Using a Description Classifier to Enhance Deductive Inference*

Robert M. MacGregor
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
macgregor@isi.edu

Abstract

The representation languages found in many expert system shells are hybrids composed of a frame language and a rule language. Unfortunately, the frame and rule components in these systems are not well integrated, and as a result they miss important classes of inferences. In place of frames, LOOM *combines a* description *language with a rule language, and uses an inference engine called a classifier to achieve a successful integration of frame-like knowledge and rule-like knowledge.* LOOM*'s ability to reason with descriptions enables us to implement a broader range of capabilities than those found in the current generation of expert system shells. For example,* LOOM *is able to detect inconsistencies in a rule base, to match against partially-specified (not fully-grounded) instances, and it implements a generalization of the traditional (e.g., CLOS-like) conception of object-oriented method dispatching.*

AI Topic: Knowledge Representation, Reasoning
Status: KR system in use by more than 15 projects in the U.S. and Germany.
Effort: 7 person years

1 Introduction

Many of todays expert system toolkits are hybrids that enhance their expressive power by including both a "frame component" and a "rule component" within a single architecture. The marriages between frames and rules achieved with these systems have significant drawbacks. In particular, their ability to reason with both kinds of knowledge is generally non-uniform. In place of frames, LOOM combines a *description* language with a rule language, and uses an inference engine called a classifier to help "bridge the gap" between the description component and the rule component, thereby achieving a successful integration of frame-like knowledge and rule-like knowledge.

The frame component present in many of todays expert system tools provides a means for introducing terms present in the model of an application domain, and for attaching constraint knowledge to individual terms. A simple form of deductive inference, called "inheritance", is commonly implemented as a part of a frame system. LOOM's description language provides a principled means for describing much of the knowledge that is commonly associated with the frame component of a knowledge representation tool. The description classifier (also called a term classifier) gives LOOM the ability to reason with descriptions. Key features of a classifier are its ability to determine subsumption relationships between descriptions (when one description is implied by another) and to combine (unify) sets of descriptions to form new descriptions. These additional kinds of inference extend considerably beyond simple inheritance. This additional deductive power enables LOOM to provide capabilities not found in current generation KR tools.

LOOM had its genesis in the KL-ONE-family[BS85] of taxonomic representation languages. LOOM is designed to support the construction of intelligent software applications. It promotes a "model-based" approach to programming wherein the core of an application program is a (dynamically changing) knowledge base of definitions, rules, and facts that collectively define the state of a domain model. The LOOM language incorporates multiple programming paradigms: logic programming, data-driven programming (productions), and object-oriented programming (methods). Each of the paradigms is fully-integrated with LOOM's knowledge base component.

Our discussion of LOOM focuses on how it solves the problem of reasoning with both rule-like knowledge and frame-like knowledge. In the discussion below, we first take a look at how current generation systems represent knowledge using rules and frames. We charac-

*This research was sponsored by the Defense Advanced Research Projects Agency under contract MDA903-87-C-0641.

terize the notion of a "constraint rule", and indicate why these systems do a poor job of reasoning with constraint rules. Next, we introduce the LOOM description language, and show how descriptions can be used to extend the expressive power of a rule language. After that, we describe the role that a classifier plays in reasoning with descriptions. As a footnote, we contrast the rules-plus-descriptions technology used in LOOM with the inference technology developed for logic programming languages.

Finally, we describe some specific benefits that accompany the availability of classifier technology. We discuss enhanced forms of terminology maintenance, reasoning with incomplete descriptions, and the use of specificity computations to enhance the data-driven and object-oriented programming paradigms.

2 Rules

A typical deductive rule has the form

```
    IF <antecedent> THEN <consequent>
```

with the interpretation "Whenever the antecedent of a rule is satisfied, the consequent is also true." For example, the rule

```
    IF Person(X) and (age(X) >= 60)
    THEN Senior-Citizen(X)
```

states "If X is a person, and the age of X is at least 60, then X is a senior citizen."

Current generation systems place significant restrictions on the expressive power of the consequent side of a rule. In general, the syntax of a rule consequent is designed to mirror the syntax used to assert facts in a knowledge base. Most of these systems can only assert facts expressible as grounded atomic formulae (tuples), e.g., "`Person(Fred)`" (Fred is a person), or "`age(Fred,50)`" (Fred's age is 50). They cannot express facts such as "Fred is at least 60 years old", or "Bicycle B3 has exactly two wheels." Accordingly, their rule languages do not permit one to state such rules as

"If X is a senior citizen,
then the age of X is at least 60", *or*
"If X is a bicycle, then X has exactly two wheels."

Knowledge representation systems place syntactic restrictions on their fact and rule languages for the following reason: In order for a system to reason with knowledge expressed as facts or rules, it must be able to *combine* the relevant knowledge structures to deduce new facts. For example, the fact "Fred is a man" and the rule "all men are persons" combine to yield "Fred is a person". The formal term for the operation that combines the structural components of facts and rules is *unification*. The restrictions that typical current generation systems place on the syntax of facts and consequents of rules stem directly from the fact that they implement relatively simple unification algorithms that are only capable of unifying terms having the form of atomic predications. As we shall see in a moment, LOOM implements a more sophisticated form of unification, and consequently, it places fewer restrictions on the syntax of facts and rules.

We use the term *constraint rule* to describe a rule whose right hand side does not have the form of an atomic predication (an atomic predication is a predicate applied to a list of variables or constants, e.g., "`Person(?x)`" or "`color(?x,Red)`"). For example, the rules "a senior-citizen's age is at least 60" and "a bicycle must have exactly two wheels" are constraint rules. For the remainder of this discussion, we will conceptually partition our rules into two classes – a rule whose consequent is represented as an atomic predication will be called a "simple rule", while a more complex rule will be called a "constraint rule".

Some frame languages can express limited kinds of constraint rules. For example, in KEE, the statement "the age of a senior citizen is at least 60" is represented by attaching a type restriction ("greater than or equal to 60") to the slot "`age`" on the frame "`Senior-Citizen`", while the statement "a bicycle has exactly two wheels" is represented by attaching a number restriction ("exactly two slot fillers") to the slot `has-wheel` on the frame `Bicycle`.[1] Thus, the representational power of a hybrid language combining frames and rules may exceed that of a simple-rule-only language (or a frame-only language).

Unfortunately, current generation frame-plus-rule (F+R) systems give uneven treatment to knowledge placed in the system. In particular, their ability to *reason* with constraint rules is quite limited. Systems such as KEE, ART, and Knowledge Craft are capable of applying constraint rules to test for consistency, but they lack the ability to "chain" using constraint rules. Suppose a knowledge base contains the simple rule

```
    IF age(X) >= 21 THEN Eligible-to-Vote(X)
```

and also contains the constraint rule

```
    IF Senior-Citizen(X) THEN age(X) >= 60.
```

If we assert `Senior-Citizen(Fred)` then `Eligible-to-Vote(Fred)` is necessarily true. However, these systems cannot find this inference. This missing piece of knowledge is the rule of inference "age at least 60 implies age at least 21."

[1]KEE employs the terms "value class" and "cardinality" to refer to the notions of type restriction and number restriction, respectively.

LOOM description	*English equivalent*
`Person`	"person"
`(>= age 60)`	"thing with age at least 60"
`(:exactly 2 has-wheel)`	"thing with exactly two wheels"
`(:all has-wheel Flat)`	"thing all of whose wheels are flat"
`(:the color Blue)`	"thing with color of type blue"

Figure 1

As another example, if we state the constraint rule

```
IF Younger-than-Jack-Benny(X)
THEN age(X) < 39
```

and then tell the system both `Senior-Citizen(Fred)` and `Younger-than-Jack-Benny(Fred)`, then both of the following (contradictory) facts can be proved: "age(Fred) < 39" and "age(Fred) >= 60". These facts are not directly representable in most F+R systems (because these facts do not have the form of atomic predications), and hence most F+R systems will fail to detect this contradiction.

The missed inferences just described *can* be inferred by a system that has the ability to perform unification over constraint expressions. Such a system would detect that the constraints "age(X) >= 60" and "age(X) >= 21" are unifiable (yielding the result "age(X) >= 60"), and it would detect that "age(X) < 39" and "age(X) >= 60" are *not* unifiable. LOOM calls these expressions *descriptions*, and uses the classifier to perform unification over descriptions.

3 Descriptions

This section introduces the notion of a "description", and describes the kinds of inferences that are made by a description classifier.

A description language is designed to facilitate the construction of expressions that describe classes of individuals. For example, the English description "blue, 2-wheeled cycle" characterizes a subset of the class of all cycles. Figure 2 above contains examples of simple descriptions, phrased using LOOM syntax on the left, and in English on the right:

Compound descriptions are formed using the connectives `:and` (intersection), `:or` (union), and `:not` (relative complement). For example, in LOOM, the description

```
(:and Person (>= age 60))
```

might describe the class of senior citizens, while

```
(:and Cycle
      (:exactly 2 has-wheel)
      (:the color Blue))
```

describes the class of blue bicycles.

A *definition* binds a symbol to a description, i.e., it gives a name to a description. The knowledge base object representing this definition is called a *concept*. Definitions are stated in LOOM using the operator "defconcept", e.g.,

```
(defconcept Bicycle
    (:and Cycle (:exactly 2 has-wheel)
                (:all has-wheel Wheel)))
```

defines a concept named `Bicycle` with relation `has-wheel` constrained to refer to exactly two `Wheels`. LOOM substitutes the notion of a concept in place of the frame constructs found in F+R systems.

Table 2 lists some of the constructs available in LOOM for expressing simple descriptions.[2]

Given any two descriptions *D1* and *D2*, a classifier can be called upon to answer the following questions:

1. Does *D1* denote a subclass of *D2*?
2. Are *D1* and *D2* equivalent?
3. Are *D1* and *D2* disjoint?

Because the problem of answering each of these questions is undecidable in the general case, a classifier can be counted on to generate sound but not necessarily complete responses to these questions.[3]

For description languages, unification corresponds to intersection—descriptions *D1* and *D2* are unifiable if and only if their intersection is not equivalent to the description denoting the empty set.[4]

[2]LOOM also supports descriptions representing non-unary predicates; they are not discussed in this paper.

[3]Both LOOM and the more complex expert system shells such as KEE, ART, etc. are incomplete. The significant difference is that LOOM finds important classes of inference (as illustrated in this paper) that these other systems miss.

[4]See the discussion in [Kni89, pages 105–108] on unifying feature structures.

LOOM Expression e	*Interpretation* $[\![e]\!]$
`(:and` C_1 C_2`)`	$\lambda x.\, [\![C_1]\!](x) \wedge [\![C_2]\!](x)$
`(:or` C_1 C_2`)`	$\lambda x.\, [\![C_1]\!](x) \vee [\![C_2]\!](x)$
`(:all` R C`)`	$\lambda x.\, \forall y.\, [\![R]\!](x,y) \rightarrow [\![C]\!](y)$
`(:at-least` k R`)`	$\lambda x.\, \exists\, k\ \textit{distinct}\ y\ \wedge_i [\![R]\!](x,y)$
`(:at-most` k R`)`	$\lambda x.\ \nexists\, k+1\ \textit{distinct}\ y\ \wedge_i [\![R]\!](x,y)$
`(:same-as` R_1 R_2`)`	$\lambda x.\, \forall y.\, [\![R_1]\!](x,y) \leftrightarrow [\![R_w]\!](x,y)$
`(>=` R c`)`	$\lambda x.\, \forall y.\, [\![R]\!](x,y) \rightarrow (y \geq [\![C]\!])$
`(:exactly` k R`)`	`(:and (:at-least` k R`) (:at-most` k R`))`
`(:the` R C`)`	`(:and (:exactly` 1 R`) (:all` R C`))`

Figure 2

4 Rules in LOOM

LOOM permits arbitrary descriptions to be attached to the right-hand sides of rules. Here are two earlier examples of rules repeated using LOOM syntax:

```
    (implies (>= age 21) Eligible-to-Vote)
    (implies Senior-Citizen (>= age 60))
```

Suppose we tell LOOM that Fred is a senior citizen. Because the LOOM classifier can determine that "`(>= age 60)`" implies "`(>= age 21)`", it can "chain" from the consequent of the second rule to the antecedent of the first rule, and hence it will infer that Fred is eligible to vote. Next, suppose we state the rule

```
      (implies Younger-than-Jack-Benny
               (< age 39))
```

and assert both `Younger-than-Jack-Benny(Fred)` and `Senior-Citizen(Fred)`. LOOM automatically unifies all descriptions asserted or inferred about an object. In this case, LOOM will derive a contradiction when it attempts to unify "`(< age 39)`" and "`(>= age 60)`". LOOM notes the contradiction by marking the object Fred as *incoherent.*

The LOOM classifier organizes all descriptions introduced into the system into a taxonomy, with more general descriptions placed above more specific ones. Thus, for example, the description "`(>= age 60)`" would be placed below the description "`(>= age 21)`". A description "`(:and (<= age 39) (>= age 39))`" would merge with the description "`(= age 39)`", since the two descriptions are equivalent. A by-product of this computation is that consequents and antecedents of rules composed only of descriptions are unified at compile time. In this guise, the classifier performs a function analogous to that of a rule compiler in some other systems.

The idea of combining rules with descriptions in the manner just described is a relatively recent innovation within the field of classification-based KR systems, but it seems to be gaining quick acceptance within that community. CLASSIC [BBMR89] and MESON [OK88] are examples of other KR systems that have adopted the same approach. However, rules in these other systems are limited in that they only contain monadic (one-argument) predicates.

5 Description Unification versus Prolog-style Unification

The unification mechanisms of F+R systems are generally weaker than those found in languages like Prolog. Prolog derives significant inferential power from its ability to unify terms containing partially-instantiated list structures. The style of programming that exploits this kind of unification tends to promote clever, frequently obscure, encodings of knowledge. Perhaps for this reason, a logic programming style of representing knowledge appears to be antithetical to the object-centered approach associated with frame systems. In any case, most F+R systems provide only a simple form of unification wherein arguments to predicates must be either simple variables or constants.

LOOM's representation language encompasses a definition language (that binds predicate symbols to descriptions) and a function-free Horn logic. A logic that combines these two languages is provably more expressive than ordinary Horn logic *with* function symbols (and in particular, it is more expressive than pure Prolog). This result tells us that we can exclude embedded function symbols from the LOOM language without sacrificing representational power. Our decision to exclude embedded function symbols restricts the kinds of syntactic encodings that can be represented by the language. We consider this restriction a feature, since it encourages a more object-centered approach to representing knowledge than the encodings employed in logic programming applications.

6 Benefits of the Description Technology

The ability to classify descriptions and to unify sets of descriptions enables new features not available using ordinary F+R technology. In this section, we list some of these benefits.

6.1 Terminology Maintenance

As definitions and rules are entered into a LOOM knowledge base, they are compiled by the classifier into a taxonomically organized network. During this process, the classifier automatically unifies the consequents of rules that share a common antecedent. This compilation activity augments the standard notion of frame inheritance. For example, suppose we are given the following rules and definitions

```
  (implies Mil-Spec-Part
           (:all subpart Mil-Spec-Part))
  (implies 5-Volt-Part
           (:all subpart 5-Volt-Part))
  (defconcept 5-Volt-Mil-Spec-Part
       (:and 5-Volt-Part Mil-Spec-Part))
```

When classifying `5-Volt-Mil-Spec-Part`, the classifier will inherit the descriptions
"`(:all subpart 5-Volt-Part)`" and
"`(:all subpart Mil-Spec-Part)`"
from `5-Volt-Part` and `Mil-Spec-Part`, respectively. These two descriptions will be unified, yielding the single description

```
  "(:all subpart
         (:and 5-Volt-Part Mil-Spec-Part))."
```

Normalization of this unified description produces the equivalent description
"`(:all subpart 5-Volt-Mil-Spec-Part)`."
The classifier will attach the normalized rule

```
 (implies 5-Volt-Mil-Spec-Part
          (:all subpart
                5-Volt-Mil-Spec-Part))
```

to the concept `5-Volt-Mil-Spec-Part` (and also it will eliminate the two descriptions inherited to `5-Volt-Mil-Spec-Part` since they are now redundant).

The taxonomy constructed by the classifier can be visually inspected by a model builder. Not infrequently, such a user will note and fix anomalies (bugs) in the model while browsing such a taxonomy. In this way, significant percentage of anomalies or inconsistencies in a knowledge base can be discovered at compile time, rather than at run time. Thus, the classifier provides a valuable service in support of the knowledge acquisition process.

6.2 Reasoning with Incomplete Knowledge

Traditional rule-based technology can only perform inference on terms that are "fully-grounded." A classifier enables systems like LOOM to reason with incomplete or indefinite knowledge. For example, suppose we have the following knowledge

```
    (disjoint Male Female)
    (implies Pregnant Female)
```

(these assertions state that male and female are disjoint concepts, and that pregnant things are necessarily female things) and suppose we assert about Fred the two descriptions "`Male`" and "`(:or Fat Pregnant)`", meaning that Fred is a male, and Fred is either fat or pregnant. In the process of unifying these two descriptions, LOOM will eliminate the disjunct `Pregnant` (because it implies `Female`, which does not unify with `Male`), arriving at the single description for Fred "`(:and Male Fat)`". This kind of reasoning is being exploited by a natural language parsing application that is being implemented using LOOM [Kas89]. In that application, a significant percentage of the semantic knowledge that attaches to terms in a sentence takes the form of disjunctive descriptions. Especially in the early stages of parsing, significant benefits (manifested by a reduction in the search space) arise whenever the process of unification resolves an ambiguity.

The notion of using a classifier to detect inconsistencies during problem solving generalizes to other tasks besides parsing natural language. We conjecture that other important classes of problem solving will benefit from the introduction of classifier technology.

6.3 Enhancing the Data-Driven and Object-Oriented Programming Paradigms

It is common in many applications to encounter production rules containing significant numbers of conditions within a single antecedent. These complex rules are difficult to understand, and hard to debug. A term definition facility enables frequently occurring sets of conditions to be bundled together and given a name. By substituting these names in place of references to their sets of conditions, the sizes of rules (measured in numbers of conditions) can often be substantially reduced. This tends to make the rules easier to comprehend and debug, and can have beneficial effects on system performance. [Yen89] elaborates on the benefits that result when definitions are combined with production rules. [Mac88] describes how the expressive power of productions can be increased when references to defined terms are permitted.

LOOM provides a method dispatching facility that takes advantage of its ability to unify descriptions. The method dispatching facilities found in object-oriented programming languages provide two important functions: First, given a generic function and a set of arguments, the dispatcher eliminates all methods whose types (paired with their formal parameters) are incompatible with the types of the actual parameters. Second, the dispatcher prefers methods tied to more specific types/classes over methods attached to more general types. LOOM generalizes traditional method dispatching by permitting an arbitrary pattern to be associated with each method [MY88]. When a generic function is called in LOOM, the LOOM method dispatcher eliminates all methods whose patterns are not satisfied when their free variables are bound to the actual parameters. In LOOM, a method is preferred over another when the pattern attached to the former method specializes the pattern attached to the latter method. [Yen90] describes how classifier technology can be extended to compute subsumption relations between patterns. Thus, LOOM's generalization of the method dispatching paradigm is predicated on the availability of a classifier.

7 Discussion

Much of the technology described in this paper is not unique to LOOM[Mac90]. Classification-based reasoning represents one of the most active research areas in the field of KR systems, as exemplified by such systems as BACK [Nv88], CLASSIC, MESON, and SB-ONE [Kob90]. LOOM stands at the high end of the current group of classification-based systems with respect to the expressivity of its description and rule languages, and with respect to its support for a broad range of non-classifier-related KR utilities (including a first-order query language, default reasoning, production rules, multiple knowledge bases, multiple worlds, and object-oriented methods).

The notions of *description*, *definition*, *subsumption*, and *classification* as presented in this paper have become standard within the "terminological logic" community [PS+90]. Not yet standard is our thesis that description *unification* (performed by a classifier) should play the same role in reasoning with a description-plus-rule language that ordinary (syntactic) unification plays in reasoning with a relation-based logic. This thesis is a key component in LOOM's approach to integrating classifier technology into a broader deductive reasoning architecture.

8 Summary

In this article, we have briefly described some of the emerging techniques and uses of classifier-based reasoning systems, specifically as they apply to the LOOM knowledge representation system.

We made the assertion that current generation expert system tools fail to achieve a satisfactory integration of frame knowledge and rule knowledge. We then described a class of languages, exemplified by LOOM, that combine descriptions and rules to form a hybrid logic that *does* achieve a satisfactory level of integration. The use of classifier technology enables a form of unification over descriptions that fills a gap present in the F+R technology. In addition, classification-based inference technology is more powerful than the inference technology found in languages such as (pure) Prolog. A classifier's ability to automatically organize definitions and to detect many kinds of inconsistency can significantly benefit the task of knowledge acquisition. The unique capabilities of the classifier can be applied to enhance existing programming paradigms—we have pointed to specific enhancements to the production rule and object oriented programming paradigms.

References

[BBMR89] A. Borgida, R.J. Brachman, D.L. McGuinness, and L.A. Resnick. CLASSIC: A structural data model for objects. In *Proc. ACM-SIGMOD-89*, Portland, Oregon, 1989.

[BS85] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, pages 171–216, August 1985.

[Kas89] Robert Kasper. Unification and classification: An experiment in information-based parsing. In *Proceedings of the International Workshop on Parsing Technologies*, Pittsburg, PA, August 1989.

[Kni89] Kevin Knight. Unification: A multidisciplinary survey. *ACM Computing Surveys*, 21(1), March 1989.

[Kob90] Alfred Kobsa. Utilizing knowledge: The components of the SB-ONE knowledge representation workbench. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan-Kaufman, 1990.

[Mac88] Robert MacGregor. A deductive pattern matcher. In *Proceedings of AAAI-88, The National Conference on Artificial Intelligence*, St. Paul, MINN, August 1988. AAAI.

[Mac90] Robert MacGregor. The evolving technology of classification-based knowledge representation systems. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan-Kaufman, 1990.

[MY88] Robert MacGregor and John Yen. LOOM: Integrating multiple AI programming paradigms. Unpublished, 1988.

[Nv88] Bernhard Nebel and Kai von Luck. Hybrid reasoning in BACK. *Methodologies for Intelligent Systems*, 3:260–269, 1988.

[OK88] Bernard Owsnicki-Klewe. Configuration as a consistency maintenance task. In W. Hoeppner, editor, *GWAI-88*, pages 77–87. Springer, Berlin Germany, 1988.

[PS$^+$90] Patel-Schneider et al. Term subsumption languages in knowledge representation. *AI Magazine*, pages 16–23, Summer 1990.

[Yen89] John Yen. Using terminological models to enhance the rule-based paradigm. In *Proceedings Second International Symposium on Artificial Intelligence*, Monterrey, Mexico, October 1989.

[Yen90] John Yen. A principled approach to reasoning about the specificity of rules. In *AAAI-90, Proceedings of the National Conference on Artificial Intelligence*, Boston, MA, July 1990. AAAI.