A Translation from Logic to English with Dynamic Semantics*

Elizabeth Coppock and David Baxter

Cycorp, Inc., Austin TX 78731, USA

Abstract. We present a procedure for translating predicate logic into English, which generates both referring and non-referring expressions using a dynamically updated context representation. The system treats referring and non-referring expressions within a unified framework, capturing their common properties – both bound and referential anaphora require an accessible antedecent – and the special properties of non-referring expressions: Non-referring expressions are introduced with quantificational determiners, and correspond to short-term discourse referents.

Keywords: natural language generation, dynamic semantics, predicate logic, quantification, anaphora.

1 Introduction

The goal of the present work is to define an algorithm for translating formulas of predicate logic into concise, natural-sounding English, with quantificational expressions, proper names, indefinites, definite descriptions, and pronouns, wherever appropriate (examples are given in §2). Because predicate logic contains both constants and variables, this algorithm should generate both referring and non-referring expressions.

Work in the field of generating referring expressions [1–16] is designed for the task of providing an appropriate means of referring to a given object in a domain. For example, the input might be a situation in which there is more than one book, and the book in question is on a unique table. An appropriate output for this situation would be *the book on the table*. Because of the nature of the input and the task, these systems only generate expressions that genuinely refer to objects or groups of objects. Even those systems within this tradition of research that generate "quantificational" expressions [7, 12] such as *those women who have fewer than two children* or *the people who work for exactly 2 employers* really only generate referential expressions, referring to groups of objects. The field 'generating referring expressions' is thus appropriately named, so far, because it deals only with the generation of genuinely referring expressions.

* Thanks to the LENLS organizing committee and audience, David Beaver, Cleo Condoravdi, Nicholas Asher, Lucas Champollion and Elias Ponvert for feedback. This work was partially supported under the DARPA Rapid Knowledge Formation program.

K. Nakakoji, Y. Murakami, and E. McCready (Eds.): JSAI-isAI, LNAI 6284, pp. 197–216, 2010.

A separate line of research, known as 'tactical generation' or 'realization' concentrates on sentence generation based on formal grammatical theories such as Head-Driven Phrase Structure Grammar (HPSG) [17–20], Lexical Functional Grammar (LFG) [21–24] and Combinatory Categorial Grammar (CCG) [25–27]. Some systems in this category take as input a logical semantic representation that contains quantifiers and variables. For example, the HPSG generation system described in [17] takes as input the typed feature structure corresponding to the CONTENT value of the top-level HPSG sign using Pollard and Yoo's HPSG analysis of quantification [28], which makes the input a notational variant of quantificational logic. Systems in this category may potentially generate non-referring expressions, such as *no man* and *himself* in no_i *man likes himself*$_i$.

Tactical realization systems based purely on existing grammatical formalisms like [17] lack a representation of the discourse and a theory of antecedent-accessibility that could be used to decide, for example, when a pronoun, definite description, or indefinite description, would be an appropriate way to realize a given discourse referent. They are incomplete without algorithms for generating referring expressions. But the problem cannot be solved simply by adjoining a tactical realization system to a system for generating referring expressions, because there are common constraints between referential and bound variable anaphora. For example, both types of anaphora require a cognitively accessible antecedent. The fields of tactical realization and generating referring expressions should not be kept separate; rather, referring and non-referring expressions should be treated within a unified framework.

The idea that referential and bound variable anaphora have certain commonalities is one of the insights underlying dynamic theories of semantics, such as Discourse Representation Theory (DRT; [29]) and File Change Semantics [30]. Under such theories, both referring and non-referring expressions correspond to 'discourse referents' [31], which are not actual referents but elements of the discourse. According to Karttunen's definition (p. 4), "the appearance of an indefinite noun phrase [or any noun phrase, for our purposes] establishes a 'discourse referent' just in case it justifies the occurrence of a coreferential pronoun or a definite noun phrase later in the text." Pronouns can express either bound variables or constants, so a discourse referent may or may not correspond a genuine referent.

A representation containing discourse referents such as DRT's Discourse Representation Structures (DRSs) might therefore seem to be a natural starting point for a system designed to capture the commonalities between referential and bound variable anaphora. Combinations of HPSG and DRT (see [32] and references cited therein) could potentially be used in a practical natural language generation system; indeed, Minimal Recursion Semantics (MRS; [19]) is a form of Underspecified Discourse Representation Theory [33], and MRS-based natural language generation systems exist [6]. However, this strategy does not capture phenomena that reflect changes to the discourse context as the discourse proceeds, because if the input is a fully-formed DRS, then the discourse representation will remain fixed

throughout the generation procedure. The dynamic nature of the framework is not utilized under such an approach.

The framework described in the present paper is inspired by some of the ideas underlying DRT, including the notion of the discourse referent. However, rather than using a complete Discourse Representation Structure as an input, a (somewhat more minimal) representation of the discourse is built up as the sentence is generated. Using this, the generation system keeps track of the changes that take place as the sentence is in progress, including the introduction of new discourse referents. It is thus dynamic, in the sense that it updates the discourse as it goes along.

2 Problems with Non-referential 'Discourse Referents'

Strictly speaking, it is not necessary to be able to generate quantificational determiners and bound variable anaphora, if the goal is to produce an English sentence for any given formula of predicate calculus. For an input like this:

(1) $\forall x[\mathbf{isa}(x, \mathbf{Man}) \rightarrow \mathbf{loves}(x, x)]$

one could give a direct translation like this:

(2) For every x, if x is a man, then x loves x.

This output counts as English as long as explicit variables like 'x' count as English. But it is more desirable to produce the following kind of output:

(3) Every man loves himself.

If the goal is to produce *concise* English translations of first-order logic formulas, then it is necessary to produce these kinds of non-referring expressions. This leads to special challenges having to do with determiner selection and capturing lifespan limitations.

2.1 Determiner Selection

In simple cases, universally quantified variables as in (4) are introduced with *every,* as in (5), and existentially quantified variables as in (6) are introduced with *some,* as in (7):

(4) $\forall x[\mathbf{loves}(\mathbf{Mary}, x)]$

(5) Mary loves everything.

(6) $\exists x[\mathbf{loves}(\mathbf{Mary}, x)]$

(7) Mary loves something.

A first pass at a determiner-selection algorithm would be then: If the variable is universally quantified, use *every*; if it is existentially quantified, use *some*. But this simple algorithm would fail at cases where a universally quantified variable is introduced with an indefinite determiner, as in donkey sentences. The universally quantified variables in the formula in (8) are introduced as indefinites in (9):

(8) $\forall x \forall y[[\mathbf{isa}(x, \mathbf{Donkey}) \wedge \mathbf{isa}(x, \mathbf{Farmer}) \wedge \mathbf{owns}(x, y)] \rightarrow \mathbf{beats}(x, y)]$

(9) If a farmer owns a donkey, then he beats it.

Using *every* instead would not express the same idea:

(10) If every farmer owns every donkey, then he beats it.

Thus, universally quantified variables are not always introduced with *every*.

One might then refine the algorithm to say that when the variable occurs in the antecedent of a conditional, then it is introduced with an indefinite. But there are cases of this type in which the variable is introduced by *every*:

(11) $\forall x[\mathbf{isa}(x, \mathbf{Donkey}) \rightarrow \mathbf{loves}(\mathbf{Mary}, x)]$

(12) Mary loves every donkey.

When the *if-then* structure of the logical input formula is lost in the English translation, the variable is introduced with *every*.

Universally quantified variables can also be introduced with determiners other than *every* in the presence of negation. A formula like (13) has two concise renditions, (14) and (15), and neither uses *every*.

(13) $\forall x[\mathbf{isa}(x, \mathbf{Donkey}) \rightarrow \neg\mathbf{loves}(\mathbf{Mary}, x)]$

(14) Mary doesn't love any donkey(s).

(15) Mary loves no donkey(s).

When negation is expressed on the verb, the universally quantified variable is expressed with *any*; in the other case, the determiner *no* expresses both negation and universal quantification.

The facts are slightly different when the variable is in subject position. Consider an example in which x is the first argument of **loves**:

(16) $\forall x[\mathbf{isa}(x, \mathbf{Donkey}) \rightarrow \neg\mathbf{loves}(x, \mathbf{Mary})]$

In such a case, only *no* is possible; verbal negation with *any* is not possible:

(17) No donkeys love Mary.

(18) *Any donkeys don't love Mary.

Thus, syntactic considerations appear to play a role in determiner selection as well.

Negation does not always cause a universally-quantified variable to be realized with *any* or *no*; when the negation outscopes the universal quantifier, then *every* is used, as in the following example:

(19) $\neg\forall x[\mathbf{isa}(x, \mathbf{Donkey}) \rightarrow \mathbf{loves}(\mathbf{Mary}, x)]$

(20) Mary doesn't love every donkey.

Thus, whether a universally quantified variable should be realized as *every, some/a, any,* or *no* depends (at least) on whether it is in the protasis of a conditional, the presence and relative scope of negation, and syntactic position.

Existentially bound variables are also sensitive to negation. When the existential quantifier outscopes negation, *some* is used, as usual, whether in subject or object position:

(21) $\exists x[\neg\mathbf{loves}(x, \mathbf{Mary})]$

(22) Someone doesn't love Mary.

(23) $\exists x[\neg\mathbf{loves}(\mathbf{Mary}, x)]$

(24) Mary doesn't love someone.

However, when negation outscopes the existential quantifier, *no* and *any* become appropriate, following the pattern observed with universal quantifiers:

(25) $\neg\exists x[\mathbf{loves}(x, \mathbf{Mary})]$

(26) Noone loves Mary. / *Anyone doesn't love Mary.

(27) $\neg\exists x[\mathbf{loves}(\mathbf{Mary}, x)]$

(28) Mary doesn't love anyone. / Mary loves noone.

Thus, whereas existential quantifiers give rise to *any* and *no* only when they are in the scope of negation, universal quantifiers give rise to them only when they outscope negation.

Since negation of an existential is equivalent to universal quantification over a negation, one might argue that *any* never really corresponds to a universal quantifier; to get an output with *any* one would convert a formula with a universal quantifier outscoping a negation to an equivalent formula with an existential quantifier outscoped by it. So (29) would be converted to (30) before producing a sentence with *any*:

(29) $\forall x[\neg\mathbf{loves}(\mathbf{Mary}, x)]$

(30) $\neg\exists x[\mathbf{loves}(\mathbf{Mary}, x)]$

This would allow one maintain the generalization that *any* corresponds to an existential quantifier, as is sometimes assumed (e.g. [34]). We assume, however, that *any* can correspond to universally quantified variables as well as existentially quantified ones. One argument for this view is that when multiple universal quantifiers outscope negation as in (31), both may correspond to an *any* phrase as in (32).

(31) $\forall x[\forall y[\neg \mathbf{loves}(x, y)]]$

(32) It is not the case that anyone loves anyone (else).

Only one of the quantifiers in (31) can be "swapped" with a negation in order to produce an existential quantifier, yet there are two *any* phrases. At least one of them must correspond to a universal quantifier. Since we must allow universally quantified variables to be realized with *any* in such cases, we might as well allow it in general.

The facts listed above show that the choice of determiner to realize a variable is a non-trivial function of the logical operators present in the input formula and their relative scope. The determiner selection algorithm given in §3.3 captures these patterns.

2.2 Lifespan Limitations

Like referring expressions, non-referring expressions can be anaphoric, when there is an accessible antecedent in the discourse. The second realization of a given discourse referent should, barring potential ambiguity, take the form of a pronoun, whether the discourse referent corresponds to a constant or a variable:

(33) $\mathbf{loves}(\mathbf{Mary}, \mathbf{Mary})$

(34) Mary loves herself.

(35) $\forall x \neg \mathbf{loves}(x, x)$

(36) No woman loves herself.

However, unlike discourse referents corresponding to constants, discourse referents corresponding to variables (realized with non-referring expressions) exist in the discourse only temporarily, and thus have a limited 'lifespan' [31]. This is exemplified in (37), from Heim [30], and (38).

(37) Everyone found a cat and kept it. #Then it ran away.

(38) No_i self-respecting lady will give you her_i phone number. #I know her_i.

On the reading of (37) on which the universal quantifier outscopes the existential quantifier, the *it* in the second sentence cannot corefer with the *it* in the first sentence. Similarly, the discourse referent introduced by *no self-respecting lady* in (38) in is a *short-term referent* [31], whose lifespan ends with the end of the

first sentence. From a parsing perspective, the challenge is to assign the right interpretations to pronouns. From a generation perspective, the challenge is to avoid generating pronouns that are anaphoric to discourse referents that are no longer active.

This is a challenge that does not arise with referential expressions. Contrast (38) with the following example:

(39) Jane$_i$ will give you her$_i$ phone number. I know her$_i$.

In (39), the discourse referent survives into the second sentence, because proper names are referential, and the lifespan of a discourse referent introduced with a referential expression is in principle unlimited. If referring and non-referring expressions are not distinguished, then this difference between them cannot be captured. Thus, although bound variable and referential anaphora should be treated in a unified fashion, the treatment should not be so unified as to blur the distinction between them.

The lifespan of a short-term discourse referent does not always correspond to the enclosing tensed sentence. Whereas a discourse referent introduced by *every* is limited to the protasis of a conditional, a discourse referent introduced with an indefinite in the protasis may extend to the apodosis [29, 30]:

(40) *If every$_i$ farmer owns every$_j$ donkey, then he$_i$ beats it$_j$.

(41) If a$_i$ farmer owns a$_j$ donkey, then he$_i$ beats it$_j$.

The lifespan of the indefinite is not indefinite, however:

(42) If a$_i$ farmer owns a$_j$ donkey, then he$_i$ beats it$_j$. #I know him$_i$.

The lifespan of an indefinite introduced in the protasis of a conditional ends with the apodosis.

3 The Cyc NLG System

We now present a procedure for translating predicate calculus into English, which treats referring and non-referring expressions in a unified framework, and captures all of the facts described in the previous section, regarding both determiner selection and lifespan limitations. The system we present is the natural language generation (NLG) system for Cyc [35], a large-scale commonsense knowledge base and reasoning engine. Cyc is based on CycL, a logic that subsumes first order logic [36],[1] and the system we describe translates from CycL to English. Here, we concentrate on the first order portion of CycL, making limited use of Cyc-specific ontological distinctions, in order to maximize the applicability of our model. The input is described in detail in §3.1.

[1] The majority of the assertions in the Cyc Knowledge Base are statement of first-order logic; the majority of the remaining assertions can be transformed into statements first-order logic [36].

Our procedure uses two forms of dynamically updated context: the *discourse context,* which lists the discourse referents that have been mentioned, and the *operator context,* which stores information that is stripped away from the input formula. The discourse context is discussed in §3.2; the operator context will be discussed in §3.3.

3.1 Input: First-Order Predicate Calculus Part of CycL

The input to the Cyc NLG system is a formula of CycL, which is a higher-order logic built on first-order predicate calculus [37]. CycL has a number of fancy features, such as quoting, meta-assertions, lambda expressions (forming terms through variable abstraction), and kappa expressions (forming predicates through variable abstraction), most of which will not concern us here. In this paper, we will focus on inputs from the first-order portion of CycL. The set of expressions within this first-order portion contains variables (e.g. x, y, z), and atomic constants denoting individuals (e.g. **Mary**), collections (e.g. **Donkey**) predicates, (e.g. **loves**), and functions (e.g. **MotherOf**). The predicates and functions can in principle be of any arity. In CycL, arguments of predicates and functions can in principle be any other CycL expression, so CycL is higher order, but we can restrict our attention to first-order predicates and functions. The set of non-atomic expressions contains:[2]

- Non-Atomic Terms: if γ is a function and $\xi_1 ... \xi_n$ a sequence of arguments matching γ's argument constraints, then $\gamma(\xi_1 ... \xi_n)$ is a term.
- Atomic Sentences: if π is a predicate and $\xi_1 ... \xi_n$ a sequence of arguments matching π's argument constraints, then $\pi(\xi_1 ... \xi_n)$ is a sentence.
- Negations: if ϕ is a sentence then $\neg\phi$ is a sentence.
- Conjunctions: if ϕ is a sentence and ψ is a sentence than $\phi \wedge \psi$ is a sentence.
- Disjunctions: if ϕ is a sentence and ψ is a sentence then $\phi \vee \psi$ is a sentence.
- Implications: if ϕ is a sentence and ψ is a sentence then $\phi \rightarrow \psi$ is a sentence.
- Universals: if ϕ is a sentence and ξ is a variable then $\forall \xi \phi$ is a sentence.
- Existentials: if ϕ is a sentence and ξ is a variable then $\exists \xi \phi$ is a sentence.

This logic can be given a standard model-theoretic semantics for predicate calculus.

Other than the distinction between individuals and collections, this logic is perfectly standard. In Cyc, collections are often used in place of one-place predicates, so rather than $\mathbf{farmer}(x)$, we will have $\mathbf{isa}(x, \mathbf{Farmer})$, where **Farmer** represents the collection of all farmers, and **isa** is a predicate relating an individual to a collection, which holds if the individual is an instance of the collection.

[2] Regarding notation: We use the standard way of using parentheses in logic, rather than using the Lisp-style notation that is normally used for CycL. Constants are indicated with bold face (whereas in CycL they are prefixed with '#$') and variables with italics (rather than being prefixed with '?' as in CycL). Following CycL conventions, however, we use initial lowercase letters for predicates and initial uppercase letters for individuals and collections. Variables of all types are lowercase.

(We use the term *instance* for collections, rather than *member,* which we reserve for sets; the idea is that collections represent concepts, while sets are merely extensionally defined.) To give a more standard logic, one could replace all collections with single-arity predicates, so the distinction between individuals and collections is not completely crucial. However, it does happen to play a role in the grammar, so the grammar would have to be adapted if that distinction were eliminated.

The Cyc NLG system has full access to the Cyc Knowledge Base (KB), which contains an English lexicon. The lexicon includes a set of *generation templates,* which describe an English sentence or phrase corresponding to a function or predicate, with open slots for the arguments. For example, the predicate **likes** is associated with a template describing a transitive sentence in which the subject is the realization of the first argument, the verb is a form of *like* that agrees with the subject, and the object is a realization of the second argument. These templates thus accomplish argument linking. (Rather than stipulating the syntactic realization of arguments on a case-by-case basis, one could derive these templates from more general principles, so the present system is not crucially tied to a stipulative linking theory; we just take linking as given.) Aside from what is specified in generation templates, the grammatical structure of a generated utterance is determined procedurally. Therefore, the system that we describe here is not only a natural language generation system, but also a grammar.

Because it was developed for the purpose of generation rather than parsing, the theoretical constructs that this system uses are different from the ones that have been developed under the parsing perspective. In particular, there are two forms of context: *discourse context* and *operator context.* These are described in the following two subsections.

3.2 Discourse Context

Definition. A discourse context D is a set of *discourse referents* [31]. Like Heim's 'file cards' [30, 38] and the elements of DRT universes [29], these discourse referents need not correspond to any particular entity in the situation described by the sentence. Each discourse referent r is associated with a logical expression α, which can be either a variable or a closed term, composed entirely of constants (either atomic, e.g. **Mary**, or non-atomic, e.g. **Mother**(**Mary**) 'Mary's mother').

Insofar as the logical expression associated with these discourse referents can be either a variable or a closed term, they are similar to Muskens's 'registers' [39], and unlike the elements of DRT universes, which correspond only to variables. As Muskens points out [39], allowing proper names to be translated with constants eliminates the need for DRT's 'external anchoring' device. From a generation perspective, this design choice is quite natural; it would be a waste to convert constants in the input formula to variables before listing them among the discourse referents.

Discourse referents are also associated with *index features*: person, number and gender [40, 41]. Index features are computed on the basis of morphosyntactic information if it is available, or encyclopedic knowledge in the Cyc Knowledge Base (KB) otherwise. For example, the individual **Mary** is asserted to be a human female in the KB, so corresponding discourse referents will have a feminine gender feature. These index features determine the form that pronouns take.

Side effects. We recursively define a generation function $G(\alpha)$, where α can be any expression of the logic, which depends not only on α, but also on a global discourse D and a global operator context O. We subdivide the definition of $G(\alpha)$ into cases based on the logical expression type of α. A fundamental case is when α is an *atomic formula*, as in (43).

(43) **loves**(**Mary**, **Mary**)

The most appropriate method for atomic formulas uses the generation template for the predicate – a mapping from a logical predicate to a partial specification of a sentence in natural language. The generation template for **loves** specifies a template from which a syntax tree is built. A syntax tree is like an HPSG sign [40], with "phonological", semantic, and syntactic information, including daughters for phrases. (Since we are computing textual output, the value of the so-called PHON feature is an orthographic string.)

A simplified rendition of the phrase that is ultimately generated for (43) is in Fig. 1. The tree is traversed left to right, depth first, and may be expanded during the traversal. Each time a node is visited, the value of the PHON feature is computed. We say that a subexpression of the logical formula has been *realized* if the PHON value of the phrase it corresponds to has been set. The PHON value of the mother is the concatenation of the PHON values of the daughters. The value of G is the PHON value of the top-level phrase.

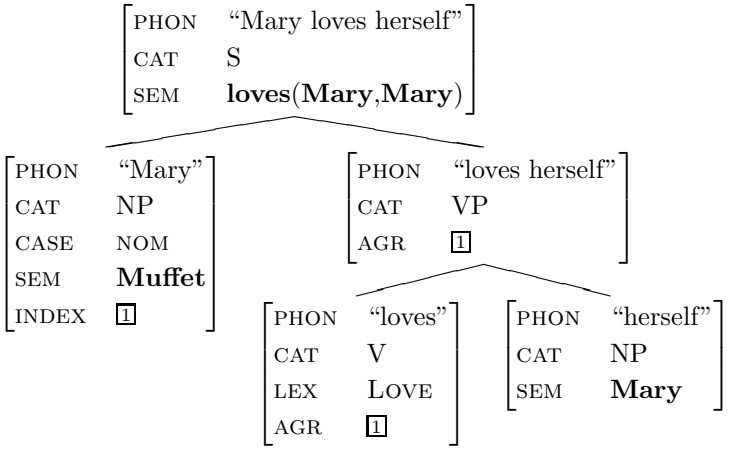

Fig. 1. Phrase generated for the input '**loves**(**Mary**, **Mary**)' (simplified)

What makes the semantics dynamic is that the computation of G can have *side effects* in the form of updates to D and O. The discourse context D is updated when a discourse referent is realized, as part of the algorithm for generating a constant or a variable. If α is an individual-denoting constant, and is not listed among the discourse referents, then $G(\alpha)$ returns a proper name, *and the discourse context is updated.* For example, the expression **Mary** in (43) is added to D after its first instance in the input formula is realized.

If α is listed in D, then an anaphoric expression is used. The algorithm for generating an anaphor is: If a pronoun would be ambiguous, then use a definite description or name; otherwise, use a pronoun. There is a great deal of sophistication and subtlety this treatment could acquire [42–48], but this is not our focus here. A (reflexive) pronoun is appropriate for the second instance of **Mary** in (43), so the formula is rendered, "Mary loves herself."

Like other discourse referents, variables are added to D after being mentioned, so as to be made eligible for subsequent rendering with anaphora. Thus, after *everything* is generated in (44), the variable x is added to the set of discourse referents, so it can be realized with a pronoun on its second mention:

(44) Everything likes itself.
$\forall x[\mathbf{likes}(x, x)]$

Lifespan limitations. Constants added to D during the execution of an instance of G may remain in D. However, it is necessary to remove variables from D when their "lifespan" is exhausted. Thus, if the argument to G is a quantificational sentence binding a variable ξ, then the following line must be executed before G exits:

$$D \leftarrow D - R(\xi)$$

where $R(\xi)$ stands for the discourse referent associated with ξ. Suppose the input formula were:

(45) $\forall x \neg[\mathbf{loves}(\mathbf{Doug}, x)] \wedge \forall x[\mathbf{loves}(\mathbf{Mary}, x)]$

This would be rendered correctly as: "Doug loves nothing and Mary loves everything" rather than "Doug loves nothing and Mary loves it."

Removal of variables from D is the key to accounting for the limited lifespan of the discourse referent introduced by *no self-respecting woman* in (38) ("No self-respecting woman will give you her phone number. #I know her.") At the end of the sentence, the variable bound by the universal quantifier is removed from the discourse context, making it impossible for there to be subsequent anaphoric references to it.[3] If a pronoun is generated in the following sentence, it will have to be associated with some other discourse referent.

[3] Some of Karttunen's observations in [31] suggest that discourse referents corresponding to existentials with maximally wide scope should not be removed from the discourse context. For example, consider the sentence "I have a proof of this theorem but it won't fit in this margin." One is tempted to analyze the first sentence using an existential quantifier; but then how can there be subsequent anaphoric reference to it? One possibility is to introduce a Skolem constant for wide-scope existentials.

3.3 Operator Context

$G(\alpha)$ depends not only on a discourse context D, but also on an operator context O. As we describe futher below, logical operators and negations are stripped away from the input formula, leaving a "clausal skeleton," and the information stripped away is stored in the operator context.

Definition. We define an operator context O as a tuple $\langle V, S, n\rangle$ where V is a set of variable type entries, S is a stack of logical symbols, and n is an integer representing the number of negations remaining to be expressed.

A variable type entry $v \in V$ is a tuple $\langle \alpha, \theta, \tau\rangle$, where α is a variable over individuals, θ is a quantifier symbol, and τ is a type. Types are Cyc *collections,* such as **Donkey**, the collection of all donkeys. As mentioned above, Cyc collections are like sets except that they are meant to represent concepts and have instances, rather than members [49].[4] Associating a type with a variable is conceptually similar to identifying the type as the 'restriction' of the quantifier, even though formally, quantifiers in first order logic do not specify restrictions, unlike generalized quantifiers.

The stack of symbols S contains the variables and logical operators with scope over the element of the formula that is currently being realized. We represent stacks as tuples $\langle \alpha_1 \ldots \alpha_n\rangle$, where α_n is at the top of the stack. The wider the scope of the operator, the deeper on the stack it is. For the formula $\forall x \neg[\mathbf{loves}(\mathbf{Mary}, x)]$, at the point when **Mary** is being generated, $S = \langle x, \neg\rangle$. We use the variable x rather than the quantifier $\forall$ to indicate the scope of the associated universal quantifier, because the variable uniquely identifies the quantifier in question, while there may be many universal quantifiers in a formula. The quantifier associated with the variable is computable from the variable's type entry in V.

Finally, n is used to keep track of unexpressed negations. As we will see in the next section, negations can be removed from formulas in the course of constructing the clausal skeleton, and this counter helps to ensure that every negation is expressed exactly once. (Note that the value of n is not derivable from S, as any negation on S may be either expressed or unexpressed.)

Clausal skeletons. In the generation of some formulas, parts of the input formula are stripped away, leaving a *clausal skeleton.* For example, the clausal skeleton of (46) is (47).

(46) $\forall x \forall y[[\mathbf{isa}(x, \mathbf{Man}) \wedge \mathbf{isa}(y, \mathbf{Donkey}) \wedge \mathbf{owns}(x, y)] \rightarrow \mathbf{loves}(x, y)]$

(47) $\mathbf{owns}(x, y) \rightarrow \mathbf{loves}(x, y)$

The two **isa** statements in the antecedent of the conditional in (46) are *variable typing clauses.* The binary predicate **isa** relates an individual to a collection,

[4] Collections can also be complex; CycL contains collection-forming functions with which concepts like "the collection of farmers who own a donkey" can be expressed.

and signifies that the individual is an instance of the collection. Variable typing clauses are removed, along with the universal quantifiers (as mentioned in [50]).

For any input formula of the form $\forall\xi\alpha$, where ξ is a variable and α is a sentence, a simplified version of the algorithm for constructing clausal skeletons σ and new variable type entries v is as follows (where **Thing** is the most general collection, and $\sim$ stands for "is of the form" or "matches"):

(48)
- If $\alpha \sim [\psi \rightarrow \phi]$:
  - If $\psi \sim [\mathbf{isa}(\xi, \gamma)]$:
    $\sigma = \phi$
    $v = \langle \xi, \forall, \gamma \rangle$
  - Else if $\psi \sim [\delta_1 \wedge ... \wedge \delta_n]$ where $\delta_i \sim [\mathbf{isa}(\xi, \gamma)]$:
    $\sigma = [\delta_1 \wedge ... \wedge \delta_{i-1} \wedge \delta_{i+1} \wedge ... \wedge \delta_n] \rightarrow \phi$
    $v = \langle \xi, \forall, \gamma \rangle$
  - Else:
    $\sigma = \alpha$
    $v = \langle \xi, \forall, \mathbf{Thing} \rangle$
- Else:
  $\sigma = \alpha$
  $v = \langle \xi, \forall, \mathbf{Thing} \rangle$

For a formula of the form $\exists\xi\alpha$, there are two cases:

(49)
- If $\alpha \sim [\delta_1 \wedge ... \wedge \delta_n]$ where $\delta_i \sim [\mathbf{isa}(\xi, \gamma)]$:
  $\sigma = [\delta_1 \wedge ... \wedge \delta_{i-1} \wedge \delta_{i+1} \wedge ... \wedge \delta_n]$
  $v = \langle \xi, \exists, \gamma \rangle$
- Else:
  $\sigma = \alpha$
  $v = \langle \xi, \exists, \mathbf{Thing} \rangle$

The variable type entry v is added to the set V of variable type entries in the operator context O. After the operator context is updated, $G(\sigma)$ is computed; in other words, the clausal skeleton is realized. Then, importantly, the operator context is restored to its previous state. The information stored in the operator context surfaces when the variable is expressed, as described in §3.3.

The clausal skeleton is isomorphic, clause for clause, to the resulting English sentence. Thus the realization of (46) has two clauses, just as its clausal skeleton (47) has:

(50) If a farmer owns a donkey, then he loves it.

When the antecedent of the clause consists entirely of a variable typing clause, all that remains in the clausal skeleton is the consequent. An input formula such as (51) will be realized as in (52), a single clause.

(51) $\mathbf{isa}(x, \mathbf{Man}) \rightarrow \mathbf{loves}(x, \mathbf{Mary})$

(52) Every man loves Mary.

Thus, all of the content that is expressed below the clause level (in quantified noun phrases) comes from the operator context, and every clause in the English translation is part of the clausal skeleton.[5]

Negation stripping. When α is of the form $\neg\phi$, the clausal skeleton of α is ϕ. No variable type entries are produced in this case, of course, but the counter representing the number of unexpressed negations, n, is incremented. "Stripping" the negation in this way makes it possible for negations to be expressed sub-clausally, using a member of the *no*-series (*nobody, nothing*, etc.). We use clausal negation as a "back-up strategy" when expressing negation sub-clausally fails; if $n > 0$ after ϕ is realized, either the verb is negated or negation is expressed periphrastically with, for example, "It is false that...".

Updating the operator stack. The main purpose of the operator stack is to determine when NPIs are licensed. The generation procedure $G(\alpha)$ updates the operator stack whenever α is a formula whose operator is in the set $\{\forall, \exists, \neg, \rightarrow\}$. If α is a quantificational formula such as $\forall x\phi$, then the variable (x) is pushed onto the operator stack, and popped off of it at the completion of $G(\alpha)$. If α is a negative formula, then a $\neg$ symbol is pushed onto the operator stack and again, popped off of it after $G(\alpha)$ is computed. If α is an implication, then the symbol $\rightarrow$ is pushed onto the stack and the stack is popped at the completion of the generation of the *antecedent,* because implications only license NPIs in the antecedent.

Determiner selection algorithm. The definition of $G(\alpha)$ where α is a variable (*variable realization*) involves the operator context as well as the discourse context. As mentioned in §3.2, variables are realized as pronouns when they are listed in D and a pronoun would not be ambiguous. If a pronoun is not appropriate for realizing a variable, a lexical noun phrase containing a determiner and a noun is used. The noun is the realization of the variable's type, the τ such that $\langle \alpha, \theta, \tau \rangle \in V$. τ represents a collection, e.g. **Man** (the collection of all men), and can be realized as, for example, *man.* There are several types of determiner that may accompany the noun: DEFINITE (*the*), NO (*no*), UNIVERSAL (*every*), INDEFINITE (*a, some*), NPI (*any, a* or *some*).[6]

The first step in the algorithm for choosing a determiner type is to compute whether or not a variable could be expressed as an NPI, i.e., with *any.* Both universally and existentially bound variables can be realized with *any,* but the

[5] If the variable typing clauses were not removed, the variables would be registered with type **Thing** and the output would contain more clauses: "If something is a farmer and some other thing is a donkey then the thing loves the other thing." If the variables were not registered in the operator context at all, then the universal quantifier would not be stripped from the formula, and the output would be as follows, with explicit variables: "For every x, for every y, if x is a farmer and y is a donkey and x owns y, then x loves y."

[6] Another determiner type is WH (*what, which*). *Wh-* determiners are used for unbound variables in formulas to be generated with interrogative force. We ignore questions here for the sake of simplicity.

two quantifier types differ with respect to the scope that they must have relative to an NPI licenser. In order to qualify for being realized with *any,* an instance of a variable must be bound by either a universally quantified variable outscoping an NPI licenser, or an existentially bound quantifier outscoped by one. For example, (29), repeated below is rendered as in (54).

(53) $\forall x[\neg \mathbf{loves}(\mathbf{Mary}, x)]$

(54) Mary doesn't love anything.

In contrast, when the universal quantifier is inside the scope of negation, the output should be *Mary doesn't love everything.*

The algorithm for determining whether or not NPI *any* is licensed is as follows: First, look up the quantifier of the variable in question; then if the quantifier is $\forall$, the question is whether the variable is deeper on the stack than an NPI licenser; if the quantifier is $\exists$, the question is whether the NPI licenser is deeper than it. Call the NPI licenser π; if the variable has no NPI licenser, then π is null.

Given π, a variable type entry $\langle \alpha, \theta, \tau \rangle$ for variable ξ, and an unexpressed negation counter n, the determiner is chosen according to the following algorithm:

(55)
- If $R(\chi) \in D$ (i.e., a previous instance of the variable has been realized), return DEFINITE.
- If π is non-null:
  - If $\pi = \neg$ and $n > 0$, then return NO and decrement n by one.
  - Otherwise, return NPI.[7]
- If $\theta = \forall$, return UNIVERSAL.
- Otherwise, return INDEFINITE.

Note at this point that the set of variable type entries V is not redundant with the discourse context D, despite the fact that they may simultaneously contain the same variable as an entry. The discourse context is used for referents that have already been realized, but variable type entries are used in the formulation of the first mention of the variable.

Now consider example (46) again, repeated here as (56).

(56) $\forall x \forall y[[\mathbf{isa}(x, \mathbf{Man}) \wedge \mathbf{isa}(y, \mathbf{Donkey}) \wedge \mathbf{owns}(x, y)] \rightarrow \mathbf{loves}(x, y)]$

Right before x is generated for the first time, $\langle x, \mathbf{Man}, \forall \rangle \in V$ (i.e. x is universally quantified, and has type **Man**), $S = \langle x, \rightarrow \rangle$ (i.e., the current expression is inside the antecedent of a conditional), $n = 0$ (there are no unexpressed negations), and x is not in D (so x has not previously been realized). Therefore $G(x)$ will realized with an NPI-type determiner, as either *any man* or with an indefinite (*a* or *some*). After x is realized, it will be in D, so it will be realized as a

[7] In the generation of variables with NPI-type determiners, we make the stylistic choice to use *any* when there is only one mention of the variable, and an indefinite otherwise.

masculine pronoun or as *the man*. If instead, S were merely $\langle x \rangle$ (so the current expression is not in the scope of an NPI licenser), then π would be null, and the appropriate determiner type would be EVERY.

Another case where NPI-type determiners are chosen is in the scope of negation, when there is no negation to be expressed, i.e., when $\pi = \neg$ and $n \not> 0$. Such a situation arises when negation is expressed on the verb, as in *Mary doesn't love anything*. When $n > 0$ (there are unexpressed negations) and $\pi = \neg$ (the NPI licenser for the variable is negation), the variable can be used to express negation, as in *Mary loves nothing*.

We are now in a position return to the contrast between indefinite determiners and *every* with respect to the lifespan of the discourse referents they introduce. Recall the fact that whereas discourse referents introduced by indefinites in the protasis of a conditional extend to the apodosis, those introduced by *every* are limited to the protasis:

(57) If a_i farmer owns a_j/*every_j donkey, then he_i loves it_j.

We have just seen how the acceptable variant of (57), with an indefinite determiner in the antecedent, is generated for an input like (46). The only way for *every* to be generated in the antecedent of a conditional is for the universally quantified variable to be outscoped by $\rightarrow$, as in the following example:

(58) $\forall x[\mathbf{isa}(x, \mathbf{Farmer}) \wedge \forall y[\mathbf{isa}(x, \mathbf{Donkey}) \rightarrow \mathbf{owns}(x, y)]] \rightarrow \mathbf{loves}(x, \mathbf{Mary})$

The formula in (58) would be rendered as follows:

(59) If a farmer owns every donkey, then he loves Mary.

When it is time to realize the variable y for the first time, $S = \langle x, \rightarrow, y \rangle$, so the computation of π for y will yield a null value, since y is universally quantified and universally quantified variables must be outside the scope of an NPI licenser. The determiner selection algorithm will therefore correctly choose *every*. But in this case, the scope of the quantifier will have ended by the time the consequent of the conditional is reached. Thus, discourse referents introduced by *every* in the protasis of a conditional will never extend to the apodosis.

4 Summary and Outlook

We have presented an algorithm for translating predicate logic to English that uses dynamically updated information states. It deals with referring and non-referring expressions in a unified framework, capturing the fact that both require an accessible antecedent. This is formalized using the discourse context, where discourse referents are placed after they are introduced. The system also captures special features of non-referring expressions, which correspond to logical variables. Discourse referents associated with variables have limited lifespans and are introduced with quantificational determiners, whose use is governed by a complex set of factors, modelled with the operator context.

Just as general frameworks for generating semantic representations from English sentences (e.g. [51], [29]) are semantic theories, the framework presented here is in a sense a theory of semantics (or what we might call 'inverse semantics'). It differs in its use of theoretical primitives from semantic theories that were developed from a parsing perspective, as the theoretical constructs that we found useful in generation are slightly different from those that were found to be useful in parsing. Given such differences, the generation perspective could potentially shed light on the theory of semantics more generally, and provide more elegant or even more empirically adequate accounts of certain phenomena.

One area where the generation perspective may shed light is in NPI licensing. NPI *any* is not always licensed in the semantic scope of negation:

(60) *Anyone doesn't love me.

On our analysis, this configuration is blocked by *Noone loves me.* Thus, it is not necessary to associate syntactic constraints with NPI *any* to rule (60) out. In general, implicit in the notion of a 'determiner selection algorithm' is the idea of blocking, an idea that is natural from the generation perspective, and we believe it may be worthwhile to pursue this view of quantificational determiners further.

Secondly, the notion of *accessibility* between pronouns and their antecedents receives quite a different treatment here from the one in Discourse Represenation Theory. Whereas the accessibility relationship is characterized in DRT as a structural relationship within Discourse Representation Structures, accessibility is formalized here as existence in the discourse context, a potentially transient state that ends for variables when the logical expression corresponding to the quantifier that binds them has been realized. In DRT, proper names always "float to the top" of a DRS, so they are always available; this corresponds to the fact that constants are never removed from the discourse context. We feel that the present view on accessibility has a certain intuitive appeal, and it would be worthwhile to compare the empirical predictions of the two approaches to see if they differ.

Natural language generation also puts presupposition in a new light. Definite descriptions and pronouns, for example, are usually analyzed as containing uniqueness presuppositions. The association of these items with uniqueness is encoded in the present system via conditions on the choice of referring expression type. Presuppositions in general need not be represented declaratively, but can be implicitly encoded in a procedural generation algorithm. This view on presupposition would capture the fact that pronouns are quite easy to process, and would therefore seem to carry a very simple message, contrary to what one would expect if they came associated with complex presuppositional content.

References

1. Horacek, H.: An algorithm for generating referential descriptions with flexible interfaces. In: Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics, pp. 206–213 (1988)
2. Dale, R.: Cooking up referring expressions. In: Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics (1989)

3. Reiter, E.: The computational complexity of avoiding false implicatures. In: Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics (1990)
4. Reiter, E., Dale, R.: A fast algorithm for the generation of referring expressions. In: Proceedings of the 14th International Conference on Computational Linguistics, Nantes, pp. 232–238 (1992)
5. Dale, R., Reiter, E.: Computational interpretations of the Gricean maxims in the generation of referring expressions. Cognitive Science 19, 233–263 (1994)
6. Copestake, A., Flickinger, D., Malouf, R., Riehemann, S., Sag, I.: Translation using minimal recursion semantics. In: Proceedings of the Sixth International Conference on Theoretical and Methodological Issues in Machine Translation, Leuven, Belgium (1995)
7. Shaw, J., McKeown, K.: Generating referring quantified expressions. In: Proceedings of the first international conference on natural language generation, Mitzpe Ramon, Israel, pp. 100–107 (2000)
8. Krahmer, E., Van Erk, S., Verleg, A.: Graph-based generation of referring expressions. Computational Linguistics (2003)
9. Van Deemter, K.: Generating referring expressions: Boolean extensions of the incremental algorithm. Computational Linguistics 28, 37–52 (2002)
10. Siddharthan, A., Copestake, A.: Generating anaphora for simplifying text. In: Proceedings of the 4th Discourse Anaphora and Anaphor Resolution Colloquium, pp. 199–204 (2002)
11. Siddharthan, A., Copestake, A.: Generating referring expressions in open domains. In: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, Barcelona, Spain, pp. 408–415 (2004)
12. Varges, S., Van Deemter, K.: Generating referring expressions containing quantifiers. In: Proceedings of the 6th International Workshop on Computational Semantics, pp. 1–13 (2005)
13. Kelleher, J.D., Kruijff, G.J.M.: Incremental generation of spatial referring expressions in situated dialog. In: Proceedings of COLING/ACL 2006 (2006)
14. Paraboni, I., Van Deemter, K., Masthoff, J.: Generating referring expressions: Making referents easy to identify. Computational Linguistics 33, 229–254 (2007)
15. Van Deemter, K., Krahmer, E.: Graphs and booleans: on the generation of referring expressions. In: Bunt, H., Muskins, R. (eds.) Computing Meaning, vol. 3, pp. 397–422. Springer, Dordrecht (2008)
16. Areces, C., Koller, A., Striegnitz, K.: Referring expressions as formulas of description logic. In: White, M., Nakatsu, C., McDonald, D. (eds.) Proceedings of the Fifth International Natural Language Generation Conference, Salt Fork, Ohio, pp. 42–49. Association for Computational Linguistics (2008)
17. Wilcock, G., Matsumoto, Y.: Head-driven generation with HPSG. In: Proceedings of COLING-ACL 1998: Workshop on Usage of WordNet in Natural Language Processing Systems, pp. 1393–1397 (1998)
18. Carroll, J., Flickinger, D., Copestake, A., Poznanski, V.: An efficient chart generator for (semi-)lexicalist grammars. In: Proceedings of the 7th European Workshop on Natural Language Generation, Toulouse, France (1990)
19. Copestake, A., Flickinger, D., Pollard, C., Sag, I.A.: Minimal recursion semantics: An introduction. Research on Language and Computation 3, 281–332 (2005)
20. Carroll, J., Oepen, S.: High efficiency realization for a wide-coverage unification grammar. In: Dale, R., Wong, K.F. (eds.) Proceedings of the Second International Joint Conference on Natural Language Processing (IJNLP 2005), Springer, Heidelberg (2005)

21. Wedekind, J., Kaplan, R.M.: Ambiguity-preserving generation with LFG- and PATR-style grammars. Computational Linguistics 22, 555–558 (1996)
22. Wedekind, J.: Semantic-driven generation with LFG- and PATR-style grammars. Computational Linguistics 25, 277–281 (1999)
23. Kaplan, R.M., Wedekind, J.: LFG generation produces context-free languages. In: Proceedings of the 18th Conference on Computational Linguistics, Saarbrücken, Germany, pp. 425–431 (2000)
24. Cahill, A., van Genabith, J.: Robust pcfg-based generation using automatically acquired lfg approximations. In: Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL, Sydney, Australia, pp. 1033–1040. Association for Computational Linguistics (2006)
25. Calder, J., Reape, M., Zeevat, H.: An algorithm for generation in unification categorial grammar. In: Proceedings of the 4th Conference of the European Chapter of the Association for Computational Linguistics, Manchester, UK, pp. 233–240 (1989)
26. Phillips, J.D.: Generation of text from logical formulae. Machine Translation 8, 209–235 (1993)
27. White, M.: Reining in CCG chart realization. In: Belz, A., Evans, R., Piwek, P. (eds.) INLG 2004. LNCS (LNAI), vol. 3123, pp. 182–191. Springer, Heidelberg (2004)
28. Pollard, C., Yoo, E.J.: A unified theory of scope for quantifiers and *wh-* phrases. Journal of Linguistics 34(2), 415–445 (1998)
29. Kamp, H., Reyle, U.: From Discourse to Logic. Kluwer Academic Publishers, Dordrecht (1993)
30. Heim, I.: The Semantics of Definite and Indefinite Noun Phrases. PhD thesis, MIT (1982)
31. Karttunen, L.: Discourse referents. In: McCawley, J.D. (ed.) Syntax and Semantics 7: Notes from the Linguistic Underground, pp. 363–385. Academic Press, New York (1976)
32. Sailer, M.: Npi licensing, intervention and discourse representation structures in hpsg. In: Müller, S. (ed.) Proceedings of the HPSG 2007 Conference. CSLI Publications, Stanford (2007)
33. Reyle, U.: Dealing with ambiguities by underspecification: Construction, representation, and deduction. Journal of Semantics 10(2), 123–179 (1993)
34. De Swart, H.: Licensing of negative polarity items under inverse scope. Lingua 105, 175–200 (1998)
35. Lenat, D.: Cyc: A large-scale investment in knowledge infrastructure. Communications of the ACM 38 (1995)
36. Ramachandran, D., Reagan, P., Goolsbey, K.: First-orderized ResearchCyc: Expressivity and efficiency in a common-sense ontology. In: Papers from the AAAI Workshop on Contexts and Ontologies: Theory, Practice and Applications, Pittsburg, PA (2005)
37. Matuszek, C., Cabral, J., Witbrock, M., DeOliveira, J.: An introduction to the syntax and content of Cyc. In: Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering, Stanford, CA (2006)
38. Heim, I.: File change semantics and the familiarity theory of definiteness. In: Baurle, R., Schwarze, C., Von Stechow, A. (eds.) Meaning, Use, and the Interpretation of Language, pp. 164–189. Walter de Gruyter, Berlin (1983)
39. Muskens, R.: Combining Montague semantics and discourse representation. Linguistics and Philosophy 19, 143–186 (1996)

40. Pollard, C., Sag, I.A.: Head-Driven Phrase Structure Grammar. University of Chicago Press, Chicago (1994)
41. Wechsler, S., Zlatić, L.: The Many Faces of Agreement. Center for the Study of Language and Information, Stanford (2003)
42. Chafe, W.L.: Givenness, contrastiveness, definiteness, subjects, topics and point of view. In: Li, C.N. (ed.) Subject and topic, pp. 25–55. Academic Press, New York (1976)
43. Ariel, M.: Accessing NP antecedents. Routledge, London (1990)
44. Gundel, J.K., Hedberg, N., Zacharski, R.: Cognitive status and the form of referring expressions in discourse. Language 69, 274–307 (1993)
45. Brennan, S.: Centering attention in discourse. Language and Cognitive Processes 10, 137–167 (1995)
46. Grosz, B.J., Joshi, A.K., Weinstein, S.: Providing a unified account of definite noun phrases in discourse. In: Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics, Cambridge, MA, pp. 44–49 (1983)
47. Grosz, B.J., Joshi, A.K., Weinstein, S.: Centering: A framework for modeling the local coherence of discourse. Computational Linguistics 21, 203–226 (1995)
48. Beaver, D.I.: The optimization of discourse anaphora. Linguistics and Philosophy 27, 3–56 (2004)
49. Lenat, D.B., Guha, R.V.: Building Large Knowledge-Based Systems. Addison-Wesley, Reading (1990)
50. Baxter, D., Shepard, B., Siegel, N., Gottesman, B., Schneider, D.: Interactive natural language explanations of cyc inferences. In: Proceedings of AAAI 2005: International Symposium on Explanation-aware Computing, Washington, D.C. (2005)
51. Heim, I., Kratzer, A.: Semantics in Generative Grammar. Blackwell, Oxford (1998)