CLASP: Integrating Term Subsumption Systems and Production Systems

John Yen, member, ieee

Department of Computer Science

Texas A&M University

College Station, TX 77843

(409) 845-5466

Yen@CSSUN.TAMU.EDU

Robert Neches and Robert MacGregor

USC / Information Sciences Institute

4676 Admiralty Way,

Marina del Rey, CA 90292

Appeared in *IEEE Transactions on Knowledge and Data Engineering, Vol. 3, No. 1, pp. 25 - 32, 1991.* The research described in this paper was supported by Engineering Excellence Fund at Texas A&M University, by DARPA under Contract No. MDA903-87-C-0641, and by the Air Force Logistics Command under Contract No. F33600-87-C-7047. Views and conclusions contained in this paper are those of the authors, and should not be interpreted as representing the official opinion or policy of the sponsoring agencies.

Abstract— Rules and frames are two knowledge representation schemes whose strengths and weakness are complementary to each other. Although several previous systems have attempted to integrate the two, few efforts have been made to incorporate the terminological knowledge of the frame-based systems into the rule-based paradigm. To achieve a deep integration of the two schemes, we have developed and implemented a CLASsification-based Production system (CLASP). This paper describes two major processes of CLASP: a semantic pattern matcher and a pattern classifier. The semantic pattern matcher extends the pattern matching capabilities of rule-based systems through the use of terminological knowledge. The pattern classifier enables the system to compute a rule's specificity, which is useful for conflict resolution, based on the semantics of its left hand side. The paradigm not only enhances the reasoning capabilities of rule-based systems, but also helps to reduce the cost of maintaining such systems because definitional knowledge is explicitly represented in a form that facilitates sharing and minimizes duplication of effort.

Index Terms: Rule-based Systems, Knowledge Representations, Expert Systems, AI architectures, Software Engineering.

I. Introduction

This paper describes a unified AI programming paradigm that tightly integrates rule-based and frame-based paradigms, by providing the capability to use *terminological reasoning* within the pattern matching and control components of a rule processing system architecture. CLASP, an implementation of such a paradigm, integrates a term subsumption language (LOOM) [1], production rules, and methods in object-oriented programming.

The term *Term Subsumption Languages* refers to knowledge representation formalisms that employ a formal language, with a formal semantics, for the definition of terms (more commonly referred to as concept or classes), and that deduce whether one term subsumes (is more general that) another [2]. These formalisms generally descend from the ideas presented in KL-ONE [3]. Term subsumption languages are more principled than both semantic networks and frames because the languages have well-defined semantics, which is often missing from frames and semantic networks [4, 5]. In the last few years many knowledge representation systems have been built using term subsumption languages, including KRYPTON, KL-TWO, NIKL, BACK, SB-ONE, LOOM, and CLASSIC.

We will use the following example to illustrate definitional knowledge that can be expressed in term subsumption languages. Suppose **Successful-father** is defined as a father all whose children are college graduates. This can be expressed as[1]

```
(defconcept Successful-Father (:and Father (:all Child College-Graduate)))
```

with the following logic interpretation:

$\forall\, x$ `Successful-father`$(x) \Longleftrightarrow$ `Father`$(x) \wedge [\, \forall\, y$ `Child`$(x, y) \Rightarrow$ `College-Graduate`$(y)\,]$

The major strength of term subsumption systems is the reasoning capabilities offered by a *classifier*. The classifier is a special purpose reasoner that automatically infers and maintains a consistent and accurate taxonomic lattice of logical subsumption relations between concepts [7]. Based on such inferential power, term subsumption systems tidily handle the pattern matching problem of recognizing John as a successful-father, given facts such as "John is a male person", "John has two children", "Philip is John's son", "Angela is John's daughter", "both Philip and Angela are college graduates".

The integration of terminological capabilities with rules is intended to address three problems with

[1] We use the syntax of LOOM knowledge representation system[6] to define concepts and relations in this paper.

rule-based systems that critics have identified as hindering system maintenance and limiting the ability to generate high-quality explanations and justifications [8, 9]. First, rules fail to explicitly separate different kinds of knowledge; different clauses in the same rule may implicitly serve to represent contexts, affect control, or capture structural knowledge [10, 11]. Because the intent behind them is unclear, it is hard to explain rules and difficult to determine how to correctly add or revise them. Second, the meaning of the terminology used by the rules is often ill-defined [12]. This makes it difficult to determine when rules are, or should be, relevant to some shared abstraction – which, in turn, makes it difficult to find and change abstractions. Third, it is difficult to structure a large set of rules [13]. This makes it difficult to decompose the set into smaller, more comprehensible and maintainable subsets.

This paper will focus on two components of the CLASP architecture that serve to alleviate these problems. The first is a semantic pattern matcher that combines LOOM's KL-ONE style automatic classifier with a RETE match algorithm. This allows relatively efficient triggering of rules against data items which do not match the rules' conditions symbol-for-symbol, but which can be determined inferentially to satisfy terminology utilized in the rules' condition sides. The second component is a pattern classifier, which extends the concept classification algorithm in term subsumption system to enable computation of specificity relations between arbitrary conjunctive patterns in the condition sides of rules. In the sections following, we will briefly review related work, describe the architecture of CLASP with emphasis upon the pattern matching and pattern classification components, and summarize collateral research issues raised by the work.

II. Related Work

KEE, ART, and Knowledge Craft combine frames and rules, but do not support automatic classifiers. Thus, the frames provide a vocabulary that can be used within the rules, and a means for partitioning, indexing, and organizing rules[13]. But the pattern matching process can not avail itself of terminological inferences and the burden of correctly maintaining the frame hierarchy falls totally upon the user.

KL-TWO also provides a *noticer* mechanism for users to define demons that get executed when their conditions are met [14]. Although the noticer has improved the expressive power and control of demons, it lacks a global control mechanism like the recognize-act cycle of production systems. Moreover, the

noticer does not fully support matching facility for conjunctive patterns.

CONSUL[15, 16] was the first attempt to integrate rules into a term subsumption system. Rules in CONSUL demonstrated a new philosophy on the use of rules: they were used solely for mapping one description (which was represented as a concept) to another until the system knew how to act on it (i.e., until the description was transformed to an "actionable" one). Similar transformation rules have also been used to map linguistic structures to domain-specific knowledge in an natural language application built using SB-ONE [17]. Built in NIKL[18], CONSUL used its classifier to match data with rules. It also used the taxonomic structure of its knowledge base to infer specificity relations between rules. However, its inference architecture and rule language were not as general as that of a production system. Moreover, due to the limitations of NIKL, CONSUL could only operate upon class concepts and could not match rules against data instances. An integration of rules and inheritance networks has also been reported in [19].

III. A Classification-based Production System

CLASP extends the rule-based paradigm by taking advantage of terminological knowledge and classification reasoning in term subsumption systems.

A. The Representation of Productions

A production rule has two major components: a condition side that describes its triggering condition and an action side that contains the actions to be executed when the triggering condition is met. In CLASP, the conditions of a production can refer to terms defined in the terminological knowledge base, and the action-side explicitly describes the task that the rule intends to perform. Each task is associated with a set of *methods*, which describe various ways to accomplish the task in different situations. Examples of rules and methods are shown in Figure 1

A condition-side pattern is a conjunction of conditions, each of which consists of a type, a predicate, and a list of arguments. Possible types of a condition are `:TRUE` `:NOT-TRUE` and `:FAIL`. A `:TRUE` or `:NOT-TRUE` condition checks that its parameters hold or do not hold for a given predicate. A single `:FAIL`

```
(defrule display-car-status-rule
  :when (:and (Door-opened ?car))
  :perform (display-status ?car))

(defmethod display-status (?x)
  :situation (:and (Car ?x))
  :action  (print ``All systems go''))

(defmethod display-status (?x)
  :situation (:and (Car ?x)
                   (Has-tire ?x ?tire)
                   (Has-pressure ?tire Low))
  :action (print ``Low Tire Pressure''))
```

Figure 1: Examples of Productions and Methods

condition is allowed in a left-hand side and always appears last. A rule with a `:FAIL` condition will only instantiate when the rest of the condition is true and the `:FAIL` condition has just changed from true to not-true. For example, a rule with the condition `(:and (Foreigner ?x) (:Fail (Student ?x)))` will trigger whenever a foreign student loses his/her student status. A condition's predicate is a concept or a relation defined in the terminological knowledge base. A concept serves as a unary predicate, and a relation serves as a binary predicate. Arguments can be either constants or variables [2].

B. An Overview of the CLASP Architecture

Figure 2 shows the general architecture of CLASP. Productions, methods, and facts are stored in a rule base, a method knowledge base, and a facts database, respectively. The rule compiler translates definitions of individual productions and methods into their internal representations, and uses the pattern classifier to compute the specificity of rules. The facts manager updates the facts database whenever the factual knowledge is modified. An important component of the facts manager is a semantic pattern

[2] Variables are denoted by symbols starting with the question mark "?".

Figure 2: The architecture of the classification-based production system

matcher that detects changes to the conflict set arising from changes to the assertional data. The production interpreter selects productions from the conflict set and executes their actions, which invokes tasks to be performed, retrieves applicable methods, selects methods from those retrieved, and, finally, executes the bodies of selected methods. The rule specificity computed by the pattern classifier is used both for selecting productions and selecting methods. The semantic pattern matcher and the pattern classifier are the primary extensions of the conventional rule-based paradigm. Hence, we will focus the remaining discussion on these two components.

C. The Semantic Pattern Matcher

The semantic pattern matcher uses terminological knowledge to match data with rule conditions. To illustrate this, consider the example shown in Figure 4. The facts do not match the condition of rule at the symbol level; however, they will match R1 (with the variable bindings ?x = Bob, ?y = Lina) if we also consider the definitions of **Daughter** and **Car-owner**, which are shown in Figure 3.

A semantic pattern matcher augments conventional (symbolic) pattern matcher with a deductive

```
(defconcept Person (:primitive))
(defconcept Male (:and Person :primitive))
(defconcept Female (:and Person :primitive))
(defconcept College-graduate (:and Person :primitive))
(defconcept Female-College-graduate (:and Female College-graduate))
(defrelation Child (:and :primitive (:domain Person) (:range Person)))
(defrelation Daughter (:and Child (:range Female)))
(defconcept Father (:and Male (:at-least 1 Child )))
(defconcept Successful-Father (:and Father (:all Child College-graduate)))
(defrelation Has-car (:and :primitive (:domain Person) (:range Vehicle)))
(defconcept Car-owner (:and Person (:at-least 1 Has-car)))
```

Figure 3: An Example of Terminological Knowledge

```
(defrule R1
        :when (:and (Daughter ?x ?y)
                    (Car-owner ?y))
        ... )

Facts: (Child Bob Lina)
       (Female Lina)
       (Has-car Lina Bob's-old-car)
```

Figure 4: An example of semantic pattern matching

reasoning component, which is based on the inference rules captured by the terminological knowledge. The **realizer** in hybrid term subsumption systems offers this kind of deductive reasoning capability. Therefore, an efficient semantic pattern matcher can be implemented by integrating the realizer with an efficient pattern matching algorithm (e.g., RETE match algorithm).

C.1 The CONCRETE Matching Network

The semantic pattern matcher in CLASP is implemented by combining Forgy's Rete matching algorithm[20] with the deductive matcher of LOOM [6], which is a counterpart of Vilain's KL-TWO's realizer[14]. The rule compiler builds a *CONcept Classification RETE* (CONCRETE) net as rules are loaded into the rule base. As external changes are made to the facts database, the LOOM matcher computes assertional changes that can be deduced from the terminological knowledge, and it informs the CONCRETE net about relevant changes. For example, when the fact **(Has-car Lina Bob's-old-car)** is added to the facts database, the LOOM matcher deduces, among other things, the proposition that Lina is a car owner, i.e., **(Car-owner Lina)**. This infered fact, together with the asserted facts (e.g., **(Has-car Lina Bob's-old-car**), is sent to the relevant top level nodes in CONCRETE. The CONCRETE net stores partial matching results, propagates assertional changes informed by the LOOM matcher down the network, and generates addition or deletion of rule instantiations, which are used to update the conflict set. To achieve an efficient net structure, we do a data dependency analysis on the patterns to avoid long chains of CONCRETE nodes and early unnecessary joins.

D. The Pattern Classifier

The pattern classifier organizes patterns into a lattice where more specific patterns are below more general ones, based on the definitions of terms referred to in the patterns. Using the pattern classifier, CLASP can compute a well-defined specificity relation between rules during compile time [21]. *Specificity* is a classic conflict resolution heuristic used by many production system languages (e.g., OPS5) [22]. In addition, common sense reasoning often relies on the specificity of a rule's antecedents to override conclusions drawn by more general rules when they contradict the more specific rule. Our approach gives specificity a definition based on semantics, where previously it was definable only in terms of structural correlates like the number of condition clauses.

D.1 Defining Pattern Subsumption Relations

Conceptually, a pattern p_2 is more specific than (i.e., is subsumed by) a pattern p_1 if, for all states of the facts database, a match with p_2 implies a match with p_1. A state of the facts database in a term

subsumption systems is specified by a set of assertions (i.e., `tell` operations), which we will call a *world description* $\mathcal{W}$. The expression $p^{\mathcal{W}}(\vec{x})$ denotes that $\vec{x}$ satisfies the condition of the pattern p **in the world described by** $\mathcal{W}$[3].

Definition 1 *Suppose p_1 and p_2 are two patterns whose predicates are defined in a terminological knowledge base $\mathcal{T}$. The pattern p_1 subsumes p_2, denoted as $p_1 \succeq p_2$, iff*

$$\forall\, \mathcal{W}\ (\ \exists\, \vec{x}\ \ p_2^{\mathcal{W}}(\vec{x}) \Rightarrow \exists\, \vec{y}\ \ p_1^{\mathcal{W}}(\vec{y})\ ) \tag{1}$$

where $\vec{x}$ and $\vec{y}$ denote vectors of variable bindings.

The definition allows patterns with different number of variables to be compared with each other. This is important for using the subsumption of patterns as a useful measure of the specificity of rules, for the condition of a specific rule often introduce extra variables to test a situation that is more complicated than the condition of a general rule. Enforcing that two subsuming patterns have same number of variables will render the pattern subsumption taxonomy useless for controlling the firing of rules.

To determine whether a conjunctive pattern p_1 (i.e., a conjunction of non-negated literal) subsumes another conjunctive pattern p_2, we need to find a substitution that replaces variables in p_1 by arguments in p_2 such that the latter terminological implies the former under the substitution. *Terminological implication*, denoted as $\stackrel{\mathcal{T}}{\Rightarrow}$, is defined as follows: $p_2 \stackrel{\mathcal{T}}{\Rightarrow} p_1$ iff

$$\forall \mathcal{W}\ \ \left[\forall \vec{x}\ \left(p_2^{\mathcal{W}}(\vec{x}) \Rightarrow p_1^{\mathcal{W}}(\vec{x'})\right)\right] \tag{2}$$

where $\vec{x'}$ is a subvector of $\vec{x}$, and $\mathcal{T}$ denotes the terminological knowledge base where the predicates of p_1 and p_2 are defined. More formally, we have the following Theorem.

Theorem 1 *Suppose p_1 and p_2 are two conjunctive patterns. The pattern p_1 subsumes p_2 iff there exists a* subsumption substitution *θ that replaces variables of p_1 by p_2's variables or constants such that p_2 terminologically implies $p_1\theta$ based on the terminological knowledge base $\mathcal{T}$, i.e.,*

$$p_1 \succeq p_2 \ \textit{iff}\ \exists \theta \ \textit{such that}\ p_2 \stackrel{\mathcal{T}}{\Rightarrow} p_1\theta. \tag{3}$$

Proof of the theorem can be found in [23].

[3] A more formal definition of pattern instantiation can be found in [23].

P1: `(:and (father ?x ?y) (father ?x ?z) )`

P2: `(father ?u ?v)`

Figure 5: An Example of Two Indifferent Patterns

The subsumption substitution S can also be viewed as a *mapping* because it maps each of p_1's variables to a variable or a constant in pattern p_2. We will use the terms "subsumption substitution" and "subsumption mapping" interchangeably in our discussion. A subsumption mapping is a proof that p_2 is more specific than p_1 because, for any instantiation of p_2's variables, we can construct an instantiation of p_1's variables from the subsumption mapping. Thus, matching p_2 implies matching p_1 if a subsumption mapping exists.

We further define the following relationships between patterns:

- Two patterns are *indifferent*, denoted by $\sim$, if and only if they subsume each other, i.e.,

$$P_1 \sim P_2 \quad \Leftrightarrow \quad P_1 \succeq P_2 \ \wedge \ P_2 \succeq P_1.$$

  Indifferent patterns are merged in the specificity lattice. Conceptually, two patterns are indifferent if, for any states of the fact database, either both patterns match or neither of them matches the fact database.

- Two patterns are *equivalent*, denoted by $\equiv$, if they are indifferent and the subsumption mapping is a one-to-one mapping between variables of the two patterns. Two indifferent patterns may not be equivalent. For instance, the patterns P1 and P2 in Figure 5 are indifferent because P1 subsumes P2 under the substitution { ?u/?x, ?v/?y, ?v/?z } and P2 subsumes P1 under the substitution { ?x/?u, ?y/?v } or { ?x/?u, ?z/?v }. But the two patterns are not equivalent because the mapping is not on-to-one, resulting in different instantiations for a given facts database.

- Two patterns are *equal*, denoted by $=$, if they are equivalent without variable substitution.

The subsumption substitution differs from substitution in unification in that it is unidirectional. It substitutes variables/constants of a child pattern for variables of a parent pattern, but not the other way. This distinction is due to the fact that a subsumption test is meant to test implications, which is directional, while unification is meant to test equality, which is bidirectional.

SubsumesP(p_1, p_2)

1. $\overline{p_1} \leftarrow$ Normalize(p_1)

   $\overline{p_2} \leftarrow$ Normalize(p_2)

2. For each literal l_1^i in p_1 Do

   For each literal l_2^j in p_2 Do

   If l_2^j is a potential subsumee of l_1^i

   If l_1^i is unary

   Then record the arguments in l_2^j as a potential image of the variable in l_1^i

   Else record the mapping constraint imposed by the pair of literals

3. Search for a mapping that satisfy the mapping constraints of all literals of p_1

Figure 6: A General Subsumption Algorithm for Conjunctive Patterns

D.2 A General Approach to Classifying Conjunctive Patterns

Having defined the subsumption of patterns, this section describes a general approach for testing the subsumption of conjunctive patterns. Our general pattern classification algorithm, which is shown in Figure 6, consists of three major steps. First, each pattern is normalized by making explicit in the pattern any unstated conditions logically implied by the patterns and the terminological knowledge. Second, the algorithm attempts to reduce the space of possible subsumption mappings using subsumption relationships between predicates. Third, it performs a dependency-directed backtracking to search for a subsumption mapping. If a subsumption mapping is found, the algorithm returns true, else it returns false.

Defining Pattern Normalization The normalization step transforms each pattern into an equivalent normalized pattern. Intuitively, a pattern is normalized if it contains no implicit conditions other than those that can be deduced easily from the subsumption of concepts and subsumption of roles. For instance, in normalizing a pattern, we do not need to transform a condition such as `(Father ?x)` into a tedious subpattern like `(Animal ?x)`$\wedge$`(Person ?x)`$\wedge$`(Male ?x)`$\wedge$`(Father ?x)`. More formally, we define a normalized pattern as follows:

Definition 2 *A pattern p is said to be* normalized *iff*

$$\forall l,\ \textit{if}\ p \stackrel{\mathcal{I}}{\Rightarrow} l,\ \textit{then}\ \exists l'\ \textit{in}\ p\ \textit{such that}\ l' \stackrel{\mathcal{I}}{\Rightarrow} l \tag{4}$$

where l and l's are literals with the same number of arguments.

We say a pattern $\overline{p}$ is a *normalized form* of p if and only if $\overline{p}$ is normalized and p equals $\overline{p}$ (i.e., they are equivalent without variable substitution). The definition of normalized patterns is illustrated using the rules in Figure 8. The left-hand-side condition of R3 is not normalized because no unary condition in the pattern implies `(College-Graduate ?w)` even though it is implied by the pattern based on the definition of `Successful-father`. Examples of normalized rules can be found in Figure 9.

Normalizing a pattern is analogous to completing a concept definition in KL-ONE's classifier[7]. Both of them attempt to compute the deductive closure of the objects to be classified before actually classifying them for the same reason: to gain efficiency for the subsumption test. The actual algorithm for normalizing patterns depends on the language used for defining terms.

The rationale behind normalizing patterns is to simplify the search step. Without the normalization process, the search for a subsumption substitution would have to consider the possibility that a condition in the parent pattern subsumes a conjunctive subpattern of the child pattern. For example, consider the rules **R2** and **R3** in Figure 8. The condition `(College-graduate ?y)` in R2 subsumes the subpattern `(Successful-Father ?z)` $\wedge$ `(Child ?z ?w)` of R3's condition under the substitution `?y/?w`. Having deduced the conditions implied by these conjunctive subpatterns during the normalization process, the subsumption test only needs to consider pairs of conditions (one from the parent pattern, one from the child pattern) with the same arity for testing subsumption possibility of the two patterns. Thus, normalizing patterns significantly reduces the complexity of the subsumption test. The following theorem formally states the impact of pattern normalization to the pattern subsumption test.

Theorem 2 *Suppose p_1 and p_2 are two normalized conjunctive patterns:*

$$p_1 = l_1^1 \wedge l_2^1 \wedge ...l_n^1 \tag{5}$$

$$p_2 = l_1^2 \wedge l_2^2 \wedge ...l_m^2 \tag{6}$$

where l_i^1 and l_j^2 are literals without negations. The pattern p_1 subsumes p_2 if and only if there exists a subsumption substitution S such that every literal l_i^1 in p_1 subsumes at least one literal in p_2 with the

Expression e	*Interpretation* $[\![e]\!]$
(**:and** C_1 C_2)	$\lambda x.\, [\![C_1]\!](x) \wedge [\![C_2]\!](x)$
(**:and** R_1 R_2)	$\lambda xy.\, [\![R_1]\!](x,y) \wedge [\![R_2]\!](x,y)$
(**:at-least** 1 R)	$\lambda x.\, \exists y.\, [\![R]\!](x,y)$
(**:all** R C)	$\lambda x.\, \forall y.\, [\![R]\!](x,y) \rightarrow [\![C]\!](y)$
(**:domain** C)	$\lambda xy.\, [\![C]\!](x)$
(**:range** C)	$\lambda xy.\, [\![C]\!](y)$

Figure 7: Semantics of Some Term-Forming Expressions

same arity, i.e.,

$$p_1 \succeq p_2 \quad \Leftrightarrow \quad \exists\theta \left[\forall l_i^1 \ \textit{in}\ p_1,\ \exists l_j^2 \ \textit{in}\ p_2,\ \textit{such that}\ l_j^2 \stackrel{\mathcal{T}}{\Rightarrow} l_i^1\theta\right] \tag{7}$$

where l_i^1 *and* l_j^2 *have the same number of arguments.*

We will call l_j^2 the *subsumee* of l_i^1. Comparing Equations 3 and 7, we can see that we have significantly reduced the complexity of subsumption test by normalizing the patterns.

E. Types of Normalization Steps

Five types of normalization steps have been implemented in CLASP: (1) domain and range deductions, (2) normalizing unary conditions, (3) normalizing binary conditions, (4) value restriction deductions, and (5) at-least-one deductions. Each normalization step will be described and illustrated with examples, based on Figures 3 and 8. These normalization steps are correct because each one transforms a pattern into an equivalent one based on the semantics of LOOM's term-forming expressions in Figure 7.

1. *Domain and Range Deduction:* This step deduces unary conditions about variables that appear in a binary condition using the domain and the range of the binary condition's predicate (i.e., a relation). For instance, this step will infer an implicit condition for R2 `(Female ?y)` from the range of `Daughter` relation.

```
(defrule R2
        :when (:and (College-graduate ?y)
                    (Daughter ?x ?y)
                    (Car-Owner ?y))
        ... )

(defrule R3
        :when (:and (Successful-Father ?z)
                    (Child ?z ?w)
                    (Female ?w)
                    (Has-Car ?w ?c))
        ... )
```

Figure 8: An example of two rules before normalization

2. *Normalizing Unary Conditions:* Unary conditions that involve the same variables are replaced by one unary condition whose predicate is the conjunction of the unary predicates (i.e., concepts) in the original pattern. This ensures that all patterns are transformed into a canonical form where each variable has at most one unary condition. The condition-side of R2 thus is normalized to combine three unary conditions about the variable "?x" into one condition `(Female-College-graduate-Car-Owner ?y)` where `Female-College-graduate-Car-Owner` is the conjunct of `Female`, `College-graduate`, and `Car-Owner`.

3. *Normalizing Binary Conditions:* Binary conditions with the same arguments are collected, and replaced by a new composite binary condition that takes into account the unary conditions of its domain variable and its range variable. This ensures that all normalized patterns have at most two binary conditions for each variable pair (the argument position of the variables can be switched). For instance, the subpattern in R3 `(Child ?z ?w)` $\wedge$ `(Female ?w)` can be further transformed to `(Daughter ?z ?w)` $\wedge$ `(Female ?w)` by this normalization step.

4. *Value Restriction Deduction:* Suppose a pattern contains conditions of the form

   `(:and (`C_1 `?x) (R ?x ?y) ... )`,

and the definition of C_1 in the terminological space has a value restriction on R, say C_2. Then the pattern is equivalent to a pattern that has an additional unary condition (C_2 ?y). For example, the literal `(College-graduate ?w)` will be added to the condition of R3 by this normalization step because successful-father has been defined as a father all whose children, which include daughters, are college graduates as shown in Figure 3.

5. *At-least-one Deduction:* A pattern containing two conditions in the form of

   `(:and ... (C ?x) ... (R ?x` α`) ...),`

   where α is either a variable or a constant, is transformed to one that replaces C by the concept C' defined below, which has an additional at-least-one number restriction on the relation R.

   ```
   (defconcept C' (:and C (:at-least 1 R))).
   ```

   This will cause the literal `(Car-Owner ?w)` to be added to R3's condition because `Car-Owner` has been defined to be a person who has at least one car.

Figure 9 shows the condition-sides of R2 and R3 after they have been normalized. It is easier to see that R3 is actually more specific than R2, which was not obvious prior to normalization.

F. Reducing the Search Space

Although an exhaustive search that considers all possible subsumption mappings can not be avoided in the worst case, the search space can be significantly reduced in most cases using information about the subsumption of predicates. Normally, the condition pattern of a rule consists of several different predicates, only a small percentage of which are subsumed by a predicate in another pattern. Thus, using the subsumption relationships between predicates, we can significantly reduce the search space for finding a subsumption mapping.

In general, comparing unary conditions of two patterns generates a set of potential candidates (which we call *potential images*) that a variable can map to under a subsumption mapping. Comparing binary conditions of two patterns generates *mapping constraints* on how pairs of variables should be mapped. Potential images are used to reduce the branching factor of the search space, and mapping constraints are

```
(defrule R2
        :when (:and (Person ?x)
                    (Female-College-graduate-Car-Owner ?y)
                    (Daughter ?x ?y))
        ... )

(defrule R3
        :when (:and (Successful-Father ?z)
                    (Female-College-graduate-Car-Owner ?w)
                    (Daughter ?z ?w)
                    (Has-Car ?w ?c))
        ... )
```

Figure 9: Two rules after normalization

used to prune the search tree. This is illustrated using the example in Figure 9. The condition `(Person ?x)` has two potential subsumee, `(Successful-Father ?z)` and `(Female-College-graduate-Car-Owner ?w)`; therefore, the potential images of `?x` are { `?z, ?w` } . Similarly, the binary condition `(Daughter ?x ?y)` has only one potential subsumees `(Daughter ?z ?w)`, which means that the subsumption mapping has to map `?x` to `?z` and `?y` to `?w` simultaneously.

The process of reducing the search space can also detect early failure of the subsumption test. The test terminates and returns false whenever (1) it fails to find any potential images for a variable in p_2; or (2) a binary condition in p_1 fails to find any binary condition in p_2 as a potential subsumee.

G. Searching for a Subsumption Substitution

To search for a subsumption mapping that satisfies all the constraints generated from the previous step, the pattern classifier first sorts the parent variables in increasing order of the number of their potential images, then it performs a dependency-directed backtracking. The position of a variable in the sorted list corresponds to the level it's image is assigned in the search tree. At each node in the tree, the

variables' assigned images are checked to see if they satisfy the mapping constraints. If anyone of the mapping constraints is not satisfied, the algorithm backtracks to the closest node whose assignment causes a constraint violation. If a mapping that satisfies all the constraints is found, the subsumption test returns true. Otherwise, it returns false.

H. Discussion

We have shown elsewhere that CLASP's pattern classification algorithm is sound [23]. It is also complete for a simple term subsumption language whose expressiveness is equivalent to that of $\mathcal{FL}^-$ in [24]. Further discussions on the issues regarding soundness and completeness of the subsumption algorithm can be found in [23].

Determining the subsumption of normalized conjunctive patterns is NP-complete, for it can be reduced from the problem of determining subgraph isomorphism for directed graphs, which is known to be NP-complete [25]. However, worst case rarely occur in practice. To analyze the behavior of an algorithm in reality, we have defined *normal cases* [4] and have shown that the complexity of the algorithm for normal cases is polynomial [23].

Brachman and Levesque have demonstrated that there is an important tradeoff between the expressiveness of a terminological language and the complexity of its reasoner [24]. A similarly tradeoff between the computational complexity of the normalization process and the expressiveness of the terminological language has also been investigated [23].

IV. Summary

We have presented the general architecture and an implementation of a CLASsification-based Production system (CLASP). Our main objective is to extend the benefits of classification capabilities in frame systems to the developers of rule-based systems. By structuring the condition-sides using predicates defined in the terminological spaces, the paradigm improves conventional rule-based programming in several respects. First, the pattern matching operation is based on the terminological definitions of the

[4] Using normal cases to analyze the complexity of intractable algorithm has been suggested by Bernard Nebel [26].

symbols, not just the symbols themselves. Second, conflict resolution can be based on a well-defined specificity relationship between rules, which is computed by a pattern classifier using terminological knowledge and the subsumption lattice precomputed by its classifier. Third, the paradigm encourages the development of a rich and coherent terminological knowledge base, which is shared across rules.

Representing terminological knowledge explicitly can also help to reduce the maintenance costs of rule-based systems because duplication of efforts made to define identical terms in different rule systems can be avoided if the same terminological definitions are used. Finally, the pattern classifier described in this paper can be used to extend other AI reasoning systems or programming paradigms (e.g., common-sense reasoning, planning, and problem solving) where the specificity of patterns plays an important role.

Acknowledgements

We would like to thank Paul Rosenbloom, John Granacki, Leonard Friedman, Brian Harp, and Pedro Szekely for their comments on earlier drafts of the paper. Many thanks also go to Bill Swartout and Peter Patel-Schneider for their comments on the pattern classifier, and David Benjamin for his contribution to the design of the semantic pattern matcher and the coding of its parser, and to Pat Langley for giving us access to the PRISIM code, which helps the design of CONCRETE significantly.

References

[1] R. MacGregor and R. Bates, "The loom knowledge representation language," Technical Report ISI/RS-87-188, USC/Information Sciences Institute, 1987.

[2] P. F. Patel-Schneider, B. Owsnicki-Klewe, A. Kobsa, N. Guarino, R. MacGregor, W. S. Mark, D. McGuinness, B. Nebel, A. Schmiedel, and J. Yen, "Term subsumption languages in knowledge representation," *AI Magazine*, vol. 11, no. 2, pp. 16–23, 1990.

[3] R. Brachman and J. Schmolze, "An overview of the KL-ONE knowledge representation system," *Cognitive Science*, vol. 9, no. 2, pp. 171–216, August 1985.

[4] W. A. Woods, "Whats's in a link: Foundations for semantic networks," In *Representation and Understanding: Studies in Cognitive Science*, D. Bobrow and A. Collins, editors, Academic Press, 1975.

[5] R. J. Brachman, "What is-a is and isn't: An analysis of taxonomic links in semantic networks," *Computer*, vol. 16, no. 10, pp. 30–36, October 1983.

[6] R. M. MacGregor, "A deductive pattern matcher," In *Proceedings of AAAI-88*, 1988.

[7] J. Schmolze and T. Lipkis, "Classification in the KL-ONE knowledge representation system," In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 330–332. IJCAI, 1983.

[8] W. Swartout, "XPLAIN: A system for creating and explaining expert consulting systems," *Artificial Intelligence*, vol. 21, no. 3, pp. 285–325, September 1983.

[9] R. Neches, W. Swartout, , and J. Moore, "Enhanced maintenance and explanation of expert systems through explicit models of their development," *Transactions On Software Engineering*, vol. SE-11, no. 11, pp. 1337–1351, November 1985.

[10] W. Clancey, "The epistemology of a rule-based expert system: A framework for explanation," *Artificial Intelligence*, vol. 20, no. 3, pp. 215–251, May 1983.

[11] J. S. Aikins, "Prototypes and production rules: A knowledge representation for computer consultations," Technical Report STAN-CS-80-814, Department of Computer Science, Stanford University, 1980.

[12] W. Swartout and R. Neches, "The shifting terminological space: An impediment to evolvability," In *AAAI-86, Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, AAAI.

[13] R. Fikes and T. Kehler, "The role of frame-based representation in reasoning," *Communication of the ACM*, vol. 28, no. 9, , September 1985.

[14] M. Vilain, "Kl-two, a hybrid knowledge representation system," Technical Report 5694, Bolt Beranak and Newman, September 1984.

[15] W. Mark, "Rule-based inference in large knowledge bases," In *Proceedings of the National Conference on Artificial Intelligence*. AAAI, August 1980.

[16] W. Mark, "Representation and inference in the consul system," In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 375–381. IJCAI, Morgan Kaufman, 1981.

[17] A. Kobsa, "The sb-one knowledge representation workbench," In *Proceedings of the Workshop on Formal Aspects of Semantic Networks*, February 1989.

[18] M. Moser, "An overview of NIKL, the new implementation of KL-ONE," In *Research in Natural Language Understanding*, Bolt, Beranek, and Newman, Inc., Cambridge, MA, 1983.

[19] T. Daly, J. Kastner, and E. Mays, "Integrating rules and inheritance networks in a knowledge based financial marketing consultation system," In *Hawaii Int. conf. on System Science*, January 1988.

[20] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," *Artificial Intelligence*, vol. 19, pp. 17–37, 1982.

[21] J. Yen, "A principled approach to reasoning about the specificity of rules," In *Proc. National Conf. on Artificial Intelligence*, pp. 701–707, Boston, August 1990.

[22] J. McDermott and C. Forgy, "Production system conflict resolution strategies," In *Pattern-Directed Inference Systems*, D. A. Waterman and F. Hayes-Roth, editors, Academic Press, New York, 1978.

[23] J. Yen, "Reasoning about the specificity of patterns in term subsumption-based systems," Technical Report TAMU 90-003, Department of Computer Science, Texas A&M University, February 1990.

[24] R. J. Brachman and H. J. Levesque, "The tractability of subsumption in frame-based description languages," In *Proceedings of AAAI-84*, pp. 34–37, Austin, Texas, August 1984.

[25] M. R. Garey and D. S. Johnson, *Computers and Intractability - A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, Cal., 1979.

[26] B. Nebel, "Terminological reasoning is inherently intractable," Technical Report IWBS Report 82, IWBS, IBM Deutschland, W. Germany, October 1989.