what other rules are present in the rulebase.

3.2 Example

As an example consider the problem of configuring an automobile using a rule-based system. A car in this example is modelled using five variables i.e. package, frame, engine, transmission and type. The domain of a variable is the set of the possible values for this variable and specific customer requirements are simply values for the variables type and package. The five variables and their possible values are described in more detail below.

1. **Package** with values **Deluxe Luxury Standard**
   The package variable describes the equipment of the car. The deluxe package will include both an airconditioner and a central locking system. The luxury package includes only the central locking system. None of the equipment items is included within the standard package.

2. **Type** with values **Sportscar Familycar**
   The marketing decided to sell only in two car segments. No pickups or other kind of cars can be sold except of sportscars and familycars.

3. **Engine** with values **A B**
   Three different engine types are available. Engine A is a sport version, might therefore be used for the sportscar. Engine B is a smooth-running version for limousines.

4. **Transmission** with values **manual automatic half-automatic**
   Automatic and half-automatic transmissions are normally found in limousines. Manual transmissions are used for off-road cars and sportscars.

5. **Frame** with values **convertible hatchback sedan**
   Cars can be produced with three diffrent frametypes. Hatchback and sedan frametypes

are well suited for families. Convertible frames on the other hand are generally used for sportcars.

The process of determining values for the other variables, given some specific customer requirements and the rulebase below, is called the configuration process. The configuration can be represented as a kind of bill-of-material [7] with layout information.

The Rulebase All the rules are of the form IF premise THEN conclusion. The forward-chaining algorithm selects the rules where the premises are true and triggers them to conclude new facts. Knowing for example, that the customer wishes a sportscar the algorithm concludes immediately by rule 8 (see below), that the car will have a manual transmission.

Rule1 IF Package = Deluxe and Frame = convertible THEN Engine A

Rule2 IF Package = Deluxe and Frame = hatchback THEN Engine B

Rule3 IF Package = Standard and Frame = convertible THEN Engine A

Rule4 IF Engine = A THEN Transmission = manual

Rule5 IF Engine = B THEN Transmission = automatic

Rule6 IF Type = Sportscar THEN Frame = convertible

Rule7 IF Type = Familycar THEN Frame = sedan

Rule8 IF Type = Sportscar THEN Transmission = manual

Given the customer requirements "deluxe sportscar" (values for the variables package and type) one can determine the configuration by a forward-chaining procedure on rules 1-8. The configuration or the output of the configuration process is the five tuple:

- type = sportscar (customer input)
- package = deluxe (customer input)
- frame = convertible (by rule 6 given the input type = sportscar)
- engine = A (by rule 1 given Package = deluxe and frame = convertible)
- transmission = manual (by rule 8 given the input type = sportscar or by rule 4 give the deduced fact engine = A)

The deduction process in a rulebase system can be illustrated using a derivation graph *(Figure 1.)*. An arrow indicates the deduction of a value for a variable at the end of the arrow given the premise (a value for the variable) at the begin of the arrow. Cycles in the derivation graph or paths ending at the same vertex are possible, because knowledge may come from different sources. Rule 8 in the example above might stem from marketing regulations, whereas rules 4 and 5 are determined by technical considerations.

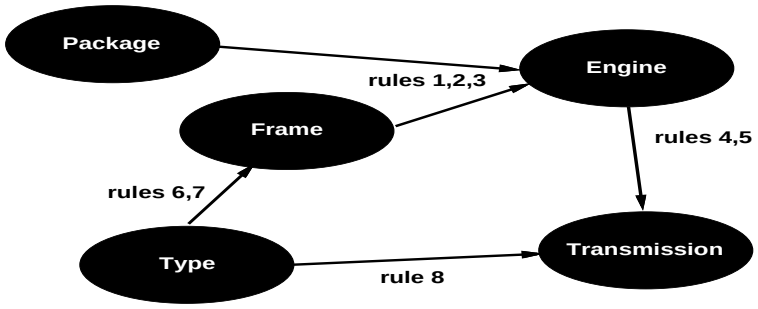

Figure 1: *The derivation graph*

4 Constraint-based configuration systems

4.1 Inference by abduction

Abductive inference is the process where we conclude from the rule A $\Rightarrow$ B and the observation that B is true, that A might have caused B to be true. It is an approximate inference, meaning that abduction, in contrast to deduction, is not sound. Engineering design and configuration are likely to be abductive rather than deductive . It is the task offinding a structure given a functional specification. When a designer tries to realize a function F1 for the artifact to be designed, and his design knowledge tells him that a component C1 beside others realizes F1 i.e. C1 $\Rightarrow$ F1, then the designer concludes by abduction to select C1. If in later design phases inconsistencies occurs, then he probably replaces C1 by another component, which can also realize F1. Abductive reasoning can also be found in the area of medical diagnostics. Given rules in the form disease $\Rightarrow$ symptoms, then the doctor concludes by abduction a disease because of the observed symptoms.

In order to build a abductive inference engine we need a component which is responsible for the generation of the hypotheses and a deductive inference engine. First a hypotheses H (component or disease) accounting for the fact F (function or symptoms) will be generated. The deductive inference engine tries to prove F on the basis of H. To mechanise such a abductive inference engine it is not that much a problem to generate hypotheses and test their validity by deduction; it is the problem of finding "good" hypotheses in a controlled manner to deal with the enourmous search space of possible hypotheses in the field of design or diagnosis.

4.2 Constraint Satisfaction, a mechanism for abduction

A finite, discrete constraint-based knowledge representation consists of:

- variables
- domains
- constraints, expressing relations and possible variable-value combinations

The following example differs from the example in section 3 only in the way knowledge is represented. Here the configuration knowledge is described by constraints, whereas rules were used in the previous example.

Constraints A constraint describes a relation between components (variables) and all the allowed combinations of values for the variables in the relation. All variable-value combinations, which are not explicitly mentioned, are assumed to be not allowed. The goal of the constraint-satisfaction process is to find for each variable values satisfying all the constraints or to realize that there exists no solution for the constraint satisfaction problem. A allowed value-combination tuple may have the following form: (Familycar, automatic) meaning the value Familycar for the variable Type and the value automatic for the variable Transmission are compatible with each other. Knowing that this tuple (Familycar, automatic) holds and that the observation Type = Familycar is true, we can conclude by abduction that the transmission might be automatic. Constraints and rules have different interpretations. Consider for example the allowed value-combination tuple (A manual) in C2 (see below), which must be interpreted as follows:
"engine A is compatible with manual transmission"
while the interpretation of rule 4 in the previous section is:
"every car with engine A will get a manual transmission"

C1: Package Frame Engine
allowed value-combinations:
(Deluxe convertible A)(Deluxe hatchback B)(Deluxe convertible C)...

C2: Engine Transmission
allowed value-combinations:
(A manual)(B automatic)(C half-automatic)(A half-automatic)...

C3: Type Frame
allowed value-combinations:
(Sportscar convertible)(Familycar sedan)...

C4: Type Transmission
allowed value-combinations:
(Sportscar manual)(Familycar half-automatic)...

Constraint satisfaction can now be explained as a recursive abductive process, where at each stage of the reasoning process new hypotheses are build using validated previous ones until hypotheses are found satisfying all constraints (the solutions). In the general case finding solutions or testing if a solution exists is a NP-complete problem. Different versions of backtrack search with complexity bound of order $O(k^n)$ where n is the number of variables and k is the largest domain size of the variables, are normally used to solve constraint satisfaction problems. In practice, however, one can lower the complexity enormously by using preprocessing steps to guarantee a certain degree of consistency [5] e.g. arc-, path-consistency or by using clustering [2] or decomposition techniques [4]. Furthmore it is possible to apply design-compilation

techniques [8] in order to get more efficient runtime systems. On modern workstations, the solutions to realistic size configuration problems can be enumerated in almost realtime.

5 Rules versus Constraints

Ease of maintenance is one of the most important requirement of todays design systems. A comparison of how the maintenance effort differs in both approaches when changes of the expert knowledge must be encountered will be presented below using the example from previous sections. A new type of cars, namely funcars, should be produced and sold especially to young people. A funcar is a mixture between an off-road car and a sportscar and should have a half-automatic transmission and a convertible frame.

5.1 Rule-based reasoning:

Marketing decides to introduce a new funcar type. The following new rules must be included in the rulebase:

Rule10 IF Type = Funcar THEN Frame = convertible

Rule11 IF Type = Funcar THEN Transmission = half-automatic

Suppose a young customer requires a "funcar deluxe". Then we obtain by forward reasoning :

Step 1 IF Type = Funcar THEN Frame = convertible (by Rule 10)

Step 2 IF Package = Deluxe and Frame = convertible THEN Engine A (by Rule 1)

Step 3 IF Engine A THEN Transmission manual (by Rule 4)

Step 4 IF Funcar THEN Transmission half-automatic (by Rule 11)

$=>$ There is a contradiction between Step 3 and Step 4.

This contradiction requires changes in the rulebase. After adding the funcar rules it becomes clear, that rule 4 is no longer valid: "It is not true, that all cars with engine A will get a manual transmission". The knowledge engineer must revise rule 4 by making it context-dependent!

Step 1 removing Rule 4

Step 2 adding Rule 4a
IF Engine = A and Type = Funcar THEN Transmission = half-automatic

Step 3 adding Rule 4b
IF Engine = A and Type = Sportcar or Familycar
THEN Transmission = manual

Furthermore the knowledge engineer must verify that Engine A is compatible with half-automatic transmission by asking a technician.

To guarantee consistency of a rulebase, it is necessary to eliminate all the conflicts which might occur whenever different paths of the derivation graph end at the same vertex (e.g. vertex transmission in Fig. 1). These conflicts are caused by the integration of different sources of knowledge within one system. Each knowledge source might be consistent by its own, but putting them together might lead to inconsistencies when working in the deductive framework. This is the reason why many expert systems fail to be adapted to changes in new environments and thus quickly become obsolete.

5.2 Constraint-based reasoning:

The marketing department decides to introduce the new funcar type and presents the following constraints:

C3' Type Frame
allowed value-combinations:
(Funcar convertible)

C4' Type Transmission
allowed value-combinations:
(Funcar half-automatic)

The tuple (A manual) of constraint C2 is still valid due to the context independence of constraint based knowledge representation and there is no need to describe the context in which this constraint is valid as is was necessary for rule 4 in the rule-based approach.

Given the customer requirements "funcar deluxe", the configuration system can determine the other parameters by applying a backtrack search algorithm to the constraint satisfaction problem. The configuration will be:

- type = Funcar (input variable)
- package = deluxe (input variable)
- frame = convertible
- engine = A
- transmission = half-automatic

Knowledge from various sources can be integrated in a constraint-based system **without any subsequent modification of the existing knowledge**, thereby facilitating the maintenance and the incremental development of configuration systems.

5.3 Comparison

In systems built using deductive rules, in particular expert systems, the context-dependence results in severe problems of *maintenance* of knowledge in the face of a dynamic world. Even minute changes of technology or changes in the marketing policy require revision of the entire rule set, which can be very costly. In the rule-based approach, adding a new car-type forced us to create a new relation between engine, type and transmission. Expressing knowledge without looking at the context is one of the major advantages of constraint based reasoning. No relation between engine, type and transmission is necessary in this context.

One must furthermore realize, that assigning a single source (e.g. marketing regulations) to the modified deductive rules is impossible. Rules from different sources are mixed together in new rules only to get a consistent chaining behaviour (rule 4a and 4b). These new rules can be considered as "interface rules" between knowledge sources and they are artificial in the sense that only the chaining behaviour is responsible for their existence. Furthermore, proving the correctness of those rules becomes troublesome. The problem of systems built using deductive rules is therefore not only the revision of the rules, which can be very costly. The revisions itself will make the maintenance of the rulebase even more difficult, leading to systems which are no longer manageable.

Mechanising the knowledge engineering process is straightforward within the constraint-based framework, because whenever a relation between variables is "established", the knowledge engineer must enter all valid variable-value combinations for that relation. Knowledge engineering in the rule-based framework on the other hand resembles a more handcrafted approach, since it is possible to delay the engineering of rules until these rules are required by the reasoning process. Consider the relation between engine and transmission in the above examples, where the tuple (A half-automatic) in the constraint-based system was valid from the very first moment. In the rule-based system on the other hand one could find this piece of knowledge only in "decoded" form within rule 4b, which was added after the first contradiction was detected.

Finally, we conclude that there is no clear separation between knowledge and control in rulebase systems thus making maintenance hard, whereas the clear separation of knowledge and control in constraint-based systems facilitates maintenance and allows further investigation of the knowledge, as we will show in the next paragraph.

6 Solving configuration management problems

Constraint-based reasoning Besides generating actual solutions, constraints can also be manipulated in different ways which amount to interesting options in the context of configuration. For example:

- the set of constraints can be analyzed to find a more compact representation of the set of potential solutions, using a process called *constraint propagation*. This allows examining the configuration problem at more qualitative levels of abstraction. This process could also be used to generate concise descriptions of the range of possible products within a certain technology.

- the constraint set can be transformed into a different formulation which admits the same solutions, but has a more convenient form, for example only involving constraints between pairs of variables, or only involving constraints between intervals. This is particularly interesting for detecting and creating softness [1]: a product which can be modelled by simple constraints is very soft and easily adaptable to new environments.

- the constraint set, or parts thereof, can be matched to analogous *cases* and solved by reinstantiating or adapting an earlier solution. In configuration, this means reusing a solution found for another client.

- the constraints can be *solved* to eliminate intermediate variables and thus create perspectives on the problem from particular viewpoints. In configuration, one could generate a viewpoint where requirements are directly related to minimum cost of a product satisfying them.

- Constraint-based reasoning methods can be *embedded* in deep knowledge structures and be used whenever these methods are appropriate.

Dynamic constraint systems When large amount of constraints must be handled then it is important for reasons of modularity and efficency to allow the introduction and retraction of constraints resp. variables during problem solving. Constraints in static systems are applicable only where there are well-defined variables. In *dynamic constraint systems* [6] one can define the set of variables for a particular problem using *rules* which introduce variables and constraints as required by the problem. Example: The object pipe-system is relevant only when working with hydraulic motor-types, whereas no pipe-system variable is needed and no pipe-system constraints are active when configuring electrical motor-types.
The nonmonotonic nature of the configuration task can be made explicit in dynamic constraint systems. By adding new variables, conflicts can be eliminated as the following example shows: In the domain of the elevator configuration an expert states that above a critical cabin-weight a certain motor can only be used in combination with a motor-fan. The activation of the variable motor-fan makes certain motor cabin combinations valid, which were not possible without a motor-fan. It was shown, that configuration knowledge can be represented in a very clear and explicit way using the dynamic constraint satisfaction framework [3].

7 Conclusion

Knowledge maintenance in configuration systems must be facilited, because configuration knowledge of todays products evolves over the whole product life cycle. The context independence of constraint-based knowledge representation has been identified as an important feature for facilitating the incremental developement and maintenance of large evolving knowledgebases. The knowledge representation in deductive rule-based systems on the other hand will always be context dependent and there seems to be no way to solve the problems resulting from this context dependency.

We have shown that constraint based representaion and reasoning offers promising possibilities for configuration management systems with respect to maintainability and structured knowledge engineering and other aspects, as mentioned in the previous section. Nevertheles to enable these possibilities for productive constraint based configuration systems a lot of problems, for example limitations when combining discrete and continuous constraints, hierarchical constraints, efficency problems, etc. must be solved.

References

[1] GNOSIS Workpackage 4. Soft Machinery. Report GNOSIS/TW4/All/R/D/20Oct93/C, IMS-GNOSIS, available from ftp.cpsc.ucalgary.ca in directory pub/KSI/GNOSIS/tw4/, 1993.

[2] Rina Dechter and Judea Pearl. Tree Clustering for Constraint Networks. *Artificial Intelligence*, 38:353–366, 1989.

[3] A. Haselböck and M. Stumptner. An Integrated Approach for Modelling Complex Configuration Domains. In *Proc. AI-ES-NL*, pages 625–634, Avignon, 1993.

[4] Philippe Jégou. Decomposition of domains based on the micro-structure of Finite Constraint-Satisfaction Problems. In *Proc. of AAAI-93*, pages 731–736, Washington, DC, 1993.

[5] Alan K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[6] S. Mittal and B. Falkenhainer. Dynamic Constraint Satisfaction Problems. In *Proc. of AAAI-90*, pages 25–32, Boston, MA, 1990.

[7] Paul Schönsleben. *Praktische Betriebsinformatik*. Springer, Berlin, 1994.

[8] C. Tong. A Divide-and-Conquer Approach to Knowledge Compilation. In M. Lowry and Mc Carthey, editors, *Automated Software Design*, pages 261–287. AAAI Press, 1991.