Statement of Interest

Bill Swartout

1989

Most of my experience with term subsumption languages stems from using NIKL as part of the knowledge representation for the Explainable Expert Systems (EES) project. The goals of the EES project have been to create a framework for building expert systems that 1) enhances the explanations that an expert system can offer, and 2) eases maintenance and extension of expert systems. The EES framework achieves these goals by adopting a specification-based approach to expert system creation. These goals have been difficult to achieve in conventional shells due to their use of rules as a single, low-level representation for knowledge. As we and others have pointed out [Swartout 83, Clancey 83], rule-based architectures tend to confound different kinds of knowledge and leave implicit much of the "design rationale" behind an expert system. These problems impact both maintenance and explanation.

Unlike other shells, to build an expert system in EES, a system builder does not begin by writing down a set of rules. Rather, he creates a knowledge base that explicitly distinguishes different kinds of knowledge, such as domain descriptive knowledge, problem solving knowledge, and terminology. This knowledge base may contain both domain independent as well as domain specific knowledge. This knowledge base specifies how problem solving should be done in a particular domain, however its generality means that performance will be slow if the knowledge base is directly interpreted. EES also provides an automatic programmer to create an efficient implementation from the knowledge base. The automatic programmer treats the knowledge base as a specification and derives an expert system from it. The programmer records the design decisions it makes in a design history. This design history provides explanation routines with the rationale they need to produce better
explanations, particularly justifications.

The EES framework imposes some severe demands on a knowledge representation formalism. EES needs to represent problem solving knowledge, domain facts, and terminology. Furthermore, the representation must represent knowledge in a way that is effective for problem solving, but also explicit and declarative enough to support explanation. We realized that unadorned NIKL would not satisfy all of these needs. For example, NIKL has no predefined semantics for representing problem solving knowledge. We used NIKL as a component of our knowledge base, to represent terminology and some kinds of domain descriptive knowledge, and we created extensions for representing other kinds of knowledge such as problem solving knowledge. We used NIKL's classifier to organize the terminological base. During problem solving, we also used the classifier as a sort of pattern matcher to find plans that could be applied to goals and to help perform goal reformulation.

Using NIKL in this way, we were able to create a version of the EES framework (see [Neches et al 85]). We used this framework to create some demonstration-sized expert systems. Recently, we have shown how the framework can be used to provide significantly enhanced explanations [Moore and Swartout 89].

Despite this success, we frequently ran into difficulties in representing knowledge due to limitations imposed by NIKL. Some of these limitations were just a consequence of the particular implementation we were using, but others were more fundamental. These latter limitations stem from some basic design decisions that were made in KL-ONE and adopted by NIKL. These decisions are outlined briefly below, and followed by a list of questions about the design of knowledge representation systems.

The design of KL-ONE-style languages (like NIKL) was guided in part by the insight that the overall performance of a general purpose reasoning system can often be improved if some of its reasoning can be taken over by special purpose reasoners that exploit specialized representations. Because the capabilities of the special purpose reasoners and representations are limited, they can be tailored to perform their tasks very efficiently. NIKL provided a specialized representation for terms and their definition, while NIKL's classifier performed the specialized reasoning function of organizing the terms into a subsumption hierarchy based on their definitions.

Unfortunately, as Levesque and Brachman discovered [Brachman 84], there are many kinds of naturally occurring terms that cannot be represented
if the classifier is to be both complete and efficient. Thus, the designers of NIKL were faced with a quandry. They either had to restrict the expressive power of the language, or give up on the completeness or efficiency of the classifier. For the most part, they chose to restrict the expressiveness of the representation.

While this choice might be right for some applications, it was not clear that it was the best choice for the concerns of EES. Because the representation of terminology was restricted, some terms could not be defined. It was necessary to either make those terms primitive (that is, not give them any definition) or to create an unnatural definition that would cause the classifier to "do the right thing." From the standpoint of explanation, neither approach was very good. Obviously, if a term lacked a definition, it couldn't be explained. On the other hand, if an artificial definition had been created because the natural one couldn't be expressed then that artificiality would be embarrassingly reflected in the system's explanations. Thus, while the decisions that NIKL's designers made might be appropriate for many systems, they created some serious problems for us.

Problems such as these have led me to a number of questions about the design of knowledge representation systems:

1. Is it possible to resolve the tradeoff described in the Levesque and Brachman paper without taking into account the particulars of an application?

2. Could we create a knowledge representation system that lets a system builder resolve the choice as he sees fit?

3. NIKL and Krypton both were attempts to create specialized representations and then marry them with existing general purpose reasoners. I think it's fair to say that these marriages were not entirely happy. Wouldn't it make sense to start the other way around, i.e. to begin with a general purpose reasoner and define a clean interface for introducing special purpose representations and reasoners?

4. KL-ONE-style languages provided classifiers and limited the expressiveness of their representations because it was assumed that reasoning about the subsumption between terms was an important kind of reasoning. Based on subsequent experience, what evidence do we have to
support/deny that assumption? How much is it used in real applications?

References:


[Swartout 83] Swartout, W., "XPLAIN: A system for creating and explaining expert consulting systems," Artificial Intelligence 21, (3), September 1983, 285-325. Also available as ISI/RS-83-4