

## A Knowledge Representation System for Integration of General and Case-Specific Knowledge

Agnar Aamodt

University of Trondheim, Department of Informatics  
N-7055 Dragvoll, Norway  
agnar@ifi.unit.no

### Abstract

*Combining various knowledge types - and reasoning methods - in knowledge-based systems is a challenge to the knowledge representation task. The paper describes an object-oriented, frame-based knowledge representation system aimed at unifying case-specific and general domain knowledge within a single representation system. It is targeted at the representational needs that have emerged from research in knowledge-intensive case-based reasoning, addressing complex problem solving in open and weak theory domains. Emphasis is put on representational expressiveness, on flexible reasoning and control schemes, and on easy inspection of cases and other knowledge objects.*

### Introduction

In a knowledge-based system, knowledge about real-world phenomena is represented in a way that enables the system to utilize the knowledge for its reasoning tasks. Representing knowledge in a suitable way for a certain purpose is far from an easy problem, and a major part of AI research has been, and still is, actively investigating this *knowledge representation problem*. Traditionally, the type of knowledge assumed to be of interest for development of knowledge-based systems, and hence the target of knowledge representation methods, has almost exclusively been *general knowledge* about domain concepts and relationships. Examples are conceptual and relational models in the form of semantic networks, frame systems describing general concepts and their interrelations, and heuristic rules associating particular premises with their corresponding consequences.

An increasing amount of work is now focused on a knowledge-based paradigm in which knowledge is captured as past experiences in the form of *case-specific knowledge*. This type of knowledge forms the basis for case-based reasoning (CBR) methods, in which past problem solving episodes - cases - are recalled and used to solve new problems. After the input problem has been solved, it is retained as a new case or merged with another similar case, enabling the system to learn from each

problem solving experience. A recent overview of case-based reasoning is given in [6], while a thorough treatment is found in [11].

Knowledge representation, in a general sense, has so far not been a major issue of case-based reasoning research. Knowledge representation issues in AI has almost exclusively dealt with the representation of general domain knowledge. By capturing specific knowledge, indexed in a suitable memory structure, CBR has offered an alternative and presumably simpler representational approach. Even if many CBR systems use some type of general knowledge in their methods, e.g. to improve the assessment of similarity between a new problem and the set of past cases, such knowledge is often compiled into global metrics or heuristics. In systems that reason from explicit general knowledge, the knowledge models have been of minor complexity and have had limited roles within the systems.

This situation is about to change, however. The reason is the current trend in CBR towards integrated systems that do case-based as well as generalization-based reasoning. And, correspondingly, towards systems that combine case knowledge with more extensive models of general domain knowledge (e.g. [14], [5]).

In the following, a knowledge representation system, CreekL, is described that integrates general and case-specific knowledge within a unified language. The next section gives an overview of the system in terms of its basic philosophy and underlying assumptions. This is followed by a characteristic of the language syntax, the language semantics, and current system status. The final section concludes the paper and points out future work.

### Basic language philosophy

In order to get the desired effects of integrating different knowledge and reasoning types, the system parts involved need to share a common understanding of the terms they jointly will be reasoning about. A piece of knowledge - i.e. a concept definition or a proposition - should have a basic interpretation which is the same for the various reasoning methods involved. Upon this

common core view, different methods may have local views - perspectives. In CreekL this is facilitated by having explicit definitions of each type of concept - and local aspects of a concept - that will be reasoned about.

The type of applications addressed by the CreekL system is *user-interactive* systems in open and *weak theory* domains, i.e. domains that are open-textured and dynamically changing, and that do not have a strong theory from which statements in general can be proved true or false. A weak domain theory is characterized by uncertain relationships between domain concepts. Example domains are medical diagnosis, corporate decision-making, and most engineering domains. A concept definition is most often a *typical* and *partial* description, not cannot be assumed as universally quantified and complete. Instances (exemplars, cases) may extend a concept definition, or - as exceptions - serve to constrain a prototypical definition.

To capture these knowledge types, an expressive and flexible representation formalism is needed. Expressiveness is related to the ability to represent objects and phenomena of the real world according to their real-world meaning. Expressiveness is therefore not primarily a syntactic problem, but a semantic one. Flexibility is related to the ability to build representational constructs that are not pre-defined, and to extend the language by incorporating new constructs. Frame systems are particularly suited for describing and reasoning with prototypical concept definitions, and provide the necessary degree of flexibility [13].

An important role of a representation system for the type of systems addressed here is to achieve a common consensus between the user/expert and the system as to the meaning of the actual knowledge that is represented. Hence, the semantics of the language should be clearly defined. Flexibility - in the sense of an easily extendible representation language - and a clearly defined semantics may easily be viewed as conflicting requirements. In CreekL this conflict is resolved by explicitly defining the semantics of added representational constructs within the representation system itself. For example, if a new interpreter is needed (e.g. a new mechanism for uncertainty handling), it may be incorporated into the representation system, but in order to preserve a clear semantics of the representation language, the properties of the new interpreter have to be explicitly defined within the system. A particular inference method (such as constraint propagation) may be defined by the conditions that have to be satisfied in order for the inference procedure to apply, the type of results it produces, its dependencies on lower level inference methods (like, e.g., default inheritance, backward chaining), etc.

CreekL has strong similarities with other open frame systems such as RLL [9], KODIAK [19], and CYCL [12], but differs from strictly logical-oriented languages such as the KL-ONE type of systems [7] which assume stronger

domain theories, and which do not have default inheritance.

### Basic language syntax - example structures

A concept in CreekL is represented as a *frame*. A frame is an object which consists of a list of *slots* that define the concept by relating it to other concepts or values. Slots therefore correspond to relations between objects. Referring to a semantic network view, a frame corresponds to a network node, a slot corresponds to a link between a concept node and a slot value.

Slots are typed according to what role the slot value serves, i.e. what type of value the slot filler holds. Slot value roles are called *facets*. Typical facets are regular values, default values, value constraints, procedural definitions. Concepts are therefore represented as 4-level structures of frame, slots, facets, and values (with annotations).

Below, part of a frame representing the concept car with its slots, facets and values is exemplified. The frame is shown as a pretty-printed lisp structure with all but significant parentheses removed.

```
car
  subclass-of      value      vehicle sporting-gear
  has-part         value      electrical-system engine
  has-numb-of-wheels default    4
  has-fault        value-class car-fault
  has-numb-of-seats value-constraint (and (> input-value 0)
                                         (< input-value 9))
  has-age          value-dimension years
                    if-needed    (time-difference
                                   *current--year*
                                   self.has-production-year)
```

Facets are here used to represent specific value types (actual values, defaults, value-dimensions), constraints on legal values (value-class), and demons - attached Lisp functions that return a value or perform an operation (if-needed). The latter works as follows : When we want to know the age of a particular car, say *car#1* (an instance of *car*), the *has-age* slot is inherited, and the *time-difference* function is called with the two arguments shown. The dotted notation 'a.b' refers to the value of slot b in frame a, i.e. the value of the *has-production-year* slot of *car#1* (after substitution of self).

CreekL not only enables but also enforces an explicit definitions of relations and symbolic values. For example, if the user wants to introduce a new slot, called *has-color*, on the *car* frame, the system automatically creates the frame *has-color* and gives it a slot called *used-to-describe* with the value *car*. The user may enter additional slots on the *has-color* frame to better describe what it means to have color. The system also automatically creates a frame for each symbolic values entered in a value expression.

Below is a frame showing a partial representation of a solved case, described by its feature names (slot names) and values.

```
case#54
  instance-of      value      car-starting-case diagnostic-case
  has-task         value      find-car-starting-fault
```



particular drug (A) that causes a medical effect (E) and the tablet (B) containing the drug, which also causes the effect.

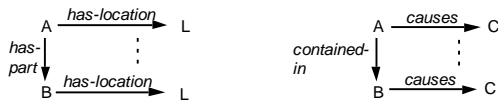


Figure 4. Plausible inheritance examples

Plausible inheritance properties of relations are represented in CreekL by storing, in the frame representing the inheritance relation, the relations that may get inherited. For example:

```

has-part
inheritance-relation-for  value  has-location
inheritance-condition    value  (has-location physical-object)

has-location
inherits-via             value  has-part

```

Explanation generation is assumed to be controlled by control knowledge relevant to the explanation task, including what type of inheritance to apply at various stages of an explanation generation process. An explanation in Creek is a structure consisting of a single fact (proposition) or a chain of facts. A fact corresponds to a relationship, i.e. a <concept.relation.value> triplet. For general concepts, an explanation is attached to the particular value of a target concept that it explains. Since an explanation usually is associated with a single value, the explanation is - in general - contained in an annotation field of the value expression. The first element of an explanation list is a numeric value indicating the *explanatory strength* of the explanation. This is a measure for the degree of support a fact has in the underlying domain model, and hence the degree of belief in the fact. For example, if an explanation for why a starter motor (target concept) will not turn is that it has a weak battery, this will be represented as:

```

starter-motor-1
instance-of      (starter-motor nil)
connected-to    (battery-1 confirmed-by-user)
has-battery-circuit-status (closed confirmed-by-user)
has-turning-status (does-not-turn
                   (0.9
                    (battery-1.has-voltage.very-low)
                    (battery-1.instance-
                     ((battery.has-
                      causes
                       (starter-motor.has-turning-status.
                        does-not-turn) )
                      (starter-motor.has-instance.starter-motor-1)
                      (starter-motor-1.has-turning-status
                       does-not-turn))))

```

## Development status

CreekL is based on an early frame language developed for the METATOOL knowledge acquisition environment [1]. The basic system has been implemented (in Common Lisp) and used in some application systems. An earlier derivative of the language, called SFL [3], is part of the knowledge acquisition workbench KEW of the Esprit

project Acknowledge [2]. A version of the language was further used in a knowledge-based front-end to an information retrieval system, developed for the European Space Agency, ESA [16]. In these applications only general domain knowledge was represented. The case-based capabilities is an extension [4], and still under development. The current updates are done in conjunction with the implementation of a method for explanation-driven case-based reasoning [5].

## Conclusion and Future Work

The CreekL representation enables the tight integration of case-specific and general domain knowledge by emphasizing a through, explicit representation of all concepts to be reasoned about - including system-internal terms and methods. It takes a non-logicist, procedural approach to representing semantics, and enables integration by explicitly defining the characteristics of local interpreters.

In addition to our focus on knowledge-intensive CBR, we are studying the language as a general basis for knowledge acquisition and knowledge level modeling [18]. Since it has a procedural semantics, it is possible to express informal and vague models - at the knowledge level - as well as specialized and detailed models for symbol level implementation. Hence the remodeling involved in moving between the levels is performed within the same basic object structure, and build upon the same representational syntax. The role of CreekL within the framework of bridging the knowledge level - symbol level gap, is studied within the Components of Expertise framework and workbench [15]. Ongoing language extensions include improved mechanisms for retrieving, reusing and storing of cases, implementing plausible inheritance methods, and a mechanism for guided reasoning and learning through context focusing.

## References

- [1] G. Aakvik, M. Vestli, I. Nordbø, I. Sølberg, T. Amble, A. Aamodt: METAKREK; METATOOL documentation. SINTEF-DELAB Report STF14 A88055. SINTEF, Trondheim, 1988.
- [2] G. Aakvik, O. J. Mengshoel, I. Nordbø, M. Vestli: SFL - The Sintef Frame Language. The Acknowledge Project, ACK-STF-T3.1-WP-003-A. Trondheim, 1990.
- [3] G. Aakvik, A. Aamodt, I. Nordbø: A knowledge Representation Framework Supporting Knowledge Modelling. *Proceedings EKAW-91*, Crieff, Scotland, May 1991.
- [4] A. Aamodt: *A knowledge-intensive approach to problem solving and sustained learning*, Ph.D. dissertation, University of Trondheim, Norwegian Institute of Technology, May 1991. University Microfilms PUB 92-08460.
- [5] A. Aamodt: Explanation-driven case-based reasoning, In S. Wess, K. Althoff, M. Richter (eds.): *Topics in Case-based reasoning*. Springer Verlag, 1994, pp 274-288.
- [6] A. Aamodt, E. Plaza: Case-Based Reasoning: Foundational issues, current state, and future trends, *AI Communications*, March 1994, pp 39-59.

- [7] R. Brachman, S. Schmolze: An overview of the KL-ONE knowledge representation system. *Cognitive Science*, Vol 9, no 1, pp 3-28. January 1985.
- [8] P. Cohen, C. Loisel: Beyond ISA: Structures for plausible inference in semantic networks. In *Proceedings of the AAAI-88*, pp. 415-420.
- [9] R. Greiner, D. Lenat: A representation language language. *Proceedings AAAI-80*, Morgan Kaufmann, pp. 165-169.
- [10] J. Josephson, S. Josephson: *Abductive inference, computation, philosophy, technology*. Cambridge University Press, 1994.
- [11] J. Kolodner: *Case-based reasoning*. Morgan Kaufmann. 1993.
- [12] D. Lenat, R. Guha: *Building Large Knowledge-Based Systems*. Addison-Wesley, 1989.
- [13] M. Minsky: A framework for representing knowledge. In P. Winston (ed.): *The psychology of computer vision*. McGraw-Hill, 1975. pp. 211-277.
- [14] C.B. Skalak, E. Rissland : Arguments and cases: An inevitable twining. *Artificial Intelligence and Law, An International Journal*, 1(1), 1992, pp.3-48.
- [15] L. Steels: The componential framework and its role in reusability, In J-M. David et al (eds.), *Second generation expert systems* , Springer, 1993, pp 273-298.
- [16] I. Sølvberg, I. Nordbø, A. Aamodt: Knowledge-based information retrieval. *Future Generation Computer Systems*, Vol.7, 1991/92, pp 379-390.
- [17] P. Thagard: Explanatory Coherence. *Behavioral and Brain Sciences* 12, 1989, 435-467.
- [18] W. Van de Velde: Issues in knowledge level modelling, In J-M. David, J-P. Krivine, R. Simmons (eds.), *Second generation expert systems* , Springer Verlag, pp. 211-231.
- [19] R. Wilensky (1986): Knowledge representation, a critique and a proposal. In J.L. Kolodner, C.K. Riesbeck (eds.): *Experience, memory and reasoning*. Lawrence Erlbaum, pp. 15-28.