INTERPRETING DESCRIPTIONS IN A PROLOG-BASED KNOWLEDGE REPRESENTATION SYSTEM

Randy Goebel

Logic Programming and Artificial Intelligence Group
Computer Science Department
University of Waterloo
Waterloo, Ontario
Canada N2L 3GI

Abstract

Descriptions provide a syntactic device for abbreviating expressions of a formal language. We discuss the motivation for descriptions in a system called DLOG. We describe two approaches to specifying their semantics, and a method for implementing their use. We explain why some descriptions should be given a higher order interpretation, and explain how such descriptions can be interpreted in the simpler logic of Prolog. The essential idea is to constrain the domain of descriptions so that an extended unification procedure can determine description equivalence within the Prolog framework.

Introduction

A description is a syntactic device for abbreviating expressions of a formal language. For example, "The king of America is an old cowboy" might be rendered in first order predicate calculus as

$$\exists x \forall y [[\mathit{king-of-America}(y) \equiv x = y] \wedge \mathit{old-cowboy}(x)]$$

and paraphrased as "there is an x who is the unique king of America, and that x has the property 'old-cowboy'." This expression can be abbreviated, in the classical way, as $\mathit{old-cowboy}(\iota x[\mathit{king-of-America}(x)])$; we say that "$\iota$" is used to form a definite description. In first order logic, predicate arguments denote individual objects; the description operator ι has syntactically "objectified" a portion of the original sentence.

The description operator has created a new syntactic object, but the meaning or denotation of that object may not be well-defined. In this particular case, the existence and uniqueness properties presupposed by the above description have been the issue of much debate (e.g., see [Kaplan75]).

Another example of descriptive abbreviation relaxes the uniqueness assumption. For example, the expression "there is a person who is an old cowboy, and who is a Republican" might be rendered as $\exists x[\mathit{person}(x) \wedge \mathit{old-cowboy}(x) \wedge \mathit{Republican}(x)]$. We might use another description operator, say ξ (read "an"), to provide the following abbreviations:

$$\mathit{person}(\xi x[\mathit{old-cowboy}(x) \wedge \mathit{Republican}(x)]).$$
$$\mathit{old-cowboy}(\xi x[\mathit{person}(x) \wedge \mathit{Republican}(x)]).$$
$$\mathit{Republican}(\xi x[\mathit{person}(x) \wedge \mathit{old-cowboy}(x)]).$$

Each such abbreviation uses the description operator "ξ" to focus syntactic attention on the predicate "person," "old-cowboy," and "Republican," respectively. Again we must be concerned with the meaning of such descriptions. It is somewhat natural to treat all of the above abbreviations are semantically synonymous—that all assert the existence of an individual object with the three properties. But this creates the need to explain their obvious syntactic differences in some other way. One hint comes from Hilbert and his use of description operators (e.g., see [Leisenring69, Robinson79]). Hilbert used the transformations

$$\forall x I(x) \rightarrow I(\epsilon x \neg I(x))$$
$$\exists x I(x) \rightarrow I(\epsilon x I(x))$$

to simplify derivations. In other words, the transformations provided a computational advantage even though their meanings are identical. Robinson nicely expresses the intuition:

"It would seem that Hilbert terms...do capture a certain intuitive manoeuvre, which is worth formalizing: to introduce a unique name for an entity whose *existence* has been established by some previous part of an argument, so as to continue with the argument and be able conveniently to refer to it if need arises." [Robinson79, p. 291]

This rationalization of the syntactic differences is related to the object-orientation of AI representation systems, where the ability to refer to and manipulate objects is argued to be of conceptual advantage in specifying symbolic representations of domains (e.g., [Moore76, Schubert76, Norman79, Bobrow77a, Bobrow77b, Hewitt80, Steels80, Attardi81]). (Descriptions have also been used within the logic database literature (e.g., [Dilger78, Dahl80, Dahl82]).)

In AI, part of the reluctance to adopt a logical interpretation of descriptions must be attributed to the long-standing confusion and controversy about their meaning—logicians do not generally agree on the semantics of descriptive terms (e.g., see [Carroll78]). The reluctance to analyze descriptions in a logical framework probably results from a traditional misunderstanding of the role of logic (cf. [Hayes77]).

Descriptive terms in DLOG

DLOG is a representation system implemented in Prolog. It is a "system" in the sense that it provides a general representation language (including various kinds of descriptions), a query evaluation mechanism, a simple integrity maintenance scheme, and an abstract description of the intended semantics of the representation language independent of implementation. Although originally developed to to help describe concepts in the Department Database (DDB) domain, the DLOG system is domain independent in the sense of traditional data base management systems.

The descriptions in DLOG are motivated by a desire for brevity in describing undergraduate degree program requirements in the DDB. For example, suppose that "CS115 or a 1.5 unit elective" is a requirement. The desire is to avoid an expression like

$$\begin{aligned} &\mathit{requirement}(BScCS, CS115) \\ &\quad \vee [\exists x[\mathit{requirement}(BScCS, x) \\ &\qquad \wedge \mathit{units}(x, 1.5) \wedge \mathit{elective}(BScCS, x)]] \end{aligned}$$

in favour of something like

$$\begin{aligned} &\mathit{requirement}(BScCS, \\ &\quad \kappa x[x = CS115 \\ &\qquad \vee x = \kappa y[\mathit{elective}(BScCS, y) \\ &\qquad\quad \wedge \mathit{units}(y, 1.5)]]) \end{aligned}$$

by using some appropriately defined description operator "K." Similarly, descriptions of sets were seen to be useful. For example, the expression "at least 12 CS courses" might suggest the use of an expression like that in fig. 1. where the components grouped bv the left brace must be written once for each set of twelve CS courser. This is clearly a tiresome way to express the assertion, and furthermore, would require extensive modification after any new CS course was created (e.g., by adding an assertion like "CS-course(CSI23)"). We would prefer something like

$$\begin{aligned}&requirement(BScCS,\\ &\qquad \kappa x|set(x)\\ &\qquad\qquad \wedge cardinality-of(x)=12\\ &\qquad\qquad \wedge\forall y[y\in x\supset CS-course(y)]|)\end{aligned}$$

where "K" is another appropriate description forming operator.

Similar motivations have given rise to the following description forms in DLOG.

Definite individuals. DLOG's definite individual provides a shorthand syntax for referring to a unique individual whose name is unknown. Intuitively, the variable binding symbol V can be read as the English definite article "the." For example, we might refer to "the head of Computer Science" as

$$\iota x|head-of(x,ComputerScience)|$$

If our description distinguished the intended individual, then we need never know which individual constant actually names that domain individual. We normally expect that the variable bound with the symbol V appears somewhere in the formula that constitutes the body of the description.

Indefinite individuals. When we need to refer to "any old a" with some property specified by a formula '(a)\ we can use an indefinite individual. We use the variable binding symbol V as the English indefinite article "a" or "an." For example, "a course with course number greater than 300" might be referred to by the indefinite individual

$$\epsilon x_1|course(x_1)\wedge\exists x_2|course-no(x_1,x_2)\wedge x_2\geq 300||$$

As for definite individuals, we normally expect that the variable bound with the symbol 'E' appears somewhere in the formula that constitutes the body of the description

Definite sets. A definite set is used to refer to a set consisting of all individuals that satisfy some property. The name "definite" is used to correspond to its use in "definite individual." A definite set is "definite" because it refers to all individuals *in the current knowledge base* that satisfy the specified property. For example, "the set of all numerical analysis courses" might be designated as

$$\{x_1:course(x_1)\wedge topic-of(x_1,NA)\}$$

Here the braces "{ }" serve as the description forming symbol.

Indefinite sets. Indefinite sets are "indefinite" in the sense that they refer to one of a set of sets. Like indefinite individuals, they are intended to be used to refer to "any old set" that is an element of *the set of sets* that satisfy the specified properties. For example, the indefinite set

$$\{x_1,X_1:course(x_1)\wedge cardinality-of(X_1,3)\}$$

is the DLOG term that represents an arbitrary set that is a member of the set of "all 3 element sets of courses."

Lambda constants. Lambda constants were introduced to capture a kind of individual occurring naturally in the DDB domain: regulations. For example, in describing a typical degree program, we must classify all kinds of requirements for that program, e.g., "nobody can register if they're under 16 years old" refers to a regulation that uses the lambda constant

$$\lambda x_1[\exists x_2|age-of(x_1,x_2)\wedge x_2\geq 16|]$$

$$\left\{\begin{aligned}&requirement(BScCS,CS115)\\ &\wedge requirement(BScCS,CS215)\\ &\wedge\\ &\cdot\\ &\cdot\\ &\cdot\\ &\wedge requirement(BScCS,CS442)\end{aligned}\right.$$
$$\vee\left\{\begin{aligned}&requirement(BScCS,CS118)\\ &\wedge requirement(BScCS,CS215)\\ &\wedge\\ &\cdot\\ &\cdot\\ &\cdot\\ &\wedge requirement(BScCS,CS442)\end{aligned}\right.$$

Figure 1. "...at least 12 CS courses"

In this way regulations can be placed in relation to other individuals and sets, e.g.,

$$\begin{aligned}&program-prerequisite(BScMajors,\\ &\qquad first,\\ &\qquad\qquad \lambda x_1[\exists x_2|age-of(x_1,x_2)\wedge x_2\geq 16||)\end{aligned}$$

says "one of the regulations for the first year of a BScMajors program is that an individual be at least 16 years old."

Semantics of descriptions

We have suggested that there is more to the meaning of descriptions than their denotation in a first order language. In DLOG (and, we claim, in any representation system) there are at least two aspects to the meaning of descriptions.

One important aspect is the traditional specification of denotational semantics along Tarskian lines: given a well-defined class of formal expressions, one specifies a systematic way in which expressions and their parts can be attributed denotations in an interpretation. Two assumptions underlying this methodology are (1) that the expressions in question are being evaluated as to their truth; and (2) that the denotation of complete expressions depends solely on the denotations of their parts.

Another important aspect, often overshadowed by concerns of the former, is the *intended meaning* of such formal expressions when they are being formed (e.g., by a user), during their use in assertions (e.g., when adding facts to a knowledge base), and during their use in queries (e.g., when requesting that facts be verified with respect to a knowledge base). In this regard, the use of descriptive terms impinge on philosophical problems associated with names and their use (e.g., [Donnellan66, Brinton77, Katz77, Linsky77)). A most common example is the difference between referential and attributive use of descriptions. The issue is whether a description is intended to refer to a known referent (referential), or unknown referent (attributive). Apparently only a few AI researchers have considered the problem (e.g., |Schubert76, Ortony77]). These and related issues are further discussed in [Goebe!84, Goebel85]

Approaches to specifying DLOG's semantics

To attempt a coherent description of a Prolog-based mechanism for proving existential formulas in DLOG, one must first select a methodology for specifying the meaning of descriptions. The most common method is Russell's contextual definition [Kaplan75j. Contextual definition is essentially macro definition, e.g., any string of the form $\Phi(\iota x[\Psi(x)])$ is replaced with

$$\exists x \forall y[[\Psi(y) \equiv x = y] \wedge \Phi(x)].$$

The meaning of the description is specified by the logic from which the definition is taken.

A related issue is the meaning of descriptive terms when their proof-theoretic preconditions fail. For example, a constructive proof of $\exists x\, \Phi(x) \wedge \Psi(x)$ will produce a referent of the description in $\Phi(\xi[\Psi(x)])$, but what does the latter mean when a proof of the former fails? Under one popular theory [Kaplan75, p. 215], the meaning of such descriptions whose logical preconditions fail have been specified by convention, e.g., a failing description refers to a designated null constant that lies outside the domain of discourse.

First order semantics

One possible choice for the description defining language is first order logic. The overwhelming advantage of first order semantics is simplicity; an abstract, implementation independent specification of semantics is worthwhile in that it provides a simple way to understand the complexity of the actual system.

Using first order logic, most of the intended meaning of DLOG descriptions can be specified in a relatively straightforward way. The individual descriptions (definite and indefinite) are specified as above; sets can be axiomatized with a *set* relation and a set membership relation "$\in$" (DLOG set theory is finite, thus very simple). However, lambda terms rely on semantic notions foreign to first order logic; their definition here requires the use of meta language concepts.

Contextual definitions for set descriptions are defined as follows. The sentence

$$(\Phi(\{x_1 : \Psi(x_1)\})$$

contains no set variables—the term $\{x_1 : \Psi(x_1)\}$\ describes a set consisting of *all* individuals *a* such that $\Psi(\alpha)$ is true. In a first order language that distinguishes set variables X_1 X_2, X_3,..., the definition of sentence (1) is rendered as

$$\exists X_1.[\Phi(X_1) \wedge \forall x_1.[x_1 \in X_1 \equiv \Psi(x_1)]]$$

This can be read as "there is a set X_1 that has property Φ, and all individuals x, in the set have property ¥." Because the only defining property of a definite set is the property attributed to each of its members, its uniqueness is easy to establish. In contrast to definite individuals, there is only one extension for each definite set so a further specification of uniqueness is not required.

The contextual definition of indefinite sets can be approached in a similar way. We can view

$$\Phi(\{x_1, X_1 : \Psi(x_1) \wedge \Omega(X_1)\})$$

as having the definition

$$\exists X_1 \forall x_1[[x_1 \in X_1 \supset \Psi(x_1)] \wedge \Omega(X_1) \wedge \Phi(X_1)]$$

This says that "some set X_1 that has the property O and whose elements each have property Ψ, also has property Φ." Intuitively, the indefinite set construction specifies a set that fits the description, similar to the way that an indefinite individual specifies an individual that fits its description.

DLOG lambda constants provide the user with a method of asserting axioms about unary predicate abstractions, intuitively interpreted as regulations. For example, the assertion

$$\Phi(\lambda x_1 \Psi(x_1))$$

can be interpreted as asserting that the property Φ is true of the regulation named by $\lambda x_1 \Psi(x_1)$). These terms are useful because they allow a user to assert relations about properties. Intuitively, lambda constants are most reasonably interpreted as a special kind of constant, indexed for retrieval by the terms they appear with. However, they cannot be manipulated without the definition of an application mechanism. This definition relies on a meta relation *satisfies*, which is defined in terms of the provability meta relation *derivable* (cf. [Bowen82]). The *satisfies* meta predicate then provides a method for testing whether an individual satisfies the relation denoted by a DLOG lambda constant. That is,

$$\forall x_1[satisfies(x_1, \lambda x_2 \Psi) \equiv derivable(DB, \Psi(x_1))]$$

For any individual constant a of a DLOG database DB, $satisfies(\alpha, \lambda x_1 \Psi(x_1))$ holds if and only if $\Psi(\alpha)$ holds in DB. An assertion of the form

$$satisfies(\alpha, \lambda x_1 \Phi(x_1)) \qquad (2)$$

is interpreted to mean that, in the current database,

$$\Phi(\alpha) \qquad (3)$$

is derivable. Indeed (2) is a dumsy alternative to (3), but by using lambda constants in this way, we not only provide a way of asserting axioms about regulations, but also a way of using those regulations in question answering. The *satisfies* predicate provides the mechanism for applying lambda constants as unary predicates of the current database.

An example will illustrate. The experimental DDB domain requires the description of degree requirements, which can often be expressed as lambda constants, e.g., the assertion

$$enrolment-requirement(BScCS, \lambda x[\exists y[age-of(x,y) \wedge y \geq 16]]) \qquad (4)$$

states that "an enrolment requirement of the BScCS degree is that the candidate's age is greater than or equal to sixteen." The lambda constant format allows the requirement to be asserted and queried, and the *satisfies* predicate provides the mechanism to pose a query like

$$\exists x[enrolment-requirement(BScCS, x) \wedge satisfies(John, x)]? \qquad (5)$$

that can be read as "Has John satisfied an enrolment requirement for the BScCS program?"

Notice that, in the DDB domain, degree requirements are most naturally conceived as conditions which must be satisfied. Since degree programs are distinguished by their various requirements, it is most straightforward to describe degree program requirements as relations on degree names and conditions to be satisfied—in DLOG, as lambda terms.

Of course there are alternatives to the use of this special term. For example, the meaning of sentence (4) might be rephrased in. terms of a standard first order language as

$$\forall x[satisfied-requirements(x, BScCS) \supset \exists y[age-of(x,y) \wedge y \geq 16]] \qquad (6)$$

where we would use *BScCS* as the name of a degree program and modify the predicate *satisfies* to correspond more closely to our intuition regarding what one must do with degree requirements. This alternative has a more straightforward meaning since there are no "special" forms. But now there is no way of asking what the requirements of the BScCS program are, short of providing another non-first order primitive for manipulating sentences. For example, to answer the equivalent of query (5) in the alternative notation, we require an operation that retrieves a sentence of the form (6) from the current database, and then returns the consequent of that sentence as an answer

Lambda terms can be manipulated with a standard (sorted) proof procedure to answer existential queries about requirements; they are simply retrieved and bound to existential lambda variables as in normal answer extraction. Furthermore, they can be used in conjunction with the *satisfies* predicate to determine if an individual has satisfied a particular requirement.

The case for higher order semantics

The dear disadvantage of first order semantics is an inability to directly deal with higher order concepts. Though DLOG domains are restricted to be finite and no abstraction is permitted in the DLOG proof mechanism, the specification of certain DLOG expressions in a first order way is contorted and mitigates against the desired semantic simplicity. This is most obvious in the way that lambda terms must be explained in terms of meta relations.

One alternative is to use a second-order intensional logic, as used by Montague to explain such concepts as "obligation," "event," and "task " For example, Montague's formalization of the concept of obligation [Montague74, p. 151ff.] corresponds well with the use of lambda terms in the DDB application of DLOG.

Montague's system provides a natural semantics for DLOG's lambda terms, and is obviously powerful enough to be used to describe the rest of DLOG's descriptive terms (individuals, sets). Only DLOG lambda terms require this treatment, but Montague's system provides a rather more uniform treatment of DLOG's semantics than is possible in weaker systems.

The complete picture of Montague's system requires careful study, but the essence can be explained in a relatively straightforward manner. An essentia) concept is the classification of individuals into categories of two different kinds. Each n place predicate constant has an associated type <s_0, s_u ' ' ' S_{n-1}-> that indicates the kind of object that can appear in each term position: , —1 specifies a standard* individual; s, -0 specifies a proposition; and S_i specifies a S_i,-place predicate.**

For example, a predicate constant *P* of type <—1,1> takes individual constants in its first position and unary predicates in its second. In the Department Data Base domain, the *satisfies* predicate constant has type <—1,1>, e.g., the assertion

*satisfies(/red,*Xx *[completed (x ,cs* 115)])

has an individual constant 'fred' in the first argument position, and a lambda constant in the second argument position. The first denotes an individual object (the person with name 'fred'), and the second denotes a predicate specifying the property of "x complete ing the course CS115."

The meaning of the above assertion is assigned in a way that introduces the second and most important difference of Montague's system. The assignment of truth values to sentences is an inherently two phase process. As Montague explains [Montague74, p. 157], an *interpretation* assigns *intensions* to symbols, and a *model* assigns *extensions.* Extensions include the standard objects well-known from traditional Tarksian semantics, as well as sets of sequences of individuals. Intensions are functions from possible worlds to the universe of individuals. They are introduced in order to distinguish the sense or abstract meaning of a predicate from its denotation in a particular possible world.

The complexity of Montague's complete system can be perplexing, the essence of the system provides a rich specification language for DLOG's complex objects. Some of the complexity dissolves because of the simplicity of DLOG theories: they are finite, and the intended interpretation is over a highly restricted domain. This simplicity constrains the number of possible worlds that can serve as interpretations for DLOG theories (thus, for example, providing a restricted interpretation of "□"). In the DDB example, the intended interpretation together with partial knowledge of each particular student identifies *the* intended possible world for semantic interpretation.

In Montague's second order logic, the meaning of DLOG lambda expressions is given by expressing them as unary predicate constants. For example, the DLOG formula

$$requirement(BScCS, \lambda x\,[completed(x, CS115)]) \tag{10}$$

is written as

$$requirement(BScCS, \hat{x}completed(x, CS115)) \tag{11}$$

In general, the Φ syntax is shorthand for

$$\top Q \wedge u \Box [Q[u] \Rightarrow \Phi$$

Montague uses the symbols 'A' and V for 'V and '3', respectively. He also uses brackets where parentheses arc typical, e.g., P[x] for P(x). In addition Montague employs the symbols HP and Φ', read as "the" and "necessarily," respectively. These latter symbols are used to form names of predicates. DLOG's lambda symbol 'X' plays the same role as Montague's '"' symbol.

Formula (11) is intended to mean "a requirement of the BScCS program is to bear the relation *completed* to the course CS115." The intensional semantics provides a way of admitting different intensions for the *completed* relation, e.g., completing a course might have different meanings in different possible worlds. In the case of DLOG, the particular possible world in which symbols are assigned extensions is *fixed* to be the Departmental Database.

The second order power of Montague's logic provides the expressive ability to assert relations on predicates: it is the property of completing CS115 that bears the *requirement* relation to the program *BScCS,* and not any particular extension of the property.

Again, the application of lambda terms can be explained with the aid of a relation called *satisfies.* However, in Montague's language *satisfies* is a predicate constant of type <—1,1> and is interpreted (in a possible world t in a structure *<I,U,F>)* as a relation *<I,U,<I,U»* where / is the set of possible worlds and *U* the universe of possible individuals. (So *<1 ,U>* is a unary relation, *<I,U,U>* is a binary relation, etc.)

Computing with descriptions by extended unification

The mechanism for manipulating DLOG descriptions is implemented in the Horn clause logic of Prolog. Adopting one of the above approaches to semantics means to adopt the corresponding view of what the DLOG proof procedure is doing. The simplest way to view the DLOG proof procedure is as a Horn clause prover extended with meta relations to handle the non-Horn features of DLOG. However, we speculate that the theoretical foundation of a higher-order proof procedure based on unification due to Jensen and Pietrzykowski [Jensen75] will provide the corresponding view for the Montague system. Here the intuition is to consider the DLOG implementation as a restricted implementation of their unification procedure for general type theory. We have not yet investigated the possibility of adapting Jensen and Pietrzykowski's procedure for use in an intensional logic.

Instead of extending Prolog's Horn clause theorem prover to handle the expressions that arise from any method of contextual definition, the unification algorithm can be augmented to provide the correct matching of descriptive terms. As others have observed (e.g., (Clark78, van Emden84]), any assertion of the form

$$\Phi(t_1, t_2, \cdots t_n)$$

where t_i, $1 \leq i \leq n$ are terms, can be rewritten as

* Here "standard individual" means the usual notion of an individual in a first order model.

** See [Montague74, p. 150]. The notion of predicate used in this context is sometimes called a "relation in intension."

$$\Phi(x_1,x_2,\cdots x_n)\subset x_1=t_1\wedge x_2=t_2\wedge\cdots\wedge x_n=t_n$$

and implications

$$\Phi(t_1,t_2,\cdots,t_n)\subset\Psi_1\wedge\Psi_2\wedge\cdots\wedge\Psi_m$$

can be rewritten as

$$\Phi(x_1,x_2,\cdots,x_n)\subset x_1=t_1\wedge x_2=t_2\wedge\cdots\wedge x_n=t_n$$
$$\wedge\Psi_1\wedge\Psi_2\wedge\cdots\wedge\Psi_m$$

where the x_i $1<i<n$ are new variables not occuring in the original formulas. In DLOG, the equality expressions arising from this transformation are determined from within unification. In a sense, some of the complexity of derivation is off-loaded to the "pattern matcher" (cf. [Reiter75]).

The idea of extending a resolution proof procedure's power by augmenting unification was first suggested by Morris [Morris69], who proposed that equality be manipulated with so-called "E-unification." There have been many other related proposals including Stickel [Stickel75], Morgan [Morgan75], and Kahn [Kahn8l], Of related interest is the representation language KRL [Bobrow77a, Bobrow77b, Bobrow79), which relies on a complex "mapping" process on several different kinds of object descriptions called "descriptors." We argue elsewhere that KRL's mapping can best be understood as a elaborated unification scheme [Goebel85].

Returning to the handling of descriptive terms by augmenting unification, we cite Rosenschein on the advantage of embedded terms:

> ...the data object is kept small and "hierarchical" so that where an exhaustive match must be performed, failure can occur quickly. That is, deep, heterogeneous structures are preferred to broad, homogeneous structures. For example, {(){()()}} is better than {{}{}{}{}}!

We view Rosenschein's claim as support for the interpretation of descriptions as embedded terms, rather than as their contextual definition by rewriting.

The DLOG unification algorithm is invoked by the DLOG *derivable* predicate, similar to the way Prolog's derivation procedure uses a built-in unification algorithm. Intuitively, whenever a unification must be performed and there are special DLOG terms to be matched, standard unification is intercepted, and DLOG unification is used. For example, suppose that the two terms Ex<*P(x)* and *Fred* are to be unified. The applicable DLOG *unify* axiom is

$$unify(\xi x\,\Phi(x),Fred)\leftarrow apply(\lambda x\,\Phi(x),Fred)$$

where apply binds the symbol *"Fred"* to the lambda variable "x" and invokes *derivable*.

The DLOG unification definition uses an organization similar to the LOGLISP system of Robinson and Sibert [Robinson80, Robinson82]. LOGLISP consists of a logical proof theory embedded within LISP, and allows the invocation of LISP by the theorem-prover, and the theorem-prover by LISP. Similarly, the DLOG *derivable* procedure can invoke the standard Prolog proof procedure, and both are accessible from with DLOG's unification matcher.

In general, the correct "unification" of the DLOG extensions requires a derivation procedure more powerful than that provided by Prolog. For example, the equivalence of two lambda expressions, e.g., $\lambda x\,\Phi(x)$ and $\lambda y\,\Psi(y)$ can only be established if it can be shown that $\forall x\,\Phi(x)\equiv\Psi(x)$ follows from the current database. The current DLOG unification procedure uses a local context mechanism to derive this equivalence. It is also the case that disjunctive terms require a more general proof mechanism, since a proof of $\exists x\,\Phi(x)\vee\Psi(y')$) cannot be handled by the current implementation, although a special heuristic will use a notion of partial proof to retrieve facts relevant to such a query [Goebel85|.

t [Rosenschein78, p. 634].

Bobrow and Winograd's description of KRL's matching framework (see [Bobrow77a, §2.5]) also uses the notion of partial match. Their discussion about what is deductive and what is heuristic is sufficiently interesting to pursue here because DLOG already provides some of the features of KRL's "flexible" matching.

Recall that the basic data type of KRL is a frame-like structure called a "unit." A unit is a collection of "descriptors" that attribute various properties to the unit in which they appear. Of interest here are the various ways in which units can be related by matching their descriptors. For example, consider KRL's matching by "using properties of the datum elements" [Bobrow77a, pps. 23-24]:

> Consider matching the pattern descriptor *(which Owns (a Dog))* against a datum which explicitly includes a descriptor *(which Owns Pluto)*. The SELF description in the memory unit for Pluto contains a perspective indicating that he is a dog. In a semantic sense, the match should succeed. It can only do so by further reference to the information about Pluto.

This form of matching already exists in DLOG. For example, the KRL descriptors *(which Owns (a Dog))* and *(which Owns Pluto)* might be rendered as $\exists x[Owns(x,\epsilon y|dog(y)|)$ and $\exists x[Owns(x,Pluto)]$, respectively If we have the fact that Pluto is a dog (i.e., the assertion *dog(Pluto)),* DLOG unification will successfully unify the above pair by recursively proving that *dog(Pluto)* follows from the knowledge base.

Several other forms of KRL matching fall into similar categories, where a recursive proof will provide the inferences required to demonstrate the equality of descriptions. The only clear instance in which partial matches arise are due to resource limitations. Again the partial results determine whether the current line of reasoning is to continue (perhaps given further resources), or to be abandoned.

Concluding remarks

We have argued that there may be more to the meaning of descriptions than their traditional Tarskian semantics, especially as regards the way that they are manipulated within a logic-based representation language. We briefly outlined the kinds of descriptive terms included in the Prolog-based DLOG representation system, and discussed various ways in which those terms could be interpreted. Lambda terms, useful in a particular application, do not have an obvious formal meaning and suggest the need for higher-order semantics. Regardless of which semantic specification is selected, the notion of extended unification can be used to manipulate embedded descriptions. With some effort, the extended procedure can be viewed as providing either metalogical or higher-order proof theory extensions.

Finally, it is important for representation systems to exploit the computational as well as the traditional denotational meaning of descriptions. The proceduralists have been saying this for years; we claim that logic can contribute to an understanding of the computational use of certain kinds of descriptions.

Acknow lodgements

David Poole suggested many improvements to an earlier draft of this paper. Richard Robinson pointed out the relationship between DLOG's lambda terms and Montague's formalization of obligation.

References

[Attardi8l] G. Attardi and M. Simi (1981), Consistency and completeness of Omega, a logic for knowledge representation, *Proceedings of the Seventh International Joint Conference on Artificial Intelligence,* August 24-28, The University of British Columbia, Vancouver, British Columbia, 504-510.

|Bobrow77a] D.G. Bobrow and T. Winograd (1977), An overview of KRL-O, a knowledge representation language, *Cognitive Science* 1(1), 3-46.

[Bobrow77b] D.G. Bobrow and T. Winograd (1977), Experience with KRL-O, one cycle of a knowledge representation language, *Proceedings of the Fifth International Joint Conference on Artificial Intelligence,* August 22-25, MIT, Cambridge, Massachusetts, 213-222.

(Bobrow79j D.G Bobrow and T. Winograd (1979), KRL, another perspective, *Cognitiw Science* 3(1), 29-42.

[Bowen82| K. Bowen and R.A. Kowalski (1982), Amalgamating language and metalanguage in logic programming, *Logic Programming,* A.P.I.C. Studies in Data Processing 16, K.L. Clark and S.-A. Tarnhmd (eds.), Academic Press, New York, 153-172.

[Brinton77] A. Brinton (1977), Uses of definite descriptions and Russell's theory, *Philosophical Studies* 31, 261-267.

(Carroll78) J.M. Carroll (1978), Names and naming: an interdisciplinary view, Research Report RC7370, IBM Watson Research Center, Yorktown Heights, New York, October.

[Clark78] K.L. Clark (1978), Negation as failure, *Logic and Data Bases,* H. Gallaire and J. Minker (eds.), Plenum Press, New York, 293-322.

[Dan 180] V. Dahl (1980), Two solutions for the negation problem, *Proceedings of the Logic Programming Workshop,* July 14-16, Debrecen, Hungary, S.-A. Tarnlund (ed), 61-72.

[Dahl82] V. Dahl (1982), On database systems development through logic, *ACM Transactions on Database Systems* 7(1), 102-123.

[Dilgcr78] W. Dilger and G. Zifonun (1978), The predicate calculus-language KS as a query language, *Logic and Data Bases,* H. Gallaire and J. Minker (eds.), Plenum Press, New York, 377-408.

[Donnellan66] K.S. Donnellan (1966), Reference and definite descriptions, *Pfiilosophical Review* 76(3), 281-304.

[van Emden84] M.H. van Emden and J.W Lloyd (1984), A logical reconstruction of Prolog II, *Proceedings of the Second International Logic Programming Conference,* July 2-6, Uppsala University, Uppsala, Sweden, 115-125.

[Goebel84) R.G. Goebel (1984), DLOG: a logic-based data model for the machine representation of knowledge, *ACM SIC ART Newsletter* 87, 45-46 [reprinted, with corrections, from *ACM SIC ART Newsletter,* 86, 69-71],

[Goebd85] R.G. Goebel (1985), A logic-based data model for the machine representation of knowledge, Ph.D. dissertation, Computer Science Department, The University of British Columbia, Vancouver, British Columbia, (accepted with revisions in February), 247 pages.

[Hayes77] P.J. Hayes (1977), In defence of logic, *Proceeding of the Fifth International Joint Conference on Artificial Intelligence,* August 22-25, MIT, Cambridge, Massachusetts, 559-565.

[Hewitt80] C. Hewitt, G. Attardi, and M. Simi (1980), Knowledge embedding in the description system Omega, *Proceedings of the First American Association of Artificial Intelligence Conference,* August 18-21, Stanford University, Stanford, California, 157-163.

[Jensen75] DC. Jensen and T. Pietrxykowski (1975), Mechanizing to-order type theory through unfication, *Theoretical Computer Science* 3(2), 123-171.

[Kahn8l] K. Kahn (1981), UNIFORM - a language based upon unification which unifies (much of) LISP, PROLOG and ACTI, *Proceedings of the Seventh International Joint Conference on Artificial Intelligence,* August 24-28, Vancouver, British Columbia, 933-939.

|Kaplan75] D. Kaplan (1975), What is Russell's theory of descriptions?, *The Logic of Grammar,* D. Davidson and G. Harman (eds.), Dickenson, Encino, California, 210-217.

[Katz77] J.J. Katx (1977), A proper theory of names, *Philosophical Studies* 31, 1-80.

[Leisenring69] A.C. Leisenring (1969), *Mathematical Logic and Hilbert's E-symbol,* MacDonald Technical & Scientific, London, England.

|Linsky77] L. Linsky (1977), *Names and descriptions,* The University of Chicago Press.

[Montague74] R. Montague (1974), On the nature of certain philosophical entities, *Formal Philosophy,* R.H. Thomason (ed.), Yale University Press, 148-187 [reprinted from *The Monist* 53(1960), 159-194).

[Moore76| R.C. Moore (1976), D-SCR1PT, a computational theory of descriptions, *IEEE Transactions on Computers* C-25(4), 366-373.

[Morgan75] C.G. Morgan (1975), Automated hypothesis generation using extended inductive resolution, *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence,* September 3-8, Tblisi, USSR, 351-356.

[Morris69] J.B. Morris (1969), E-resolution: extension of resolution to include the equality relation, *Proceedings of the Intemationl Joint Conference on Artificial Intelligence,* May 7-9, Washington, D.C., 287-294.

|Norman79] D.A. Norman and D.G. Bobrow (1979), Descriptions: an intermediate stage in memory retrieval, *Cognitive-Psychology* 11(1), 107-123.

[Ortony77] A. Ortony and R.C. Anderson (1977), Definite descriptions and semantic memory, *Cognitive Science* 1(1), 74-83.

[Reiter75] R. Reiter (1975), Formal reasoning and language understanding systems, *Proceedings of the First Conference on 'theoretical Issues in Natural Language Processing,* June 10-13, MIT, Cambridge, Massachusetts, 175-179.

[Robinson79] J.A. Robinson (1979), *Logic: Form and Function,* Artificial Intelligence Series 6, Elsevier North Holland, New York.

[Robinson80] J.A. Robinson and E.E. Sibert (1980), Logic-programming in LISP, Report 8-80, School of Computer and Information Science, Syracuse University, Syracuse, New York, December.

[Robmson82] J.A. Robinson and E.E. Sibert (1982), LOGLISP: an alternative to PROLOG, *Machine Intelligence,* vol. 10, J.E. Hayes, D. Michie, and Y-H Pao (eds), Ellis-Horwood, 399-419.

[Rosenschein78] S.J. Rosenschein (1978), The production system: architecture and abstraction, *Pattern-Directed Inference Systems,* D.A. Waterman and F. Hayes-Roth (eds), Academic Press, New York, 525-538.

[Schubert76] L.K. Schubert (1976), Extending the expressive power of semantic networks, *Artificial Intelligence* 7(2), 163-198

[Steels80] L. Steels (1980), Description types in the XPRT-system, *Proceedings of the AISB-80 Conference on Artificial Intelligence,* July 1-4, Amsterdam, Holland, (STEELS 1-9).

[Stickel75] M.E. Stickel (1975), A complete unification algorithm for associative-commutative functions, *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence,* September 3-8, Tblisi, USSR, 71-76.