Making Computational Sense of Montague's Intensional Logic

Jerry R. Hobbs[1]

Department of Computer Sciences, City College, CUNY

Stanley J. Rosenschein[2]

Courant Institute of Mathematical Sciences, New York University

Recommended by D. E. Walker

ABSTRACT

Montague's difficult notation and complex model theory have tended to obscure potential insights for the computer scientist studying Natural Language. Despite his strict insistence on an abstract model-theoretic interpretation for his formalism, we feel that Montague's work can be related to procedural semantics in a fairly direct way. A simplified version of Montague's formalism is presented, and its key concepts are explicated in terms of computational analogues. Several examples are presented within Montague's formalism but with a view toward developing a procedural interpretation. We provide a natural translation from intensional logic into LISP. *This allows one to express the composition of meaning in much the way Montague does, using subtle patterns of functional application to distribute the meanings of individual words throughout a sentence. The paper discusses some of the insights this research has yielded on knowledge representation and suggests some new ways of looking at intensionality, context, and expectation.*

1. Introduction

With the goal of bridging the gap between linguistics and logic, the logician Richard Montague developed an apparatus for describing the syntax and semantics of English. Using a categorial grammar and the language of intensional logic, he gave a mathematically precise account of a small but semantically interesting fragment of English. A worker in natural language processing is likely to find his first encounter with Montague's work a rather unsatisfactory experience. He finds that English sentences are supposed to acquire meaning by being mapped into a universe of possible worlds, of infinite sets and functionals of functions on these sets. He finds, for example, the word "be" defined as a functional mapping a

[1] Jerry R. Hobbs, presently at SRI International, Menlo Park, California.

[2] Stanley J. Rosenschein, presently at Technion—Israel Institute of Technology, Technion City, Haifa, Israel.

function from possible worlds and points in time into entities and a function from possible worlds and points in time into functionals from functions from possible worlds and points in time into functionals from functions from possible worlds and points in time into entities into truth values into truth values into truth values.[3] It is difficult for him to see how such a representation can help with any of the problems he faces, either linguistic problems, such as devising representations for context and expectation and algorithms for finding antecedents of pronouns and resolving ambiguities, or task problems, such as question–answering and converting natural language input into the directed behavior of some device. In short, he questions its relevance to someone whose work and theory must be grounded in the need to produce working computer programs.

In this paper, we suggest that despite Montague's difficult notation and the complex model-theoretic interpretation for his formalism, there are many potential insights in Montague for the computational linguist. This is not to imply that natural language processing systems should be based on Montague's formalism. For one thing, Montague is primarily concerned with the assignment of truth values to simple declarative sentences, which is only one of the many activities language users ordinarily perform. Nevertheless, Montague's method, involving subtle patterns of functional application, suggests an interesting way of distributing meanings of individual words throughout a sentence. We show how this method can be used computationally by relating Montague's work to procedural semantics in a fairly direct way. In addition, we indicate how the method might be extended to handle certain aspects of context.

Much work has been done on Montague grammar and on relating intensional logic to the semantics of English. We will not review this literature here. Rather, we will take Montague [9] as the key paper and representative of the approach. In this paper he treats a subset of English which includes simple quantification and some intensional verbs. Many linguists are currently working on extending this subset. Additional material can be found in [10], [12].

Montague's method for assigning meanings to English sentences involves three distinct phases. An English string is assigned a syntactic analysis with respect to a categorial grammar. This is translated into an expression in the language of intensional logic. Finally, this expression undergoes model-theoretic interpretation.

Montague's language of intensional logic is a typed lambda calculus. In this language he constructs a generally complex formula representing a pattern of composition and application of functions, ultimately derived from the basic terms of the language, such terms as **man**, **seek**, **run**, etc. The meanings of surface words are taken to be very abstract functions, which take as arguments other such functions. Things "work out", so that the function assigned as the meaning of a declarative sentence evaluates at a possible world and point in time to true or false. Part of the attraction of Montague's treatment lies in the way he manages to mesh

[3] This sentence parses unambiguously.

a complex system of meaning assignments in a mathematically precise way so that the meanings do work out, with the exact details illuminating some classic problems of semantics, including intensional predicates and referential and nonreferential terms.

The goal of any semantic theory is to express English strings in terms of an antecedently understood metalanguage [12]. The metalanguage of set theory has been a favorite choice this century. Meanings for Montague are ultimately abstract set-theoretic constructs, in the tradition of Tarskian model-theoretic semantics [14]. But while these constructs may be antecedently understood by humans, they certainly are not antecedently understood by computers, and Montague makes no claim for their being computable in any sense.

To make sense to the computational linguist, something must be reduced to implementable procedures. The meaning of an expression is then the behavior of the procedure it is transformed into. Thus, while he often uses formalisms that look very much like those of the logician, the computational linguist is after a quite different type of semantic theory, one which is ultimately machine-theoretic rather than model-theoretic in its orientation (see also Davies and Isard [1], Isard [4] and Joshi and Weischedel [6]).

What guidelines, then, can Montague, the model-theorist, give the computational linguist in the task of working out the details of a procedural semantics for natural language? While it is true that Montague's semantic constructs generally involve infinite sets and functions on them, failing computability on most counts, Montague has made a significant contribution to the computational semanticist by showing possible formats for the representation of meanings of individual words and mechanisms for the combination of meanings which are considerably more elegant than most computational alternatives now in use. By replacing the bottom layer of Montague's model-theoretic edifice with an appropriate set of procedures, we hope to preserve computability while still maintaining the basic framework of Montague grammar. We hope to convey those aspects of Montague grammar which should be of interest to the artificial intelligence researcher working on knowledge representation, and to the computational linguist in particular.

In Section 2 we give an outline of the main features of Montague's formalism. In addition we suggest ways in which a computer scientist might think of its key concepts. Section 3 gives several Montague-style examples together with simple procedural, or machine-theoretic, interpretations. Section 4 describes the very little that needs to be done to Montague's expressions in intensional logic in order that they be directly interpretable as expressions in an existing programming language, LISP, augmented by a small suitable set of primitive functions. This result then demonstrates that the rather extensive research in Montague grammar is quite compatible with research in procedural semantics (e.g. [16], [15]). Furthermore, Montague's very fruitful approach to the problem of structuring functional—and by extension, procedural—knowledge indicates how the method of procedural semantics can be sharpened in just that respect which has caused it most to come under attack: the ad hoc character of its definitions. In Section 5 we speculate on what light this approach may throw onto the nature of context and expectation.

2. Montague's Formalism

This exposition will follow that of Montague [9] and use the more standard features of his notation, but will give only those rules necessary for the examples in this paper. A few rules are added to handle one example.

The categorial grammar used for syntactic analysis of English strings consists of categories into which English words and phrases may fall and rules for combining words and phrases of various categories into larger phrases and sentences. The categories, with examples of words (or basic expressions) that belong to them, are as follows:

Truth values:	t	(no basic expressions)
Entities:	e	(no basic expressions)
Intransitive verbs:	IV:	rise, trot
Terms:	T:	ninety, he_0
Transitive verbs:	TV:	seek
Common noun phrases:	CN:	man, frog, horse, temperature
Adverbials:	IAV:	rapidly
Attributive adjectives:	Adj:	slow.[4]

Phrases of the various categories may be built up out of basic expressions by the following rules (morphological fineries are ignored):

(1) (Article+Common Noun Phrase). If ζ is in category CN, then $F_0(\zeta)$, $F_1(\zeta)$, $F_2(\zeta)$ are in category T, where $F_0(\zeta) = \textit{every } \zeta$, $F_1(\zeta) = \textit{the } \zeta$, $F_2(\zeta) = \textit{a } \zeta$.

(2) (Subject+Verb Phrase). If α is in category T and δ is in category IV, then $F_4(\alpha, \delta)$ is in category t, where $F_4(\alpha, \delta) = \alpha\delta$.

(3) (Transitive Verb+Object). If δ is of category TV and β of category T, then $F_5(\delta, \beta)$ is of category IV, where $F_5(\delta, \beta) = \delta\beta$.

(4) (Verb Phrase+Adverbial). If δ is of category IAV and β of category IV, then $F_7(\delta, \beta)$ is of category IV, where $F_7(\delta, \beta) = \beta\delta$. (Note: this is the only syntactic rule which reverses the order of the elements.)

(5) (Attributive Adjective+Common Noun). If δ is of category Adj and β of category CN, then $F_7'(\delta, \beta)$ is of category CN, where $F_7'(\delta, \beta) = \delta\beta$.[5]

[4] This category is not in Montague [9].

[5] This rule is not in Montague [9].

(6) (Conjunction). If ϕ, ψ are of category t, then so is $F_8(\phi, \psi)$ where $F_8(\phi, \psi) = \phi$ *and* ψ.

(7) (Quantification). If α is of category T and not he_n; and ϕ is of category t or IV and contains he_n, then $F_{10,n}(\alpha, \phi)$ is of the same category as ϕ, where $F_{10,n}(\alpha, \phi) = \phi'$, where ϕ' is like ϕ except that the first occurrence of he_n has been replaced by α.

Rules (1)–(3), (5) and (6) are just phrase structure rules, while rules (4) and (7) can be implemented by simple transformations.

This completes our discussion of the syntactic rules. We now present the language of intensional logic.

The basic types of Montague's intensional logic are as follows:

t = truth values;
e = entities;
s = possible world-point in time pairs.

A possible world-point in time pair will be called a point of reference.

Higher types are built up as follows: if a and b are types, then $\langle a, b\rangle$ is the type consisting of all functions from a to b. Expressions in intensional logic may be built up from constants and variables of each type and from other expressions by means of logical connectives, quantification, temporal and modal operators, functional application, and lambda abstraction. For example, if u is a variable and α and β are expressions of the appropriate type, then

$$\alpha \vee \beta, \quad (\forall u)\alpha, \quad \Box\,\alpha, \quad \lambda u\alpha, \quad \alpha(\beta)$$

are also expressions. In addition, if α is an expression of type a, then $^\wedge\alpha$ (called the *intension* of α) is an expression of type $\langle s, a\rangle$. If α is an expression of type $\langle s, a\rangle$ then $^\vee\alpha$ (called the *extension* of α) is an expression of type a. The extension operator $^\vee$ applies a function whose domain is points of reference to the current point of reference. The intension operator $^\wedge$ applied to an expression creates a function whose domain is points of reference and whose value at each point of reference is the expression. (To reduce parenthesizing, we assume $^\wedge$ and $^\vee$ apply to the smallest meaningful expression to their immediate right.)

The types do not occur arbitrarily in the analysis of English. Certain types turn out to be the most useful, and for these key types it is worthwhile developing our intuitions by describing computational analogues. For this purpose, let us assume that a point of reference corresponds to a possible state of the machine at a particular moment in time. Then the extension of an expression α, $^\vee\alpha$, may be viewed as the evaluation of that expression with respect to the current state of the machine. The intension of α, $^\wedge\alpha$, on the other hand, represents an object which when evaluated with respect to any state of the machine will return the value of α in the current state. In Section 4 these notions will be refined, and some necessary elaboration will be presented.

The type e may be viewed as the set of *constants* of the "data type" available in a computer program, e.g. numbers. Type $\langle s, e\rangle$ is the set of functions from points of reference to entities. When evaluated, they give an object of type e, a constant. Thus, as a first approximation, we may view an object of type $\langle s, e\rangle$ as a *simple variable*. It associates a constant with any current state of the machine. In particular, the parameter of a procedure which evaluates to a constant is of type $\langle s, e\rangle$. This initial intuition is useful, but it will have to be modified somewhat in Section 4 below. In addition the first example of Section 3 views objects of type $\langle s, e\rangle$ in a slightly different light.

An object of type $\langle\langle s, e\rangle, t\rangle$ maps a variable into a truth value and thus may be thought of as a call-by-name *procedure* of one argument which returns a truth value. (This will hereafter be called simply " a procedure ".) An object of type $\langle s, \langle\langle s, e\rangle, t\rangle\rangle$, for any current state of the machine evaluates to a procedure, and thus may be thought of as a *procedure name*. Such a name may be attached to the same procedure throughout the operation of a program, or it may change. An object of type $\langle\langle s, \langle\langle s, e\rangle, t\rangle\rangle, t\rangle$ maps procedure names into truth values and may be thought of as a call-by-name *functional*. Objects of type $\langle s, \langle\langle s, \langle\langle s, e\rangle, t\rangle\rangle, t\rangle\rangle$ are variables ranging over functionals, hence *functional names*.

Predicate modifiers like the English word "rapidly" are realized as higher functionals of type $\langle\langle s, \langle\langle s, e\rangle, t\rangle\rangle, \langle\langle s, e\rangle, t\rangle\rangle$, which map procedure names (e.g. the procedural interpretation of "walks") into procedures (e.g. the procedure for "walks rapidly"). Transitive verbs such as "seek" and "be" are realized as objects of type $\langle\langle s, \langle\langle s, \langle\langle s, e\rangle, t\rangle\rangle, t\rangle\rangle, \langle\langle s, e\rangle, t\rangle\rangle$. This is a higher functional taking two arguments—the first a functional name representing the direct object of the verb, the second a simple variable representing the subject of the verb.

In what follows, x, y, z will be used as variables ranging over objects of type $\langle s, e\rangle$, simple variables; P, Q, R over objects of type $\langle s, \langle\langle s, e\rangle, t\rangle\rangle$, procedure names; and $\mathcal{P}$ over functional names of type $\langle s, \langle\langle s, \langle\langle s, e\rangle, t\rangle\rangle, t\rangle\rangle$.

The rules for translation from syntactic representations to expressions in intensional logic are as follows:

(1) The English words "man", "frog", "horse", "temperature", "rise", and "trot" are mapped into MAN, FROG, HORSE, TEMPERATURE, RISE, and TROT, respectively, where these are objects of type $\langle\langle s, e\rangle, t\rangle$. For Montague they are functions, i.e. sets of ordered pairs; we may view them as the procedures in a computer program which recognize or define the properties of "man", "frog", etc. "Slow" and "rapidly" map into SLOW and RAPID, respectively, which are of type $\langle\langle s, \langle\langle s, e\rangle, t\rangle\rangle, \langle\langle s, e\rangle, t\rangle\rangle$. "Seek" maps into SEEK, of type

$$\langle\langle s, \langle\langle s, \langle\langle s, e\rangle, t\rangle\rangle, t\rangle\rangle, \langle\langle s, e\rangle, t\rangle\rangle.$$

(2) "Be" maps into the expression $\lambda\mathcal{P}\lambda x[{}^{\vee}\mathcal{P}({}^{\wedge}\lambda y[{}^{\vee}x = {}^{\vee}y])]$.

(3) "Ninety" is mapped into $\lambda P[{}^{\vee}P({}^{\wedge}n)]$ where n is an entity of type e.

(4) "he$_i$" is mapped into $\lambda P[{}^{\vee}P(x_i)]$ where x_i is the ith variable of type $\langle s, e\rangle$.

In the remaining rules, α' signifies the translation of α under the rules.

(5) Every $\zeta: F_0(\zeta)$ is mapped into $\lambda P[(\forall x)\,(\zeta'(x) \supset {}^{\vee}P(x))]$,
the $\zeta: F_1(\zeta)$ is mapped into $\lambda P[(\exists y)\,((\forall x)\,(\zeta'(x) \leftrightarrow x = y)\ \&\ {}^{\vee}P(y))]$,
a $\zeta: F_2(\zeta)$ is mapped into

$$\lambda P[(\exists x)\,(\zeta'(x)\ \&\ {}^{\vee}P(x))].$$

(6) $F_4(\delta, \beta)$ is mapped into $\delta'({}^{\wedge}\beta')$,
$F_5(\delta, \beta)$ is mapped into $\delta'({}^{\wedge}\beta')$,
$F_7(\delta, \beta)$ is mapped into $\delta'({}^{\wedge}\beta')$,
$F_7'(\delta, \beta)$ is mapped into $\delta'({}^{\wedge}\beta')$.

(7) $F_8(\alpha, \beta)$ is mapped into $\alpha'\ \&\ \beta'$.

(8) $F_{10,\,n}(\alpha, \phi)$ is mapped into $\alpha'({}^{\wedge}\lambda x_n \phi')$ if α is of category T and ϕ of category t.
$F_{10,\,n}(\alpha, \delta)$ is mapped into $\lambda y[\alpha'({}^{\wedge}\lambda x_n[\delta'(y)])]$ if α is of category T and δ of category IV.

Thus, most concatenations of words in English are translated into functional applications in intensional logic.

Montague presents a standard model-theoretic interpretation for the expressions of intensional logic. We will not outline the details, for our interpretations will be quite different. Objects in intensional logic will be interpreted as constants, procedures, and functionals in a computer program.

3. Examples

3.1. Consider the sentence

The temperature is ninety and the temperature rises. (1)

This sentence has been of interest [9] because if "be" is viewed as equality and therefore as a symmetric and transitive relation,

Ninety rises

follows. The syntactic representation is

$$F_8(F_4(F_1(\text{temperature}), F_5(\text{be, ninety})), F_4(F_1(\text{temperature}), \text{rise})). \quad (2)$$

The translation rules map this into the expression

$$\begin{aligned}&\lambda P[(\exists y_1)\,((\forall x_1)\,(\textsc{temperature}(x_1) \leftrightarrow x_1 = y_1)\ \&\ {}^{\vee}P(y_1))]\\ &\quad({}^{\wedge}\lambda\mathscr{P}\lambda x_2[{}^{\vee}\mathscr{P}({}^{\wedge}\lambda y_2[{}^{\vee}x_2 = {}^{\vee}y_2])]\,({}^{\wedge}\lambda Q[{}^{\vee}Q({}^{\wedge}n)]))\\ &\quad\&\lambda R[(\exists y_3)\,((\forall x_3)\,(\textsc{temperature}(x_3) \leftrightarrow x_3 = y_3)\ \&\ {}^{\vee}R(y_3))]\,({}^{\wedge}\textsc{rise}). \quad (3)\end{aligned}$$

This expression can be simplified by symbolically applying functions to their arguments in the order indicated by (2) and using the equivalence ${}^{\vee}{}^{\wedge}\alpha \equiv \alpha$. In the first conjunct, replacing $\mathscr{P}$ by its value yields

$$\begin{aligned}&\lambda P[(\exists y_1)\,((\forall x_1)\,(\textsc{temperature}(x_1) \leftrightarrow x_1 = y_1)\ \&\ {}^{\vee}P(y_1))]\\ &\quad({}^{\wedge}\lambda x_2[\lambda Q[{}^{\vee}Q({}^{\wedge}n)]\,({}^{\wedge}\lambda y_2[{}^{\vee}x_2 = {}^{\vee}y_2])]).\end{aligned}$$

Replacing Q by its value gives

$$\lambda P[(\exists y_1)((\forall x_1)(\text{TEMPERATURE}(x_1) \leftrightarrow x_1 = y_1) \,\&\, {}^{\vee}P(y_1))] \\ ({}^{\wedge}\lambda x_2[\lambda y_2[{}^{\vee}x_2 = {}^{\vee}y_2]({}^{\wedge}n)]).$$

Replacing y_2 by its value yields

$$\lambda P[(\exists y_1)((\forall x_1)(\text{TEMPERATURE}(x_1) \leftrightarrow x_1 = y_1) \,\&\, {}^{\vee}P(y_1))] \\ ({}^{\wedge}\lambda x_2[{}^{\vee}x_2 = n]).$$

Replacing P by its value yields

$$(\exists y_1)((\forall x_1)(\text{TEMPERATURE}(x_1) \leftrightarrow x_1 = y_1) \,\&\, \lambda x_2[{}^{\vee}x_2 = n](y_1)).$$

Replacing x_2 by its value results in

$$(\exists y_1)((\forall x_1)(\text{TEMPERATURE}(x_1) \leftrightarrow x_1 = y_1) \,\&\, {}^{\vee}y_1 = n). \tag{4}$$

(5) results from function application in the second conjunct:

$$(\exists y_3)((\forall x_3)(\text{TEMPERATURE}(x_3) \leftrightarrow x_3 = y_3) \,\&\, \text{RISE}(y_3)). \tag{5}$$

The conjunction of (4) and (5) reduces (because of the uniqueness of y) to

$$(\exists y)((\forall x)(\text{TEMPERATURE}(x) \leftrightarrow x = y) \,\&\, {}^{\vee}y = n \,\&\, \text{RISE}(y)). \tag{6}$$

For our interpretation of (6) we will imagine a system in which the temperature is measured and recorded on a graph whose horizontal axis is time. The set of possible worlds is the set of all possible graphs. Here it is most convenient to think of y not as a one-argument function from points of reference to numbers but as a two-argument function from possible worlds and points in time into numbers. Particularized to one possible world, it is then a function from points in time into numbers. The part of (6),

$$(\exists y)((\forall x)(\text{TEMPERATURE}(x) \leftrightarrow x = y) \cdots),$$

simply accesses the unique temperature checking function. The expression

$${}^{\vee}y = n$$

evaluates the function at the current time and returns TRUE if and only if the value is 90. The predicate RISE computes the left derivative of the function y at the current time; it returns TRUE if that value is positive, FALSE otherwise.

In a sense, this example runs counter to the intuition developed in the previous section about the nature of objects of type $\langle s, e\rangle$, such as y, as simple variables, for here it is used as a function from times into numbers. However, a simple variable itself may be viewed as a function from points in time into the set of values it takes on at those given times. The difference is that in a computer program, one is not able to access previous values of a variable once the value has been changed, as we would have to access previous values of y in this example to compute its left derivative.

3.2. Consider the sentence

Every man seeks a frog.

By usual accounts it is three-ways ambiguous—there are the intensional reading in which every man is seeking something which satisfies his own image of "frog" (reading 1), and the two extensional readings in which each man is seeking his own particular real frog (reading 2) and all men are looking for the same real frog (reading 3).

Montague gives the following syntactic representations:

$$F_4(F_0(\text{man}), F_5(\text{seek}, F_2(\text{frog}))) \quad \text{(reading 1)} \tag{7}$$

$$F_4(F_0(\text{man}), F_{10,0}(F_2(\text{frog}), F_5(\text{seek}, \text{he}_0))) \quad \text{(reading 2)} \tag{8}$$

$$F_{10,0}(F_2(\text{frog}), F_4(F_0(\text{man}), F_5(\text{seek}, \text{he}_0))) \quad \text{(reading 3)} \tag{9}$$

(7) translates into

$$\lambda P[(\forall x_1)(\text{MAN}(x_1) \supset {}^{\vee}P(x_1))]({}^{\wedge}\text{SEEK}({}^{\wedge}\lambda Q[(\exists y_1)(\text{FROG}(y_1) \mathbin{\&} {}^{\vee}Q(y_1))]))$$

which simplifies to

$$(\forall x_1)(\text{MAN}(x_1) \supset \text{SEEK}({}^{\wedge}\lambda Q[(\exists y_1)(\text{FROG}(y_1) \mathbin{\&} {}^{\vee}Q(y_1))])(x_1)). \tag{10}$$

Thus the existence of the frog y_1 is within the scope of SEEK. The function $\lambda Q[(\exists y_1)(\text{FROG}(y_1) \mathbin{\&} {}^{\vee}Q(y_1))]$ will be applied to its argument within the function SEEK. y_1 stands for the object and Q will ultimately be replaced by the function which expresses the core of the meaning of "seek", in the same way as in the previous example containing the transitive verb "be", Q was replaced by $\lambda y_2[{}^{\vee}x_2 = {}^{\vee}y_2]$.

(8) translates into

$$\lambda P[(\forall x_1)(\text{MAN}(x_1) \supset {}^{\vee}P(x_1))]({}^{\wedge}\lambda x_2[\lambda Q[(\exists y_1)(\text{FROG}(y_1) \mathbin{\&} {}^{\vee}Q(y_1))]$$
$$({}^{\wedge}\lambda x_0[\text{SEEK}({}^{\wedge}\lambda R[{}^{\vee}R(x_0)])(x_2)])]),$$

which reduces to

$$(\forall x_1)(\text{MAN}(x_1) \supset (\exists y_1)(\text{FROG}(y_1)$$
$$\mathbin{\&} \lambda x_0[\text{SEEK}({}^{\wedge}\lambda R[{}^{\vee}R(x_0)])(x_1)](y_1))). \tag{11}$$

(9) translates into

$$\lambda Q[(\exists y_1)(\text{FROG}(y_1) \mathbin{\&} {}^{\vee}Q(y_1))]({}^{\wedge}\lambda x_0[\lambda P[(\forall x_1)(\text{MAN}(x_1) \supset {}^{\vee}P(x_1))]$$
$$({}^{\wedge}\text{SEEK}({}^{\wedge}\lambda R[{}^{\vee}R(x_0)]))])$$

which simplifies to

$$(\exists y_1)(\text{FROG}(y_1) \mathbin{\&} (\forall x_1)(\text{MAN}(x_1) \supset \text{SEEK}({}^{\wedge}\lambda R[{}^{\vee}R(y_1)])(x_1))). \tag{12}$$

In Montague's treatment, sentences like "John is a man" are also syntactically three-ways ambiguous, the three readings paralleling (7), (8), and (9). This is because in the syntactic analysis, common nouns (category CN), verb phrases (IV), and sentences (t) can all be quantified into. But semantically they all collapse to the same expression in intensional logic. "Be" is defined in such a way as to allow the existential quantifier to pass beyond its scope. Also "John" introduces no universal quantifier to block the existential's passage to the outside. The sentence "Every man is a king" has two readings in the semantics of Montague grammar—one in which every man is a different king and one in which every man is the same king.

No difference shows up in Montague's exposition between intensional verbs like "seek" and nonintensional verbs like "see". It is the responsibility of the one who defines these verbs to construct them in such a way that "see" allows the existential to pass out of its scope and "seek" does not. Montague has given no guidance in the latter task. We will offer a suggestion as to how this might be done.

The verb "seek" could be lexically decomposed into the conjunction of two components: a mental component which states among other things that if A seeks a frog then A wants to have a frog; and an operational component which states that if A seeks a frog then if A is near a frog, he takes the frog.[6] We will confine ourselves to the operational component and show that it can be used to exhibit the distinction between the three readings of the verb "seek" by transforming it into scope distinctions of logical operators like the conditional and negation.

This definition of "seek" can be captured within Montague's framework by adding to the translation rules, paralleling rule 2 which defines "be", the rule

(2′) "seek" maps into the expression

$$\lambda\mathscr{P}\lambda x_2[\sim{}^{\vee}\mathscr{P}({}^{\wedge}\lambda y_2[\textsc{near}(x_2, y_2)\ \&\ \sim\textsc{take}(x_2, y_2)])]. \tag{13}$$

The expression for the object of "seek", which in the intensional reading contains an existential quantifier, replaces $\mathscr{P}$. The negation will then be outside and the propositions $\textsc{near}(x_2, y_2)$ and $\sim\textsc{take}(x_2, y_2)$ within the scope of the existential quantifier. The negation sign to the left of $\mathscr{P}$ in (13) prevents the passage of the existential quantifier to the left. It is one of the beauties of Montague's approach that the meaning of a word can be distributed in this fashion. (10) becomes

$$(\forall x_1)\,(\textsc{man}(x_1) \supset \lambda\mathscr{P}[\lambda x_2[\sim{}^{\vee}\mathscr{P}({}^{\wedge}\lambda y_2[\textsc{near}(x_2, y_2)\ \&\sim\textsc{take}(x_2, y_2)])]]$$
$$({}^{\wedge}\lambda Q[(\exists y_1)\,(\textsc{frog}(y_1)\ \&\ {}^{\vee}Q(y_1))])\,(x_1))$$

or

$$(\forall x_1)\,(\textsc{man}(x_1) \supset (\forall y_1)\,(\textsc{frog}(y_1) \supset (\textsc{near}(x_1, y_1) \supset \textsc{take}(x_1, y_1)))). \tag{14}$$

Applying (13) to (11) yields

$$(\forall x_1)\,(\textsc{man}(x_1) \supset (\exists y_1)\,(\textsc{frog}(y_1)\ \&\ (\textsc{near}(x_1, y_1) \supset \textsc{take}(x_1, y_1)))).$$

Applying (13) to (12) yields

$$(\exists y_1)\,(\textsc{frog}(y_1)\ \&\ (\forall x_1)\,(\textsc{man}(x_1) \supset (\textsc{near}(x_1, y_1) \supset \textsc{take}(x_1, y_1)))).$$

The three readings are then distinguished by three different quantifier structures.

[6] Dowty [2] has proposed similar lexical decompositions and in fact Montague has used a lexical decomposition of sorts by including a meaning postulate reducing "be" to an expression involving equality.

For our model we can now imagine a data base which contains a number of entities and a number of properties associated with these entities. In particular, it records the species of each entity and for each relevant moment in time, the locations of the entities and the facts about possession of one entity by another. Typical items in the data base might be

(MAN X1)
(FROG X2)
(AT X1 (54 40) 1846)
(AT Y1 (55 39) 1846)
(HAVE X1 Y1 1847).

A possible world for this example is a possible set of such entities and properties. The most naive interpretation of the existential quantifier is a procedure which searches through the entities until it finds one with the required properties. The corresponding interpretation of the universal quantifier is a procedure which searches through all the entities to verify that all have the required properties. NEAR is defined in terms of distance. TAKE checks for a change from nonpossession to possession.

Although definition (13) distinguishes between the several readings, it has the disadvantage that in our model we cannot determine the truth or falsity of "Every man seeks a frog" except after the fact, and then (unreliably) only if the seeking was successful. For example, if several men took distinct frogs after being near others, it must be reading 2, in which each man is looking for his own particular real frog. However, if only one man came near a frog and he took it, any of the three readings may apply. In addition, each man may have his own, possibly erroneous, image of a frog, and if there were no such thing as frogs, the expression (14) would always be vacuously true. We cannot hope to resolve these difficulties in general without modeling mental states.

Definition (13) could profit from the nicety of a time condition stating that the nearness was true just before the taking occurred. But these changes would greatly complicate the exposition at the expense of clarity.

3.3. Let us now consider the sentence

"A slow horse trots rapidly,"

with the syntactic structure

$$F_4(F_1(F_7'(\text{slow, horse})), F_7(\text{rapidly, trot})).$$

This translates into

$$\lambda P[(\exists x)\ (\text{SLOW}({}^\wedge\text{HORSE})\ (x)\ \&\ {}^\vee P(x))]\ ({}^\wedge\text{RAPID}({}^\wedge\text{TROT})),$$

which simplifies to

$$(\exists x)\ (\text{SLOW}({}^\wedge\text{HORSE})\ (x)\ \&\ \text{RAPID}(\cdot\ \text{TROT})\ (x)).$$

In a procedural interpretation, SLOW and RAPID must be defined as higher functionals which in a sense modify the definitions of the functions HORSE and TROT. There are several ways one can imagine this happening. The method we present, while unorthodox for pure lambda calculus in that it involves capturing free variables, is commonplace in programming languages, such as LISP, which are based on lambda calculus. In effect, this section anticipates the treatment given to such variables in LISP examples to be presented below.

Suppose we are given an entity called a "scale" which, for simplicity, we can think of as an ordered pair ⟨lo-point, hi-point⟩. Assume in addition that we are given two function names, LO and HI, which are initially bound to the functions which map a scale into its lo-point and hi-point respectively. We may then visualize the outlines of a HORSE function as

$$\begin{aligned}\text{HORSE} = \lambda x[\lambda(\cdots \text{gallopspeed} \cdots)\\ [\cdots (\text{speed}(x) > \text{LO}(\text{gallopspeed}))\\ \& (\text{speed}(x) < \text{HI}(\text{gallopspeed}))\\ \cdots](\cdots < 20, 35 > \cdots)].\end{aligned}$$

Gallopspeed may be taken to be the default speed scale for HORSE. A lambda application within the definition of HORSE binds gallopspeed to a particular scale to which the functions LO and HI are applied. The verb TROT is handled similarly:

$$\begin{aligned}\text{TROT} = \lambda x[\lambda(\cdots \text{speedscale} \cdots)\\ [\cdots (\text{speed}(x) > \text{LO}(\text{speedscale}))\\ \& (\text{speed}(x) < \text{HI}(\text{speedscale}))\\ \cdots](\cdots < 14, 26 > \cdots)].\end{aligned}$$

Now we can examine the roles of SLOW and RAPID as mappings from intensions of objects like HORSE and TROT to objects representing a slow horse and a rapid trotting respectively. SLOW can be defined as follows:

$$\begin{aligned}\text{SLOW} = \lambda P[\lambda\, \text{HI}[\lambda x[{}^{\vee}P(x)]]\\ (\lambda\, \text{scale}\, [\text{LO}(\text{scale}) + (\text{HI}(\text{scale}) - \text{LO}(\text{scale}))/3])].\end{aligned}$$

Similarly:

$$\begin{aligned}\text{RAPID} = \lambda P[\lambda\, \text{LO}[\lambda x[{}^{\vee}P(x)]]\\ (\lambda\, \text{scale}\, [\text{HI}(\text{scale}) - (\text{HI}(\text{scale}) - \text{LO}(\text{scale}))/3])].\end{aligned}$$

That is, SLOW redefines HI to return a lower upper limit on a speed scale, and RAPID redefines LO to return a higher lower limit. Now the meaning of "The slow horse trots rapidly" can be seen to reduce to

$$\begin{aligned}(\exists x)[\cdots (\text{speed}_1(x) > 20) \& (\text{speed}_1(x) < 25)\\ \cdots (\text{speed}_2(x) > 22) \& (\text{speed}_2(x) < 26) \cdots]. \qquad (15)\end{aligned}$$

The subscripted function names, speed_1 and speed_2, had their origin within the scope of HORSE and TROT respectively and hence may refer to the same or different speed functions. It is seen from the final reduction that although SLOW and RAPID have opposite effects, the local nature of the scopes of HI and LO allow the correct meaning composition to be obtained.

4. Correspondences with LISP

4.1.

The fact that Montague chose a lambda calculus for the language of his intensional logic immediately suggests the programming language LISP as the computational analogue. In this section we show how Montague's intensional logic expressions can be translated almost directly into LISP expressions which can be evaluated, or executed, in some environment to yield a result. Our analogue of intension will be the procedures. Points of reference will be incorporated within the environment in which the procedures are executed, and the results will correspond to extensions. Some difficulties naturally arise in precisely those places where an infinite computation seems to be implied by Montague's formalism, as in the interpretation of the universal quantifier over all possible worlds, a clearly infinite set in most models. Our approach has been to replace infinite constructions, usually "sets", by finite ones, such as "procedures", without destroying the overall framework of functional composition and application as the basic method for building up the meaning of a sentence.

Before proceeding, we would like to stress the distinction between intension and description. "Description" refers to a linguistic object, while "intension" refers to a function. Different descriptions may have the same intension. Likewise we distinguish between a LISP function, which is only applied, and its various symbolic representations as *s*-expressions. There has been confusion on this point in the natural language processing literature.

In the next few paragraphs we present a brief discussion of the relevant features of LISP. Those who desire a fuller treatment may consult McCarthy et al. [7].

Following McCarthy et al. [7] we view the LISP interpreter as consisting of two mutually recursive meta-functions: *apply* and *eval*. The function *apply* [*f*; *x*; *a*] returns the result of applying function *f* to arguments *x* in environment *a*; *eval* [*e*; *a*] evaluates expression *e* in environment *a*. The notion of an environment was originally realized concretely as the *a*-list, which pairs variables with their values. The substitution semantics of the lambda calculus are captured in LISP not by direct substitution into evaluated expressions but rather by the creation of a new environment which differs from that specified by *a* in precisely those bindings which define the substitution.

This method of "deferred" substitution gives rise to anomalies in the case of functional arguments containing free variables. If the same variables are rebound within the function calling the functional argument, the initial binding of the free variable may be overridden. These anomalies are corrected by allowing for closures, i.e. functions with frozen environments, to be created by *eval* and applied by *apply*.

This is done classically through the use of the operator FUNCTION which creates a closure or FUNARG [7], [11]. Furthermore, it is convenient to assume that the interpreter is such that *eval* [(LAMBDA · · ·); *a*] is equivalent to *eval* [(FUNCTION (LAMBDA · · ·)); *a*]; that is, a LAMBDA expression evaluates to its closure (see also [13]).

The simplest way to exhibit Montague's formalism in LISP is to identify a point of reference with a binding environment, or *a*-list, with respect to which an expression is evaluated. Then we let *eval* [*e*; *a*] correspond to the model-theoretic interpretation of an expression *e* with respect to a point of reference. In this view, the expression $^\wedge\alpha$ corresponds to (LIST (QUOTE QUOTE) α). It gives an object to which can be applied the operator $^\vee$ (corresponding to EVAL, the object language invocation of the meta-function *eval*) which in turn yields an object of the same type as α.

The first few translation rules are:

(1) α, a constant	$\rightarrow$(QUOTE α),
(2) α, a variable of type $\langle s, b\rangle$ for any b	$\rightarrow$(QUOTE α),
(3) $^\wedge\alpha$	$\rightarrow$(LIST(QUOTE QUOTE)α),
(4) $^\vee\alpha$	$\rightarrow$(EVAL α).

In rule (2) the variable must be quoted if the calling function is to be given the option of evaluating or not evaluating the variable. As a first approximation it is useful to look at intension and extension as "QUOTE the value" and "EVAL", respectively, to make firm some of our intuitions about these concepts, which behave formally in much the same way. For example, the identity

$$\text{Interpretation-of}[^\vee\,^\wedge\alpha] = \text{Interpretation-of}[\alpha]$$

is preserved in the translation:

$$eval[(\text{EVAL}(\text{LIST}(\text{QUOTE QUOTE})\alpha)); a] = eval[\alpha; a]$$

for all α and for all a.[7]

As appealing as this analogy is, however, it is desirable to treat intension and extension in another way, relating reference points to environment indirectly. We may assume there is a variable named * to the value of which intensions are applied to produce the corresponding extensions. We need place no restrictions on what * can be bound to. For example it could be an arbitrarily complex object corresponding to a model of a possible world, implemented as a data base, a list of functions, or any other suitable structure. For brevity we will call the data type of * "*s*-list", after the "*s*" in Montague's hierarchy of types.

Now rules (1)–(4) are replaced by rules (1a)–(4a):

(1a) α, a constant $\rightarrow$ (QUOTE α),

(2a) α, a variable $\rightarrow$ α,

[7] There is a certain clumsiness in rules (1)–(3). This results from the need for EVAL to serve a double role: modeling the extension operator and implementing the substitution semantics of LISP.

(3a) $^\wedge\alpha$ $\rightarrow$ (INT* α), where
INT* = (LAMBDA(G)
(LAMBDA(*) G)),

(4a) $^\vee\alpha$ $\rightarrow$ (α *).

Note that here too $^{\vee\wedge}\alpha$ has the same interpretation as α, i.e.

$$\begin{aligned} eval[((\mathrm{INT}^*\ \alpha)\ *);\ a] &= eval[(((\mathrm{LAMBDA}(G)\ (\mathrm{LAMBDA}(*)\ G))\ \alpha)\ *);\ a] \\ &= eval[\alpha;\ a]. \end{aligned}$$

The remaining translation rules are the same for either approach:

(5) $\lambda u\alpha$ $\rightarrow$ (LAMBDA(U) α),
(6) $\alpha(\beta)$ $\rightarrow$ (α β),
(7) $\alpha = \beta$ $\rightarrow$ (EQUALF α β),
(8) $\sim\phi$ $\rightarrow$ (NOT ϕ),
(9) ϕ & ψ $\rightarrow$ (AND ϕ ψ),
(10) $\phi \vee \psi$ $\rightarrow$ (OR ϕ ψ),
(11) $\phi \supset \psi$ $\rightarrow$ (IMPLIES ϕ ψ),
(12) $\phi \leftrightarrow \psi$ $\rightarrow$ (IFF ϕ ψ),
(13) $\exists u\phi$ $\rightarrow$ (FORSOME ⟨range of u⟩ (LAMBDA(U) ϕ)),
(14) $\forall u\phi$ $\rightarrow$ (FORALL ⟨range of u⟩ (LAMBDA(U) ϕ)),
(15) $\Box\phi$ $\rightarrow$ (NEC(QUOTE ϕ)),
(16) $W\phi$ $\rightarrow$ (FUTURE(QUOTE ϕ)),
(17) $H\phi$ $\rightarrow$ (PAST(QUOTE ϕ)).

The functions NOT, AND, OR, IMPLIES, and IFF are self-explanatory.

In rules (13) and (14), a naive extensional reduction of FORSOME and FORALL would be a procedure in which the expression used for the range of u would actually evaluate to a finite list. The predicate which is the second argument would be applied to the members of the list. In the more sophisticated definition of FORALL and FORSOME required for infinite sets, the range and predicate arguments would be "models" or "descriptors". FORALL, for example, would then seek to prove that if an element is in the range, the predicate is true for that element.

In rule (7) we could have used the LISP function EQUAL which tests for equality of symbolic expressions. This would have captured equality of entities and truth values. It would not work for higher types, however, since a single function can be expressed by many distinct symbolic expressions. For this reason we have postulated the operator EQUALF, which would seek to *prove* the equality of higher-type arguments, or descriptors of such arguments.

In rules (15)–(17), the reason that the proposition ϕ is quoted is that Montague's formal statement of the interpretation of the modal and tense operators does not call for the application of a function to an evaluated proposition, but rather it directs a different evaluation to take place. In LISP, evaluation must be explicitly blocked, hence the QUOTE. Note that rule (6) *does* call for evaluation.

The procedure NEC must show that the proposition ϕ is true for all possible worlds. A naive extensional reduction like that for FORALL, is not available for NEC—we cannot cycle through all possible a-lists or s-lists. Therefore we must use the second method: NEC must show that the set of constraints which define the possible worlds under consideration imply the truth of the proposition ϕ. For the operators FUTURE and PAST one could imagine interpretations based on both proof-theoretic methods and on extensional reduction.

All of this highlights the need for a deeper treatment of infinite objects.

4.2.

In this section we shall re-do the example of Section 3.1, show its translation into a LISP program, and present a possible, though oversimplified, computational interpretation.

Let us suppose the translation of "temperature" is the function TEMP which takes a procedure as its argument and checks whether it is a known "temperature-checking" procedure. By the rules given in Section 4.1 for translating into LISP, "the temperature" becomes:

```
(LAMBDA(P)
  (FORSOME entity-concepts
    (LAMBDA(Y)
      (FORALL entity-concepts
        (LAMBDA(X)
          (AND(IFF(TEMP X) (EQUAL X Y))
              ((P *) Y))))))).
```

We can define BE as follows:

```
BE = (LAMBDA(P)
        (LAMBDA(X) ((P *) (INT*(LAMBDA(Y)
                                    (EQUAL(X *) (Y *))))))).
```

Similarly, NINETY can be defined

```
NINETY = (LAMBDA(Q) ((Q *) (INT* 90))).
```

Thus, "be ninety" becomes

```
(BE(INT*(FUNCTION NINETY))),
```

which reduces to

```
(FUNARG
  (LAMBDA(X) ((P *) (INT*(LAMBDA(Y) (EQUAL(X *) (Y *))))))
  ((P · (FUNARG
    (LAMBDA(*) G)
    ((G · (FUNARG
      (LAMBDA(Q) ((Q *) (INT* 90)))
      NIL))))))).
```

If THE were defined in a manner similar to BE and NINETY, then the cumbersome lambda expressions could be replaced by their atomic designators. In this case, the form actually eval-ed is:

((THE(FUNCTION TEMP)) (INT*(BE(INT*(FUNCTION NINETY))))).

The result of applying the translation of "the temperature" to the intension of the translation of "be ninety" (as required by the semantic rules) will be *T* just in case there is one temperature-reading function and that function applied to the current * returns 90; otherwise the result will be NIL.

5. Context and Expectation

Much has been written about the role of context in the interpretation of words and sentences. Frequently the context is specified by a natural language description of any circumstances, whether mental, textual, or environmental, which impinge upon the meaning of the item in question in any way. However, it is desirable to replace this by something more precise. One possibility is to view context as the state which the initial conditions and previous text have left the text processor in. Thus a text would serve as a context for a sentence exactly to the extent that the state of the processor was altered during the interpretation of the previous text.

Section 4 suggests a very concise description of the state of a processor in which the *a*-list plays a prominent role. A complete state description would contain control information as well, but for simplicity we will concentrate on the *a*-list. By employing techniques of dynamically binding variables to valid function specifications, we can circumvent the use of special, very complex data structures and still satisfy the dictum that knowledge "comes in large chunks".

Among the proposals for handling contextual effects is Minsky's [8] notion of "frames". A frame is a large, complex data structure, possibly with procedures attached, which expresses the normally true general knowledge about stereotyped situations. When a frame is accessed, subsequent processing becomes a matter of filling the slots and noting the exceptions. The claimed strengths of the frames approach include the existence of default values for unspecified arguments and the view of expectation as the ability to access pre-stored relevant pieces of knowledge efficiently. However, since frames as they have been used are amalgams of data structures and arbitrary procedures, the problems of representation become heavily involved with issues of encoding. Furthermore, it is unclear how one gets into a frame, whether one can be in more than one frame at a time, how one gets out of a frame, how two or more frames can be merged to understand novel texts and situations, etc. In short, these are structures for which there is no well understood interpreter.

By using Montague's functional approach to full advantage one can preserve the spirit of "frames" while overcoming these deficiencies. The key is to think of knowledge as residing in functions, each of which embodies a "core meaning" of a word (or concept) embedded in a meaningful pattern of function applications, both referring to the surrounding binding environment. In this view, the "large chunks of knowledge" are accessed because when a function is applied, it in turn calls other functions. The way that functions communicate is by binding and evaluating variables. Context is the set of active bindings, and the contextual effects of one function on another are expressed in the ways variables are shared by the functions. Context is changed when the bindings are updated by the application of a lambda expression to its arguments. The effect of updating the binding of a function on the interpretation of another function may be great or small, depending on how central or pervasive the first function is in the body of the second. This appears to us as plausible and as rich a method for context switching with selective override and default as any that have been proposed. The method is both subtle and fluid. The effects can be made abrupt or slight. The creation of new co[illegible] goes on all the time without the need for any special context-switching mechanism.

In order to be more concrete, let us consider the following sentences:

John approached Minneapolis. (16)

John approached maturity. (17)

In (16) "approach" is to be interpreted as motion along a scale of physical distances, in (17) a scale of development toward realization of some set of properties. In Jackendoff's [5] formalism "approach" in (16) is in the positional mode, in (17) in the identificational mode. Which scale or mode is relevant depends on the rest of the sentence, in particular on the direct object.

With Montague-style patterns of functional application, we can define the words "approach", "maturity", and "Minneapolis" in the following way:

$$\begin{aligned}
\text{approach} &= \lambda\mathscr{P}[\lambda x[{}^{\vee}\mathscr{P}({}^{\wedge}\lambda z[(\exists w_1)\,(\exists w_2)\\
&\qquad(\text{go}(x, w_1, w_2, \text{Scale})\\
&\qquad\ \&\ \text{exceed}(w_2, w_1, \text{Scale})\ \&\ \textsc{hi}(\text{Scale}) = {}^{\vee}z)])]],\\
\text{maturity} &= \lambda Q[\lambda\text{Scale}[{}^{\vee}Q({}^{\wedge}\textsc{hi}(\text{Scale}))]\,(\text{growth-scale})],\\
\text{Minneapolis} &= \lambda Q[\textsc{loc}(Q({}^{\wedge}\text{Minn}))\,({}^{\wedge}\text{Minn})].
\end{aligned} \tag{18}$$

In the definition for "approach", "go(x, w_1, w_2, Scale)" says that x goes from w_1 to w_2 on Scale; "exceed(w_2, w_1, Scale)" says that w_2 is closer to the high end of Scale than w_1 is; "HI(Scale) = $^{\vee}z$" says that the high end of Scale is $^{\vee}z$. In the definition for "maturity", "λScale[· · ·] (growth-scale)" will bind Scale to growth-scale within "approach" when "approach" is applied to "maturity". The core of "maturity" is HI(Scale); it seems appropriate to define maturity as the final point along a scale of maturation rather than as an arbitrary individual representing "perfect maturity". The expression "approach maturity" reduces to

$$\begin{aligned}
\lambda x[(\exists w_1)\,(\exists w_2)\,(\text{go}(x, w_1, w_2, \text{growth-scale})\ \&\ \text{exceed}(w_2, w_1, \text{growth-scale})\\
\&\ \textsc{hi}(\text{growth-scale}) = \textsc{hi}(\text{growth-scale}))].
\end{aligned}$$

In the definition of "Minneapolis", "Minn" is an entity corresponding to the individual Minneapolis. "LOC" is an operator whose effect is to bring in bindings which are appropriate by virtue of Minneapolis being a location. It is one device for encoding the "is-*a*" or superset relation within the functional approach. LOC is defined

$$\text{LOC} = \lambda \text{core}[\lambda z[\lambda \text{Scale}[{}^{\vee}\text{core}]\,(\text{distance-scale-toward}(z))]].$$

When it is applied within (18),

$$\text{Minneapolis} = \lambda Q[\lambda \text{Scale}[{}^{\vee}Q({}^{\wedge}\text{Minn})]\,(\text{distance-scale-toward}({}^{\wedge}\text{Minn}))]$$

results. Thus, Scale is bound to distance-scale-toward(^Minn). "Approach Minneapolis" reduces to

$$\begin{aligned}\lambda x[(\exists w_1)\,(\exists w_2)\,(&\text{go}(x, w_1, w_2, \text{distance-scale-toward}({}^{\wedge}\text{Minn}))\\ &\&\ \text{exceed}(w_2, w_1, \text{distance-scale-toward}({}^{\wedge}\text{Minn}))\\ &\&\ \text{HI}(\text{distance-scale-toward}({}^{\wedge}\text{Minn})) = \text{Minn})].\end{aligned}$$

Here contextual knowledge is brought to bear indirectly by the binding patterns, and the verb "approach" need not even check whether its object is a physical location or an attributed state.

The frames approach is oriented toward using knowledge of the situation described in the text, and it seems to be rather weak in utilizing the structure of the text itself. This may be remedied by adopting an approach closer to the one we have described.

One of the strengths of Montague's approach is his way of attaching meaning to the intermediate results, to sentence fragments. For example,

The old overstuffed chair in a dark corner of the room (19)

at the same time creates an image and leaves the reader with a sense of expectation. In Montague's approach, the image is captured by the "core meaning" of the function corresponding to (19), and the sense of expectation lies in the fact that the function has not yet been applied to its argument.

This view of expectation as a function waiting for its argument is adequate within the boundaries of a sentence. But since each sentence is of type t, there is no function which is still waiting for its argument intersententially. We might postulate that an effect of a sentence with respect to the entire text is to set up a "megafunction" which gets applied to the similar megastructures resulting from other sentences in the text in much the same way as an English word sets up a function which gets applied to the other words in the sentence. For example, a sentence which describes a change of state might be viewed as a function which takes sentence arguments of a certain type. Sentences are of this type if they presuppose or assert the final state of the change. This is a very suggestive way of looking at the notion of expectation. Whether it is a fruitful way remains to be seen.

ACKNOWLEDGMENTS

We are indebted to Ralph Grishman and Richard Smaby for their valuable suggestions. This work was partially supported by ONR Grant N00014-75-C-0571 and CUNY Faculty Research Award Program Grants Nos. 11233 and 11655.

REFERENCES

1. Davies, D. J. M. and Isard, S. D., Utterances as programs, in: B. Meltzer and D. Michie, eds., *Machine Intelligence* 7, New York (1972).
2. Dowty, David R., Montague Grammar and the Lexical Decomposition of Causative Verbs, in: Barbara H. Partee, ed., *Montague Grammar* (Academic Press, New York, 1976).
3. Hobbs, Jerry, A model for natural language semantics, part I: the model, Yale University Department of Computer Science Research Report No. 36 (November 1974).
4. Isard, Stephen, D., What would you have done if...?, *Theoretical Linguistics* **1** (3) (1974).
5. Jackendoff, Ray, Toward an explanatory semantic representation, *Linguistic Inquiry* **7** (1) (Winter 1976) 89-150.
6. Joshi, A. K. and Weischedel, R. M., Some frills for the modal tic-tac-toe of Davies and Isard: semantics of predicate complement constructions, *Proc. Third International Joint Conference on Artificial Intelligence*, Stanford, CA (1973).
7. McCarthy, John, et al., *LISP 1.5 Programmer's Manual* (M.I.T. Press, Cambridge, MA, 1965).
8. Minsky, Marvin, A framework for representing knowledge, in: Patrick H. Winston, ed., *The Psychology of Computer Vision* (McGraw-Hill, New York, 1975).
9. Montague, Richard, The proper treatment of quantification in ordinary English, *Approaches to Natural Language: Proceedings of the* 1970 *Stanford Workshop on Grammar and Semantics*, Dordrecht (D. Reidel Publishing Company, 1973).
10. Montague, Richard, *Formal Philosophy* (Selected Papers of Richard Montague, ed. and with an introduction by Richmond Thomason) (Yale University Press, New Haven and London, 1974).
11. Moses, Joel, The function of FUNCTION in LISP, AI Memo No. 199 (M.I.T. AI Lab., Cambridge, MA, July 1970).
12. Partee, Barbara Hall, ed., *Montague Grammar* (Academic Press, New York, 1976).
13. Sussman, Gerald, and Steele, Guy, SCHEME: An interpreter for extended lambda calculus, AI Memo No. 349 (M.I.T. AI Lab., Cambridge, MA, December 1975).
14. Tarski, Alfred, Der Wahrheitsbegriff in dem formalisierten Sprachen, *Studia Philosophica* **I** (1936) 261-405. Translated as: The concept of truth in formalized languages, *Logic, Semantics, Metamathematics*, Oxford (1956).
15. Winograd, Terry, *Understanding Natural Language* (Academic Press, New York, 1972).
16. Woods, William, Procedural semantics for a question-answering machine, *Proc. AFIPS 1968 Fall Joint Computer Conference* 33 (1968) 457-471 (Thompson Book Company, Washington, D.C.).

Received December 1976; revised version received October 1977