American Journal of Computational Linguistics *Microfiche 74*

TWO PAPERS ON SEMANTIC INTERPRETATION IN MONTAGUE GRAMMAR

Joyce Friedman
Douglas B. Moran
David S. Warren

Department of Computer and Communication Sciences
The University of Michigan
Ann Arbor, Michigan 48109

CONTENTS

EXPLICIT FINITE INTENSIONAL MODELS FOR PTQ

AN INTERPRETATION SYSTEM FOR MONTAGUE GRAMMAR

N-3
Computer Studies in
Formal Linguistics

November 1977
Revised March 1978

Department of Computer and
Communication Sciences
THE UNIVERSITY OF MICHIGAN
Ann Arbor, Michigan 48109

EXPLICIT FINITE INTENSIONAL MODELS FOR PTQ

Joyce Friedman, Douglas B. Moran and David S. Warren

ABSTRACT

The semantics of Montague's *The proper treatment of quantification in ordinary English* (PTQ) uses an intensional model to evaluate formulas. In this primarily tutorial paper we show how a model can be explicitly constructed and used. Examples of the evaluation of formulas are worked through carefully. Particular attention is paid to the role of the meaning postulates of PTQ in restricting the choice of models.

An abbreviatory notation, giving names to complex elements, is used to simplify the process of constructing a model. At each level, complex elements are formed from simpler elements already named.

The size of a finite model for PTQ based on two points of reference and two entities is calculated and its implications discussed.

This research is supported in part by National Science Foundation Grants BNS 76-23940 and MCS 76-0497

I. INTRODUCTION

In *The proper treatment of quantification in ordinary English* (PTQ), Richard Montague sets up a system which uses model-theoretic semantics to provide meanings for English sentences. Expressions of intensional logic hold a position intermediate between the English syntax and the model. For each syntactic structure of an English phrase there is a corresponding formula of intensional logic. The meaning of the English phrase is taken to be the interpretation of the logical formula in the model.

In this paper, which is primarily tutorial, we show by example how a model can be explicitly constructed and how a logical formula is interpreted in a model. Our paper provides concrete examples of the semantic model and the definition of interpretation, given only formally by Montague. It is intended to be helpful to readers of PTQ. The reader of this paper may need to have a copy of PTQ in hand.

We first review the definition of intensional model, and begin to specify a model. Then we examine the way in which meaning postulates constrain the model to be reasonable. In a reasonable model the interpretations of English words are consistent with their usual meanings. We select a particular reasonable model and use it to evaluate some formulas. Problems in building a larger explicit model are illustrated by considering the case of adverbs. We conclude with calculations of the size of a model and a brief discussion of the possible use of computers for Montague grammar.

II. INTENSIONAL MODELS

An *intensional model (or interpretation)* $\mathfrak{A}$ is a quintuple $\mathfrak{A} = \langle A, I, J, \leq, F \rangle$ such that

1) A, I, and J are non-empty sets, the set of entities, the set of possible worlds, and the set of moments in time, respectively.
2) $\leq$ is a simple (linear) ordering on J.
3) F is the meaning function, which assigns each logical constant an appropriate element from the model. If the constant is of type a, the value of F is of type $\langle s,a \rangle$.

Each element of $I \times J$, the set of co-ordinates, is a single *point of reference* or *index*. To simplify the notation, we use S for this cross-product. A model then becomes a quadruple $\mathfrak{A} = \langle A, S, \leq, F \rangle$. For a given A and S the set of possible denotations of type α, $D_{\alpha,A,S}$ is given by:

- for simple types

$D_{e,A,S} = A$ (the set of entities)

$D_{t,A,S} = \{0,1\}$ (0 - falsehood, 1 - truth)

- for complex types

$D_{\langle s,a \rangle,A;S} = D^S_{a,A,S}$ (the set of total functions from S to $D_{a,A,S}$)

$$D_{\langle a,b\rangle,A,S} = D_{b,A,S}^{D_{a,A,S}}$$ (the set of total functions from $D_{a,A,S}$ to $D_{b,A,S}$)

Where no confusion can arise, the subscripts A and S are omitted in symbols for sets of possible denotations.

The rules for evaluating an expression of the logic are given in PTQ. An evaluation is performed with respect to a model $\mathfrak{A}$, a point of reference i in S, and a variable assignment g. This function g assigns a denotation of the appropriate type to each variable in the expression, that is, for any variable u of type a, $g(u) \in D_a$. The result of evaluating an expression α of type a is a possible denotation of type a, i.e., a member of D_a. This value is denoted by $\alpha^{\mathfrak{A},i,g}$ and is called the denotation or extension of α with respect to $\mathfrak{A}$, i, and g.

III. SPECIFYING A MODEL

The first step is to give the set A of entities and the set S of points of reference. These two sets uniquely determine for each type a each set D_a of possible denotations of expressions of type a. To complete the model, the meaning function F must be specified. The values of F for constants of type a are functions from S to D_a.

We now begin to build the intensional model to be used in our examples. Because we wish to write out the sets D_a explicitly, we construct a finite model. While it would be possible to write functions explicitly in ordered pair notation, the result would be long and cumbersome, because in general the elements of the pairs are themselves functions. We overcome this difficulty by introducing names for functions and by taking advantage of the type system of the model. We use the type system to provide an order in which to consider the denotation sets and their elements. The ordering is such that at each stage the new functions can be specified as ordered pairs of names already introduced. The meaning function F is also specified using these function names.

Let the set of points of reference S be $\{I1, I2\}$ and the set of entities, A or D_e, be $\{Jo, Un\}$.

It is important to distinguish words in the English vocabulary from constants in the logic from elements of the model; for example, *John* and *walk* are English words, j and $walk'$ are logical constants, and Jo is an entity, an element of D_e. English words are given in italics. Logical constants that are direct translations of English words are primed.

The function F, which defines the relationship between constants and elements of the model, assigns to each constant of type e a function from indices to entities. For example, for the logical constant j, $F(j) \in D_{\langle s,e\rangle}$ where $D_{\langle s,e\rangle} = D_e^S$. In our finite model, there are only four members of $D_{\langle s,e\rangle}$, and we use $\alpha_0, \alpha_1, \alpha_2, \alpha_3$ as their names. They are defined by:

$$D_{\langle s, e\rangle} = \begin{vmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{vmatrix} = \begin{vmatrix} \{(I1\ Jo)\ (I2\ Jo)\} \\ \{(I1\ Jo)\ (I2\ Un)\} \\ \{(I1\ Un)\ (I2\ Jo)\} \\ \{(I1\ Un)\ (I2\ Un)\} \end{vmatrix}$$

If $F(j) = \alpha_2$, then j is assigned $\{(I1\ Un)\ (I2\ Jo)\}$, that is, the function whose value at index $I1$ is Un and whose value at index $I2$ is Jo.

Words in both of the syntactic categories CN and IV translate into logical constants of type $\langle\langle s, e\rangle, t\rangle$, so the values of F for these constants are functions from indices to elements of type $\langle\langle s, e\rangle, t\rangle$. For example,

$F(unicorn')$ and $F(walk') \in D_{\langle s, \langle\langle s, e\rangle, t\rangle\rangle}$

$$\text{where } D_{\langle s, \langle\langle s, e\rangle, t\rangle\rangle} = D^{S}_{\langle\langle s, e\rangle, t\rangle} = (D_t^{D_{\langle s, e\rangle}})^{S}$$

and

$F(unicorn')(i)$ and $F(walk')(i) \in D_{\langle\langle s, e\rangle, t\rangle}$

$$\text{where } D_{\langle\langle s, e\rangle, t\rangle} = D_t^{D_{\langle s, e\rangle}}$$

In the model, there are 16 members of $D_{\langle\langle s, e\rangle, t\rangle}$ and we use $\beta_0, \ldots, \beta_{15}$ as their names; they are defined by:

$$D_{\langle\langle s,e\rangle,t\rangle} = \left|\begin{array}{l} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \\ \beta_6 \\ \beta_7 \\ \beta_8 \\ \beta_9 \\ \beta_{10} \\ \beta_{11} \\ \beta_{12} \\ \beta_{13} \\ \beta_{14} \\ \beta_{15} \end{array}\right| \quad \left|\begin{array}{llll} (\alpha_0\ 0) & (\alpha_1\ 0) & (\alpha_2\ 0) & (\alpha_3\ 0) \\ (\alpha_0\ 0) & (\alpha_1\ 0) & (\alpha_2\ 0) & (\alpha_3\ 1) \\ (\alpha_0\ 0) & (\alpha_1\ 0) & (\alpha_2\ 1) & (\alpha_3\ 0) \\ (\alpha_0\ 0) & (\alpha_1\ 0) & (\alpha_2\ 1) & (\alpha_3\ 1) \\ (\alpha_0\ 0) & (\alpha_1\ 1) & (\alpha_2\ 0) & (\alpha_3\ 0) \\ (\alpha_0\ 0) & (\alpha_1\ 1) & (\alpha_2\ 0) & (\alpha_3\ 1) \\ (\alpha_0\ 0) & (\alpha_1\ 1) & (\alpha_2\ 1) & (\alpha_3\ 0) \\ (\alpha_0\ 0) & (\alpha_1\ 1) & (\alpha_2\ 1) & (\alpha_3\ 1) \\ (\alpha_0\ 1) & (\alpha_1\ 0) & (\alpha_2\ 0) & (\alpha_3\ 0) \\ (\alpha_0\ 1) & (\alpha_1\ 0) & (\alpha_2\ 0) & (\alpha_3\ 1) \\ (\alpha_0\ 1) & (\alpha_1\ 0) & (\alpha_2\ 1) & (\alpha_3\ 0) \\ (\alpha_0\ 1) & (\alpha_1\ 0) & (\alpha_2\ 1) & (\alpha_3\ 1) \\ (\alpha_0\ 1) & (\alpha_1\ 1) & (\alpha_2\ 0) & (\alpha_3\ 0) \\ (\alpha_0\ 1) & (\alpha_1\ 1) & (\alpha_2\ 0) & (\alpha_3\ 1) \\ (\alpha_0\ 1) & (\alpha_1\ 1) & (\alpha_2\ 1) & (\alpha_3\ 0) \\ (\alpha_0\ 1) & (\alpha_1\ 1) & (\alpha_2\ 1) & (\alpha_3\ 1) \end{array}\right|$$

There are $16^2 = 256$ members of $D_{\langle s,\langle\langle s,e\rangle,t\rangle\rangle}$ and they will not be enumerated here.

The set of *entities*, $A = D_e$, is also known as the set of *individuals*. The members of the set $D_{\langle s,e\rangle}$ of possible denotations of type $\langle s,e\rangle$ are called *individual concepts*.

The members of $D_{\langle\langle s,e\rangle,t\rangle}$ are functions from $D_{\langle s,e\rangle}$ to {0,1}, and thus a member of $D_{\langle\langle s,e\rangle,t\rangle}$ can be viewed as a *set of individual concepts* for which the function is 1 (true). An element of $D_{\langle s,\langle\langle s,e\rangle,t\rangle\rangle}$ is a *property of individual concepts.*

IV. REASONABLE MODELS

Not all models that can be constructed are reasonable. Montague's meaning postulates are restrictions on models; they limit the choices for the values of the meaning function F, that is, they constrain the possible meaning of certain English words.

Meaning Postulate 1

We now examine how Meaning Postulate 1 affects the choice of values of F for constants of type e. As an example, we consider $F(j)$. Meaning Postulate 1 for j is $(\exists u)\ \Box\ (u=j)$. The denotation of this meaning postulate is $[(\exists u)\ \Box\ (u=j)]^{\mathfrak{A},i,g}$. This denotation must be 1 if the model is to be reasonable. Following the recursive definition in PTQ [p. 258]:

$[(\exists u)\ \Box\ (u=j)]^{\mathfrak{A},i,g}$ is 1 iff there exists $\nu \in D_e$ ($D_e = \{Jo, Un\}$) such that $[\Box(u=j)]^{\mathfrak{A},i,g'}$ is 1, where g' is a $\mathfrak{A}$-assignment like g except that $g'(u)$ is ν.

$[\Box(u=j)]^{\mathfrak{A},i,g'}$ is 1 iff $(u=j)^{\mathfrak{A},i',g'}$ is 1 for all $i' \in S$.

$(u=j)^{\mathfrak{A},i',g'}$ is 1 iff $u^{\mathfrak{A},i',g'}$ is $j^{\mathfrak{A},i',g'}$.

$u^{\mathfrak{A},i',g'}$ is $g'(u)$, which from above is ν, and $j^{\mathfrak{A},i',g'}$ is $F(j)(i')$.

I.e., there exists $\nu \in D_e$ such that for all $i' \in S$, $F(j)(i')$ is ν. In other words, Meaning Postulate 1 requires that the value of F for the argument j be a constant function, i.e., a function that has the same value at all points of reference. Thus $F(j)$ cannot be just any member of $D_{\langle s,e\rangle}$; it must be either $\alpha_0 = \{(I1,Jo)\ (I2,Jo)\}$ or $\alpha_3 = \{(I1,Un)(I2,Un)\}$.

Meaning Postulate 2

For Meaning Postulate 2 a similar analysis applies. This meaning postulate restricts the choice of values of F for the constants of type $\langle\langle s,e\rangle,t\rangle$ which are translations of extensional common nouns. For example, Meaning Postulate 2 for *unicorn'* is $\square[unicorn'(x) \rightarrow (\exists u)\ x = {}^\wedge u]$. A reasonable model must make it true. Again, following the recursive definition:

$(\square[unicorn'(x) \rightarrow (\exists u)\ x = {}^\wedge u])^{\mathfrak{A},i,g}$ is 1 iff
  $[unicorn'(x) \rightarrow (\exists u)\ x = {}^\wedge u])^{\mathfrak{A},i',g}$ is 1 for all $i' \in S$.

$[unicorn'(x) \rightarrow (\exists u)\ x = {}^\wedge u]^{\mathfrak{A},i',g}$ is 1 iff whenever
  $[unicorn'(x)]^{\mathfrak{A},i',g}$ is 1, then $[(\exists u)\ x = {}^\wedge u]^{\mathfrak{A},i',g}$ is 1.

$[unicorn'(x)]^{\mathfrak{A},i',g}$ is $[unicorn']^{\mathfrak{A},i',g}(x)^{\mathfrak{A},i',g}$.

$[unicorn']^{\mathfrak{A},i',g}$ is $F(unicorn')(i')$ and $x^{\mathfrak{A},i',g}$ is $g(x)$.

$[unicorn'(x)]^{\mathfrak{A},i',g}$ is $[F(unicorn')(i')]\ (g(x))$.

$[(\exists u)\ x = {}^\wedge u]^{\mathfrak{A},i',g}$ is 1 iff there exists a $\nu \in D_e$ such
  $[x = {}^\wedge u]^{\mathfrak{A},i',g'}$ is 1 where g' is a $\mathfrak{A}$-assignment like
  g except that $g'(u)$ is ν.

$[x = {}^\wedge u]^{\mathfrak{A},i',g'}$ is 1 iff $x^{\mathfrak{A},i',g'}$ is $({}^\wedge u)^{\mathfrak{A},i',g'}$.

$x^{\mathfrak{A}, i', g'}$ is $g'(x)$ and from above, $g'(x)$ is $g(x)$, and $({}^{\wedge}u)^{\mathfrak{A}, i', g'}$ is the function h such that for all $i'' \in S$, $h(i'')$ is $u^{\mathfrak{A}, i'', g'}$, but $u^{\mathfrak{A}, i'', g'}$ is $g'(u)$ is ν, so h is $\lambda^* i''(\nu)$.[1]

$[x = {}^{\wedge}u]^{\mathfrak{A}, i', g'}$ is 1 iff $g(x)$ is $\lambda^* i''(\nu)$.

For all $i' \in S$, whenever $[F(unicorn')(i')](g(x))$ is 1, then there exists $\nu \in D_e$ such that $g(x)$ is $\lambda^* i''(\nu)$.

The consequent of the result says that the individual concept that is the value of (x) must be a constant function, i.e., it must evaluate to the same entity ν at every point of reference. In our model, only α_0 and α_3 are constant individual concepts.

Meaning Postulate 2 restricts the possible denotations for these logical constants of type $\langle\langle s,e\rangle,t\rangle$ to subsets of the individual concepts that are constant functions. In any reasonable model, we must choose F so that

$F(unicorn')(i) \in \{\beta_0, \beta_1, \beta_8, \beta_9\}$,

that is, *unicornhood* can be true of only those individual concepts which yield the same entity at every point of reference.

---

[1]We use λ^* as the λ-operator of the metalanguage. Thus, $\lambda^* i''(\nu)$ denotes the function from S to D_e with the constant value ν.

Meaning Postulate 3

Meaning Postulate 3 restricts the choice of values of F for the constants that are translations of extensional intransitive verbs. Meaning Postulate 3 for $walk'$ is

$$(\exists M)(\forall x)\ \Box\ [walk'(x) \leftrightarrow [{}^{\vee}M]({}^{\vee}x)]$$

where M and x are variables of type $\langle s, \langle e, t\rangle\rangle$ and $\langle s, e\rangle$, respectively. A reasonable model must make this meaning postulate true.

$[(\exists M)(\forall x)\Box[walk'(x) \leftrightarrow [{}^{\vee}M]({}^{\vee}x)]]^{\mathfrak{A},i,g}$ is 1 iff there exists an $m \in D_{\langle s, \langle e, t\rangle\rangle}$ such that $[(\forall x)\Box[walk'(x) \leftrightarrow [{}^{\vee}M]({}^{\vee}x)]]^{\mathfrak{A},i,g'}$ is 1 where g' is a $\mathfrak{A}$-assignment like g except that $g'(M)$ is m.

$[(\forall x)\Box[walk'(x) \leftrightarrow [{}^{\vee}M]({}^{\vee}x)]]^{\mathfrak{A},i,g'}$ is 1 iff for all $\chi \in D_{\langle s, e\rangle}$ $[\Box[walk'(x) \leftrightarrow [{}^{\vee}M]({}^{\vee}x)]]^{\mathfrak{A},i,g''}$ is 1 where g'' is a $\mathfrak{A}$-assignment like g' except that $g''(x)$ is χ.

$[\Box[walk'(x) \leftrightarrow [{}^{\vee}M]({}^{\vee}x)]]^{\mathfrak{A},i,g''}$ is 1 iff $[walk'(x) \leftrightarrow [{}^{\vee}M]({}^{\vee}x)]]^{\mathfrak{A},i',g''}$ is 1 for all $i' \in S$.

$[walk'(x) \leftrightarrow [{}^{\vee}M]({}^{\vee}x)]^{\mathfrak{A},i',g''}$ is 1 iff $[walk'(x)]^{\mathfrak{A},i',g''}$ is $[[{}^{\vee}M]({}^{\vee}x)]^{\mathfrak{A},i',g''}$.

$[walk'(x)]^{\mathfrak{A},i',g''}$ is $[walk']^{\mathfrak{A},i',g''}(x)^{\mathfrak{A},i',g''}$

which is $[F(walk')(i')](g''(x))$

which is $[F(walk')(i')](\chi)$.

$[[{}^{\vee}M]({}^{\vee}x)]^{\mathfrak{A},i',g''}$ is $[{}^{\vee}M]^{\mathfrak{A},i',g''}({}^{\vee}x)^{\mathfrak{A},i',g''}$

which is $[M^{\mathfrak{A},i',g''}(i')](x^{\mathfrak{A},i',g''}(i'))$

which is $[g''(M)(i')](g''(x)(i'))$

which is $[m(i')](\lambda(i'))$.

Thus the meaning postulate is true just in case there exists $m \in D_{\langle s,\langle e,t\rangle\rangle}$ such that for all $\lambda \in D_{\langle s,e\rangle}$, $[F(walk')(i')](\lambda)$ is $[m(i')](\lambda(i'))$ for all $i \in S$. Thus, $walk'$ depends only on the entity and the point of reference. If $walk'$ is true at a point of reference for an individual concept, then $walk'$ is true at that point of reference for every individual concept that gives the same entity at that point of reference.

Relating this to our model, we see that since

$$\alpha_0(I1) = \alpha_1(I1) \text{ and } \alpha_2(I1) = \alpha_3(I1),$$

$$\alpha_0(I2) = \alpha_2(I2) \text{ and } \alpha_1(I2) = \alpha_3(I2)$$

Meaning Postulate 3 requires:

$$[F(walk')(I1)](\alpha_0) = [F(walk')(I1)](\alpha_1)$$

$$[F(walk')(I1)](\alpha_2) = [F(walk')(I1)](\alpha_3)$$

$$[F(walk')(I2)](\alpha_0) = [F(walk')(I2)](\alpha_2)$$

$$[F(walk')(I2)](\alpha_1) = [F(walk')(I2)](\alpha_3)$$

Thus, we must choose F so that

$$F(walk')(I1) \in \{\beta_0, \beta_3, \beta_{12}, \beta_{15}\}$$

$$F(walk')(I2) \in \{\beta_0, \beta_5, \beta_{10}, \beta_{15}\}$$

V. USING A MODEL

We now explicitly construct a particular reasonable model in which to evaluate some formulas.

It must satisfy the constraints on F developed above. We assign to the constant j the entity *Jo* at all points of reference:

$F(j) = \alpha_0 = \{(I1\ Jo)(I2\ Jo)\}$.

We stipulate that there are no unicorns at point of reference *I1* and one unicorn, entity *Un*, at point of reference *I2*:

$$F(unicorn') = \{(I1\ \beta_0)(I2\ \beta_1)\}$$
$$= \{(I1\ \{(\alpha_0\ 0)(\alpha_1\ 0)(\alpha_2\ 0)(\alpha_3\ 0)\})$$
$$(I2\ \{(\alpha_0\ 0)(\alpha_1\ 0)(\alpha_2\ 0)(\alpha_3\ 1)\})\}.$$

At point of reference *I1*, entities *Jo* and *Un* walk and at point of reference *I2*, only *Jo* walks:

$$F(walk') = \{(I1\ \beta_{15})(I2\ \beta_{10})\}$$
$$= \{(I1\ \{(\alpha_0\ 1)(\alpha_1\ 1)(\alpha_2\ 1)(\alpha_3\ 1)\})$$
$$(I2\ \{(\alpha_0\ 1)(\alpha_1\ 0)(\alpha_2\ 1)(\alpha_3\ 0)\})\}.$$

Our first evaluation using this model will be for the expression which is the translation of *John*:

$[\lambda P[{}^{\vee}P](\hat{}j)]$

Finding the denotation of this expression:

$\ulcorner\lambda P[{}^{\vee}P](\hat{}j)]^{\mathfrak{A},i,g}$ is a function h_{j*} such that $h_{j*}(\rho)$ is $[[{}^{\vee}P](\hat{}j)]^{\mathfrak{A},i,g'}$ where g' is a $\mathfrak{A}$-assignment like g except that $g'(P)$ is ρ.

$[{}^{\vee}P](\hat{}j)]^{\mathfrak{A},i,g'}$ is $[{}^{\vee}P]^{\mathfrak{A},i,g'}(\hat{}j)^{\mathfrak{A},i,g'}$

which is $[P^{\mathfrak{A},i,g'}(i)](\lambda{*}i'[j^{\mathfrak{A},i',g'}])$

which is $[g'(P)(i)](\lambda{*}i'[F(j)(i')])$

which is $[\rho(i)](F(j))$.

Thus, the expression denotes the function $h_{j*} = \lambda{*}\rho[[\rho(i)](F(j))]$ So h_{j*} is the set of properties which, when evaluated at point of reference i, are true of the individual concept that is the value of F for j. In this example of a reasonable model, $F(j)$ is α_0 and the possible denotations, $\rho(i)$, for which α_0 gets true are β_8 through β_{15}. At point of reference $I1$, the value of the function which is the denotation of the expression that translates *John* is 1 for the arguments

$\{(I1\ \beta_m)(I2\ \beta_n)\}$ where $m \geq 8$

and at point of reference $I2$, it is 1 for the arguments

$\{(I1\ \beta_m)(I2\ \beta_n)\}$ where $n \geq 8$.

Now, we evaluate the direct translation of the sentence *John walks*:

$[\lambda P[{}^{\vee}P](^{\wedge}j)](^{\wedge}walk')$.

At point of reference $I1$, the denotation of this expression is:

$[[\lambda P[{}^{\vee}P](^{\wedge}j)](^{\wedge}walk')]^{\mathfrak{A},I1,g}$

$[\lambda P[{}^{\vee}P](^{\wedge}j)]^{\mathfrak{A},I1,g}(^{\wedge}walk')^{\mathfrak{A},I1,g}$

$[\lambda P[{}^{\vee}P](^{\wedge}j)]^{\mathfrak{A},I1,g}(F(walk'))$

$h_{j*}\ (\{(I1\ \beta_{15})(I2\ \beta_{10})\})$

1

The model can be also used to illustrate the general fact that the denotations of an expression and any of its reductions are the same. We have shown elsewhere [Friedman, 1978] that each expression of the logic has a unique reduced form in which no further contraction is possible. We now evaluate the reduced form of the translation of *John walks*, $[walk'(\hat{}j)]$.

At point of reference *I1*:

$[walk'(\hat{}j)]^{\mathfrak{A},I1,g}$

$walk'^{\mathfrak{A},I1,g}(\hat{}j)^{\mathfrak{A},I1,g}$

$[F(walk')(I1)](F(j))$

$[\{(I1\ \beta_{15})(I2\ \beta_{10})\}(I1)](F(j))$

$[\beta_{15}](F(j))$

$[\{(\alpha_0\ 1)(\alpha_1\ 1)(\alpha_2\ 1)(\alpha_3\ 1)\}](\alpha_0)$

1

As another example, consider the logical expression $[\lambda P[{}^{\vee}P](\hat{}j)](\hat{}unicorn')$.

Evaluating in world *I1*

$[[\lambda P[{}^{\vee}P](\hat{}j)](\hat{}unicorn')]^{\mathfrak{A},I1,g}$

$[\lambda P[{}^{\vee}P](\hat{}j)]^{\mathfrak{A},I1,g}(F(unicorn'))$

$h_{j*}\ (\{(I1\ \beta_0)(I2\ \beta_1)\})$

0

The reduced form of the expression is $unicorn'(\hat{}j)$.

Evaluating in world *I1*

$[unicorn'(\hat{}j)]^{\mathfrak{A},I1,g}$

$[F(unicorn')(I1)](F(j))$

$[\{(I1\ \beta_0)(I2\ \beta_1)\}(I1)](F(j))$

$[\beta_0](F(j))$

$[\{(\alpha_0\ 0)(\alpha_1\ 0)(\alpha_2\ 0)(\alpha_3\ 0)\}](\alpha_0)$

0

VI. THE MORE COMPLEX CASE OF ADVERBS

We now consider extending our model to accommodate sentences with adverbs such as *slowly*. We have that

$$F(slowly') \in D_{\langle s,\langle\langle s,\langle\langle s,e\rangle,t\rangle\rangle,\langle\langle s,e\rangle,t\rangle\rangle\rangle} = \left(D_{\langle\langle s,e\rangle,t\rangle}^{D^S_{\langle\langle s,e\rangle,t\rangle}}\right)^S$$

There are not only too many possible denotations to consider enumerating them, but in each possible denotation, there are too many components to consider fully specifying it (see below). Consequently, in this example, only that portion of $F(slowly')$ which is needed will be specified.

The denotation for the sentence *John walks slowly* is:

$$[[slowly'(\hat{\ }walk')](\hat{\ }j)]^{\mathfrak{A},i,g}$$

$$[slowly'^{\mathfrak{A},i,g}(\hat{\ }walk')^{\mathfrak{A},i,g}](\hat{\ }j)^{\mathfrak{A},i,g}$$

$$[[F(slowly')(i)](F(walk'))](F(j))$$

At point of reference *I1*, *Jo* and *Un* are both walking, and choosing that *Jo* is walking slowly:

$$F(slowly') = \{(I1\ \{\ldots(\{(I1\ \beta_{15})(I2\ \beta_{10})\}\ \beta_{12})\ldots\})$$
$$(I2\ \{\ \cdots\ \ .\ \ .\ \ .\ \ \})\}$$

and

$$[[F(slowly')(I1)](F(walk'))](F(j))$$

$$[[\{(I1\{\ldots(\{(I1\ \beta_{15})(I2\ \beta_{10})\}\ \beta_{12})\ldots\})(I2\{\ldots\})\}(I1)](F(walk'))](F(j))$$

$$[\{\ldots(\{(I1\ \beta_{15})(I2\ \beta_{10})\}\beta_{12})\ldots\}(\{(I1\ \beta_{15})(I2\ \beta_{10})\})](F(j))$$

$$\beta_{12}](F(j))$$

$$[\{(\alpha_0\ 1)(\alpha_1\ 1)(\alpha_2\ 0)(\alpha_3\ 0)\}](\alpha_0)$$

$$1$$

VII. SIZE OF A MODEL

The smallest interesting model for PTQ has two points of reference and two entities. For such a model, the size of the sets of possible denotations and the size of their elements (in ordered pair representation) are as given in the following table.

Set of Possible Denotations	Number of Possible Denotations	Number of Ordered Pairs in Possible Denotation
D_t	2	-
$D_{\langle s,t\rangle} = D_t^S$	$2^2 = 4$	2
D_e	2	-
$D_{\langle s,e\rangle} = D_e^S$	$2^2 = 4$	2
$D_{f(CN)} = D_{f(IV)} = D_{\langle\langle s,e\rangle,t\rangle} = D_t^{D_{\langle s,e\rangle}}$	$2^4 = 16$	4
$D_{\langle s,f(CN)\rangle} = D_{\langle s,f(IV)\rangle} = D^S_{f(CN)} = D^S_{f(IV)}$	$16^2 = 2^8 = 256$	2
$D_{f(IV/t)} = D_{\langle\langle s,t\rangle,f(IV)\rangle} = D_{f(IV)}^{D_{\langle s,t\rangle}}$	$16^4 = 2^{16}$	4
$D_{\langle s,f(IV/t)\rangle} = D^S_{f(IV/t)}$	$(2^{16})^2 = 2^{32}$	2
$D_{f(IAV)} = D_{f(IV/IV)} = D_{\langle\langle s,f(IV)\rangle,f(IV)\rangle} = D_{f(IV)}^{D_{\langle s,f(IV)\rangle}}$	$(2^4)^{256} = 2^{1024}$	256
$D_{\langle s,f(IAV)\rangle} = D_{\langle s,f(IV/IV)\rangle} = D^S_{f(IAV)} = D^S_{f(IV/IV)}$	$(2^{1024})^2 = 2^{2048}$	2
$D_{f(T)} = D_{\langle\langle s,f(IV)\rangle,t\rangle} = D_t^{D_{\langle s,f(IV)\rangle}}$	2^{256}	256
$D_{\langle s,f(T)\rangle} = D^S_{f(T)}$	$(2^{256})^2 = 2^{512}$	2
$D_{f(TV)} = D_{\langle\langle s,f(T)\rangle,f(IV)\rangle} = D_{f(IV)}^{D_{\langle s,f(T)\rangle}}$	$(2^{2^2})^{2^{512}} = 2^{2^{514}}$	2^{512}
$D_{\langle s,f(TV)\rangle} = D^S_{f(TV)}$	$(2^{2^{514}})^2 = 2^{2^{515}}$	2
$D_{f(IAV/T)} = D_{\langle\langle s,f(T)\rangle,f(IAV)\rangle} = D_{f(IAV)}^{D_{\langle s,f(T)\rangle}}$	$(2^{2^{10}})^{2^{512}} = 2^{2^{522}}$	2^{512}
$D_{\langle s,f(IAV/T)\rangle} = D^S_{f(IAV/T)}$	$(2^{2^{522}})^2 = 2^{2^{523}}$	2

These size computations have relevance to a possible computer implementation. For a typical large-scale computer, the number of addressable memory locations is on the order of 2^{24} (about 16 million), yet 9 of the 16 sets of possible denotations have more members than this. This means that a full model cannot be explicitly represented in a computer. However, in evaluating expressions only some of the possible denotations are actually used. Implementation on a computer might allow a partial specification of a model, with only those possible denotations that are actually denoted by the expressions considered. In this context it is also interesting to note that the number of neurons in the human brain is currently estimated to be about 2^{33} (10 billion).

VIII. CONCLUSIONS

We have explicitly constructed an intensional model to illustrate the basic notions of the model-theoretic semantics of PTQ. We showed how the meaning postulates constrain the model to be reasonable and evaluated some simple formulas. The number of possible denotations in a small model was shown to present a problem to be solved if explicit representations of models are to be used. This problem is discussed further in Friedman, Moran, and Warren [1978].

REFERENCES

J. Friedman, Notes on an intensional logic for English II: A unique reduced form for wffs of IL N-10, Department of Computer and Communication Sciences, The University of Michigan, Ann Arbor, Mich., 1978.

J. Friedman, D. Moran, and D. Warren, An interpretation system for Montague grammar, to appear, American Journal of Computational Linguistics, 1978.

J. Hintikka, J. Moravcsik, and P. Suppes (eds.), Approaches to Natural Language, Reidel, Dordrecht. 1973.

Richard Montague, The proper treatment of quantification in ordinary English, (PTQ), in Hintikka et al., 1973, 221-242; reprinted in Montague, 1974, 247-270.

Richard Montague, Formal Philosophy: Selected Papers of Richard Montague, edited and with an introduction by Richmond Thomason Yale University Press, New Haven, 1974.

N - 4
Computer Studies in
Formal Linguistics

Department of Computer and
Communication Sciences
THE UNIVERSITY OF MICHIGAN
Ann Arbor, Michigan 48109

October 1977
Revised April 1978

AN INTERPRETATION SYSTEM FOR MONTAGUE GRAMMAR

Joyce Friedman, Douglas B. Moran, and David S. Warren

ABSTRACT

Model-theoretic semantics is one important approach to meaning in natural-language. In this approach, logical formulas obtained from English sentences are evaluated in an intensional model to determine their truth values. In this paper we describe an interactive computer system for specifying finite intensional model and evaluating logical formulas. We first give an overview of the system and its intended uses. We then provide a detailed description of the system to show what is entailed in actually carrying out this approach. Extensive examples are given in the Appendices.

Moran is responsible for the design and coding of the program. This research is supported in part by National Science Foundation Grants BNS 76-23840 and MCS 76-04297.

I. INTRODUCTION

Model-theoretic semantics is one important approach to analyzing the meaning of expressions of natural-language. Tarski's valuation method for formulas of mathematical logic can be extended to natural-language either directly or by an intermediate stage of translation into a mathematical logic. This formal truth-conditional treatment provides a precise means of assigning meanings to natural-language sentences. In model-theoretic semantics, the meaning of a formula is defined by a recursive process of evaluation in a model.

The formal system presented by Richard Montague in <u>The proper treatment of quantification in ordinary English</u> (PTQ) applies model-theoretic semantics to English and forms the basis of the work described in this paper. In PTQ the meaning of an English phrase is obtained by interpreting a corresponding logical formula in an appropriate model. PTQ uses a tensed intensional logic; the fragment of English emphasizes quantifiers and intensional constructs.

PTQ raises a number of interesting questions if one actually attempts to use the ideas in a computer system. For example, there are various ways in which a model might be represented. There are also problems in finding algorithms for the processes suggested by PTQ. We have made specific choices in our solutions to these questions and are pursuing them to find their limits.

At another time we may wish to consider some alternative approaches.

Our computer implementation of PTQ currently includes:

- a parser that finds all the 'interesting' derivations of a sentence [Friedman and Warren, forthcoming]
- a translator that produces a logical formula for each derivation and obtains a unique reduced form and an extensional form [Warren 1975; Friedman 1978]
- a model-builder that guides the specification of a logical model
- an interpreter that evaluates formulas in a model.

The latter two programs are the subject of this paper.

The model-builder and the interpreter constitute the semantic component of the PTQ system. They are designed to facilitate constructing, modifying and experimenting with models. The model-builder guides the specification of a model by prompting the user for components. The system determines an order of specification that is natural but flexible. It allows the user to delay some specifications if desired. The system permits dynamic models in which only those items that will be used need be entered. At any time the user may request interpretation of a logical formula; the formula is evaluated in the current model. If, during interpretation, the system finds that a model is under-specified, it requests the needed information and then completes the interpretation. The system also contains facilities to allow the user to add to or delete from the model.

The system is intended for use in linguistic research and for exploring potential applications of Montague grammar. Montague grammar presents some difficulties because of the complexity of the logic and its semantics. The logic of PTQ is a typed lambda calculus with intensional types. Interpretation is based on possible-worlds semantics. The meaning of a formula depends on the actual world in which it is evaluated, although the actual world does not appear explicitly in the formula. Informally, we might consider the actual world as a hidden free variable in the formula. Some accustomed laws, in particular substitutivity of equals and universal instantiation, fail to hold. Results are thus often counter-intuitive. Our computer implementation of interpretation has been valuable to us in our own research and as an aid in verifying or disproving conjectures that we have found in the literature.

Natural-language question answering systems translate an English sentence to internal representation and respond after processing the representation against stored data. PTQ is a formal framework in which an English sentence has a representation as a logical formula, and formulas have values defined using a logical model. This formal framework provides an approach to the natural-language question-answering problem, if computer representations and algorithms can be found for its implementation. Our implementation is of interest in this context; we plan to explore its applications further. (For another way of looking at Montague's framework computationally, see Hobbs and Rosenschein (1977).)

This document describes the model-building and interpretation system and how to use it. The underlying intensional logic is presented in Section II. The operation of the program is given in Sections III and IV. Section V explains the commands used in interacting with the system.

Examples are given in the text and three sample runs are provided in the Appendices. In all examples lines that are entered by the user are lower-case left-justified and prefixed by '*', whereas output lines are upper-case, prefixed by > and indented. In our introductory presentation of the model-theoretic semantics of PTQ (Friedman, Moran, and Warren, this fiche), a simple model in this notation is developed and used; Appendix A shows the use of the computer program to develop the model and interpret formulas in it. The user may want a model which satisfies a particular set of formulas; Appendix B shows the development of such a model during the interpretation of those formulas. In Appendix C, we develop a model as a counter-example to a formula given as a theorem in PTQ (p. 265).

A programmers description of the interpretation system is provided in Moran (N-8 forthcoming). Readers who might wish to adapt this program to typed lambda calculi with other types are referred to Moran (N-7, forthcoming)

II. INTENSIONAL LOGIC

The system builds models and interprets formulas in them. Before discussing the particulars of our program, we present here the underlying formal theory. The tensed intensional logic in the interpretation system is that of Montague (1970; 1973). We have found Gallin (1975) helpful as a source for a formal treatment of intensional logic, but have modified his notation to meet our needs.

A. Meaningful Expressions

The types of intensional logic are defined as follows: (1) E and TS are elementary types, (2) whenever a and b are types, <a,b> is a type, and (3) whenever a is a type, <S,a> is a type. For each type there is a set of constants and a set of variables. The following, and only the following, are meaningful expressions (ME's):

1) Every constant of type a is a ME of type a.
2) Every variable of type a is a ME of type a.
3) If A is a ME of type a and V is a variable of type b

(LAMBDA V, A) is a ME of type <b,a>.

4) If A is a ME of type <b,c> and B is a ME of type <u>b</u>, then (A B) is a ME of type <u>c</u>.

5) If A is a ME of type <c,<b,d>> and B and C are ME's of types <u>b</u> and <u>c</u>, then (A B C) is a ME of type <u>d</u>. (This expression and ((A C) B) will have the same meaning.)

6) If A and B are ME's of the same type, (EQUAL A B) is a ME of type TS.

7) If P and Q are ME's of type TS, so are (NOT P), (AND P Q), (OR P Q), (IMPLIES P Q), and (IFF P Q).

8) If P is a ME of type TS, so are (NECESSARILY P), (FUTURE P), and (PAST P).

9) If P is a ME of type TS and V is a variable, then (THERE-IS V P) and (FOR-ALL V P) are ME's of type TS.

10) If A is a ME of type <u>a</u>, (INT/ A) is a ME of type <S,a>.

11) If A is a ME of type <S,a>, (EXT A) is a ME of type <u>a</u>.

B. <u>Models of Intensional Logic</u>

<u>Models</u>

Let A and S be non-empty sets of entities and points of reference, respectively. A <u>frame based on A and S</u> is an indexed family of denotation-sets, (D_a) a ε <u>types</u>, where

(i) D_E = A

(ii) D_TS = {0,1}

(iii) D_<a,b> is a subset of $D_b^{D_a}$

(iv) $D_{\langle S,a\rangle}$ is a subset of $D_a{}^S$

For the elementary type E the set of possible denotations is the set of entities. For the elementary type IS the denotation-set has the two elements, (false) and 1 (true). For a complex type $\langle a,b\rangle$, the denotation-set $D_{\langle a,b\rangle}$ is a set of total functions with domain D_a and range D_b. We define D_S to be S, so that the special case $D_a{}^{D_S}$ is handled by the general rule.

A <u>model M on A and IxJ</u> is a pair $\langle (D_a)_{a \in \mathit{types}}, F\rangle$ where

1) $(D_a)_{a \in \mathit{types}}$ is a frame based on A and IxJ, where I is a set of possible worlds and J is an ordered set of moments in time.

2) F is the meaning function which assigns to each logical constant of type <u>a</u> an element of the denotation-set $D_{\langle S,a\rangle}$.

<u>Evaluation</u>

Evaluation is a relationship between meaningful expressions and models. We define recursively the value or <u>denotation</u> $d[B;M,i,j,g]$ of a meaningful expression B with respect to a model M and i, j, and g, where M is a model on A and IxJ, $i \in I$, $j \in J$, and g is a variable assignment. g assigns to each variable of type <u>a</u> an element of D_a.

1) If B is a constant, then $d[B;M,i,j,g]$ is $F(B)(\langle i,j\rangle)$.

2) If B is a variable, then $d[B;M,i,j,g]$ is $g(B)$.

3) If $B \in ME_a$ and V is a variable of type <u>b</u>, then d[(LAMBDA

V B);M,i,j,g] is that function h with domain D_b such that whenever x is in that domain, h(x) is d[B;M,i,j,g'], where g' is a variable assignment like g except for the possible difference that g'(V) is x.

4) If B ε ME_<a,b> and C ε ME_a, then d[(B C);M,i,j,g] is the value of the function d[B;M,i,j,g] for the argument d[C;M,i,j,g].

5) If B ε ME_<d,<c,b>>, C ε ME_c and D ε ME_d, then d[(B C D);M,i,j,g] is d[((B D) C);M,i,j,g].

6) If B, C ε ME_a, d[(EQUAL B C);M,i,j,g] is 1 if and only if d[B;M,i,j,g] is d[C;M,i,j,g].

7) If P, Q ε ME_TS, then d[(NOT P);M,i,j,g] is 1 if and only if d[P;M,i,j,g] is Similarly for AND, OR, IMPLIES, IFF.

8) If P ε ME_TS, then d[(NECESSARILY P);M,i,j,g] is 1 if and only if d[P;M,i',j',g] is 1 for all i' ε I and j' ε J. d[(FUTURE P);M,i,j,g] is 1 if and only if d[P;M,i,j',g] is 1 for some j' such that j<j'. d[(PAST P);M,i,j,g] is 1 if and only if d[P;M,i,j',g] is 1 for some j' such that j'<j.

9) If P ε ME_TS and V is a variable of type a, then d[(THERE-IS V P);M,i,j,g] is 1 if and only if there exists x ε D_a such that d[B;M,i,j,g'] is 1, where g' is as in 3. Similarly for d[(FOR-ALL V P);M,i,j,g].

10) If B ε ME_a, then d[(INT B);M,i,j,g] is that function h with domain IxJ such that whenever <i',j'> ε IxJ,

h(<i',j'>) is d[B;M,i',j',g].

11) If B ∈ ME_<S,a>, then d[(EXT B);M,i,j,g] is d[B;M,i,j,g](<i,j>).

Standard Models

A <u>standard frame based on A and S</u> is a frame based on A and S in which all possible functions occur in the denotation-sets, that is, $D_{\langle a,b\rangle} = D_b^{D_a}$ and $D_{\langle S,a\rangle} = D_a^{D_S}$. A <u>standard model</u> is a model based on a standard frame.

General Models

In a general model (g-model), the denotation-sets of the frame are less constrained. As above, D_S = A and D_TS = {0,1}. However, for a complex type <a,b> we have $\emptyset \neq D_{\langle a,b\rangle} \subsetneq D_b^{D_a}$. It is further required that the denotations for all expressions be in the model. This means in effect that the functions h which are the denotations of LAMBDA-expressions and INT-expressions must exist in the appropriate denotation-sets. The definition of evaluation in a g-model is exactly as in a standard model. Note, however, that LAMBDA-abstraction and quantification are restricted to the denotation-sets of the model.

Named Models

The models used in the computer system do not correspond exactly to the standard or general models of formal logic. We therefore introduce a new formal definition of 'named models'. This definition allows a model that may be smaller than a g-model, but expands toward a g-model when new functions are required.

A named model (n-model) is based on a frame which need not be closed under valuation. The valuation function d is a partial function, undefined where the valuation function as defined above takes a value not in the model. $D_{\langle a,b \rangle} \subseteq D_b^{D_a}$ and every element has a unique name. A named model need not be a g-model; however, a 'covering g-model' must exist. The covering g-model is the closure of the n-model under the condition that the functions required for evaluation exist. LAMBDA-abstraction and quantification in an n model are restricted to the named elements in the denotation-sets.

Dynamic Named Models

The models used in our interpretation system are dynamic n-models. They are not closed under the valuation function; however, whenever a denotation of type <a,b> is created by evaluation of an expression with LAMBDA or INT, the dynamic n-model is immediately expanded. The function is named, added to the appropriate denotation-set, and all functions with this denotation-set as domain are expanded to include values for the new argument. LAMBDA-abstraction and quantification in a dynamic n-model are restricted to the current named elements in the denotation-sets. This has the consequence that they cannot be precisely defined independently of the order of evaluation of a formula.

A dynamic n-model may be thought of either as one expanding n-model or as a sequence of n-models. It is not yet clear which view will be most productive.

Finite Models

While the intensional logic has infinitely many types, only a finite number occur in expressions of PTQ and thus only a finite number of sets of possible denotations are needed. In the current version of the interpretation system the sets A and IxJ, and hence all denotation-sets, are finite. In the finite case the notions of standard model and g-model are the same, under some weak conditions. We have introduced the distinction because we think of our dynamic models as 'potentially infinite', and find the notion of covering g-model suggestive. One might think of the g-model as a representation of 'reality' and an n-model as a finite representation of the system's knowledge. A dynamic n-model is a representation that is forced to expand toward reality as the system needs further knowledge to carry out its tasks. We plan elsewhere to consider these models further, both formally and as possible psychological models.

C. Models in the System

This system accepts only finite models. A finite model can reveal the principles involved in interpretation of formulas, and most of the interesting problems in the evaluation of formulas can arise there. In using the system in research, we have encountered no problems related to size. However, even a small standard or general model is too large to be practical. The closure property of these models causes the inclusion of elements that may not be needed. Dynamic named models are practical. They need include only those elements whose use is anticipated, and elements can be added to the model to meet new or unanticipated uses.

The functions in standard and general models are total. The functions in named models are also total, although they may be incompletely (partially) specified, that is, the value of the function may be given for only some of the arguments. The unspecified values for a function are regarded as determined, but unknown to the system. If the interpretation of an expression causes a function to be applied to an argument for which its value is unspecified, the interpretation is suspended and the user is prompted for the needed value. This approach using partially-specified total functions and dynamic models contrasts with approaches using partial functions in static models

(Kutschera, 1975).

In a named model, a D_set is a subset of the set of all possible functions of its type. With a dynamic model, the user can enter a function of type <a,b> that has a value not in D_b; this causes the element to be added to D_b. Similarly, the addition of an element to D_a causes the expansion of the specifications of all current functions of type <a,b>.

In a dynamic model, it may be reasonable to have two functions with the same specification; an as yet unspecified value may be different for the two functions, or they may have different values for some argument not yet in the model.

The denotation of a LAMBDA-expression or an INT-expression is a function and if that function is not an element of the model it is added. If an element is added to D_a during the interpretation of a LAMBDA-expression of type <a,b> or an argument to which it is applied, the body of the LAMBDA-expression is interpreted for this new element and the specification of the denotation of the LAMBDA-expression is expanded to include this argument and its computed value.

This ability to expand the model dynamically during the interpretation of formulas gives the user a second means of constructing models: starting with a minimal outline of the model, the user interprets expressions describing the desired model. When prompted for unspecified values, the user can respond with entries that will make the expression true.

The model can also be expanded under the direct control of the user. Elements can be added to the model or unspecified values of functions can be entered.

III. IMPLEMENTATION OF NAMED MODELS

A. Names for Types and D sets

Semantic types play a significant role in the interactions between the user and the system, but the names of complex types as constructed in PTQ are cumbersome. For example, words in the syntactic category IAV/TE (e.g. 'in' and 'about') are translated into logical constants of type:

f(IAV/TE) = <<s,<<s,<<s,e>,t>>,t>>,<<s,<<s,e>,t>>,<<s,e>,t>>>

To facilitate interaction, the system uses new type names that are meaningful, simple, and easily distinguished from each other. The convention is to use the names of the syntactic categories as names for the corresponding semantic types, i.e., for syntactic category x, the name of the corresponding type, f(x), will also be x. This can be done here without confusion because the syntactic categories do not play any part in this system. Where PTQ uses special symbols, e.g., IV, IV, for the compound syntactic categories, TS/B, IV/TE, we also use these special symbols. However, there are types that do not correspond to syntactic categories, e.g., the types for the intensions of the types that correspond to syntactic categories; the name we use for one of these is the combination of the names for its component types, e.g.,.<S,IAV/TE>.

It is possible to have several names for the same type, for example, f(CN) = f(IV). In such a case, another (neutral) name is needed - the convention is that this name is formed by combining the types using an equal sign as a separator (e.g., CN=IV). Type names such as CN and IV will be referred to as <u>subtype names</u> when it is necessary to distinguish them from the other type names (e.g., CN=IV or E). The user can usually refer to a type using either a subtype or type name (in the case of f(CN) = f(IV), by CN, IV or CN=IV), whichever is most natural.

When the user is prompted for a type, the preferred response is one of these new type names. However, the user may enter any equivalent form. For example, IAV=IV/IV, <<S,IV>,IV>, and <<S,<<S,F>,TS>>,<<S,I>,TS>> all name the same type. Since a type and its subtypes all have the same components, these equivalent forms refer to the type.

Names are also assigned to the sets of possible denotations of each type. The name for a set of possible denotations of a type is formed by concatenating 'D_' and the name for that type. The full type name must be used here, not a subtype name. That is, the name D_CN is not recognized by the system; it expects D_CN=IV instead.

B. Built-In Types, Logical Constants, and Variable Prefixes

The system contains the tensed intensional logic used in PTQ. There is also an extensional version without intensionality, modality, or tense; we do not discuss it in this paper. The types are those occurring in formulas that are translations of English sentences or are meaning postulates. The types and subtypes for intensional models are:

```
S
E
<S,E>
CN=IV:  <<S,E>,TS>
    SUBTYPES:  CN  IV
<S,CN=IV>:  <S,<<S,E>,TS>>
    SUBTYPES:  <S,CN>  <S,IV>
<E,TS>
<S,<E,TS>>
IAV=IV/IV:  <<S,<<S,E>,TS>>,<<S,E>,TS>>
    SUBTYPES:  IAV  IV/IV
<S,IAV=IV/IV>:  <S,<<S,<<S,E>,TS>>,<<S,E>,TS>>>
    SUBTYPES:  <S,IAV>  <S,IV/IV>
<<E,TS>,<E,TS>>
<S,<<E,TS>,<E,TS>>>
TE:  <<S,<<S,E>,TS>>,TS>
<S,TE>:  <S,<<S,<<S,E>,TS>>,TS>>
TV:  <<S,<<S,<<S,E>,TS>>,TS>>,<<S,E>,TS>>
<S,TV>:  <S,<<S,<<S,<<S,E>,TS>>,TS>>,<<S,E>,TS>>>
<E,<E,TS>>
<S,<E,<E,TS>>>
IAV/TE:  <<S,<<S,<<S,E>,TS>>,TS>>,<<S,<<S,E>,TS>>,<<S,E>,TS>>>
<S,IAV/TE>:  <S,<<S,<<S,<<S,E>,TS>>,TS>>,<<S,<<S,E>,TS>>,
               <<S,E>,TS>>>>
<E,<<E,TS>,<E,TS>>>
<S,<E,<<E,TS>,<E,TS>>>>
TS
```

```
<S,TS>
IV/TS:  <<S,TS>,<<S,E>,TS>>
<S,IV/TS>:  <S,<<S,TS>,<<S,E>,TS>>>
<TS,<E,TS>>
<S,<TS,<E,TS>>>
TS/TS:  <<S,TS>,TS>
<S,TS/TS>:  <S,<<S,TS>,TS>>
```

The logical constants are formed by capitalizing the words of which they are the translations. For example, the word "walk" translates to the constant "WALK". The logical constants are:

```
TYPE E: J, M, B, N.
TYPE CN=IV:
  SUBTYPE CN: MAN, WOMAN, PARK, FISH, PEN, UNICORN, PRICE,
              TEMPERATURE.
  SUBTYPE IV: RUN, WALK, TALK, RISE, CHANGE.
TYPE IAV=IV/IV:
  SUBTYPE IAV: RAPIDLY, SLOWLY, VOLUNTARILY, ALLEGEDLY.
  SUBTYPE IV/IV: TRY-TO, WISH-TO
TYPE TV: FIND, LOSE, EAT, LOVE, DATE, SEEK, CONCEIVE.
TYPE IAV/TE: IN, ABOUT.
TYPE IV/TS: BELIEVE-THAT, ASSERT-THAT.
```

To permit an unrestricted number of variables, certain letters are designated as variable prefixes, and a variable of type <u>a</u> is a variable prefix of type <u>a</u> followed by zero or more digits. The variable prefixes are:

```
TYPE E: U, V.
TYPE <S,E>: X, Y.
TYPE <S,CN=IV>: P, Q.
TYPE <S,<E,TS>>: R.
TYPE <S,TE>: B.
TYPE <S,<E,<E,TS>>>: S.
TYPE <S,<E,<<E,TS>,<E,TS>>>>: G.
TYPE <S,TS>: K.
```

Changes to the built-in logical constants and variable prefixes can be accomplished with the commands in Section V.

The logic is built into the system with a series of declarations giving the types, logical constants, and variable prefixes, and an ordering of the types. This ordering, which need not contain all the types, gives the sequence in which the system will prompt the user to enter the sets of possible denotations. The built-in logic can easily be modified or replaced by changing these declarations (Moran, N-7, forthcoming)

C. <u>Sets of Possible Denotations of Complex Type</u>

A set of possible denotations of a complex type <a,b> is a set of functions of type <a,b>. Each function is entered by giving its name and specification. Even though functions are understood to be total, functions can be entered with partial specifications. For each set of functions, the program prints:

1) the name of the set D_<a,b>,
2) the name of the set D_a that is the domain, and the names of elements in the domain.
3) the name of the set D_b that is the range, and the names of elements in the range. These are the names to be used as values in entering the function.
4) the logical constants whose possible denotations or denotations of their intensions are in this set.

Following this preliminary information, the program asks for the name for the first element to be entered. This name can be chosen almost arbitrarily. There are a few illegal names, but they are refused by the system. The following cannot be used: the name of another element of any set; the name of any of the model components (e.g. 'ENTITIES', 'F'); the name for a type, a set of possible denotations, or a point of reference; the name of a command; a name beginning with the character '='; a number or a system atom (T, NIL).

After the name for an element is entered, the program leads the user through its specification, and then asks for the name of the next element. To indicate the end of a set of elements, the user responds with NIL. When the specification of a set is terminated, the program displays its elements and then requests specification of the next set.

```
>    ****ENTERING ELEMENTS OF D_<S,E> - INDIVIDUAL CONCEPTS
>    THEY ARE FUNCTIONS FROM D_S TO D_E,
>    THAT IS, FROM POINTS OF REFERENCE TO ENTITIES.
>    THEY WILL BE THE VALUES OF F FOR THE LOGICAL CONSTANTS:
>    J, M, B, N.
>
>    D_S = {I1, I2}
>    D_E = {JO, UN}
>
>    ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*a0
```

D. Specification of Functions

After the name for a function has been entered, the program guides the user through its specification. For each element in the domain, the program prints its name and its specification and the user responds with the name of the corresponding value.

If the function is to be unspecified for this argument, the user enters NIL. Any element entered as a value should already be in the model. However, if an element that is not yet in the model is needed, the user may enter the name intended for that element. The program detects this as a potential error and asks the user for instructions (see F below).

```
>    ENTER NAME FOR ELEMENT. (NIL = NO MORE)
*a2
>    ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>    I1 = <HERE,THEN>
*jo
>    I2 = <HERE,NOW>
*jo
>         A2 ENTERED.
```

```
>    ****D_<S,E> - INDIVIDUAL CONCEPTS
>      DOMAIN:  D_S = {I1, I2}
>      RANGE:   D_E = {JO, UN}
>           A2 = {<I1,JO>, <I2,JO>}
>           A3 = {<I1,UN>, <I2,UN>}
```

When the specification of a function is completed, the model is checked or an equivalent function. In such a situation, there would be two names for the same element and, in functions for which this element is an argument, a value will be specified for each name. If these values are not the same, the interpretation of an expression can depend on the name used. To prevent this, one of the equivalent elements should be deleted or modified.

E. Functions Used Before They Are Specified

During the specification of a function, the value for a particular argument is entered by giving the name of an element of the range. If the user enters a name which is not in the model, the program asks for clarification. If the user chooses to specify the element immmediately, the specification of the function is suspended while it is entered. For example:

```
>    ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*C0
>    ENTER A VALUE OF NIL FOR EACH ARGUMENT:
>    I1 = <HERE,THEN>
*b0
>    I2 = <HERE,NOW>
*b1
>    NO SUCH ELEMENT.  EXPLAIN.
>     (1-WRONG NAME; 2-WILL BE ENTERED LATER; 3-ENTER NOW;
>      4-ENTER FUNCTION AS UNSPECIFIED FOR ALL ARGUMENTS).
*3
>    THIS ELEMENT OF D_CN=IV IS A FUNCTION FROM D_<S,E> TO D_TS,
>    THAT IS, FROM INDIVIDUAL CONCEPTS TO TRUTH VALUES.
>
>    D_<S,E> = {A0, A3}
>    D_TS = {0, 1}
>
>    ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>    A0 = {<I1,JO>, <I2,JO>}
*0
>    A3 = {<I1,UN>, <I2,UN>}
*1
>         B1 ENTERED.
>
>    ****D_CN=IV - SETS OF INDIVIDUAL CONCEPTS
>      DOMAIN:  D_<S,E> = {A0, A3}
>      RANGE:   D_TS = {0, 1}
>         B0 = {}
>         B15 = {A   A3}
>         B1 = {A3}
>
>    RESUME SPECIFICATION OF C0:
>
>         C0 ENTERED.
```

If the user chooses to delay the entry of the new element, the specification of the function continues uninterrupted. When the entry of functions into a set is finished, the user is prompted to enter the specifications of the deferred elements.

```
>    ****ENTERING UNSPECIFIED ELEMENTS B10, B5.
>
>    THESE ELEMENTS OF D_CN=IV ARE FUNCTIONS FROM D_<S,E> TO D_TS,
>    THAT IS, FROM INDIVIDUAL CONCEPTS TO TRUTH VALUES.
>
>    D_<S,E> = {A0, A3}
>    D_TS = {0, 1}
>
>    ENTERING B10
>    ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>    A0 = {<I1,JO>, <I2,JO>}
*1
>    A3 = {<I1,UN>, <I2,UN>}
*0
>        B10 ENTERED.
>
>    ENTERING B5
>    ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>    A0 = {<I1,JO>, <I2,JO>}
*0
>    A3 = {<I1,UN>, <I2,UN>}
*1
>        B5 ENTERED.
>    **WARNING - B5 IS THE SAME AS B1
>
>
>    ****D_CN=IV - SETS OF INDIVIDUAL CONCEPTS
>      DOMAIN:  D_<S,E> = {A0, A3}
>      RANGE:   D_TS = {0, 1}
>           B0 = {}
>           B15 = {A0, A3}
>           B1 = {A3}
>           B10 = {A0}
>           B5 = {A3}
>
>    **WARNING - D_CN=IV CONTAINS EQUIVALENT ELEMENTS.
>    EQUIVALENT ELEMENTS: B1, B5.
```

F. New Elements in a Domain

If in entering a model or adding to one, the user enters elements into a set which is the domain of some previously specified function, the specification of that function will need to be expanded to include the new elements. The system will suspend its current project and prompt for this expansion.

```
>    EXPANDING ELEMENTS IN D_CN=IV = {SET1-WALKERS, SET2-WALKERS,
>   SET-MEN}
>    THEY ARE FUNCTIONS FROM D_<S,E> TO D_TS,
>    THAT IS, FROM INDIVIDUAL CONCEPTS TO TRUTH VALUES.
>
>    D_<S,E> = {IC-JO, IC-MA, IC-BI, IC-NA}
>    D_TS = {0, 1}
>
>    NEW ARGUMENTS IN D_<S,E>:  IC-BI, IC-NA.
>    FOR EACH OF THE FOLLOWING FUNCTIONS, ENTER A VALUE
>    OR NIL FOR EACH OF THE NEW ARGUMENTS.
>
>    EXPANDING SPECIFICATION OF SET1-WALKERS = {IC-JO}
>    IC-BI = {<I1,BI>, <I2,BI>}
*0
>    IC-NA = {<I1,NA>, <I2,NA>}
*1
>
>    EXPANDING SPECIFICATION OF SET2-WALKERS = {IC-MA}
>    IC-BI = {<I1,BI>, <I2,BI>}
*1
>    IC-NA = {<I1,NA>, <I2,NA>}
*0
>
>    EXPANDING SPECIFICATION OF SET-MEN = {IC-JO}
>    IC-BI = {<I1,BI>, <I2,BI>}
*1
>    IC-NA = {<I1,NA>, <I2,NA>}
*0
>
>    ****D_CN=IV - SETS OF INDIVIDUAL CONCEPTS
>      DOMAIN:  D_<S,E> = {IC-JO, IC-MA, IC-BI, IC-NA}
>      RANGE:   D_TS = {0, 1}
>           SET1-WALKERS = {IC-JO, IC-NA}
>           SET2-WALKERS = {IC-MA, IC-BI}
>           SET-MEN = {IC-JO, IC-BI}
```

G. The Meaning Function F

In a model, the meaning function F assigns to each logical constant a function that is the denotation of the intension of that constant. If the constant is of type a, the value of F is of type <S,a>. After entering the set D_<S,a>, the user is prompted to enter the value of F for each of the constants of type a.

```
>    ****ENTERING THE FUNCTION F  FOR LOGICAL CONSTANTS OF TYPE E
>    THE VALUES OF  F  FOR THESE CONSTANTS ARE ELEMENTS OF
>   D_<S,E> (INDIVIDUAL CONCEPTS).
>
>    LOGICAL CONSTANTS: J, M, B, N.
>    D_<S,E> = {A1, A3}
>
>    FOR EACH CONSTANT, ENTER THE VALUE OF F OR NIL:
>    J
*a1
>    M
*a3
>    B
*nil
>    N
*nil
```

No check is made that a constant satisfies any meaning postulates, since it can be interesting to investigate both models that do and do not satisfy them. The user can find out whether a meaning postulate is satisfied by interpreting it in the model, as is done, for example, in Appendix C.

IV. INTERPRETATION IN A MODEL

An intensional model is used in evaluating meaningful expressions of intensional logic. These expressions may be written directly or they may be obtained using other related computer programs. For each English sentence our parsing system produces the set of derivation trees that give the structure of the sentence. Our translator applies the rules T1-T17 of PTQ to obtain a meaningful expression in intensional logic. The expression obtained is the 'direct translation' of the parse tree; a normal form is obtained by application of lambda-reduction and extension-intension removal. The interpretation system works for any of these meaningful expressions.

Meaningful expressions are interpreted with respect to the dynamic n-model and two parameters: a point of reference and a variable assignment G, whose initial values are provided by the user.

During the interpretation of a meaningful expression, the interpretation of each of its sub expressions is printed Each expression and the current values of the two parameters are printed as evaluation begins, and the expression and its denotation are printed when the evaluation is completed.

```
*(walk x3)
>   EXPRESSION IS OF TYPE IS
>   FREE VARIABLES:  X3.
>   OK?
*go
```

```
>
>   ENTER POINT OF REFERENCE (NIL = NO MORE):
*i2
>   ENTER A VALUE FOR EACH VARIABLE (NIL =CANCEL):
>   X3
*a0
>
>   INITIAL VARIABLE ASSIGNMENT G1: X3=A0.
>   COMPUTING DENOTATION OF  (WALK X3)  FOR I2 AND G1
>   :  DENOTATION OF  WALK  IS  B10 = {A0}
>   :  DENOTATION OF  X3  IS  A0 = {<I1,JO>, <I2,JO>}
>   DENOTATION IS  1
```

The interpretation of a LAMBDA-expression or a quantified expression requires a series of new variable assignments, one for each possible denotation of the bound variable. These variable assignments are generated by the system and are just like the previous assignment with the exception of the assignment for the bound variable. The system also generates a name Gi for these assignments.

Each function in the model is total, but it may be only partially specified. If a function is applied to an argument for which its value is unspecified (NIL), the system suspends the interpretation and prompts the user for the value. Similarly, if the meaning function F is unspecified for a constant in the expression, the system prompts the user for the value.

```
>   :   :   :   COMPUTING DENOTATION OF  (MAN X)   FOR I1 AND G5
>   :   :   :   :  DENOTATION OF  MAN. IS  SET-MEN = {IC-JO}
>   :   :   :   :  DENOTATION OF  X  IS  IC-MA = {<I1,MA>, <I2,MA>}
>   :   :   :   :| THE VALUE OF SET-MEN
>   :   :   :   :|IS UNSPECIFIED FOR THE ARGUMENT IC-MA
>   :   :   :   :| THE POSSIBLE VALUES ARE:
>   :   :   :   :| D_TS = {0; 1}
>   :   :   :   :|
>   :   :   :   :| ENTER THE VALUE OF SET-MEN FOR:
```

```
>   :  :  :  :| IC-MA = {<I1,MA>, <I2,MA>}
*C
>   :  :  :  :|        SET-MEN = {IC-JO}
>   :  :  :  :| DONE - SET-MEN RESET.
>   :  :  :  :|
>   :  :  :  DENOTATION IS
```

Two functions are EQUAL only if they have the same name or they have the same full specification. Two functions with the same partial specification cannot be said to be equal because they may differ in some of their as yet unspecified values. In testing the equality of partially specified functions, the system prompts the user for each of the unspecified values until the specifications are found to be different. The user may respond to a prompt for a value by leaving it unspecified, in which case the system may prompt for the value later. This allows the user to avoid specifying a value for a particular argument when the functions are going to be unequal for some later argument.

```
>   INITIAL VARIABLE ASSIGNMENT G1: Y=IC2, X=IC1.
>   COMPUTING DENOTATION OF  (EQUAL X Y)  FOR I1 AND G1
>   :  DENOTATION OF  X  IS  IC1 = {<I1,NIL>, <I2,JO>, <I3,NIL>}
>   :  DENOTATION OF  Y  IS  IC2 = {<I1,MA>, <I2,JO>, <I3,MA>}
>   :  IC1 IS UNSPECIFIED FOR ARGUMENTS:  I1, I3.
>   :
>   :  THIS ELEMENT OF D<S,E> IS A FUNCTION FROM DS TO DE,
>   :  THAT IS, FROM POINTS OF REFERENCE TO ENTITIES.
>   :
>   :  DS = {I1, I2, I3}
>   :  DE = {JO, MA}
>   :
>   :  ***COMPLETING SPECIFICATION OF IC1
>   :  ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>   :  I1 = <1,1>
*ma
>   :  I3 = <3,1>
*jo
>   DENOTATION IS  1
```

Interpretation of an expression beginning with FOR-ALL, THERE-IS, NECESSARILY, FUTURE, or PAST calls for interpretation of its body with respect to a series of variable assignments or points of reference. Interpretation of the expression stops as soon as the value of the expression is determined. This is also true for expressions with the logical connectives AND, OR, or IMPLIES.

Interpretation of LAMBDA-expressions and INT-expressions can add new elements to the model. The denotation of a LAMBDA-expression, such as (LAMBDA X (WALK X)), is a function whose arguments are the possible denotations that can be assigned to the variable, X, and whose values are found by interpreting the body of the expression, (WALK X), for those assignments to the variable. If this function is not already an element of the model, the user is asked to name it and the new element is then added to the model. Denotations for INT-expressions are generated in the same manner, with the points of reference as the domain of the function

```
>   :  COMPUTING DENOTATION OF  (LAMBDA P ((EXT P) (INT J)))
>   :  FOR I1 AND G1


>   :  :. DENOTATION OF  (LAMBDA P ((EXT P) (INT J)))  IS {C1}
>   :  :  BUT IS NOT AN ELEMENT IN THE MODEL.
>   :  :  ENTER A NAME FOR THIS ELEMENT IN D_TE:
*j*1
>   :  DENOTATION IS  J*1 = {C1}
```

New elements can be added to the model during interpretation by being given as the previously unspecified value of a function or by being the interpretation of a LAMBDA-expression or an INT-expression. As part of the addition of a new element to the model, the system prompts the user to expand the specifications of the functions to whose domain the element has been added. However, the addition of an element may cause the expansion of a function that is the denotation of a LAMBDA-expression. Rather than assuming that the user will correctly expand the denotation of the LAMBDA-expression, the system first has the user expand all the functions in the model, and then it expands the LAMBDA-expression by interpreting its body for the new denotation of the LAMBDA-variable, and adds the denotation to the model if it is new.

In interpreting a LAMBDA-expression or a quantified expression, there may be no possible denotations for the bound variable, i.e. the D_set is empty. The system assumes that the user intended some as yet unspecified function to be in that D_set and allows the user to name that function. This function is entered as being unspecified for each of its arguments; the user is prompted for values as they are needed. The user may choose to leave the D_set empty, in which case the denotation of the expression is true for universal quantification, false for existential quantification, and a function with an empty domain for a LAMBDA-expression. For a LAMBDA-expression that is applied to an argument, this latter course has the effect of delaying the interpretation of the LAMBDA-expression until after the interpretation of its argument.

```
>   COMPUTING DENOTATION OF  (THERE-IS E (FOR-ALL X (NECESSARILY
>  (IFF (WALK X) ((EXT E) (EXT X))))))  FOR I2 AND G1
>   : THERE ARE NO POSSIBLE DENOTATIONS FOR THE BOUND VARIABLE
>   : F.
>   :  UNSPECIFIED FUNCTION BEING ADDED TO D_<S,<E,TS>>.
>   : ENTER NAME FOR THIS ELEMENT (NIL = DON'T ADD):
*prop-walk*
>   :  *NEW VARIABLE ASSIGNMENT G2: F=*PROP-WALK*.
>   :  COMPUTING DENOTATION OF  (FOR-ALL X (NECESSARILY (IFF (
>   : WALK X) ((EXT E) (EXT X)))))  FOR I2 AND G2
```

At first consideration, this treatment of LAMBDA-expressions seems inefficient; it would seem more efficient to interpret the body of the LAMBDA-expression only for the argument to which it is applied. However, the logic includes meaningful expressions in which a LAMBDA-expression is <u>not</u> applied to an argument. It is also possible for the argument of a LAMBDA-expression to be another LAMBDA-expression. Meaningful expressions of both these forms are generated in PTQ. Furthermore, even where this simplification is applicable, this simplification is a syntactic reduction and should be performed before the expression is interpreted. Therefore, when interpreting a LAMBDA-expression applied to an argument the system assumes that the user does not want this simplification.

V. USING THE SYSTEM

A. Introduction

This system has a simple, flexible command language which supports the needs of both the novice and the experienced user. The novice can follow a direct path through the system, simply responding to prompts as they are given. With the prompts he is reminded of the relevant past entries that form the context for his current decisions. The prompts follow an order from simple to more complex so that the building blocks are always available when they are needed. The system checks all responses for errors, and if one is found, the user is advised and allowed to make a correct entry.

The advanced user need not follow the direct path. The order of entry of a model can be varied to correspond more naturally to the problem at hand. The model can also be changed in order to extend the model, o correct errors, or to explore alternatives. An important and extremely useful feature of the interaction is that the user may interleave model-building and formula-interpretation. The specification of a function may be deferred until the function is actually needed in the interpretation of a formula.

B. Interacting with the Program

The commands in this system are LISP function calls; the command and its operands (if any) are enclosed in parentheses:

```
(INTERP)
(ENTER MODEL)
(DISPLAY D_SETS)
```

The system is implemented in MTS/LISP and is currently available only at the University of Michigan. To use this system, run the LISP interpreter and use the function RESTORE to load the program from the file SHEF:INTERP

```
#$run *lisp
*(restore shef:interp)
```

Normally in MTS/LISP angle brackets are super-parentheses and commas are separators between elements in lists. However, we use these characters in type names, e.g., '<S,E>', so they have been redefined as normal characters.

C. Entering a Model

The entry of the model follows the order implicit in its definition. First the sets A of entities, I of possible worlds, and J of moments in time are entered. From the possible worlds and moments in time the system generates the set of points of reference IxJ. The sets A and IxJ determine the frame for a standard model.

The models in this system are named models. The D_sets for the elementary types are the same as in a standard model. Each D_set of a complex type is entered by the user with names for each function entered in the set. Interleaved with the entry of the D_sets is the specification of the meaning function F; the values of F for logical constants of a type are specified as soon as the D_set containing the possible values is entered.

The entry of a model is initiated with the command (ENTER MODEL). The system first asks whether an intensional or extensional model is desired. The user is then given the opportunity to change the built-in logical constants before the specification of F begins. Next the user is prompted for the lists of entities, possible worlds, and moments in time. The elements in these lists can be numbers or strings of characters, i.e. LISP atomic objects. Because commas are used in type names, they cannot be used as separators between elements in lists. The moments in time are entered in order, earliest to latest.

```
*(enter model)
>   DO YOU WANT AN INTENSIONAL MODEL?
*y
>   DO YOU WISH TO CHANGE THE DEFAULT LOGICAL CONSTANTS?
*n
>
>   REMINDER: COMMAS CANNOT BE USED AS SEPARATORS IN LISTS.
>   ENTER LIST OF ENTITIES.
*(jo un)
>   ENTER LIST OF POSSIBLE WORLDS.
*(here)
>   ENTER LIST OF MOMENTS IN TIME, IN INCREASING ORDER.
*(then now)
```

```
>   POINTS OF REFERENCE (INDICES):
>     I1 = <HERE,THEN>
>     I2 = <HERE,NOW>
>
>
>   ****ENTERING ELEMENTS OF D_<S,E> - INDIVIDUAL CONCEPTS
```

The entry of the sets of possible denotations is ordered so that before a D_set is entered the D_sets which are its domain and its range are entered. This ordering is built into the system as part of the type specification. This ordering also causes the system not to prompt the user for D_sets that have empty domains. Neither will the system prompt the user for any D_set not in this ordering.

The system can contain only one model at a time. If the system already contains a model, the user must delete it, using the command

(DELETE MODEL)

before entering the new model.

D. Interpreting Meaningful Expressions

The command INTERP starts the interpretation of meaningful expressions. The user is prompted for a meaningful expression and then for a point of reference (if using an intensional model) and an initial variable assignment (if the expression contains free variables). The system interprets the meaningful expression with respect to the current model and the given point of reference and variable assignment. The user is then prompted for another point of reference and variable assignment. The user indicates the end of the interpretations of an expression by entering NIL in place of a point of reference. When the interpretation of an expression is completed, the user is prompted for another expression. When there are no more expressions to be interpreted, the user enters NIL to terminate the execution of the INTERP command.

When an expression is entered, the system prints its type and its free variables and then asks for confirmation that this expression is the one intended. For each interpretation of the expression, the initial variable assignment G1 is obtained by prompting the user for the denotation of each free variable.

The following example gives one possible interaction during the initial stages of evaluating a meaningful expression:

```
    ENTER MEANINGFUL EXPRESSION (NIL = NO MORE):
*(walk x3)
>   EXPRESSION IS OF TYPE TS
>   FREE VARIABLES:  X3.
>   OK?
*go
>
>   ENTER POINT OF REFERENCE (NIL = NO MORE):
*i2
>   ENTER A VALUE FOR EACH VARIABLE (NIL =CANCEL):
>   X3
*a0
>
>   INITIAL VARIABLE ASSIGNMENT G1: X3=A0.
```

The standard method of entering a meaningful expression is to give it explicitly, as shown above. Two additional methods are also available.

A second method of giving a meaningful expression is to enter PREVIOUS thereby making the previous expression the current one. This allows the user to get out of INTERP, change the model, and then repeat the interpretation of the same expression. Also, if the expression is syntactically incorrect because of an undeclared logical constant or variable prefix the user can make the needed addition and then re-interpret the expression. The previous expression used by INTERP can also be set by the auxiliary function TYPE_OF_EXP (See Section J).

A third method involves using the LISP system directly. Between commands, a knowledgeable user can create and name expressions using LISP. These expressions can be entered during INTERP by giving their names. Named expressions are useful for dealing with larger expressions; expressions can be built up from sub-expressions using the standard LISP functions, or they can be modifications of other expressions (e.g. by replacing "$EEK" with "FIND"). Named expressions also can be easily decomposed into their sub-expressions. This is useful for verifying that a change in the model produces the intended change in the interpretation of that sub-expression. Names for expressions cannot be anything that could be mistaken for an expression, that is, the name cannot be a logical constant or a variable.

```
*(setq mp1-j (quote (there-is u (necessarily (equal u j)))))
```

```
>    (THERE-IS U (NECESSARILY (EQUAL U J)))
*(interp)
>
>    ENTER MEANINGFUL EXPRESSION (NIL = NO MORE)
*mp1-j
>    (THERE-IS U (NECESSARILY (EQUAL U J)))
>    EXPRESSION IS OF TYPE TS
>    FREE VARIABLES:  NONE
>    OK?
```

E. Saving and Restoring a Model

MTS/LISP includes a facility for saving and restoring the internal form of a data structure. The current model is a list named MODEL; it can be saved for later use with the command

(CHECKPOINT model.filename MODEL)

This command will terminate the execution of the program, and therefore only complete models should be saved. When the model is later restored, changes can be made only with the commands in the following sections.

After restoring the program, the user can restore a saved model with the command

(RESTORE model.filename)

If the system already contains a model, that model must be deleted with the command

(DELETE MODEL)

before the saved model is restored because restoring a model does not completely replace a previous model.

F. Changing the Model

The model can be changed by adding or deleting functions or by modifying the specifications of possible denotations or by modifying the specification of the meaning function F. The set of types, and the sets of entities, possible worlds, moments in time, and points of reference cannot be changed.

Addition of functions to the model is initiated by the command ADD (or ENTER) with the operand D_SETS (or D_SET, FUNCTIONS, FUNCTION, DENOTATIONS, or DENOTATION). The system prompt the user for the type of the functions to be entered and then follows the prompting sequence used in the initial entry of functions into that D_set. The user is then prompted for the type of the next group of functions to be entered and this sequence is repeated until all the new functions have been entered

```
*(add function)
>   ENTER TYPE OF ELEMENTS TO BE ADDED (NIL = NO MORE):
*<s,e>
>
>   ****ENTERING ELEMENTS OF D_<S,E> - INDIVIDUAL CONCEPTS
>   THEY ARE FUNCTIONS FROM D_S TO D_E,
>   THAT IS, FROM POINTS OF REFERENCE TO ENTITIES.
>   THEY WILL BE THE VALUES OF F FOR THE LOGICAL CONSTANTS:
>   J, M, B, N.
>
>   D_S = {I1, I2}
>   D_E = {JO, MA, BI, NA}
>
>   ENTER NAME FOR ELEMENT.. (NIL = NO MORE).
*ic-ma
>   ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>   I1 = <1,1>
*ma
>   I2 = <1,2>
```

```
*ma
>       IC-MA ENTERED.
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*nil
>   ****D_<S,E> - INDIVIDUAL CONCEPTS
>     DOMAIN:   D_S = {I1, I2}
>     RANGE:    D_E = {JO, MA, BI, NA}
>         IC-JO = {<I1,JO>, <I2,JO>}
>         IC-MA = {<I1,MA>, <I2,MA>}
>
>   ENTER TYPE OF ELEMENTS TO BE ADDED (NIL = NO MORE):
*nil
```

Functions are deleted from the model with the command DELETE and the operand FUNCTIONS (or FUNCTION, DENOTATIONS, DENOTATION). The user is prompted for the list of functions to be deleted. The functions in this list may be in any order and of any mix of types.

```
*(delete function)
>   ENTER LIST OF FUNCTIONS TO BE DELETED:
*(set1-men).
>   SET1-MEN DELETED.
```

The specifications of functions can be modified with the command MODIFY and the operand FUNCTIONS (or FUNCTION, DENOTATIONS, DENOTATION). The system requests the name of the function to be modified, and then the list of arguments for which the value of the function is to be changed. The specification of the new values for these arguments follows the method used to originally specify the values. This sequence is repeated for each of the functions to be modified.

```
*(modify function)
>   ENTER FUNCTION TO BE MODIFIED (NIL = NO MORE):
*io
>   THIS ELEMENT OF D_IAV=IV/IV IS A FUNCTION FROM D_<S,CN=IV>
```

```
>  TO D_CN=IV.
>   THAT IS, FROM PROPERTIES OF INDIVIDUAL CONCEPTS TO SETS OF
>  INDIVIDUAL CONCEPTS.
>
>   D_<S,CN=IV> = {C0, C1, C2}
>   D_CN=IV = {B0, B15, B1, B10, B5, B12}
>
>         D0 = {<{}, C0>, <{A0}, C1>, <{}, C2>}
>
>   ENTER LIST OF ARGUMENTS WHOSE VALUES ARE TO BE CHANGED:
*(c2)
>   ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>   C2 = {<I1, {}>, <I2, {A3}>}
*b0
>      MODIFICATION DONE
>        D0 = {<{}, C0>, <{A0}, C1>, <{}, C2>}
>
>   ENTER FUNCTION TO BE MODIFIED (NIL = NO MORE):
*nil
```

The specification of the meaning function F is modified with the command (MODIFY F). The modification proceeds in steps: the system prompts the user for a type and the list of logical constants of that type for which the value of F is to be changed, and then prompts the user for the new values in the same manner as in the original specification of F.

```
*(modify f)
>   ENTER TYPE OF LOGICAL CONSTANTS WHOSE VALUES OF F
>     ARE TO BE MODIFIED (NIL = NO MORE):
*e
>   ENTER THE LIST OF THESE LOGICAL CONSTANTS:
*(b n)
>   THE VALUES OF F FOR THESE CONSTANTS ARE ELEMENTS OF
>  D_<S,E> (INDIVIDUAL CONCEPTS).
>
>   LOGICAL CONSTANTS: B, N.
>   D_<S,E> = {IC-JO, IC-MA}
>
>   FOR EACH CONSTANT, ENTER THE VALUE OF F OR NIL:
>   B
*ic-bi
>   N
*ic-na
>
```

```
>   ENTER TYPE OF LOGICAL CONSTANTS WHOSE VALUES OF F
>     ARE TO BE MODIFIED (NIL = NO MORE):
*nil
```

G. <u>Changing Logical Constants</u>

The logical constants of PTQ are built into the system. These constants can be changed during the initial entry of a model, as part of the (ENTER MODEL) command, or later with the ADD, ENTER, and DELETE commands.

During entry of a model, the system first prompts for the type, intensional or extensional, of model to be entered. The system then asks if the built-in constants are to be changed. If so, the system presents the constants subtype by subtype, and asks for deletions and additions to each list.

```
>   DO YOU WISH TO CHANGE THE DEFAULT LOGICAL CONSTANTS?
*y
>
>   REMINDER: COMMAS CANNOT BE USED AS SEPARATORS IN LISTS.
>   LOGICAL CONSTANTS OF TYPE E ARE:  J, M, B, N.
>   OK?
*n
>   ENTER LIST OF CONSTANTS TO BE REMOVED FROM THIS LIST.
*(n)
>   ENTER LIST OF CONSTANTS TO BE ADDED TO THIS LIST.
*nil
>   LOGICAL CONSTANTS OF TYPE E ARE:  J, M, B.
>   OK?
*y
```

After the model has been entered, constants are added with the command ADD (or ENTER) and the operand LOGICAL_CONSTANTS (or LOGICAL_CONSTANT, CONSTANTS, CONSTANT). The addition proceeds subtype by subtype. First, the system prompts for the list of new constants of that subtype, and then for the values of the meaning function F for those constants.

```
*(add constant)
>    ENTER TYPE OF LOGICAL CONSTANTS TO BE ADDED (NIL = NO MORE):
*iv
>    LOGICAL CONSTANTS OF TYPE IV ARE:  RUN, WALK, TALK, RISE,
>   CHANGE.
>    ENTER LIST OF CONSTANTS TO BE ADDED:
*(swim)
>
>    THE VALUES OF  F  FOR THESE CONSTANTS ARE ELEMENTS OF
>   D <S,CN=IV> (PROPERTIES OF INDIVIDUAL CONCEPTS).
>
>    LOGICAL CONSTANTS: SWIM.
>    D_<S,CN=IV> = {C0, C1, C2}
>
>    FOR EACH CONSTANT, ENTER THE VALUE OF F OR NIL:
>    SWIM
*c2
>
>    ENTER TYPE OF LOGICAL CONSTANTS TO BE ADDED (NIL = NO MORE):
*nil
```

Constants are deleted with the command DELETE and the operand LOGICAL_CONSTANTS (or LOGICAL_CONSTANT, CONSTANTS, CONSTANT). The user is prompted for the list of constants to be deleted. The constants in this list may be in any order and of any mix of types.

```
*(delete constants)
>    ENTER LIST OF LOGICAL CONSTANTS TO BE DELETED:
*(m rise change temperature believe-that)
>    M DELETED.
>    RISE DELETED.
>    CHANGE DELETED.
>    TEMPERATURE DELETED.
>    BELIEVE-THAT DELETED.
```

H. Changing Variable Prefixes

A set of variable prefixes is built into the system. Changes are made with the commands ADD, ENTER, DELETE and MODIFY and the operand VARIABLE_PREFIXES (or VARIABLE_PREFIX, VARIABLES, VARIABLE, PREFIXES, PREFIX).

The addition and deletion of variable prefixes proceeds as for logical constants, except that prefixes are given only for semantic types, and not for subtypes.

The command MODIFY allows the user to replace the current prefixes of a type in a single step, rather than going through the two steps of adding and deleting prefixes. If only some of the prefixes of a type are to be changed, the list of the 'new' prefixes should contain those to be be retained.

```
*(modify prefixes)
>
>  ENTER TYPE OF VARIABLE PREFIXES TO BE RE-SET (NIL = NO MORE):
*<s,<e,ts>>
>   PREVIOUS PREFIXES WERE:
>   ENTER LIST OF NEW VARIABLE PREFIXES:
*(m)
```

I. Displaying the Model and its Components

Simple displays

The basic information about a component or element of the model can be displayed by entering its name (without parentheses) instead of a command. For example, entering ENTITIES, POSSIBLE_WORLDS or MOMENTS_IN_TIME causes those objects to be displayed; POINTS_OF_REFERENCE, TYPES OR D_SETS displays the names of those entries. The name of a point of reference or of a type displays the names of its components, and the name of a set displays the names of its elements. Entering F or the name of a functional element of a D_set displays its ordered pair representation. The value of F for a single constant can be displayed by applying F to that constant as shown below for the constant FUN.

```
*(f walk)
>   C1
*d_<s,e>
>   (A0 A3)
*a0
>   ((I1 J0) (I2 J0))
*entities
>   (J0 UN)
```

<u>The command DISPLAY</u>

The command DISPLAY takes a single argument, which is the name of a component or element to be displayed. The argument can be the name of a set of possible denotations, a possible denotation, F, a type, a logical constant, or a variable prefix. If D SETS, FUNCTIONS, TYPES, LOGICAL_CONSTANTS, VARIABLE_PREFIXES, or POINTS_OF_REFERENCE (or INDICES) is used as the argument, all of those components or elements will be displayed. For the argument MODEL, the entire model is displayed.

```
*(display d_<s,e>)
>   ****D_<S,E> = INDIVIDUAL CONCEPTS
>     DOMAIN:  D_S = {I1  I2}
```

```
>     RANGE:    D_E = {JO, UN}
>          A2 = {<I1,JO>, <I2,JO>}
>          A3 = {<I1,UN>, <I2,UN>}
```

The command DISPLAY_WHERE_USED

The command DISPLAY_WHERE_USED takes as its argument one of the following: D_SETS, FUNCTIONS, the name of a set of possible denotations, or the name of a possible denotation. The display of a possible denotation gives not only its specification, but also the list of functions in which it is used as a value.

```
*(display_where_used BO)
>          BO = {}
>          APPEARS AS A VALUE IN FUNCTIONS: C1, C2.
```

J. Auxiliary Commands

Display the commands

The user can examine the available system commands with the command DISPLAY and one of three operands. If the operand COMMANDS is used, the system will print the names of the commands. The operand COMMANDS&OPERANDS also prints the possible operands for each command. When the name of a command is the operand, its possible arguments are printed.

Changing the output format of functions

The output format for functions can be changed with the command RESET_FN_FORMAT. Unless changed by the user, the system uses FULL format: functions are printed as sets, properties and relations. In the TERSE format, the conversion of functions into sets, properties and relations is done only on the top level; if the first component of a relation is a set, the system will print the name of the function giving that set rather than the set itself. The third format is ORDERED_PAIRS and in this format the functions are printed as sets of ordered pairs (<argument,value>). The command prompts the user to enter the name of the new format.

Checking the type of an expression

The well-formedness and type of an expression can be checked with the command TYPE_OF_EXP. The prompting done by this command is the same as the first part of INTERP. It also shares the PREVIOUS expression feature with INTERP.

Resetting the print line

If the user abnormally terminates INTERP in the middle of an interpretation, the characters used in the indenting scheme will be frozen in the print line, that is, they will be printed out at the beginning of subsequent lines. To remove these characters invoke the command RESET_PRINT_LINE.

REFERENCES

J. Friedman, Notes on an intensional logic for English III: A unique reduced form for wffs of IL, N-10, The University of Michigan, Ann Arbor, Mich., 1978.

J. Friedman, D. Moran, and D. Warren, An explicit finite intensional model for PTQ, _American Journal of Computational Linguistics_, this fiche, 3-22.

J. Friedman and D. S. Warren, A parsing method for Montague grammars, to appear in _Linguistics and Philosophy_.

Daniel Gallin, _Intensional and Higher-order Modal Logic_, North-Holland Publishing Company, Amsterdam, 1975.

J. Hintikka, J. Moravcsik, and P. Suppes (eds.), _Approaches to Natural Language_, Reidel, Dordrecht, 1973.

Jerry R. Hobbs and Stanley J. Rosenschein, Making computational sense of Montague's intensional logic, _Artificial Intelligence_ 9 (1978), 287-306.

Edward L. Keenan (ed.), _Formal Semantics of Natural Language_, Cambridge University Press, Cambridge, 1975.

Franz von Kutschera, Partial interpretations, in Keenan (ed.), 1975, 156-174.

Richard Montague, Universal grammar, _Theoria_ 36 (1970), 373-398; reprinted in Montague, 1974, 222-246.

Richard Montague, The proper treatment of quantification in ordinary English, (PTQ), in Hintikka _et al._, 1973, 221-242; reprinted in Montague, 1974, 247-270.

Richard Montague, _Formal Philosophy: Selected Papers of Richard Montague_, edited and with an introduction by Richmond Thomason, Yale University Press, New Haven, 1974.

D. Moran, Changing semantic types in the interpretation system, N-7, The University of Michigan, Ann Arbor, Mich., forthcoming.

D. Moran, programmers' description of the interpretation system, N-8, The University of Michigan, Ann Arbor, Mich., forthcoming.

David S. Warren, A translation program for the grammar of PTQ, The University of Michigan, Ann Arbor, Mich., 1975.

APPENDIX A

This session parallels the development and use of the simple model given in Friedman, Moran, and Warren (this fiche). In this model, there are two entities, JO and UN, and two points of reference, I1 and I2. JO walks at both points of reference, UN walks only at I2. In I2, the individual concept for UN has the property of 'unicornhood'.

```
#$run *lisp t=2
*(restore interp)
*(enter model)
>   DO YOU WANT AN INTENSIONAL MODEL?
*y
>   DO YOU WISH TO CHANGE THE DEFAULT LOGICAL CONSTANTS?
*n
>
>   REMINDER: COMMAS CANNOT BE USED AS SEPARATORS IN LISTS.
>   ENTER LIST OF ENTITIES.
*(jo un)
>   ENTER LIST OF POSSIBLE WORLDS.
*(here)
>   ENTER LIST OF MOMENTS IN TIME, IN INCREASING ORDER.
*(then now)
>   POINTS OF REFERENCE (INDICES):
>     I1 = <HERE,THEN>
>     I2 = <HERE,NOW>
>
>   ****ENTERING ELEMENTS OF D_<S,E> - INDIVIDUAL CONCEPTS
>   THEY ARE FUNCTIONS FROM D_S TO D_E,
>   THAT IS, FROM POINTS OF REFERENCE TO ENTITIES.
>   THEY WILL BE THE VALUES OF F FOR THE LOGICAL CONSTANTS:
>   J, M, B, N.
>
>   D_S = {I1, I2}
>   D_E = {JO, UN}
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*a0
>   ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>   I1 = <HERE,THEN>
*jo
>   I2 = <HERE,NOW>
*jo
>         AO ENTERED.
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*a3
>   ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>   I1 = <HERE,THEN>
*un
```

```
>   I2 = <HERE,NOW>
*un
>        A3 ENTERED.
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*nil
>   ****D_<S,E> - INDIVIDUAL CONCEPTS
>     DOMAIN:  D_S = {I1, I2}
>     RANGE:   D_E = {JO, UN}
>          A0 = {<I1,JO>, <I2,JO>}
>          A3 = {<I1,UN>, <I2,UN>}
>
>   ****ENTERING THE FUNCTION  F  FOR LOGICAL CONSTANTS OF TYPE E
>   THE VALUES OF  F  FOR THESE CONSTANTS ARE ELEMENTS OF
>  D_<S,E> ,(INDIVIDUAL CONCEPTS).
>
>   LOGICAL CONSTANTS: J, M, B, N.
>   D_<S,E> = {A0, A3}
>
>   FOR EACH CONSTANT, ENTER THE VALUE OF F OR NIL:
>   J
*a0
>   M
*a3
>   B
*nil
>   N
*nil
>
>   ****ENTERING ELEMENTS OF D_CN=IV - SETS OF INDIVIDUAL
>  CONCEPTS
>   THEY ARE FUNCTIONS FROM D_<S,E> TO D_TS,
>   THAT IS, FROM INDIVIDUAL CONCEPTS TO TRUTH VALUES.
>   THEY WILL BE THE DENOTATIONS OF THE LOGICAL CONSTANTS:
>   MAN, WOMAN, PARK, FISH, PEN, UNICORN, PRICE, TEMPERATURE,
>  RUN, WALK, TALK, RISE, CHANGE.
>
>   D_<S,E> = {A0, A3}
>   D_TS = {0, 1}
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*b0
>   ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>   A0 = {<I1,JO>  <I2 JO>}
*0
>   A3 = {<I1,UN>, <I2,UN>}
*0
>        B0 ENTERED.
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*b15
>   ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>   A  = {<I1,JO>, <I2,JO>}
```

```
*1
>    A3 = {<I1,UN>, <I2,UN>}
*1
>        B15 ENTERED.
>
>    ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*nil
>    ****D_CN=IV - SETS OF INDIVIDUAL CONCEPTS
>      DOMAIN:  D_<S,E> = {A0, A3}
>      RANGE:   D_TS = {0, 1}
>           B0 = {}
>           B15 = {A0, A3}
>
>    ****ENTERING ELEMENTS OF D_<S,CN=IV>   PROPERTIES OF
>   INDIVIDUAL CONCEPTS
>    THEY ARE FUNCTIONS FROM D_S TO D_CN=IV,
>    THAT IS, FROM POINTS OF REFERENCE TO SETS OF INDIVIDUAL
>   CONCEPTS.
>    THEY WILL BE THE VALUES OF F FOR THE LOGICAL CONSTANTS:
>    MAN, WOMAN, PARK, FISH, PEN, UNICORN, PRICE, TEMPERATURE,
>   RUN, WALK, TALK, FISH, CHANGE.
>
>    D_S = {I1, I2}
>    D_CN=IV = {B0, B15}
>
>    ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*c0
>    ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>    I1 = <HERE,THEN>
*b0
>    I2 = <HERE,NOW>
*b1
>    NO SUCH ELEMENT.  EXPLAIN.
>     (1-WRONG NAME; 2-WILL BE ENTERED LATER; 3-ENTER NOW;
>      4-ENTER FUNCTION AS UNSPECIFIED FOR ALL ARGUMENTS).
*3
>    THIS ELEMENT OF D_CN=IV IS A FUNCTION FROM D_<S,E> TO D_TS,
>    THAT IS, FROM INDIVIDUAL CONCEPTS TO TRUTH VALUES.
>
>    D_<S,E> = {A0, A3}
>    D_TS = {0, 1}
>
>    ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>    A0 = {<I1,JO>, <I2,JO>}
*1
>    A3 = {<I1,UN>, <I2,UN>}
*1
>        B1 ENTERED.
>
>    ****D_CN=IV - SETS OF INDIVIDUAL CONCEPTS
>      DOMAIN:  D_<S,E> = {A0, A3}
>      RANGE:   D_TS = {0, 1}
>           B0 = {}
```

```
>           B15 = {A0, A3}
>           B1 = {A3}
>
>   RESUME SPECIFICATION OF C0
>
>        C0 ENTERED.
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*c1
>   ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>   I1 = <HERE,THEN>
*b15
>   I2 = <HERE,NOW>
*b10
>   NO SUCH ELEMENT.  EXPLAIN.
>    (1-WRONG NAME; 2-WILL BE ENTERED LATER; 3-ENTER NOW;
>     4-ENTER FUNCTION AS UNSPECIFIED FOR ALL ARGUMENTS).
*2
>        C1 ENTERED.
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*c2
>   ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>   I1 = <HERE,THEN>
*a
>   EXPECTED AN ELEMENT OF TYPE CN=IV, BUT FOUND
>  AN ELEMENT OF TYPE <S,E>. ENTER NEW NAME.
*b00
>   NO SUCH ELEMENT.  EXPLAIN.
>    (1-WRONG NAME; 2-WILL BE ENTERED LATER; 3-ENTER NOW;
>     4-ENTER FUNCTION AS UNSPECIFIED FOR ALL ARGUMENTS).
*1
>   ENTER CORRECT NAME:
*b
>   I2 = <HERE,NOW>
*b5
>   NO SUCH ELEMENT.  EXPLAIN.
>    (1-WRONG NAME; 2-WILL BE ENTERED LATER; 3-ENTER NOW;
>     4-ENTER FUNCTION AS UNSPECIFIED FOR ALL ARGUMENTS).
*2
>        C2 ENTERED.
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*nil
>   ****ENTERING UNSPECIFIED ELEMENTS B10, B5.
>
>   THESE ELEMENTS OF D_CN=IV ARE FUNCTIONS FROM D_<S,E> TO D_TS,
>   THAT IS, FROM INDIVIDUAL CONCEPTS TO TRUTH VALUES.
>
>   D_<S,E> = {A0, A3}
>   D_TS = {0, 1}
>
>   ENTERING B1
```

```
>    ENTER A VALUE OF NIL FOR EACH ARGUMENT:
>    A0 = {<I1,JO>, <I2,JO>}
*1
>    A3 = {<I1,UN>, <I2,UN>}
*0
>         B10 ENTERED.
>
>    ENTERING B5
>    ENTER A VALUE OF NIL FOR EACH ARGUMENT:
>    A0 = {<I1,JO>, <I2,JO>}
*
>    A3 = {<I1,UN>, <I2,UN>}
*1
>         B5 ENTERED.
>    **WARNING - B5 IS THE SAME AS B1
>
>    ****D_CN=IV - SETS OF INDIVIDUAL CONCEPTS
>      DOMAIN:   D_<S,E> = {A0, A3}
>      RANGE:    D_TS = {0, 1}
>           B0 = {}
>           B15 = {A0, A3}
>           B1 = {A3}
>           B10 = {A0}
>           B5 = {A3}
>
>    **WARNING - D_CN=IV CONTAINS EQUIVALENT ELEMENTS.
>    EQUIVALENT ELEMENTS: B1, B5.
>
>    ****D_<S,CN=IV> - PROPERTIES OF INDIVIDUAL CONCEPTS
>      DOMAIN:  D_S = {I1, I2}
>      RANGE:   D_CN=IV = {B0, B15, B1, B10, B5}
>           C0 = {<I1, {}>, <I2, {A3}>}
>           C1 = {<I1, {A0, A3}>, <I2, {A0}>}
>           C2 = {<I1, {}>, <I2, {A3}>}
>
>    ****ENTERING THE FUNCTION  F  FOR LOGICAL CONSTANTS OF TYPE
>   CN=IV
>    THE VALUES OF  F  FOR THESE CONSTANTS ARE ELEMENTS OF
>   D_<S,CN=IV> (PROPERTIES OF INDIVIDUAL CONCEPTS).
>
>    LOGICAL CONSTANTS: MAN, WOMAN, PARK, FISH, PEN, UNICORN,
>   PRICE, TEMPERATURE, RUN, WALK, TALK, RISE, CHANGE.
>    D_<S,CN=IV> = {C0, C1, C2}
>
>    FOR EACH CONSTANT, ENTER THE VALUE OF F OR NIL:
>    MAN
*nil
>    WOMAN
*nil
>    PARK
*nil
>    FISH
*nil
```

```
>    PEN
*nil
>    UNICORN
*c0
>    PRICE
*nil
>    TEMPERATURE
*nil
>    RUN
*nil
>    WALK
*c1
>    TALK
*nil
>    RISE
*nil
>    CHANCE
*nil
>
>    ****ENTERING ELEMENTS OF D_IAV=IV/IV -- RELATIONS (IN
>   EXTENSION) BETWEEN INDIVIDUAL CONCEPTS AND PROPERTIES OF
>   INDIVIDUAL CONCEPTS
>    THEY ARE FUNCTIONS FROM D_<S,CN=IV> TO D_CN=IV
>    THAT IS, FROM PROPERTIES OF INDIVIDUAL CONCEPTS TO SETS OF
>   INDIVIDUAL CONCEPTS.
>    THEY WILL BE THE DENOTATIONS OF THE LOGICAL CONSTANTS:
>    RAPIDLY, SLOWLY, VOLUNTARILY, ALLEGEDLY, TRY-TO, WISH-TO.
>
>    D_<S,CN=IV> = {C0, C1, C2}
>    D_CN=IV = {B0, B15, B1, B10, B5}
>
>    ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*d0
>    ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>    C0 = {<I1, {}>, <I2, {A3}>}
*nil
>    C1 = {<I1, {A1, A3}>, <I2, {A1}>}
*b12
>    NO SUCH ELEMENT.  EXPLAIN:
>     (1-WRONG NAME; 2-WILL BE ENTERED LATER, 3-ENTER NOW;
>      4-ENTER FUNCTION AS UNSPECIFIED FOR ALL ARGUMENTS).
*2
>    C2 = {<I1, {}>, <I2, {A3}>}
*nil
>         D0 ENTERED.
>
>    ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*nil
>    ****ENTERING UNSPECIFIED ELEMENT B12.
>
>    THIS ELEMENT OF D_CN=IV IS A FUNCTION FROM D_<S,E> TO D_TS,
>    THAT IS, FROM INDIVIDUAL CONCEPTS TO TRUTH VALUES.
>
```

```
>    D_<S,F> =  {A0, A3}
>    D_TS  = {0,  1}
>
>    ENTERING B12
>    ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>    A0 =  {<I1,JO>, <I2,JO>}
*1
>    A3 =  {<I1,UN>, <I2,UN>}
*0
>          B12 ENTERED.
>    **WARNING - B12 IS THE SAME AS B10
>
>    ****D_CN=IV - SETS OF INDIVIDUAL CONCEPTS
>      DOMAIN:   D_<S,F> =  {A0, A3}
>      RANGE:    D_TS  =  {0, 1}
>           B0  =  {}
>           B15  =  {A0, A3}
>           B1  =  {A3}
>           B10  =  {A0}
>           B5  =  {A3}
>           B12  =  {A0}
>
>     **WARNING - D_CN=IV CONTAINS EQUIVALENT ELEMENTS.
>     EQUIVALENT ELEMENTS: B1; B5.
>     EQUIVALENT ELEMENTS: B10, B12.
>
>     ****D_IAV=IV/IV - RELATIONS (IN EXTENSION) BETWEEN
>   INDIVIDUAL CONCEPTS AND PROPERTIES OF INDIVIDUAL CONCEPTS
>      DOMAIN:  D_<S,CN=IV> =  {C0, C1, C2}
>      RANGE:    D_CN=IV  =  {B0, B15, B1, B10, B5, B12}
>           D0  =  {<{}, C0>, <{A0}, C1>, <{}, C2>}
>
>     ****ENTERING ELEMENTS OF D_<S,IAV=IV/IV> -
>   RELATIONS-IN-INTENSION BETWEEN INDIVIDUAL CONCEPTS AND
>   PROPERTIES OF INDIVIDUAL CONCEPTS
>    THEY ARE FUNCTIONS FROM D_S TO D_IAV=IV/IV,
>    THAT IS, FROM POINTS OF REFERENCE TO RELATIONS (IN EXTENSION)
>    BETWEEN INDIVIDUAL CONCEPTS AND PROPERTIES OF INDIVIDUAL
>   CONCEPTS.
>    THEY WILL BE THE VALUES OF F FOR THE LOGICAL CONSTANTS:
>    RAPIDLY, SLOWLY, VOLUNTARILY, ALLEGEDLY, TRY-TO, WISH-TO.
>
>    D_S  =  {I1, I2}
>    D_IAV=IV/IV  =  {D0}
>
>    ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*e0
>    ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>    I1  =  <HERE,THEN>
*d0
>    I2  =  <HERE,NOW>
*nil
>        E0 ENTERED.
```

```
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*nil
>   ****D_<S,IAV=IV/IV> - RELATIONS-IN-INTENSION BETWEEN
>  INDIVIDUAL CONCEPTS AND PROPERTIES OF INDIVIDUAL CONCEPTS
>      DOMAIN:  D_S = {I1, I2}
>      RANGE:   D_IAV=IV/IV = {D0}
>           E0 = {<I1, {<{}, C0>, <{A0}, C1>, <{}, C2>}>, <I2, {}>}
>
>   ****ENTERING THE FUNCTION  F  FOR LOGICAL CONSTANTS OF TYPE
>  IAV=IV/IV
>   THE VALUES OF  F  FOR THESE CONSTANTS ARE ELEMENTS OF
>  D_<S,IAV=IV/IV> (RELATIONS-IN-INTENSION BETWEEN INDIVIDUAL
>  CONCEPTS AND PROPERTIES OF INDIVIDUAL CONCEPTS).
>
>   LOGICAL CONSTANTS: RAPIDLY, SLOWLY, VOLUNTARILY, ALLEGEDLY,
>  TRY-TO, WISH-TO.
>   D_<S,IAV=IV/IV> = {E0}
>
>   FOR EACH CONSTANT, ENTER THE VALUE OF F OR NIL:
>   RAPIDLY
*nil
>   SLOWLY
*e0
>   VOLUNTARILY
*nil
>   ALLEGEDLY
*nil
>   TRY-TO
*nil
>   WISH-TO
*nil
>
>   ****ENTERING ELEMENTS OF D_IF - SETS OF PROPERTIES OF
>  INDIVIDUAL CONCEPTS
>   THEY ARE FUNCTIONS FROM D_<S,CN=IV> TO D_TS,
>   THAT IS, FROM PROPERTIES OF INDIVIDUAL CONCEPTS TO TRUTH
>  VALUES.
>
>   D_<S,CN=IV> = {C0, C1, C2}
>   D_TS = {0, 1}
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*nil
>
>   ****ENTERING ELEMENTS OF D_<S,TS> - PROPOSITIONS
>   THEY ARE FUNCTIONS FROM D_S TO D_TS,
>   THAT IS, FROM POINTS OF REFERENCE TO TRUTH VALUES.
>
>   D_S = {I1, I2}
>   D_TS = {0, 1}
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
```

```
*nil
>   ENTRY OF MODEL TERMINATED
>
*(display model)
>   THE MODEL IS:
>
>   ENTITIES: (JO UN)
>
>   POSSIBLE WORLDS: (HERE)
>
>   MOMENTS IN TIME: (THEN NOW)
>
>   POINTS OF REFERENCE (INDICES):
>     I1 = <HERE,THEN>
>     I2 = <HERE,NOW>
>
>   ****D_<S,E> - INDIVIDUAL CONCEPTS
>     DOMAIN:  D_S = {I1, I2}
>     RANGE:   D_E = {JO, UN}
>
>         A0 = {<I1,JO>, <I2,JO>}
>         A3 = {<I1,UN>, <I2,UN>}
>
>   ****D_CN=IV - SETS OF INDIVIDUAL CONCEPTS
>     DOMAIN:  D_<S,E> = {A0, A3}
>     RANGE:   D_TS = { , 1}
>
>         B0 = {}
>         APPEARS AS A VALUE IN FUNCTIONS: C0, C2.
>
>         B15 = {A0, A3}
>         APPEARS AS A VALUE IN FUNCTIONS: C1.
>
>         B1 = {A3}
>         APPEARS AS A VALUE IN FUNCTIONS: C0.
>
>         B10 = {A0}
>         APPEARS AS A VALUE IN FUNCTIONS: C1.
>
>         B5 = {A3}
>         APPEARS AS A VALUE IN FUNCTIONS: C2.
>
>         B12 = {A0}
>         APPEARS AS A VALUE IN FUNCTIONS: D0.
>
>   **WARNING - D_CN=IV CONTAINS EQUIVALENT ELEMENTS.
>   EQUIVALENT ELEMENTS: B1, B5.
>   EQUIVALENT ELEMENTS: B10, B12.
>
>   ****D_<S,CN=IV> - PROPERTIES OF INDIVIDUAL CONCEPTS
>     DOMAIN:  D_S = {I1, I2}
>     RANGE:   D_CN=IV = {B0, B15, B1, B10, B5, B12}
>
```

```
>           C0 = {<I1, {}>, <I2, {A3}>}
>           C1 = {<I1, {A0, A3}>, <I2, {A3}>}
>           C2 = {<I1, {}>, <I2, {A3}>}
>
>    ****D_IAV=IV/IV - RELATIONS (IN EXTENSION) BETWEEN
>  INDIVIDUAL CONCEPTS AND PROPERTIES OF INDIVIDUAL CONCEPTS
>      DOMAIN:  D_<S,CN=IV> = {C0, C1, C2}
>      RANGE:   D_CN=IV = {B0, B15, B1, B13, B5, B12}
>
>           D0 = {<{}, C0>, <{A0}, C1>, <{}, C2>}
>           APPEARS AS A VALUE IN FUNCTIONS: E0.
>
>    ****D_<S,IAV=IV/IV> - RELATIONS-IN-INTENSION BETWEEN
>  INDIVIDUAL CONCEPTS AND PROPERTIES OF INDIVIDUAL CONCEPTS
>      DOMAIN:  D_S = {I1, I2}
>      RANGE:   D_IAV=IV/IV = {D0}
>
>              = {<I1, {<{}, C >, <{A0}, C1>, <{}, C2>}>, <I2, {}>}
>
>    ****LOGICAL CONSTANTS:
>      TYPE E: J, M, B, N.
>      TYPE CN=IV:
>        SUBTYPE CN: MAN, WOMAN, PARK, FISH, PEN, UNICORN, PRICE
>     TEMPERATURE.
>        SUBTYPE IV: RUN, WALK, TALK, RISE, CHANGE.
>      TYPE IAV=IV/IV:
>        SUBTYPE IAV: RAPIDLY, SLOWLY, VOLUNTARILY, ALLEGEDLY.
>        SUBTYPE IV/IV: TRY-TO, WISH-TO.
>      TYPE TV: FIND, LOSE, EAT, LOVE, DATE, SEEK, CONCEIVE.
>      TYPE IAV/TE: IN, ABOUT.
>      TYPE IV/IS: BELIEVE-THAT, ASSERT-THAT.
>
>    THE FUNCTION "F" (IN ORDERED PAIR NOTATION) IS
>    F = {<J,A0>, <M,A3>, <UNICORN,C0>, <WALK,C1>, <SLOWLY,E0>}
>
>    ****VARIABLE PREFIXES:
>      TYPE E: U, V.
>      TYPE <S,E>: X, Y.
>      TYPE <S,CN=IV>: P, Q.
>      TYPE <S,<E,TS>>: R.
>      TYPE <S,TS>: B.
>      TYPE <S,<E,<E,TS>>>: S.
>      TYPE <S,<E,<<E,TS>,<E,TS>>>>: G.
>      TYPE <S,TS>: K.
>
>    END OF MODEL DESCRIPTION
>
*(interp)
>
>    ENTER MEANINGFUL EXPRESSION (NIL = NO MORE):
*           ;comment - direct translation of "john walks"
*((lambda p ((ext p) (int j))) (int walk))
>    EXPRESSION IS OF TYPE TS
```

```
>    FREE VARIABLES:  NONE
>    OK?
*ok
>
>    ENTER POINT OF REFERENCE (NIL = NO MORE):
*i1
>
>    INITIAL VARIABLE ASSIGNMENT G1: NIL.
>    COMPUTING DENOTATION OF  ((LAMBDA P ((EXT P) (INT J))) (INT
>   WALK))  FOR I1 AND G1
>    :  COMPUTING DENOTATION OF  (LAMBDA P ((EXT P) (INT J)))
>    :  FOR I1 AND G1
>    :  :  *NEW VARIABLE ASSIGNMENT G2: P=C0.
>    :  :  COMPUTING DENOTATION OF  ((EXT P) (INT J))  FOR I1 AND
>    :  :  G2
>    :  :  :  COMPUTING DENOTATION OF  (EXT P)  FOR I1 AND G2
>    :  :  :  :  DENOTATION OF  P  IS  C0 = {<I1, {}>, <I2 {A3}>}
>    :  :  :  DENOTATION IS  B0 = {}
>    :  :  :  COMPUTING DENOTATION OF  (INT J)  FOR I1 AND G2
>    :  :  :  DENOTATION IS  A4 = {<I1,JO>, <I2,JO>}
>    :  :  DENOTATION IS  0
>    :  :  *NEW VARIABLE ASSIGNMENT G3: P=C1.
>    :  :  COMPUTING DENOTATION OF  ((EXT P) (INT J))  FOR I1 AND
>    :  :  G3
>    :  :  :  COMPUTING DENOTATION OF  (EXT P)  FOR I1 AND G3
>    :  :  :  :  DENOTATION OF  P  IS  C1 = {<I1, {A2, A3}>, <I2
>    :  :  :  :  , {A3}>}
>    :  :  :  DENOTATION IS  B15 = {A2, A3}
>    :  :  :  COMPUTING DENOTATION OF  (INT J)  FOR I1 AND G3
>    :  :  :  DENOTATION IS  A6 = {<I1,JO>, <I2,JO>}
>    :  :  DENOTATION IS  1
>    :  :  *NEW VARIABLE ASSIGNMENT G4: P=C2.
>    :  :  COMPUTING DENOTATION OF ((EXT P) (INT J)) FOR I1 AND G4
>    :  :  :  COMPUTING DENOTATION OF  (EXT P)  FOR I1 AND G4
>    :  :  :  :  DENOTATION OF  P  IS  C2 = {<I1, {}>, <I2, {A3}>}
>    :  :  :  DENOTATION IS  B6 = {}
>    :  :  :  COMPUTING DENOTATION OF  (INT J)  FOR I1 AND G4
>    :  :  :  DENOTATION IS  A0 = {<I1,JO>, <I2,JO>}
>    :  :  DENOTATION IS
>    :  :  DENOTATION OF  (LAMBDA P ((EXT P) (INT J)))  IS {C1}
>    :  :  BUT IS NOT AN ELEMENT IN THE MODEL.
>    :  :  ENTER A NAME FOR THIS ELEMENT IN D_TE:
*j*1
>    :  DENOTATION IS  J*1 = {C1}
>    :  COMPUTING DENOTATION OF  (INT WALK)  FOR I1 AND G1
>    :  DENOTATION IS  C1 = {<I1, {A2, A3}>, <I2, {A3}>}
>    DENOTATION IS  1
>
>    ENTER POINT OF REFERENCE (NIL = NO MORE):
*nil
>
>    ENTER MEANINGFUL EXPRESSION (NIL = NO MORE):
          ;comment - reduced form of "john walks"
```

```
*(walk (int j))
>   EXPRESSION IS OF TYPE TS
>   FREE VARIABLES:  NONE
>   OK?
*y
>
>   ENTER POINT OF REFERENCE (NIL = NO MORE):
*i1
>
>   INITIAL VARIABLE ASSIGNMENT G1: NIL.
>   COMPUTING DENOTATION OF  (WALK (INT J))  FOR I1 AND G1
>   :  DENOTATION OF  WALK  IS  B15 = {A0, A3}
>   :  COMPUTING DENOTATION OF  (INT J)  FOR I1 AND G1
>   :  DENOTATION IS  A1 = {<I1,J0>, <I2,J0>}
>   DENOTATION IS  1
>
>   ENTER POINT OF REFERENCE (NIL = NO MORE):
*nil
>
>   ENTER MEANINGFUL EXPRESSION (NIL = NO MORE):
*           ;comment - reduced form of "john walks slowly"
*((slowly (int walk)) (int j))
>   EXPRESSION IS OF TYPE TS
>   FREE VARIABLES:  NONE
>   OK?
*y
>
>   ENTER POINT OF REFERENCE (NIL = NO MORE):
*i1
>
>   INITIAL VARIABLE ASSIGNMENT G1: NIL.
>   COMPUTING DENOTATION OF  ((SLOWLY (INT WALK)) (INT J))  FOR
>  I1 AND G1
>   :  COMPUTING DENOTATION OF (SLOWLY (INT WALK)) FOR I1 AND G1
>   :  :  DENOTATION OF  SLOWLY  IS  D1 = {<{}, C0>, <{A0}, C1>
>   :  :  <{}, C2>}
>   :  :  COMPUTING DENOTATION OF  (INT WALK)  FOR I1 AND G1
>   :  :  DENOTATION IS  C1 = {<I1, {A0, A3}>, <I2, {A0}>}
>   :  DENOTATION IS  B12 = {A0}
>   :  COMPUTING DENOTATION OF  (INT J)  FOR I1 AND G1
>   :  DENOTATION IS  A  = {<I1,J0>, <I2,J0>}
>   DENOTATION IS 1
>
>   ENTER POINT OF REFERENCE (NIL = NO MORE):
*nil
>
>   ENTER MEANINGFUL EXPRESSION (NIL = NO MORE):
*nil
>   ENTER NEXT COMMAND:
*(checkpoint model-n3 model)
>CHECKPOINT DONE. SPACE=6 PAGES
#EXECUTION TERMINATED
```

APPENDIX B

Creating a model that satisfies Meaning Postulates 1, 2, and 3.

```
*(enter model)
>   DO YOU WANT AN INTENSIONAL MODEL?
*y
>   DO YOU WISH TO CHANGE THE DEFAULT LOGICAL CONSTANTS?
*n
>
>   REMINDER: COMMAS CANNOT BE USED AS SEPARATORS IN LISTS.
>   ENTER LIST OF ENTITIES.
*(jo ma)
>   ENTER LIST OF POSSIBLE WORLDS.
*(1)
>   ENTER LIST OF MOMENTS IN TIME, IN INCREASING ORDER.
*(1 2)
>   POINTS OF REFERENCE (INDICES):
>     I1 = <1,1>
>     I2 = <1,2>
>
>   ****ENTERING ELEMENTS OF D_<S,E> - INDIVIDUAL CONCEPTS
>   THEY ARE FUNCTIONS FROM D_S TO D_E,
>   THAT IS, FROM POINTS OF REFERENCE TO ENTITIES.
>   THEY WILL BE THE VALUES OF F FOR THE LOGICAL CONSTANTS:
>   J, M, B, N.
>
>   D_S = {I1, I2}
>   D_E = {JO, MA}
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*nil
>
>   ****ENTERING ELEMENTS OF D_<S,TS> - PROPOSITIONS
>   THEY ARE FUNCTIONS FROM D_S TO D_TS,
>   THAT IS, FROM POINTS OF REFERENCE TO TRUTH VALUES.
>
>   D_S = {I1, I2}
>   D_TS = {0, 1}
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*nil
>   ENTRY OF MODEL TERMINATED
>
*(interp)
>
>   ENTER MEANINGFUL EXPRESSION (NIL = NO MORE):
*     ;comment - meaning postulate 1 for john.
*(there-is u (necessarily (equal u j)))
>   EXPRESSION IS OF TYPE TS
>   FREE VARIABLES:  NONE
```

```
>   OK?
*ok
>
>   ENTER POINT OF REFERENCE (NIL = NO MORE):
*i2
>
>  INITIAL VARIABLE ASSIGNMENT G1: NIL.
>   COMPUTING DENOTATION OF  (THERE-IS U (NECESSARILY (EQUAL U J)
>  ))   FOR I2 AND G1.
>   :   *NEW VARIABLE ASSIGNMENT G2: U JO.
>   :   COMPUTING DENOTATION OF  (NECESSARILY (EQUAL U J))   FOR
>   : I2 AND G2.
>   :   :  COMPUTING DENOTATION OF  (EQUAL U J)   FOR I1 AND G2
>   :   :   :  DENOTATION OF  U  IS  JO
>   :   :   :    | THE VALUE OF F IS UNSPECIFIED FOR THE ARGUMENT J
>   :   :   :    | THE POSSIBLE VALUES ARE:
>   :   :   :    | L_<F,2> = {}
>   :   :   :    |
>   :   :   :    | ENTER THE VALUE OF F FOR:
>   :   :   :    | J
*ic-jo
>   :   :   :    | NO SUCH ELEMENT.  EXPLAIN.
>   :   :   :    |  (1-WRONG NAME; 2-WILL BE ENTERED LATER;
>   :   :   :    |   3-ENTER NOW; 4-ENTER FUNCTION AS
>   :   :   :    |   UNSPECIFIED FOR ALL ARGUMENTS).
*4
>   :   :   :    |        F(J) = IC-JO = {<I1,NIL>, <I2,NIL>}
>   :   :   :    | DONE - F RESET.
>   :   :   :    |
>   :   :   :    | THE VALUE OF IC-JO
>   :   :   :    |IS UNSPECIFIED FOR THE ARGUMENT I1
>   :   :   :    | THE POSSIBLE VALUES ARE:
>   :   :   :    | L_F = {JO, MA}
>   :   :   :    |
>   :   :   :    | ENTER THE VALUE OF IC-JO FOR:
>   :   :   :    | I1 = <1,1>
*jo
>   :   :   :    |        IC-JO = {<I1,JO>, <I2,NIL>}
>   :   :   :    |DONE - IC-JO RESET.
>   :   :   :    |
>   :   :   :  DENOTATION OF  J  IS  JO
>   :   :  DENOTATION IS 1 .
>   :   :  COMPUTING DENOTATION OF  (EQUAL U J)   FOR I2 AND G2
>   :   :   :  DENOTATION OF  U  IS  JO
>   :   :   :    | THE VALUE OF IC-JO
>   :   :   :    |IS UNSPECIFIED FOR THE ARGUMENT I2
>   :   :   :    | THE POSSIBLE VALUES ARE:
>   :   :   :    | L_F = {JO, MA}
>   :   :   :    |
>   :   :   :    | ENTER THE VALUE OF IC-JO FOR:
>   :   :   :    | I2 = <1,2>
*jo
>   :   :   :             C-JO = {<I1,JO>, <I2,JO>}
```

```
>   :  :  :    | DONE - IC-JO RESET.
>   :  :  :    |
>   :  :  :  DENOTATION OF  J  IS .JO
>   :  :  DENOTATION IS  1
>   :  DENOTATION IS  1
>   DENOTATION IS  1
>
>   ENTER POINT OF REFERENCE (NIL = NO MORE):
*nil
>
>   ENTER MEANINGFUL EXPRESSION (NIL = NO MORE):
*nil
>   ENTER NEXT COMMAND:
>
>
*(add function)
>   ENTER TYPE OF ELEMENTS TO BE ADDED (NIL = NO MORE):
*<s,e>
>
>   ****ENTERING ELEMENTS OF D_<S,E> - INDIVIDUAL CONCEPTS
>   THEY ARE FUNCTIONS FROM D_S TO D_E,
>   THAT IS, FROM POINTS OF REFERENCE TO ENTITIES.
>   THEY WILL BE THE VALUES OF F FOR THE LOGICAL CONSTANTS:
>   J, M, B, N.
>
>   D_S = {I1, I2}
>   D_E = {JO, MA}
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*ic-ma
>   ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>   I1 = <1,1>
*ma
>   I2 = <1,2>
*ma
>        IC-MA ENTERED.
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*nil
>   ****D_<S,E> - INDIVIDUAL CONCEPTS
>     DOMAIN:  D_S = {I1, I2}
>     RANGE:   D_E = {JO, MA}
>         IC-JO = {<I1,JO>, <I2,JO>}
>         IC-MA = {<I1,MA>, <I2,MA>}
>
>   ENTER TYPE OF ELEMENTS TO BE ADDED (NIL = NO MORE):
*nil
>
*(modify f)
>   ENTER TYPE OF LOGICAL CONSTANTS WHOSE VALUES OF F
>     ARE TO BE MODIFIED (NIL = NO MORE):
*e
>   ENTER THE LIST OF THESE LOGICAL CONSTANTS:
```

```
*(m)
>   THE VALUES OF  F  FOR THESE CONSTANTS ARE ELEMENTS OF
>  D_<S,F> (INDIVIDUAL CONCEPTS).
>
>   LOGICAL CONSTANTS: M.
>   D_<S,E> = {IC-JO, IC-MA}
>
>   FOR EACH CONSTANT, ENTER THE VALUE OF F OR NIL:
>   M
*ic-ma
>
>   ENTER TYPE OF LOGICAL CONSTANTS WHOSE VALUES OF F
>     ARE TO BE MODIFIED (NIL = NO MORE):
*nil
>
*(interp)
>
>   ENTER MEANINGFUL EXPRESSION (NIL = NO MORE):
*        ;comment - meaning postulate 1 for mary.
*(there-is u (necessarily (equal u m)))
>   EXPRESSION IS OF TYPE IS
>   FREE VARIABLES:  NONE
>   OK?
*ok
>
>   ENTER POINT OF REFERENCE (NIL = NO MORE):
*i1
>
>   INITIAL VARIABLE ASSIGNMENT G1: NIL.
>   COMPUTING DENOTATION OF  (THERE-IS U (NECESSARILY (EQUAL U M)
>  ))  FOR I1 AND G1
>   :  *NEW VARIABLE ASSIGNMENT G2: U=JO.
>   :  COMPUTING DENOTATION OF  (NECESSARILY (EQUAL U M))  FOR
>   :  I1 AND G2
>   :  :  COMPUTING DENOTATION OF  (EQUAL U M)  FOR I1 AND G2
>   :  :  :  DENOTATION OF  U  IS  JO
>   :  :  :  DENOTATION OF  M  IS  MA
>   :  :  DENOTATION IS
>   :  DENOTATION IS
>   :  *NEW VARIABLE ASSIGNMENT G3: U=MA.
>   :  COMPUTING DENOTATION OF  (NECESSARILY (EQUAL U M))  FOR
>   : I1 AND G3.
>   :  :  COMPUTING DENOTATION OF  (EQUAL U M)  FOR I1 AND G3
>   :  :  :  DENOTATION OF  U  IS  MA
>   :  :  :  DENOTATION OF  M  IS  MA
>   :  :  DENOTATION IS  1
>   :  :  COMPUTING DENOTATION OF  (EQUAL U M)  FOR I2 AND G3
>   :  :  :  DENOTATION OF  U  IS  MA
>   :  :  :  DENOTATION OF  M  IS  MA
>   :  :  DENOTATION IS  1
>   :  DENOTATION IS  1
>   DENOTATION IS  1
>
```

```
>    ENTER POINT OF REFERENCE (NIL = NO MORE):
*nil
>
>    ENTER MEANINGFUL EXPRESSION (NIL = NO MORE):
*        ;comment - meaning postulate 2 for man.
*(for-all x (necessarily (implies (man x) (there-is u (equal x .
*(int u))))))
>    EXPRESSION IS OF TYPE TS
>    FREE VARIABLES:  NONE
>    OK?
*ok
>
>    ENTER POINT OF REFERENCE (NIL = NO MORE):
*i1
>
>    INITIAL VARIABLE ASSIGNMENT G1: NIL.
>    COMPUTING DENOTATION OF  (FOR-ALL X (NECESSARILY (IMPLIES (
>   MAN X) (THERE-IS U (EQUAL X (INT U)))))) FOR I1 AND G1
>    :  *NEW VARIABLE ASSIGNMENT G2: X=IC-JC.
>    :  COMPUTING DENOTATION OF  (NECESSARILY (IMPLIES (MAN X) (
>    : THERE-IS U (EQUAL X (INT U))))) FOR I1 AND G2
>    :  :  COMPUTING DENOTATION OF  (IMPLIES (MAN X) (THERE-IS U
>    :  : (EQUAL X (INT U)))) FOR I1 AND G2
>    :  :  :  COMPUTING DENOTATION OF  (MAN X)  FOR I1 AND G2
>    :  :  :  :  | THE VALUE OF F
>    :  :  :  :  |IS UNSPECIFIED FOR THE ARGUMENT MAN
>    :  :  :  :  | THE POSSIBLE VALUES ARE:
>    :  :  :  :  | D_<S,CN=IV> = {}
>    :  :  :  :  |
>    :  :  :  :  | ENTER THE VALUE OF F FOR:
>    :  :  :  :  | MAN
*prop-man
>    :  :  :  :  | NO SUCH ELEMENT.  EXPLAIN.
>    :  :  :  :  |  (1-WRONG NAME; 2-WILL BE ENTERED LATER;
>    :  :  :  :  |   3-ENTER NOW; 4-ENTER FUNCTION AS
>    :  :  :  :  |   UNSPECIFIED FOR ALL ARGUMENTS).
*4
>    :  :  :  :  |       F(MAN) = PROP-MAN = {<I1, {}>, <I2, {}>}
>    :  :  :  :  | DONE - F RESET.
>    :  :  :  :  |
>    :  :  :  :  | THE VALUE OF PROP-MAN
>    :  :  :  :  |IS UNSPECIFIED FOR THE ARGUMENT I1
>    :  :  :  :  | THE POSSIBLE VALUES ARE:
>    :  :  :  :  | D_CN=IV = {}
>    :  :  :  :  |
>    :  :  :  :  | ENTER THE VALUE OF PROP-MAN FOR:
>    :  :  :  :  | I1 = <1,1>
*set-men
>    :  :  :  :  | NO SUCH ELEMENT.  EXPLAIN.
>    :  :  :  :  |  (1-WRONG NAME; 2-WILL BE ENTERED LATER;
>    :  :  :  :  |   3-ENTER NOW; 4-ENTER FUNCTION AS
>    :  :  :  :  |   UNSPECIFIED FOR ALL ARGUMENTS).
*4
```

```
>   :   :   :   :    |          PROP-MAN = {<I1, {}>, <I2, {}>}
>   :   :   :   :    | DONE - PROP-MAN RESET.
>   :   :   :   :    |
>   :   :   :   :   DENOTATION OF  MAN  IS  SET-MEN = {}
>   :   :   :   :   DENOTATION OF  X  IS  IC-JO = {<I1,JO>, <I2,JO>}
>   :   :   :   :| THE VALUE OF SET-MEN
>   :   :   :   :|IS UNSPECIFIED FOR THE ARGUMENT IC-JO
>   :   :   :   :| THE POSSIBLE VALUES ARE:
>   :   :   :   :| D_TS = {0, 1}
>   :   :   :   :|
>   :   :   :   :| ENTER THE VALUE OF SET-MEN FOR:
>   :   :   :   :| IC-JO = {<I1,JO>, <I2,JO>}
*
>   :   :   :   :|        SET-MEN = {IC-JO}
>   :   :   :   :| DONE - SET-MEN RESET.
>   :   :   :   :|
>   :   :   :  DENOTATION IS  1
>   :   :   :  COMPUTING DENOTATION OF  (THERE-IS U (EQUAL X (INT
>   :   :   : U)))  FOR I1 AND G2
>   :   :   :   :  *NEW VARIABLE ASSIGNMENT G3: X=IC-JO, U=JO.
>   :   :   :   :  COMPUTING DENOTATION OF  (EQUAL X (INT U))  FOR
>   :   :   :   :  1 AND G3
>   :   :   :   :   :  DENOTATION OF  X  IS  IC-JO = {<I1,JO>, <I2,
>   :   :   :   :   :  JO>}
>   :   :   :   :   :  COMPUTING DENOTATION OF  (INT U)  FOR I1 AND
>   :   :   :   :   :  G3
>   :   :   :   :   :   :  DENOTATION OF  U  IS  JO
>   :   :   :   :   :   :  DENOTATION OF  U  IS  JO
>   :   :   :   :   :  DENOTATION IS  IC-JO = {<I1,JO>, <I2,JO>}
>   :   :   :   :  DENOTATION IS  1
>   :   :   :  DENOTATION IS  1
>   :   :  DENOTATION IS  1
>   :   :  COMPUTING DENOTATION OF  (IMPLIES (MAN X) (THERE-IS U
>   :   : (EQUAL X (INT U))))  FOR I2 AND G2
>   :   :   :  COMPUTING DENOTATION OF  (MAN X)  FOR I2 AND G2
>   :   :   :   :    | THE VALUE OF PROP-MAN
>   :   :   :   :    |IS UNSPECIFIED FOR THE ARGUMENT I2
>   :   :   :   :    | THE POSSIBLE VALUES ARE:
>   :   :   :   :    | D_CN=IV = {SET-MEN}
>   :   :   :   :    |
>   :   :   :   :    | ENTER THE VALUE OF PROP-MAN FOR:
>   :   :   :   :    | I2 = <1,2>
*set-men
>   :   :   :   :    |          PROP-MAN = {<I1, {IC-JO}>, <I2, {IC-JO}
>   :   :   :   :    |>}
>   :   :   :   :    | DONE - PROP-MAN RESET.
>   :   :   :   :    |
>   :   :   :   :   DENOTATION OF  MAN  IS  SET-MEN = {IC-JO}
>   :   :   :   :   DENOTATION OF  X  IS  IC-JO = {<I1,JO>, <I2,JO>}
>   :   :   :  DENOTATION IS  1
>   :   :   :  COMPUTING DENOTATION OF  (THERE-IS U (EQUAL X (INT
>   :   :   : U)))  FOR I2 AND G2
>   :   :   :   :  *NEW VARIABLE ASSIGNMENT G4: X=IC-JO, U=JO.
```

```
>   :   :   :   :  COMPUTING DENOTATION OF  (EQUAL X (INT U))  FOR
>   :   :   :   : I2 AND G4
>   :   :   :   :   :  DENOTATION OF  X IS  IC-JO = {<I1,JO>, <I2,
>   :   :   :   :   : JO>}
*   :   :   :   :   :  COMPUTING DENOTATION OF (INT U) FOR I2 AND G4
>   :   :   :   :   :   :  DENOTATION OF  U  IS  JO
>   :   :   :   :   :   :  DENOTATION OF  U  IS  JO
>   :   :   :   :   :  DENOTATION IS  IC-JO = {<I1,JO>, <I2,JO>}
>   :   :   :   :  DENOTATION IS  1
>   :   :   :  DENOTATION IS  1
>   :   :  DENOTATION IS  1
>   :  DENOTATION IS  1
>   :  *NEW VARIABLE ASSIGNMENT G5: X=IC-MA.
>   :  COMPUTING DENOTATION OF  (NECESSARILY (IMPLIES (MAN X) (
>   : THERE-IS U.(EQUAL X (INT U)))))  FOR I1 AND G5
>   :   :  COMPUTING DENOTATION OF  (IMPLIES (MAN X) (THERE-IS U
>   :   : (EQUAL X (INT U))))  FOR I1 AND G5
>   :   :   :  COMPUTING DENOTATION OF  (MAN X)  FOR I1 AND G5
>   :   :   :   :  DENOTATION OF  MAN  IS  SET-MEN = {IC-JO}
>   :   :   :   :  DENOTATION OF  X  IS  IC-MA = {<I1,MA>, <I2,MA>}
>   :   :   :   :| THE VALUE OF SET-MEN
>   :   :   :   :|IS UNSPECIFIED FOR THE ARGUMENT IC-MA
>   :   :   :   :| THE POSSIBLE VALUES ARE:
>   :   :   :   :| D_IS = {0, 1}
>   :   :   :   :|
>   :   :   :   :| ENTER THE VALUE OF SET-MEN FOR:
>   :   :   :   :| IC-MA = {<I1,MA>, <I2,MA>}
*
>   :   :   :   :|     0  SET-MEN = {IC-JO}
>   :   :   :   :| DONE - SET-MEN RESET.
>   :   :   :   :|
>   :   :   :   DENOTATION IS
>   :   :   DENOTATION IS  1
>   :   :  COMPUTING DENOTATION OF  (IMPLIES (MAN X) (THERE-IS U
>   :   : (EQUAL X (INT U))))  FOR I2 AND G5
>   :   :   :  COMPUTING DENOTATION OF  (MAN X)  FOR I2 AND G5
>   :   :   :   :  DENOTATION OF  MAN  IS  SET-MEN = {IC-JO}
>   :   :   :   :  DENOTATION OF  X  IS  IC-MA = {<I1,MA>, <I2,MA>}
>   :   :   :  DENOTATION IS
>   :   :  DENOTATION IS  1
>   :  DENOTATION IS  1
>   DENOTATION IS  1
>
>   ENTER POINT OF REFERENCE (NIL = NO MORE):
*nil
>
>   ENTER MEANINGFUL EXPRESSION (NIL = NO MORE):
*      ;comment - meaning postulate 3 for walk.
*(there-is e (for-all x (necessarily (iff (walk x) ((ext e) (
*ext x))))))
>   EXPRESSION IS OF TYPE TS
>   FREE VARIABLES:  NONE
>   OK?
```

```
*y
>
>   ENTER POINT OF REFERENCE (NIL = NO MORE):
*i2
>
>   INITIAL-VARIABLE ASSIGNMENT G1: NIL.
>   COMPUTING DENOTATION OF  (THERE-IS E (FOR-ALL X (NECESSARILY
>  (IFF (WALK X) ((EXT E) (EXT X))))))  FOR I2 AND G1
>   : THERE ARE NO POSSIBLE DENOTATIONS FOR THE BOUND VARIABLE
>   : E.
>   : UNSPECIFIED FUNCTION BEING ADDED TO D_<S,<E,T>>.
>   : ENTER NAME FOR THIS ELEMENT (NIL = DON'T ADD):
*prop-walk*
>   : *NEW VARIABLE ASSIGNMENT G2: E=PROP-WALK*.
>   : COMPUTING DENOTATION OF  (FOR-ALL X (NECESSARILY (IFF (
>   : WALK X) ((EXT E) (EXT X)))))  FOR I2 AND G2
       .
       .
       .
>      DENOTATION IS
>   DENOTATION IS  1
>
>   ENTER POINT OF REFERENCE (NIL = NO MORE):
*nil
>
>   ENTER MEANINGFUL EXPRESSION (NIL = NO MORE):
*nil
>
>   ENTER NEXT COMMAND:
>
*(display functions)
>          IC-JO = {<I1,JO>, <I2,JO>}
>          IC-MA = {<I1,MA>, <I2,MA>}
>          SET-MEN = {IC-JO}
>          SET1-WALKERS = {IC-JO}
>          SET2-WALKERS = {IC-MA}
>          PROP-MAN = {<I1, {IC-JO}>, <I2, {IC-JO}>}
>          PROP-WALK = {<I1, {IC-JO}>, <I2, {IC-MA}>}
>          SET1-WALK*ERS = {JO}
>          SET2-WALK*ERS = {MA}
>          PROP-WALK* = {<I1, {JO}>, <I2, {MA}>}
>
*(checkpoint model-mp model)
*EXECUTION TERMINATED
```

APPENDIX C

Constructing a counter-example to:
    (NECESSARILY (IFF (WOMAN X) (WOMAN* (EXT X))))

```
#$run *lisp t=2
*(restore interp)
*(enter model)
>    DO YOU WANT AN INTENSIONAL MODEL?
*y
>    DO YOU WISH TO CHANGE THE DEFAULT LOGICAL CONSTANTS?
*n
>
>    REMINDER: COMMAS CANNOT BE USED AS SEPARATORS IN LISTS.
>    ENTER LIST OF ENTITIES.
*(bi ma)
>    ENTER LIST OF POSSIBLE WORLDS.
*(w1 w2)
>    ENTER LIST OF MOMENTS IN TIME, IN INCREASING ORDER.
*(t)
>    POINTS OF REFERENCE (INDICES):
>      I1 = <W1,T>
>      I2 = <W2,T>
>
>    ****ENTERING ELEMENTS OF D_<S,E> - INDIVIDUAL CONCEPTS
>    THEY ARE FUNCTIONS FROM D_S TO D_E,
>    THAT IS, FROM POINTS OF REFERENCE TO ENTITIES.
>    THEY WILL BE THE VALUES OF F FOR THE LOGICAL CONSTANTS:
>    J, M, B, N.
>
>    D_S = {I1, I2}
>    D_E = {BI, MA}
>
>    ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*pres
>    ENTER A VALUE OF NIL FOR EACH ARGUMENT:
>    I1 = <W1,T>
*ma
>    I2 = <W2,T>
*bi
>         PRES ENTERED.
>
>    ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*biic
>    ENTER A VALUE OF NIL FOR EACH ARGUMENT:
>    I1 = <W1,T>
*bi
>    I2 = <W2,T>
*bi
>         BIIC ENTERED.
>
```

```
>    ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*maic
>    ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>    I1 = <W1,T>
*ma
>    I2 = <W2,T>
*ma
>         MAIC ENTERED.
>
>    ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*nil
>    ****D_<S,E> - INDIVIDUAL CONCEPTS
>      DOMAIN:  D_S  = {I1, I2}
>      RANGE:   D_E  = {BI, MA}
>          PRES = {<I1,MA>, <I2,BI>}
>          BIIC = {<I1,BI>, <I2,BI>}
>          MAIC = {<I1,MA>, <I2,MA>}
>
>    ****ENTERING THE FUNCTION  F  FOR LOGICAL CONSTANTS OF TYPE E
>    THE VALUES OF  F  FOR THESE CONSTANTS ARE ELEMENTS OF
>   D_<S,E> (INDIVIDUAL CONCEPTS).
>
>    LOGICAL CONSTANTS: J, M, B, N.
>    D_<S,E> = {PRES, BIIC, MAIC}
>
>    FOR EACH CONSTANT, ENTER THE VALUE OF F OR NIL:
>    J
*nil
>
*maic
>    B
*biic
>    N
*nil
>
>    ****ENTERING ELEMENTS OF D_CN=IV - SETS OF INDIVIDUAL
>   CONCEPTS
>    THEY ARE FUNCTIONS FROM D_<S,E> TO D_TS,
>    THAT IS, FROM INDIVIDUAL CONCEPTS TO TRUTH VALUES.
>    THEY WILL BE THE DENOTATIONS OF THE LOGICAL CONSTANTS:
>    MAN, WOMAN, PARK, FISH, PEN, UNICORN, PRICE, TEMPERATURE,
>   RUN, WALK, TALK, RISE, CHANGE.
>
>    D_<S,E> = {PRES, BIIC, MAIC}
>    D_TS = {0, 1}
>
>    ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*womanset
>    ENTER A VALUE OR NIL FOR EACH ARGUMENT:
>    PRES = {<I1,MA>, <I2,BI>}
*
     BIIC = {<I1,BI>, <I2,BI>}
*0
```

```
>   MAIC = {<I1,MA>, <I2,MA>}
*1
>        WOMANSET ENTERED.
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*nil
>   ****D_CN=IV - SETS OF INDIVIDUAL CONCEPTS
>     DOMAIN:  D_<S,E> = {PRES, PIIC, MAIC}
>     RANGE:   D_TS = {0, 1}.
>          WOMANSET = {MAIC}
>
>   ****ENTERING ELEMENTS OF D_<S,CN=IV> - PROPERTIES OF
>  INDIVIDUAL CONCEPTS
>   THEY ARE FUNCTIONS FROM D_S TO D_CN=IV,
>   THAT IS, FROM POINTS OF REFERENCE TO SETS OF INDIVIDUAL
>  CONCEPTS.
>   THEY WILL BE THE VALUES OF F FOR THE LOGICAL CONSTANTS:
>   MAN, WOMAN, PARK, FISH, PEN, UNICORN, PRICE, TEMPERATURE,
>  RUN, WALK, TALK, RISE, CHANGE.
>
>   D_S = {I1, I2}
>   D_CN=IV = {WOMANSET}
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*womanprop
>   ENTER A VALUE OF NIL FOR EACH ARGUMENT:
>   I1 = <W1,T>
*womanset
>   I2 = <W2,T>
*womanset
>        WOMANPROP ENTERED.
>
>   ENTER NAME FOR ELEMENT. (NIL = NO MORE).
*nil
>   ****D_<S,CN=IV> - PROPERTIES OF INDIVIDUAL CONCEPTS
>     DOMAIN:  D_S = {I1, I2}
>     RANGE:   D_CN=IV = {WOMANSET}
>          WOMANPROP = {<I1, {MAIC}>, <I2, {MAIC}>}
>
>   ****ENTERING THE FUNCTION  F  FOR LOGICAL CONSTANTS OF TYPE
>  CN=IV
>   THE VALUES OF  F  FOR THESE CONSTANTS ARE ELEMENTS OF
>  D_<S,CN=IV> (PROPERTIES OF INDIVIDUAL CONCEPTS).
>
>   LOGICAL CONSTANTS: MAN, WOMAN, PARK, FISH, PEN, UNICORN,
>  PRICE, TEMPERATURE, RUN, WALK, TALK, RISE, CHANGE.
>   D_<S,CN=IV> = {WOMANPROP}
>
>   FOR EACH CONSTANT, ENTER THE VALUE OF F OR NIL:
>   MAN
*nil
>   WOMAN
*womanprop
```

```
>    DARK
*nil

>    ENTRY OF MODEL TERMINATED
>
*(interp)
>
>    ENTER MEANINGFUL EXPRESSION (NIL = NO MORE):
*              ;comment - meaning postulate 2 for "woman".
*(for-all x (necessarily (implies (woman x) (there-is u (equal
*x (int u))))))
>    EXPRESSION IS OF TYPE T S
>    FREE VARIABLES:  NONE
>    OK?
*ok*
>
>    ENTER POINT OF REFERENCE (NIL = NO MORE):
*i1
>
>    INITIAL VARIABLE ASSIGNMENT G1: NIL.
>    COMPUTING DENOTATION OF   (FOR-ALL X (NECESSARILY (IMPLIES (
>    WOMAN X) (THERE-IS U (EQUAL X (INT U)))))  FOR I1 AND G1
>    :  *NEW VARIABLE ASSIGNMENT G2: X=PRES.
>    :  COMPUTING DENOTATION OF   (NECESSARILY (IMPLIES (WOMAN X)
>    :  (THERE-IS U (EQUAL X (INT U)))))  FOR I1 AND G2
>    :  :  COMPUTING DENOTATION OF  (IMPLIES (WOMAN X) (THERE-IS
>    :  :  U (EQUAL X (INT U))))  FOR I1 AND G2
>    :  :  :  COMPUTING DENOTATION OF  (WOMAN X)  FOR I1 AND G2
>    :  :  :  :  DENOTATION OF  WOMAN  IS  WOMANSET = {MAIC}
>    :  :  :  :  DENOTATION OF  X  IS  PRES = {<I1,MA>, <I2,BI>}
>    :  :  :  DENOTATION IS  0
>    :  :  DENOTATION IS  1
>    :  :  COMPUTING DENOTATION OF  (IMPLIES (WOMAN X) (THERE-IS
>    :  :  U (EQUAL X (INT U))))  FOR I2 AND G2
>    :  :  :  COMPUTING DENOTATION OF  (WOMAN X)  FOR I2 AND G2
>    :  :  :  :  DENOTATION OF  WOMAN  IS  WOMANSET = {MAIC}
>    :  :  :  :  DENOTATION OF  X  IS  PRES = {<I1,MA>, <I2,BI>}
>    :  :  :  DENOTATION IS  0
>    :  :  DENOTATION IS  1
>    :  DENOTATION IS  1
>    :  *NEW VARIABLE ASSIGNMENT G3: X=BTIC.
>    :  COMPUTING DENOTATION OF   (NECESSARILY (IMPLIES (WOMAN X)
>    :  (THERE-IS U (EQUAL X (INT U)))))  FOR I1 AND G3

>    :  DENOTATION IS  1
>    :  *NEW VARIABLE ASSIGNMENT G4: X=MAIC.
>    :  COMPUTING DENOTATION OF   (NECESSARILY (IMPLIES (WOMAN X)
>    :  (THERE-IS U (EQUAL X (INT U)))))  FOR I1 AND G4

>    :  DENOTATION IS  1
>    DENOTATION IS  1
>
>    ENTER POINT OF REFERENCE (NIL = NO MORE):
```

```
*nil
>
>    ENTER MEANINGFUL EXPRESSION (NIL = NO MORE):
*          ;comment - replacing woman* by its definition.
*(necessarily (iff (woman x) ((lambda u (woman* (int u))) (ext x)
*)))
>    EXPRESSION IS OF TYPE TS
>    FREE VARIABLES:  X.
>    OK?
*ok
>
>    ENTER POINT OF REFERENCE (NIL = NO MORE):
*i1
>    ENTER A VALUE FOR EACH VARIABLE (NIL =CANCEL):
>    X
*pres
>
>    INITIAL VARIABLE ASSIGNMENT G1: X=PRES.
>    COMPUTING DENOTATION OF  (NECESSARILY (IFF (WOMAN X) ((
>   LAMBDA U (WOMAN (INT U))) (EXT X))))  FOR I1 AND G1
>    : COMPUTING DENOTATION OF  (IFF (WOMAN X) ((LAMBDA U (WOMAN
>    : (INT U))) (EXT X)))  FOR I1 AND G1
>    :  : COMPUTING DENOTATION OF  (WOMAN X)  FOR I1 AND G1
>    :  :  : DENOTATION OF  WOMAN  IS  WOMANSET = {MAIC}
>    :  :  : DENOTATION OF  X  IS  PRES = {<I1,MA>, <I2,BI>}
>    :  : DENOTATION IS  0
>    :  : COMPUTING DENOTATION OF  ((LAMBDA U (WOMAN (INT U)))
>    :  : (EXT X))  FOR I1 AND G1
>    :  :  : COMPUTING DENOTATION OF  (LAMBDA U (WOMAN (INT U)))
>    :  :  : )  FOR I1 AND G1
>    :  :  :  : *NEW VARIABLE ASSIGNMENT G2: U=BI.
>    :  :  :  : COMPUTING DENOTATION OF  (WOMAN (INT U))  FOR I1
>    :  :  :  : AND G2
>    :  :  :  :  : DENOTATION OF  WOMAN  IS  WOMANSET = {MAIC}
>    :  :  :  :  : COMPUTING DENOTATION OF (INT U) FOR I1 AND G2
>    :  :  :  :  :  : DENOTATION OF  U  IS  BI
>    :  :  :  :  :  : DENOTATION OF  U  IS  BI
>    :  :  :  :  : DENOTATION IS  BIIC = {<I1,BI>, <I2,BI>}
>    :  :  :  : DENOTATION IS  0
>    :  :  :  : *NEW VARIABLE ASSIGNMENT G3: U=MA.
>    :  :  :  : COMPUTING DENOTATION OF  (WOMAN (INT U))  FOR I1
>    :  :  :  : AND G3
>    :  :  :  :  : DENOTATION OF  WOMAN  IS  WOMANSET = {MAIC}
>    :  :  :  :  : COMPUTING DENOTATION OF (INT U) FOR I1 AND G3
>    :  :  :  :  :  : DENOTATION OF  U  IS  MA
>    :  :  :  :  :  : DENOTATION OF  U  IS  MA
>    :  :  :  :  : DENOTATION IS  MAIC = {<I1,MA>, <I2,MA>}
>    :  :  :  : DENOTATION IS  1
>    :  :  :  : DENOTATION OF (LAMBDA U (WOMAN (INT U))) IS {MA}
>    :  :  :  : BUT IS NOT AN ELEMENT IN THE MODEL.
>    :  :  :  : ENTER A NAME FOR THIS ELEMENT IN D_<E,TS>:
*woman*set
>    :  :  : DENOTATION IS  WOMAN*SET = {MA}
```

```
>    :   :   :   COMPUTING DENOTATION OF   (EXT X)   FOR I1 AND G1
>    :   :   :   :   DENOTATION OF   X   IS   PROPS = {<I1,MA>, <I2,BI>}
>    :   :   :   DENOTATION IS   MA
>    :   :   DENOTATION IS   1
>    :   DENOTATION IS   [illegible]
>    DENOTATION IS   [illegible]
>
>    ENTER POINT OF REFERENCE (NIL = NO MORE):
*nil
>
>    ENTER MEANINGFUL EXPRESSION (NIL = NO MORE):
*nil
>    ENTER NEXT COMMAND:
* (stop)
*EXECUTION TERMINATED
```