Denotational Semantics for "Natural" Language Question-Answering Programs[1]

Michael G. Main[2]
David B. Benson

Department of Computer Science
Washington State University
Pullman, WA 99164-1210

Scott-Strachey style denotational semantics is proposed as a suitable means of communicating the specification of "natural" language question answerers to computer programmers and software engineers. The method is exemplified by a simple question answerer communicating with a small data base. This example is partly based on treatment of fragments of English by Montague. Emphasis is placed on the semantic interpretation of questions. The "meaning" of a question is taken as a function from the set of universes to a set of possible answers.

1. Introduction

We advocate the use of Scott-Strachey denotational semantics for "natural" language question-answering programs. The majority of this paper demonstrates the use of denotational semantics for a small question answerer. The types of questions possible are similar to those in Harris (1979), Winograd (1972), and Woods (1972). The analysis is not as deep as in Karttunen (1977) or similar studies, as it is oriented to the specification of useful, but linguistically modest, capabilities.

Before the demonstration, we discuss the benefits of formal semantics and why denotational semantics is an appropriate formalization. The semantics of a question answerer is given by defining the action of the program for each possible input. An informal semantic description, perhaps in narrative form, is necessary for a potential user who wants to know what questions he may ask and what sort of answers the program will provide. Informal meanings are also valuable to the designer and implementer of a question answerer. However, the designer and implementer must understand all aspects of a proposed question answerer in a precise unambiguous way that informal methods do not provide. In short, a formal method of specifying the semantics is needed at the design and implementation stage (see Ashcroft and Wadge 1982).

Once a formal semantics has been given, it can be put to other uses as well. It can provide the basis for a rigorous proof of correctness of an implementation. Furthermore, formal specifications might allow partial automation of the implementation process in the same way that automatic compiler-writers produce parts of a compiler from a formal specification of a programming language (see Johnson 1975). With the advent of at least one commercially available "natural" language question-answering program (Harris 1979), these advantages become even more concrete.

If there is a familiarity to these arguments, it is because the same reasoning is used to justify formal semantics for programming languages. The problems of design and analysis of question answerers are much the same for programming languages – Benson (1975) argues this point at length. An obvious approach is to apply a programming language technique – denotational semantics – to the problem of formally specifying a question answerer. As a bonus, the method is understood by many programmers and software engineers through introductory textbooks such as Stoy (1977), Gordon (1979), McGettrick (1980), Pagan (1981), and Tennent (1981). Additionally, linguistic treatments of natural language, such as Montague (1973), are basically denotational and we can modify such

[1] This research was supported in part by NSF grants MCS7708486 and MCS8003433.

[2] Current Address: Department of Computer Science, University of Colorado, Boulder, CO 80309.

0362-613X/83/010011-11$03.00

treatments to meet our needs.

In denotational semantics for programming languages, evaluation functions map program segments into objects in various semantic domains. These objects are taken as the meanings of the program segments, and determine the output of a program. Similarly, in the semantics of a question answerer, evaluation functions map input (questions) into objects that determine the output (answers).

Deciding what objects are in the semantic domains has a fundamental effect on the capabilities of the overlying question answerer, as well as an effect on the usefulness and clarity of the semantic descriptions. Lewis (1972) discusses these considerations for natural language sentences and the domains described in section 2 are based on his, although the treatment of questions is closer to Ajdukiewicz (1926). Also introduced in section 2 are the lambda expressions that denote individual semantic objects. Such expressions have been used in denotational semantics of programming languages (see Scott 1970, Milne and Strachey 1976, Stoy 1977) and in treatments of fragments of English by Montague (1973).

In section 3, evaluation functions mapping inputs for a small question answerer to objects in the semantic domain are given. The allowable questions are defined by a category grammar as has been done by Ajdukiewicz (1935), Carnap (1937), Lewis (1972), Montague (1973), and others. An account of the answer relationship is given in section 4.

In the final section, we briefly describe an implementation of the question answerer and suggest some broad principles for designing question answerers.

2. Semantic Domains for Natural Language

2.1 The Domains

A semantic domain is a set of objects. The objects are meanings of syntactic constructs; in our case the syntactic constructs are natural language phrases. In denotational semantics for programming languages, the semantic domains often have some order imposed on the objects to treat recursively defined functions. The specification of a question answerer may also involve explicit recursively defined questions, although in this demonstration only quantification is used and not recursion. This does not prevent an implementation from using recursion – almost surely it will, to handle the quantification. But lack of recursion does allow us to use unordered semantic domains, as described below.

One domain is the set of objects about which questions will be asked, e.g., moon rocks, toy blocks, or whatever. We are not concerned with the internal structure of these objects; hence they are called atoms and the domain is designated **A**. Two other domains are the set of natural numbers, designated **N**, and the set of truth values {true, false}, designated **T**.

A fourth domain is the set of possible universes, designated **U**. Intuitively, a universe is a description of properties of atoms, the relationships between atoms, the relationships between relationships, etc.... A universe is usually a partial description including only the properties and relationships of interest. For example, in the toy blocks world (Winograd 1972) a universe is the specification of the size, shape, color, and position of all the blocks in the scene. A toy blocks universe does not include a description of the density, mass, or material composition of the various blocks. In application, the set of possible universes may be called a "data base", and each actual universe is a specific state or configuration of the data base. The internal structure of such a data base is left unspecified in this paper.

Other semantic domains are built from the four basic domains. For example, the set of functions from universes to truth values, designated **[U→T]**, is a new domain. In general, if **X** and **Y** designate sets, then **[X→Y]** designates the set of functions from **X** to **Y**. Other semantic domains could be conceived (e.g., product or sum domains), but function domains will be adequate for our examples.

2.2 Assigning domains to syntactic categories

Natural language phrases have been divided into many different categories (see Kahn 1978). Exactly which categories are used depends upon syntax. In the category grammar of section 3, the categories include sentence, intransitive verb, common noun, noun group, noun modifier, numeral, and so on. There are also separate interrogative categories for phrases that ask a question, such as *how many stones.*

To each syntactic category, a semantic domain is assigned. The meaning of a phrase is an object in the assigned domain. Ultimately, from the meaning of a question, we will derive its answer. But first we assign domains to non-interrogative categories, beginning with the category of sentences.

A declarative sentence is a proposition – that is, something to which a truth value can be assigned. The meaning of a sentence somehow determines what that truth value is in any possible universe. So, an appropriate semantic object for a sentence is a function from possible universes to truth values. The domain assigned to the category of sentences is therefore **[U→T]**, which we will designate **S**. If a declarative sentence has a meaning $\alpha \in \mathbf{S}$, and β is a universe, then $\alpha\beta$ is true just if the sentence is true in the universe β.[3]

[3] Lower case Greek letters are semantic objects. Juxtaposition indicates functional application and associates left-to-right. Thus, $\alpha\beta$ means the function α applied to the argument β, and $\alpha\beta\gamma$ means $(\alpha(\beta))(\gamma)$.

If there is a single fixed universe, then sentences could be assigned meanings in **T** alone. However, question answerers are generally based on a changeable universe or data base. Hence, the [**U**→**T**] approach is needed to give a fixed meaning to a sentence whose truth value may vary from universe to universe.

Both common nouns and intransitive verbs define subsets of **A**. The subset of *stone* (a common noun) is the set of atoms that are stones. The subset for the intransitive verb phrase *belong to Preston* is the set of atoms that belong to Preston. The meaning of such a phrase determines what that subset is for any possible universe. This can be done by an object in [**U**→[**A**→**T**]] (i.e., functions from **U** to [**A**→**T**]). This domain is assigned to both common noun and intransitive verb categories, and is designated by **C**. If a common noun has a meaning $\alpha \in \mathbf{C}$, and β is a universe and γ an atom, then $\alpha\beta\gamma$ is true just if γ is in the subset defined by the common noun in the universe β – and similarly for intransitive verbs.

For most other syntactic categories, the semantic domain is determined by how phrases in that category combine with sentences, common nouns, intransitive verbs, and other phrases to form new phrases. For example, a noun group combines with an intransitive verb to form a sentence. Therefore, a semantic object for a noun group should take a semantic object from **C** (the domain for intransitive verbs) and yield a semantic object from **S** (the domain for sentences). Thus, the semantic domain for noun groups is [**C**→**S**].[4]

Similarly, a noun modifier, such as *black,* combines with a common noun, such as *stone,* to form a new common noun – *black stone.* So, the semantic domain for noun modifiers is [**C**→**C**]. By examining the syntactic rules, this method can be applied to most categories.[5] Occasional exceptions can be made – for example, numerals are assigned the domain, **N**, of natural numbers.

2.3 Interrogative categories

So far, the categories include only non-interrogative phrases. Syntactically, the interrogative phrases parallel the non-interrogatives, but semantically an interrogative lacks something. For example, Ajdukiewicz (1926) would represent the meaning of the interrogative sentence *what is black and white and read all over* as: ?x(x *is black and white and read all over).* Here, x is a variable and "?" a quantifier indicating the lack. In this case, the range of the variable is the set of noun group phrases that could answer the question.

There are also interrogative phrases in other categories, such as the intransitive verb phrase *owns what.* And it is not always a noun group that is missing; for example, the noun group *how many stones* lacks a numeral. In general, any non-interrogative category is made into an interrogative category by indicating what is lacking. Hence, if *X* and *Y* are non-interrogative categories, then *X?Y* designates the category of interrogative phrases that are syntactically like *X*-phrases but semantically lack a *Y*-phrase.[6] If *Y* has semantic domain **Y** and *X* has semantic domain **X**, then *X?Y* will have semantic domain [**Y**→**X**]. Furthermore, if an interrogative phrase in *X?Y* has a meaning $\alpha \in$ [**Y**→**X**], then $\alpha\beta$ is that object of **X** that results when $\beta \in$ **Y** answers the interrogative.

For example, the interrogative sentence *who killed cock robin* will have a meaning $\alpha \in$ [[**C**→**S**]→**S**], since it is a sentence (domain **S**) lacking a noun group (domain [**C**→**S**]). If *the sparrow* has a meaning $\beta \in$ [**C**→**S**], and *the sparrow killed cock robin* has a meaning $\gamma \in$ **S**, then $\alpha\beta = \gamma$.

A similar idea can be applied to one other category: yes–no questions. Such an interrogative lacks a truth value, so the semantic domain assigned to yes–no questions is [**T**→**S**]. If $\alpha \in$ [**T**→**S**] is the meaning of a yes–no question, then α(true) $\in$ **S** is the meaning of the sentence when it is answered by *yes,* and similarly for an α(false).

2.4 Lambda expressions

Up to this point, semantic objects have been described in English. In order to be more precise, a formal notation is needed. We use a typed lambda expression for denoting functions, similar to the lambda calculi of Church (1951).

Every lambda expression has a type that indicates the semantic domain of the object denoted by the expression. These types are in one-to-one correspondence with the semantic domains (**A**, **U**, **T**, **N**, [**U**→**T**], etc....), so we will use the same letters in ordinary typescript for the expression types (A, U, T, N, [U→T], etc....).

The expressions of each type include a set of constants and a denumerable set of variables. A function that maps the constants into semantic objects is an interpretation, and generally remains fixed. A function that maps variables into semantic objects is a variable assignment and, as its name suggests, will

[4] A name, such as *John,* is one type of noun group that denotes a particular atom which does not change from universe to universe Therefore, it might be better to have names in a separate syntactic category with assigned semantic domain **A**. But, doing this would not do away with the category noun group, since some noun phrases are not names (e.g., *every stone*). But, having two categories unduly complicates the syntax, so we lump names together with other noun groups, and treat them as Montague (1973) has.

[5] This method of assigning semantic domains is an application of Frege's rule of compositionality, which says that the meaning of a compound phrase is composed from the meaning of its parts. (Frege 1892, van Emde Boas and Janssen 1979)

[6] This could be extended to include phrases, like *who does what,* which question more than one thing at a time.

change. If ψ is a variable assignment, x is a variable of any type, and α is an object from the semantic domain of x, then $\psi[\alpha/x]$ is a changed variable assignment that is like ψ except that $\psi[\alpha/x]$ assigns α to the variable x. In this way, new variable assignments are formed from old. Finally, each lambda expression denotes one semantic object, but which object this is varies as the variable assignment changes. So, for an expression b the notation $\|b\|\psi$ is to denote the semantic object for the expression b, with variable assignment ψ.[7]

Table 1 gives recursive rules defining lambda expressions and the semantic objects they denote. We take as given a set of constants and a denumerable set of variables for each type, and a fixed interpretation, φ. X and Y are arbitrary types, with corresponding semantic domains **X** and **Y**.

Some lambda expressions can be derived more than one way. Any such ambiguities are resolved as follows:

a. The expression following a lambda abstraction (i.e., λx) or a quantifier (i.e., $\exists x$, $\forall x$, $\exists nx$,) will be as long as possible – that is, to the first unmatched right parenthesis or to the end of the entire expression.
b. The unary operator $\neg$ has higher priority than any of the binary operators ($\wedge$, $\vee$, $=$, $\Rightarrow$, $\Leftrightarrow$, and functional application, which is indicated by juxtaposition). Functional application has the highest priority of the binary operators and associates left-to-right, e.g., bcd=(bc)d. The other binary operators have equal precedence and also associate left-to-right, e.g., bc$\wedge$dc$\vee$c = ((bc)$\wedge$(dc))$\vee$c.

3. Syntax and Semantics of a Small Question Answerer

3.1 Syntactic categories

To illustrate the mechanics of the previous section, we give the syntax and semantics of a small question answerer. The question answerer accepts questions about configurations of pieces in the game of Go. Briefly, a configuration consists of a 19 by 19 grid of points, labeled A-1 through S-19. Each point may contain a black or a white stone. Like-colored stones, connected horizontally or vertically, form blocks. Empty points adjacent to a block are that block's liberties. Each configuration is a universe in the semantic domain **U**.

The syntax for a question answerer must answer the question: what phrases are grammatically correct input to the question answerer? Our approach uses various syntactic categories, in the style of Montague (1973). A syntactic category is a set of phrases, including basic phrases and derived phrases. The derived phrases are specified by recursive syntactic rules that describe how phrases from various categories combine to form new phrases.

The syntax presented here has two kinds of categories. First, non-interrogative categories, whose phrases do not ask questions. These categories are listed in Table 2, along with an abbreviation for each, the lists of basic expressions, and the associated semantic domains from section 2. In the table, A-1, A-2, and so on are names of points on the Go board. The basic phrases *he*$_0$, *he*$_1$, *they*$_0$, *they*$_1$, and so on are used as variables in a manner made precise by syntactic rules given later.

The second kind of category is the interrogative category. Syntactically, an interrogative category behaves like some non-interrogative category, but it also asks a question whose answer is from some other non-interrogative category. If a phrase is syntactically like a phrase from category *X*, and asks a question whose answer is in category *Y*, then that phrase is in a category *X?Y*. This separation of interrogatives is needed because the semantic domain for interrogatives differs from that for non-interrogatives (see section 2.3). One additional interrogative category does not follow the *X?Y* pattern – the yes–no questions. We designate this category *YN* and list it in Table 3 with the other interrogative categories.

3.2 Syntactic and semantic rules

We now give syntactic rules that define the phrases of each category. With each syntactic rule is a semantic rule. For each phrase that a syntactic rule creates, the corresponding semantic rule tells precisely how to translate that phrase into a lambda expression denoting the meaning of the phrase. In these rules, italic letters (*u, v, w,* ...) are arbitrary phrases from any category and F_1, F_2, ..., are functions that combine phrases to yield new phrases. Variables in lambda expressions are of the following types.

- m,m',...of type N.
- t,t',... of type T.
- u,u',... of type U.
- x,x',... of type C.
- y,y',... of type [C→S].
- q_i is the (2i+3)th variable of type A.
- p_i is the (2i+4)th variable of type A.
- z is the first variable of type A.
- z' is the second variable of type A.

[7] The semantic object associated with a lambda expression also depends on the interpretation function, but we assume this is fixed.

LAMBDA EXPRESSION	SEMANTIC OBJECT
1. If b is a constant of type X, then b is an expression of type X.	$\|b\|\psi$ is φb (i.e., the interpretation φ applied to the constant b).
2. If x is a variable of type X, then x is an expression of type X.	$\|x\|\psi$ is ψx (i.e., the variable assignment ψ applied to the variable x).
3. If b is an expression of type X, then (b) is also an expression of type X.	$\|(b)\|\psi$ is $\|b\|\psi$.
4. If b is an expression of type Y and x is a variable of type X, then λx.b is an expression of type [X$\rightarrow$Y].	$\|\lambda x.b\|\psi$ is that function α:**X**$\rightarrow$**Y**, such that for every $\beta\in$**X**, $\alpha\beta = \|b\|\psi[\beta/x]$.
5. If b is an expression of type [X$\rightarrow$Y] and c is an expression of type X, then bc is an expression of type Y.	$\|bc\|\psi$ is the function $\|b\|\psi$ applied to the argument $\|c\|\psi$.
6. If b and c are expressions of type X, then b = c is an expression of type T.	$\|b = c\|\psi$ is true iff $\|b\|\psi$ is the same semantic object as $\|c\|\psi$.
7. If b and c are expressions of type T, then $\neg$b, b$\wedge$c, b$\vee$c, b$\Rightarrow$c, and b$\Leftrightarrow$c are expressions of type T.	$\|\neg b\|\psi$ is true iff $\|b\|\psi$ is false, and similarly for $\wedge$ (and), $\vee$ (or), $\Rightarrow$ (implication), and $\Leftrightarrow$ (coincidence).
8. If n is an expression of type N, then n+1 is an expression of type N.	$\|n+1\|\psi$ is the natural number successor of $\|n\|\psi$.
9. If b is an expression of type T and x is a variable of type X, then $\exists$x.b and $\forall$x.b are expressions of type T.	$\|\exists x.b\|\psi$ is true iff there exists an $\alpha\in$**X**, such that $\|b\|\psi[\alpha/x]$ is true. $\|\forall x.b\|\psi$ is $\|\neg\exists x.\neg b\|\psi$.
10. If b is an expression of type T, x is a variable of type X and n is an expression of type N, then $\exists$nx.b is an expression of type T.	Let $\eta = \|n\|\psi$. Then $\|\exists nx.b\|\psi$ is true iff there exist η distinct objects in **X**, so that for any of these η objects, say α, $\|b\|\psi[\alpha/x]$ is true.

Table 1. Lambda expressions and their semantic objects.

SYNTACTIC CATEGORY	SEMANTIC DOMAIN & BASIC PHRASES
SE (sentence)	**S** = [**U**→**T**]. No basic phrases.
CN (common noun)	**C** = [**U**→[**A**→**T**]]. *player, block, stone, liberty, point*
IV (intransitive verb)	**C**. *exist*
NG (noun group)	[**C**→**S**]. *Black, White,* he_0, he_1,..., $they_0$, $they_1$,..., *A-1, A-2, ...*
TV (transitive verb)	[[**C**→**S**]→**C**]. *own, belong to*
NM (noun modifier)	[**C**→**C**]. *black, white*
MG (modifying group)	[**C**→**C**]. No basic phrases.
PP (preposition or participial)	[[**C**→**S**]→[**C**→**C**]]. *at, with, owning, belonging to*
NU (numerals)	**N** *0, 1, 2, ...*

Table 2. Syntactic categories (non-interrogative).

SYNTACTIC CATEGORY	SEMANTIC DOMAIN & BASIC PHRASES
YN (yes-no question)	[**T**→**S**]. No basic phrases.
NG?NG (noun group questioning a noun group)	[[**C**→**S**]→[**C**→**S**]]. *who, what*
For any two non-interrogative categories, *X* and *Y*, *X?Y* is an interrogative category.	[**Y**→**X**], where **X** is *X*'s domain and **Y** is *Y*'s domain. None of these categories, except *NG?NG*, have basic phrases.

Table 3. Syntactic categories (interrogative).

3.2.1 The basic rule

Syntactic Rule:

R1. For each category, its basic phrases are phrases.

Semantics:

We give a translation of each basic phrase, according to which category it is in:

CN,IV: *player* translates to a constant of type C, denoted **player;** and similarly for any other basic phrase of category *CN* or *IV*.

NG: *Black* translates to λx.λu.xu(**Black**), where **Black** is a constant of type A. Having **Black** be a constant of type A emphasizes the point that *Black* is a name, associated with some particular object in **A**. The translation of other basic phrases of *NG* will be similar, except for he_i and $they_i$ (i = 0, 1, ...). In particular, he_i translates to λx.λu.xuq_i and $they_i$ translates to λx.λu.xup_i.

TV: *own* translates to λy.λu.λz.y(λu'.(**own**)u'z)u, where **own** is a constant of type [U → [A → [A → T]]]; and similarly for any other basic phrase of category *TV*. This emphasizes that a transitive verb is a relation between two objects in **A**. In particular, if **own** interprets to $\alpha \in$ [U → [A → [A → T]]], $\beta \in$ U, and γ, $\delta \in$ **A**, then $\alpha\beta\gamma\delta$ if true is the object γ owns the object δ in the universe β.[8]

NM: *black* translates to λx.λu.λz.xuz ∧ (**black**)uz, where **black** is a constant of type C; and similarly for other basic phrases of category *NM*.

PP: *at* translates to λy.λx.λu.λz.xuz ∧ y(λu'.(**at**)u'z)u, where **at** is a constant of type [U → [A → [A → T]]]; and so on.

NU: *0* translates to **0**, a constant of type N, which interprets to the natural number 0 in **N**; and similarly for other numerals.

NG?NG: *what* translates to λy.y and *who* translates to λy.λx.λu.y(**player**)u ∧ yxu.

3.2.2 The combinative rules

These rules use a set of syntactic functions, F_1 through F_4, which combine phrases in various ways involving person, plurality, and so on. We give these functions first.

$F_1(u,v)$ is uv', where v' is v (if the first noun in u is plural) or the result of replacing the first verb in v by its third person singular form (if the first noun in u is singular).

$F_2(u,v)$ is uv', where v' is the result of replacing the first noun in v by its objective form.

$F_3(u,v) = uv$.

$F_4(u,v) = vu$.

Syntactic Rules:

R2. If u is a phrase from *NG* and v is a phrase from *IV*, then $F_1(u,v)$ is a phrase from *SE*.

R3. If u is a phrase from *TV* and v is a phrase from *NG*, then $F_2(u,v)$ is a phrase from *IV*.

R4. If u is a phrase from *PP* and v is a phrase from *NG*, then $F_2(u,v)$ is a phrase from *MG*.

R5. If u is a phrase from *NM* and v is a phrase from *CN*, then $F_3(u,v)$ is a phrase from *CN*.

R6. If u is a phrase from *MG* and v is a phrase from *CN*, then $F_4(u,v)$ is a phrase from *CN*.

Interrogative Variants of Syntactic Rules:

In any of the rules R2 through R6, exactly one of the arguments, u or v, may be a phrase from category *X?Y*, where *X* is the original category specified for the argument and *Y* is any non-interrogative category. If the original result was to be from category *Z*, then the new result is in category *Z?Y*. For example, from R2, if u is a phrase from *NG?NG* and v is a phrase from *IV*, then $F_1(u,v)$ is a phrase from *SE?NG*.

Semantics:

Let $w = F_i(u,v)$, for i = 1, 2, 3, or 4. Let u translate to b and v translate to c. Then there are three cases for the translation of w:

Case (i): If neither u nor v is from an interrogative category, then w translates to bc.

Case (ii): If u is from an interrogative category *X?Y*, then w translates to λs.bsc, where s is a variable of type assigned to the category *Y*.

Case (iii): If v is from an interrogative category *X?Y*, then w translates to λs.b(cs), where s is a variable of type assigned to the category *Y*.

3.2.3 Extracategorical rules

These rules combine phrases with words from outside the categories to form new phrases. The rules use two syntactic functions given here:

$F_5(u,v)$ is vu', where u' is u (if $v = 1$), or the result of replacing the first noun in u by its plural form (if $v \neq 1$).

$F_6(u)$ is the result of replacing the first noun in u with its plural form.

Syntactic Rules and Semantics:

R7. Let u be a phrase from *CN*, translating to b.

[8] However, this scheme will not work with all transitive verbs, for example, *alleged to be*. Such verbs, called intensional verbs, require more complex translations, but the end result will still be of type [[C → S] → C]. The same complexities arise for *NM* and *PP*. Our example contains none of these intensional words.

Then:

(i) *every u* is a phrase from *NG*, translating to $\lambda x.\lambda u.\forall z.buz \Rightarrow xuz$.

(ii) *some u* is a phrase from *NG*, translating to $\lambda x.\lambda u.\exists z.buz \wedge xuz$.

(iii) *the u* is a phrase from *NG*, translating to $\lambda x.\lambda u.\exists z.xuz \wedge \forall z'.\ (buz' \Leftrightarrow (z=z'))$.

(iv) *no u* is a phrase from *NG*, translating to $\lambda x.\lambda u.\neg\exists z.buz \wedge xuz$.

R8. Let *u* be a phrase from *CN*, translating to b. Also let *v* be a phrase from *NU*, which translates to n. Then:

(i) *exactly* $F_5(u,v)$ is a phrase from *NG*, translating to $\lambda x.\lambda u.(\exists nz.buz \wedge xuz) \wedge \neg(\exists(n+1)z.buz \wedge xuz)$.

(ii) *at least* $F_5(u,v)$ is a phrase from *NG*, translating to $\lambda x.\lambda u.\exists nz.buz \wedge xuz$.

(iii) *less than* $F_5(u,v)$ is a phrase from *NG*, translating to $\lambda x.\lambda u.\neg(\exists nz.buz \wedge xuz)$.

R9. Let *u* be a phrase from *SE*, translating to b. Then *is it the case that u* is a phrase from *YN*, translating to $\lambda t.\lambda u.bu \Leftrightarrow t$.

R10. Let *u* be a phrase from *CN*, translating to b. Then:

(i) *what* $F_6(u)$ is a phrase from *NG?NG*, translating to $\lambda y.\lambda x.\lambda u.ybu \wedge yxu$.

(ii) *how many* $F_6(u)$ is a phrase from *NG?NU*, translating to $\lambda m.\lambda x.\lambda u.(\exists mz.buz \wedge xuz) \wedge \neg(\exists(m+1)z.buz \wedge xuz$.

3.2.4 The abstraction rule

This rule is to replace the variables he_i and $they_i$ by other noun phrases. The necessity for abstraction is discussed in Lewis (1972). The syntactic functions F_7 and F_8, defined below, are used. In these definitions, let u' be the result of replacing the first noun in *u* with its objective form, let u_p be the result of replacing the first noun in *u* with its plural form, and let u'_p be the result of replacing the first noun in *u* with its plural objective form.

$F_7(u,v,i)$ is the result of replacing, in *v*, all occurrences of he_i by *u* and all occurrences of him_i by u'.

$F_8(u,v,i)$ is the result of replacing in *v*, all occurrences of $they_i$ by u_p and all occurrences of $them_i$ by u'_p.

Syntactic Rule:

R11. Let *u* be a phrase from *NG*, *v* be a phrase from *SE*, *IV*, or *CN*, and i be a variable index (e.g., 0, 1, 2, ...). Then $F_7(u,v,i)$ and $F_8(u,v,i)$ are phrases from the same category as *v*.

Interrogative Variant of Syntactic Rule:

Either *u* may be from *NG?Y* or *v* may be from *SE?Y*, *IV?Y*, or *CN?Y* (but not both), where *Y* is any non-interrogative category. In all cases, the result is from *X?Y*, where *X* is the first portion of the category of *v*.

Semantics:

Let b be the translation of *u* and c be the translation of *v*. The translation of $F_7(u,v,i)$ is given in the following table, where s is a variable of type *Y*:

Case 1: *v* from *SE* and *u* from *NG*: $b(\lambda u.\lambda q_i.cu)$

Case 2: *v* from *SE* and *u* from *NG?Y*: $\lambda s.bx(\lambda u.\lambda q_i.cu)$

Case 3: *v* from *IV* or *CN* and *u* from *NG*: $\lambda u.\lambda z.b(\lambda u'.\lambda q_i.cu'z)u$

Case 4: *v* from *IV* or *CN* and *u* from *NG?Y*: $\lambda s.\lambda u.\lambda z.bs(\lambda u'.\lambda q_i.cu'z)u$

Case 5: *v* from *SE?Y* and *u* from *NG*: $\lambda s.b(\lambda u.\lambda q_i.csu)$

Case 6: *v* from *IV?Y* or *CN?Y* and *u* from *NG*: $\lambda s.\lambda u.\lambda z.b(\lambda u'.\lambda q_i.csu'z)u$

The translation is identical for F_8, except that q_i is replaced by p_i.

Some of the notions in the syntactic rules must still be formalized. We must define the plural, objective, and plural objective forms of each basic phrase in *CN*, *NG*, and *NG?NG*. A noun is then any such basic phrase or one of these forms. A verb is any basic phrase in *IV* or *TV*. For each verb we must define its third person singular form.

A brief discussion of the abstraction rules can clarify their usage and purpose. A sentence such as *every player owns some stone* has two possible meanings. It can mean: there is some particular stone owned by every player; or alternately, every player owns at least one stone, but not necessarily the same stone for each player. These two meanings will be achieved by introducing the *NG some stone* at different times. The abstraction rules allow this by delaying the introduction of a noun phrase. Initially, a variable (perhaps he_0) is put in the sentence as a place holder. The abstraction rules allow an *NG* to later replace the variable.

4. Questions and Answers

4.1 Questions

A *question* is any phrase from category *YN*, *SE?NG*, or *SE?NU*, along with its syntactic derivation. The syntactic derivation is needed because some phrases can be derived in more than one way (e.g., *is it the case that every player owns some stone).*

Derivations are represented by trees. Each leaf in a derivation tree is labeled with a basic phrase. Each internal node is labeled with a derived phrase, plus the number of the syntactic rule that is used to derive it from its daughter leaves.

From the semantic rules, every question translates to exactly one lambda expression.

4.2 Answers

Let Q be a question that translates to a lambda expression, b, of type [X → S]. Also let:

- ψ be a variable assignment,
- g be a constant of type U and $\|g\|\psi = \gamma$,
- d be a constant of type X and $\|d\|\psi = \delta$.

Then δ is an *answer* to Q, in the universe γ with variable assignment ψ iff $\|bdg\|\psi$ is true.

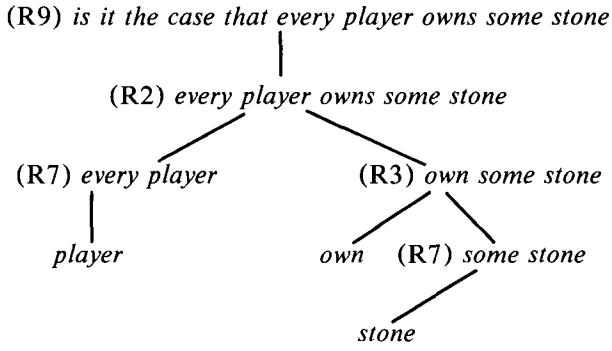
If b is an expression of type [X → S], then the answers to its corresponding question are semantic objects in X. In general, these objects depend on the choice of the universe and also on the variable assignment. However, if the question does not contain any variables (*he*$_i$ or *they*$_i$), then the set of answers is the same for any variable assignment. Such a question is called *invariable,* and we may speak of an answer without respect to a variable assignment.

4.3 Examples

All of these examples are invariable questions, so we may choose a variable assignment, ψ, at random. The types of variables are as in section 3.2, and we also use a lambda constant g, of type U. As always, there is a fixed interpretation assigning the lambda constants to semantic objects.

In translating the examples to lambda expressions, we use the semantic rules of section 3.2. However, after translating a phrase, we will sometimes alter the lambda expression in ways that cannot change the semantic object.[9]

Example 1. One derivation of the phrase *is it the case that every player owns some stone* is:



We have these translations:

own some stone:
λu.λz.∃z'.(**stone**)uz' ∧ (**own**)uzz'

every player:
λx.λu.∀z.(**player**)uz ⇒ xuz

every player owns some stone:
λu.∀z.(**player**)uz ⇒ (∃z'.(**stone**)uz' ∧ (**own**)uzz')

is it the case that every player owns some stone:
λt.λu.(∀z.(**player**)uz ⇒ (∃z'.(**stone**)uz' ∧ (**own**)uzz')) ⟺ t

Let **true** be a constant of type T that interprets to 'true'∈**T**. From the definition of an answer, 'true' is an answer to this question in the universe $\|g\|\psi$ iff:

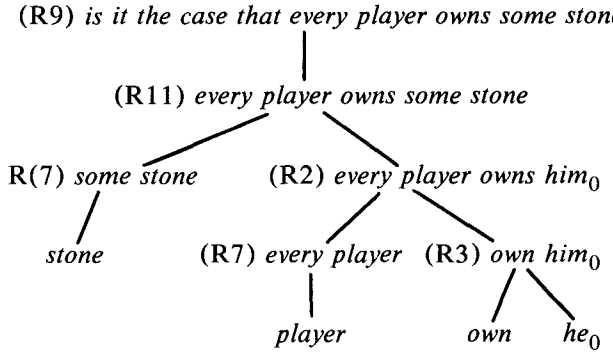
$\|$(λt.λu.(∀z.(**player**)uz ∧ (∃z'.(**stone**)uz' ∧ (**own**)uzz')) ⟺ t)(**true**)g$\|\psi$

Here, ψ can be picked at random. The above expression converts to

$\|$∀z.(**player**)gz ⇒ (∃z'.(**stone**)gz' ∧ (**own**)gzz')$\|\psi$

This is a reasonable condition for 'true' to answer the question.

Example 2. An alternative derivation for the previous phrase is:



In this case, the translations are:

own him$_0$:
λu.λz.(**own**)uzq$_0$

every player owns him$_0$:
λu.∀z.(**player**)uz ⇒ (**own**)uzq$_0$

every player owns some stone:
λu.∃z'.(**stone**)uz' ∧ (∀z.(**player**)uz ⇒ (**own**)uzz')

is it the case that every player owns some stone:
λt.λu.(∃z'.(**stone**)uz' ∧ (∀z.(**player**)uz ⇒ (**own**)uzz')) ⟺ t

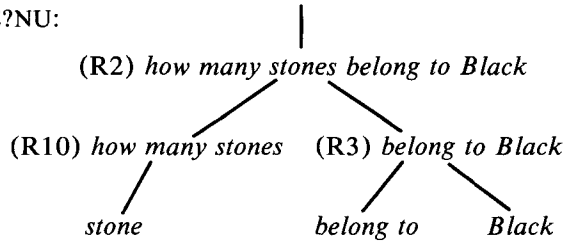
In the universe $\|g\|\psi$, 'true' is an answer to this question iff:

$\|$∃z'.(**stone**)gz' ∧ (∀z.(**player**)gz ⇒ (**own**)gzz')$\|\psi$

This contrasts with example 1.

[9] In particular, we use logical conversion (such as b ∧ b converts to b) and the α- and β-conversions of lambda calculus. α-conversion involves changing the name of a bound variable (e.g., λx.x ∧ y converts to λz.z ∧ y). β-conversion corresponds to functional application (e.g., (λx.x ∧ y)z converts to z ∧ y). Details can be found in Stoy (1977).

Example 3. Here is a derivation of a phrase from SE?NU:



The semantic rules give these translations:

belong to Black:
λu.λz.(**belong to**)uz(**Black**)

how many stones:
λm.λx.λu.(∃mz.(**stone**)uz ∧ xuz) ∧
¬(∃(m+1)z.(**stone**)uz ∧ xuz)

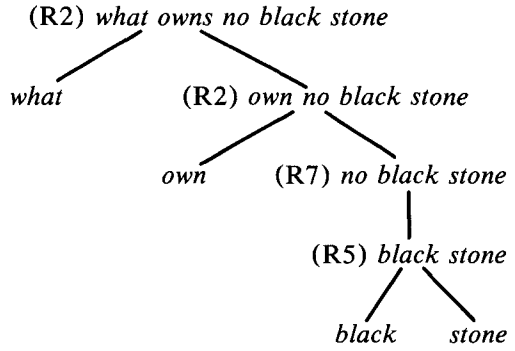
how many stones belong to Black:
λm.λu.(∃mz.(**stone**)uz ∧ (**belong to**)uz(**Black**)) ∧
¬(∃(m+1)z.(**stone**)uz ∧ (**belong to**)uz(**Black**))

Let $\eta = \|n\|\psi$ be a natural number. Then η answers this question in the universe $\|g\|\psi$, iff:

$\|$(∃nz.(**stone**)gz ∧ (**belong to**)gz(**Black**)) ∧
¬(∃(n+1)z.(**stone**)gz ∧ (**belong to**)gz(**Black**))$\|\psi$

That is, there exists η stones (but not $\eta+1$) that belong to Black in the universe $\|g\|\psi$.

Example 4. This question is from *SE?NG*.



Here are the translations to lambda expressions:

black stone:
λu.λz.(**stone**)uz ∧ (**black**)uz

no black stone:
λx.λu.¬∃z.(**stone**)uz ∧ (**black**)uz ∧ xuz

own no black stone:
λu.λz'.¬∃z.(**stone**)uz ∧ (**black**)uz ∧ (**own**)uz'z

what owns no black stone:
λy.y(λu.λz'.¬∃z.(**stone**)uz ∧ (**black**)uz ∧
(**own**)uz'z)

The semantic object corresponding to *White* (namely, $\|\lambda x.\lambda u.xu(\textbf{White})\|\psi$) answers this question in a universe, $\|g\|\psi$, iff:

$\|\neg\exists z.(\textbf{stone})gz \wedge (\textbf{black})gz \wedge (\textbf{own})g(\textbf{White})z\|\psi$

Any realistic interpretation interprets the constants **black, own,** and **White** so that $\|(\textbf{black})gz\|\psi$ and $\|(\textbf{own})g(\textbf{White})z\|\psi$ are mutually exclusive for any g, z, and ψ, so that the semantic object for *White* does answer this question.

The difficulty with examples such as these is that even when the translations of individual constructions are fairly simple the translation of a complex construction inevitably appears cryptic and the translation process is tedious. However, the formal translations are amenable to mechanical manipulations and the translating process is also easily mechanized. Hence, we leave it as an exerciue in the manipulations to show that the translation of *what blocks own at least 3 stones with less than 2 liberties* is as follows:

λy.λu.y(**block**)u ∧ y(λu.λz".∃3z'.(**stone**)uz' ∧
¬(∃2z.(**liberty**)uz ∧ (**with**)uz'z) ∧ (**own**)uz"z')u

The first part of the translation guarantees that a possible answer, y, is indeed a block. The second part checks that y owns at least 3 stones with less than 2 liberties. Note that, although there may be other parsings of this sentence in a complete English grammar, our simple syntactic rules have forced this particular translation upon us.

Answers to *SE?NG* questions are objects from [**C** → **S**]. However, it might be more convenient to have these answers be objects from **A**. A modification to the definition of an answer could allow this. In particular, let b be a lambda constant of type A, which interprets to $\beta \in \textbf{A}$. Then β could answer a question in the universe $\|g\|\psi$, with variable assignment ψ, iff $\|\lambda x.\lambda u.xub\|\psi \in$ [**C** → **S**], does.

5. Discussion

We have used denotational techniques to define the semantics of a "natural" language question answerer. The questions are defined by syntactic rules – a category grammar. Associated with each syntactic rule is a semantic rule, giving a semantic object for each phrase the syntactic rule produces. The semantic objects for questions are functions, from possible answers to propositions, where a proposition is an object that takes on a truth value. Thus, if a question, *Q*, has a semantic object that maps an answer, *A*, to a true proposition, then *A* answers *Q*. In this way, the relation between questions and answers is formalized.

While it is not our intent to give implementation details for the question answerer, an outline of the program's organization will lead to some concluding remarks. The program was written as a class project by the first author and four other students. The logical divisions of the program were as follows:

- *Lexical analysis.* The input is broken into basic component words, or "tokens", using finite automata techniques as in Johnson et al. (1968).
- *Syntactic parser.* An augmented transition network (Woods 1970) parses the input according to the category grammar with heuristic rules and interactive query to solve ambiguities. The output is a single derivation tree.
- *Semantic interpretation.* The "meaning" of the sentence is computed according to the semantic rules. The output of this phase can be thought of as "machine code" for finding answers. At this stage, the "code" is independent of the actual structure of the implementation of the underlying data base.
- *Deductive components.* The "code" from the previous stage is improved based on conversion rules of logic, such as x or $x = x$. It is also modified according to the specific structure of the data base. The latter modifications are implementation dependent.
- *Answerer.* This corresponds to a database query-retrieval program. The "code" from the previous step is executed to produce an answer.

In the actual implementation, the above stages are not strictly separate; still, the model is useful. There is a direct correspondence between the first four stages of the implementation and the initial four stages of a typical compiler (Aho and Ullman 1977). Hobbs and Rosenschein (1977) indicate how these last three stages could be developed using an augmented LISP as "code".

In this paper, we have recommended using denotational semantics as a specification technique for question-answering programs. The implementation suggests that principles of compiler design can be used as principles of question answerer design by the software engineer.

References

Aho, A.V. and Ullman, J.D. 1977 *Principles of Compiler Design.* Addison-Wesley, Reading, Massachusetts.

Ajdukiewicz, K. 1926 The Semantic Analysis of Interrogative Sentences, *Ruth Filozoficzny X.*

Ajdukiewicz, K. 1935 Syntactic Connexion. In McCall, S., Ed., *Polish Logic, 1920-1939.* Clarendon, Oxford (1967).

Ashcroft, E.A. and Wadge, W.W. 1982 R_x for Semantics. *ACM Trans. Prog. Lang. and Sys.* **4:** 283-294.

Benson, D.B. 1975 Formal Languages vis-a-vis 'Natural' Languages. In Sedelow, W. and Sedelow, S., Ed., *Computers in Language Research: Trends in Linguistics.* Mouton, the Hague (1979).

Carnap, R. 1937 *The Logical Syntax of Language.* Smeaton, A., Trans. Kegan Paul, Trench, Trubner and Son, Ltd., London.

Church, A. 1951 *The Calculi of Lambda-Conversion.* Princeton University Press, Princeton, New Jersey.

Frege, G. 1892 On Sense and Reference. In Geach, P. and Black, M., Ed., *Translations from the Philosophical Writings of Gottlob Frege.* Basil Blackwell, Oxford (1952).

Gordon, M.J.C. 1979 *The Denotational Description of Programming Languages.* Springer-Verlag, New York.

Harris, L.R. 1979 Experience with ROBOT in 12 Commercial Natural Language Data Base Query Applications, *Proc. 6th International Joint Conference on Artificial Intelligence.* Tokyo: 365-368.

Hobbs, J.R. and Rosenschein, S.J. 1977 Making Computation Sense of Montague's Intensional Logic, *Artificial Intelligence* **9:** 287-306.

Johnson, S.C. 1975 YACC – Yet Another Compiler Compiler. CSTR 32. Bell Laboratories, Murray Hill, New Jersey.

Johnson, W.L.; Porter, J.H.; Ackley, S.I.; and Ross, D.T. 1968 Automatic Generation of Efficient Lexical Analyzers Using Finite State Techniques, *Communications of the ACM* **11**(12): 805-813.

Kahn, C. 1978 Questions and Categories. In Hiz, H., Ed., *Questions.* D. Reidel Publishing Co., Dordrecht, Holland.

Karttunen, L. 1977 Syntax and Semantics of Questions, *Linguistics and Philosophy* **1:** 3-44.

Lewis, D. 1972 General semantics. In Davidson, D. and Harman, G., Ed., *Semantics of Natural Language.* D. Reidel Publishing Co., Dordrecht, Holland.

McGettrick, A.D. 1980. *The Definition of Programming Languages.* (Cambridge Computer Science Texts 11.) Cambridge University Press, Cambridge.

Milne, R. and Strachey, C. 1976 *A Theory of Programming Language Semantics.* Chapman and Hall, London.

Montague, R. 1973 The Proper Treatment of Quantification in Ordinary English. In Thomasen, R., Ed., *Formal Philosophy. Selected Papers of Richard Montague.* Yale University Press, New Haven, Connecticut (1974).

Pagen, F.G. 1981 *Formal Specification of Programming Languages: A Panoramic Primer.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Scott, D. 1976 Data Types as Lattices, *SIAM Journal of Computing* **5:** 522-587.

Stoy, J. 1977 *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, Cambridge, Massachusetts.

Tennent, R.D. 1981 *Principles of Programming Languages.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

van Emde Boas, P. and Janssen, T. 1979 The Impact of Frege's Principle of Compositionality for the Semantics of Programming and Natural Languages. Report 79-07, University of Amsterdam.

Winograd, T. 1972 *Understanding Natural Language.* Academic Press, New York, New York.

Woods, W.A. 1970 Transition Network Grammars for Natural Language Analysis, *Communications of the ACM* **13**(10): 591-602.

Woods, W.A. 1972 The Lunar Sciences Natural Language Information System. Report 2378, Bolt Beranek and Newman.