A bidirectional mapping between English and CNF-based reasoners

Steven Abney
University of Michigan
abney@umich.edu

Abstract

If language is a transduction between sound and meaning, the target of semantic interpretation should be the meaning representation expected by general cognition. Automated reasoners provide the best available fully-explicit proxies for general cognition, and they commonly expect Clause Normal Form (CNF) as input. There is a well-known algorithm for converting from unrestricted predicate calculus to CNF, but it is not invertible, leaving us without a means to transduce CNF back to English. I present a solution, with possible repercussions for the overall framework of semantic interpretation.

1 Overview

1.1 The problem

I would like to address a problem that illustrates how considerations of the place of semantic interpretation in the larger cognitive system, even very schematic considerations, can have consequences for the manner and target of interpretation.

Let us take seriously the idea that language is a mapping between sound and meaning—which is to say, essentially, an input-output device for general cognition—and let us provisionally accept current automated reasoners as the best available *fully explicit* models of general cognition. Then an important goal for a semantics of English is to define an invertible transduction between English sentences and a meaning representation that is suitable for use with an automated reasoner. Model-theoretic interpretation is good and useful, but it does not provide us with a transducer.

Standard accounts are easily recast as defining a mapping f from English sentences to predicate calculus. However, the mapping does not appear to be invertible. For one thing, not every predicate-calculus expression is in the range of f. If general cognition produces an arbitrary predicate-calculus expression ϕ to render into English, we must find a logically equivalent expression ϕ' such that $f^{-1}(\phi')$ is defined, but logical equivalence is undecideable, a problem pointed out by Shieber (1993). Even if $f^{-1}(\phi)$ is defined, it is unclear how to compute it.

In addition, the most common choice of meaning representation for automated reasoners is not general predicate calculus, but a normal form known as Clause Normal Form (CNF). Reasoners that require CNF input include systems based on resolution (McCune, 2003b), some model-building algorithms (McCune, 2003a), probabilistic reasoners using weighted model-counting (Gogate and Domingos, 2011), and more general cognitive architectures that incorporate such reasoners.

CNF is a genuine normal form, in the sense that for every expression of first-order predicate calculus (FOPC), there is a unique logically-equivalent CNF expression. Fortuitously, by mapping to CNF, we eliminate a significant part of the variation that leads to Shieber's logical-equivalence problem. But there is a catch. There is a well-known algorithm that converts FOPC expressions to CNF, but it is not invertible. That is the problem: once we have interpreted a sentence, converted the meaning to CNF, and passed it to an automated reasoner, we do not have a way of taking the CNF expressions that the reasoner produces as output and mapping them to English.

Proceedings of the Society for Computation in Linguistics (SCiL) 2018, pages 55-63.
Salt Lake City, Utah, January 4-7, 2018

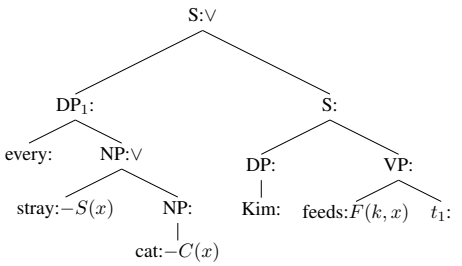

Figure 1: A tree that serves simultaneously as English LF and parse tree for the CNF translation $-S(x) \vee -C(x) \vee F(k, x)$.

1.2 A solution

The solution I propose can be stated briefly as follows. (1) Use CNF as the target of semantic translation. (2) Instead of assembling the translation in a bottom-up pass through the parse tree, creating larger and larger partial translations at each step, let us label selected nodes in the parse tree with CNF operators. In other words, take the English parse tree *to be* the CNF parse tree, albeit with some extraneous nodes and labels. The resulting tree is symmetric between English and CNF (e.g., Figure 1). In particular, the leaf nodes are labeled symmetrically with English words and CNF literals. Define a standard grammar with features to generate such trees. (3) Given a CNF expression as input from general cognition, use the grammar to parse the sequence of literals, constructing an English/CNF parse tree, and read off the English sentence.

Figure 1 provides an illustration of a combined English/CNF parse tree. The English portion is the LF for the sentence *Kim feeds every stray cat,* and the CNF portion represents the translation $-S(x) \vee -C(x) \vee F(k, x)$. Each node has a label pair α:β. For the purposes of the grammar, the label pairs are simply complex categories; we construct a single grammar that generates the pairs. When parsing English, the input consists of the English labels of leaf nodes (*Kim feeds every stray cat*) and when parsing CNF, the input consists of the CNF labels $-S(x), -C(x), F(k, x)$.

A few complications must be addressed, but they have known solutions. We must convert the tree from LF to SS before parsing English, creating a necessity for two different versions of the grammar, one for LF and one for SS. However, both versions generate the same labels, and the two-step process of parsing and converting to LF is standard and familiar. In the CNF-to-English direction, the CNF input will actually be partially parsed input: for example, $[_\vee -S(x), -C(x), F(k, x)]$. We do not pass the nonterminal nodes directly as input to the parser, but rather use them to constrain the operation of the parser. In our example, the constraint prevents the parser from constructing a node whose right label is not $\vee$. Further, CNF is a "free word order" language. Unordered inputs make for less efficient parsing, but they are manageable, and the partial-parse constraints actually ameliorate the problem. There are also more empty nodes in the CNF-to-English direction than in the other direction—for example, in Figure 1, the nodes labeled "every:," "Kim:," and "t_1:" are all empty nodes in the CNF-to-English direction—but parsers routinely deal with empty nodes, and dealing with many of them is no harder than dealing with a few. In short, handling these issues requires some care in implementation but no novel parsing techniques.

The main question I will address in the rest of the paper is how we systematically design the grammar, that is, how we determine what the CNF labels should be.

2 Direct translation constrained by feature propagation

2.1 Assigning FOPC labels

To assign CNF operators to LF nodes, I propose (at least conceptually) that we first label the tree with the usual FOPC translation, and then apply the standard conversion to CNF. I adopt the particularly direct form of translation sketched in the previous section. A key desideratum is that the assignment of semantic operators and atomic formulae to parse-tree nodes should be constrained by local feature constraints of the usual sort. The full power of feature grammars will not be required; features with atomic values will suffice.

Let us construct the LF tree for the sentence *Kim feeds every stray cat* and annotate it with the obvious FOPC translation (Figure 2). I have made one unusual assumption in the LF tree: the determiner *every* has been raised to become head of the quantifier-raising structure. This is not essential, but it will simplify the statement of certain constraints in what follows.

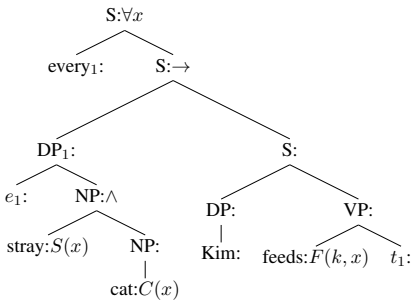

Figure 2: The LF tree labeled with the standard FOPC translation.

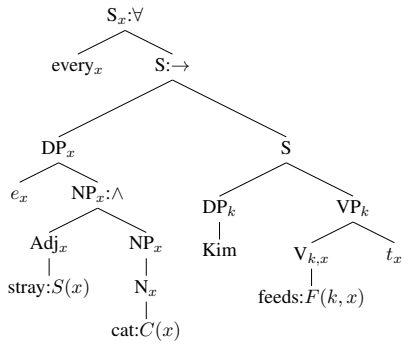

Figure 3: The results of index propagation.

How do we specify this labeling within the grammar? Constraining the occurrence of semantic operators and predicates is generally straightforward and local. For our example, we may state as a general rule that an NP representing adjectival modification translates as $\wedge$, that an S headed by a (raised) universal determiner translates as $\forall$ (we return shortly to the question of the variable), and that an S that is sibling of a universal determiner translates as $\rightarrow$. In the leaf-node translations, the predicates obviously represent the lexical translations of the corresponding English words (S for *stray,* C for *cat,* F for *feed*). It is less obvious how to constrain the choice of variables.

2.2 Index propagation

The standard approach relies heavily on lambda expressions to specify which variable goes in which position. This is a major cause of difficulty in inversion: the inverse of beta-reduction is infinitely ambiguous. Instead of using lambda expressions, let us replace numeric syntactic indices with the semantic variables themselves and propagate them through the tree by syntactic feature-passing. We can then use the syntactic indices to determine the choice of variables in the atomic formulae.

In our example, let us use the variable k for the subject DP and x for the raised object DP. Let us propagate the DP index throughout the DP, and use it to determine the variables in the atomic formulae $S(x)$ and $C(x)$, as in Figure 3. (I give a more rigorous characterization of the spreading in Constraint 2 below.)

As for the variables in $F(k, x)$, let us assume a form of syntactic concord in which the subject's index k is shared with the VP and then, because V is the head of VP, with the V. Let us also impose an object-agreement constraint on the verb, requiring its second subscript to match the object.

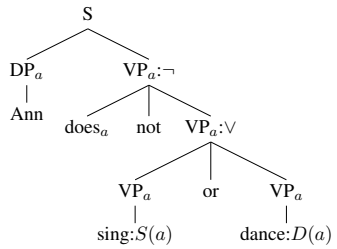

Figure 4: Negation and VP disjunction; the translation is $\neg(S(a) \vee D(a))$.

Only two minor items remain: traces obtain their indices from their antecedents in the usual way, and the variable associated with $\forall$ is now written as an index. Figure 3 shows the final result. Henceforth I omit colons when the semantic label is empty. I also usually omit preterminal nodes—Adj, N, V—to save space, but I include them when needed for clarity.

Let us consider some more examples (adapted from Heim and Kratzer (1997)). These will motivate additions to the index propagation rules, and will illustrate at least a small range of cases in which index propagation can be used in lieu of lambda expressions. The tree in Figure 4 illustrates negation and disjunction, and the trees in Figure 5 illustrate the handling of "case-marking" versus "lexical" prepositions. Relative clauses and multiple quantifiers are illustrated in later trees.

Note that Figure 4 includes an extension of the index propagation rules: the auxiliary (namely, *does*) shares its index with its complement (the disjunctive VP). In Figure 5 we have extended the object-agreement rule to apply to the two-place adjective *fond,* and we have treated *of* like an auxiliary in the

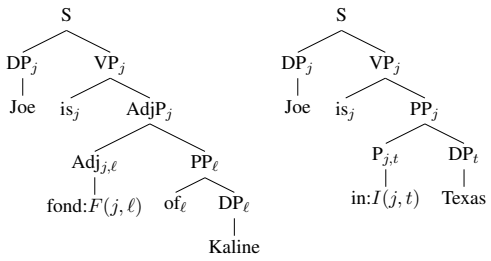

Figure 5: Differing treatments of case-marking (left) and lexical (right) prepositions.

sense that it shares its index with its complement. In the right-hand tree, *in* is treated like a transitive verb, sharing its first index with its parent and sharing its second index with its object.

2.3 Constraints on indices

Based on the examples we have considered, we may hazard a general statement of the index propagation rules. Indices are present only as required by the following two constraints.

Constraint 1 (Intrinsic Indices) *Every DP has an index (excluding pleonastics). A leaf node labeled with atomic formula $\pi(\alpha_1, \ldots, \alpha_n)$ must be child of a preterminal with syntactic indices $\alpha_1, \ldots, \alpha_n$.*

Constraint 2 (Index Propagation) *Syntactic indices are propagated as necessary to satisfy the following requirements.*

1. *A trace has the same index as its antecedent.*
2. *A modifier has the same index as the node it modifies.*
3. *A function word (e.g., auxiliary, case marker) has the same index as its complement.*
4. *An argument-taker's last index is the same as the argument's index. In this case and this case only, the index is* **discharged**.
5. *A parent inherits its head's undischarged indices.*

"Head" includes $\overline{X}$ heads, as well as the head in an adjunction structure, all heads in a coordination structure, and the relative pronoun in a relative clause.

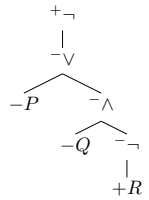

Figure 6: Local propagation of node polarity substitutes for negation lowering.

3 Conversion to CNF

The standard conversion from unrestricted FOPC to CNF involves a sequence of tree transformations: (1) rewriting conditionals, (2) lowering negation, (3) Skolemization, and (4) distribution (of disjunction over conjunction). We would like to consider how to implement the conversion via feature constraints, without altering the basic structure of our LF trees.

3.1 Negation lowering

Let us begin with negation lowering. Its effect is to eliminate the negation operator in favor of **literals**, consisting of an atomic formula and a sign (positive or negative). By adding polarity as an attribute of all nodes, not just terminal nodes, we can give a succinct characterization of negation lowering in the form of a local constraint that is readily implemented in a feature grammar.

Constraint 3 (Polarity) *(a) The polarity of the root node is positive. (b) The polarity of a child node is the same as the polarity of its parent, unless the parent node is labeled with an operator that is* **polarity-reversing** *for the child in question, in which case the child's polarity is the opposite of the parent's.*

For our purposes there are two polarity-reversing operators: negation and the conditional $\rightarrow$, which is polarity-reversing for its first child.

As an example, the FOPC expression $\neg(P \vee (Q \wedge \neg R))$ has the parse tree in Figure 6. Polarities have been added in accordance with Constraint 3. In particular, $\neg$ has inverse polarity to that of its child, but otherwise parent and child always have the same polarity. We achieve the effect of negation lowering by interpreting **signed operators** as specified in Table 1. Replacing the signed operators with their unsigned equivalents for readability, Figure 6 corresponds to the expression $-P \wedge (-Q \vee R)$, which

$^{+}\neg \ = \epsilon$	$^{-}\neg \ = \epsilon$
$^{+}\wedge \ = \wedge$	$^{-}\wedge \ = \vee$
$^{+}\vee \ = \vee$	$^{-}\vee \ = \wedge$
$^{+}\rightarrow = \vee$	$^{-}\rightarrow = \wedge$
$^{+}\forall \ = \epsilon$	$^{-}\forall \ = \epsilon$
$^{+}\exists \ = \epsilon$	$^{-}\exists \ = \epsilon$

Table 1: The interpretations of signed operators.

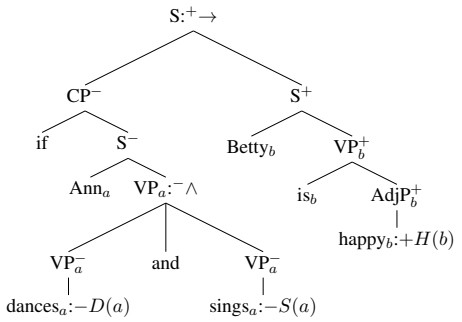

Figure 7: An LF tree illustrating polarity reversal under $\rightarrow$.

is indeed logically equivalent to $\neg(P \vee (Q \wedge \neg R))$. Note that ϵ indicates deletion of the operator.

3.2 Rewriting conditionals

In the standard conversion to CNF, rewriting conditionals precedes negation lowering. We can deal with conditionals as follows. We define the signed operator $^{+}\rightarrow$ to be a synonym for $\vee$ and $^{-}\rightarrow$ to be a synonym for $\wedge$. However, unlike $\vee$ or $\wedge$, $^{\pm}\rightarrow$ reverses the polarity of its first child. Consider the example of Figure 7. The $\rightarrow$ operator inverts the polarity of its first child, but otherwise polarities are passed unchanged from parent to child. Accordingly, Figure 7 is equivalent to $-D(a) \vee -S(a) \vee +H(b)$, which is indeed equivalent to $D(a) \wedge S(a) \rightarrow H(b)$, the natural translation for *if Ann dances and sings, Betty is happy.*

Note that Table 1 is used in the process of "reading off" the CNF expression for input to the reasoner; it is not used to eliminate signed operators from the LF tree. The signed operators do serve a purpose beyond the truth function they represent. For one thing, even though $^{+}\rightarrow$ is equivalent to $\vee$, the former reverses its first child's polarity whereas the latter does not. The signed operators also permit us to use local constraints to define the assignment of translations. An example of such a local constraint is the following: *a node has semantic operator $\rightarrow$ if it is headed by a CP headed by "if."* Such a statement remains valid whether the polarity of the node is positive or negative, though in the former case the signed operator $^{+}\rightarrow$ is interpreted as $\vee$ and in the latter case the signed operator $^{-}\rightarrow$ is interpreted as $\wedge$.

3.3 Skolemization

The third step of the conversion to CNF is Skolemization. As usually formulated, one replaces existentially bound variables with Skolem terms consisting of a Skolem function applied to the list of outscoping universal variables, then one deletes all quantifiers. The deletion is already reflected in Table 1—though, as already mentioned, the signed operators remain in the LF tree and are not actually deleted until we read off the CNF expression for input to the reasoner.

I will use the term *variable* to refer loosely to both universal variables (that is, implicitly universally-bound variables) and Skolem terms. I write Skolem functions with a dot, e.g., $\dot{x}$, to make it easy to distinguish them from universal variables. Whether a variable should be a universal variable or a Skolem term is determined by the signed operator at the variable's **home**, which I define to be the node labeled with the quantifier that originally bound it. Specifically:

Constraint 4 (Variable type determination)
(a) The syntactic index of a node whose signed operator is $^{+}\forall$ or $^{-}\exists$ must be a universal variable, and (b) the syntactic index of a node whose signed operator is $^{+}\exists$ or $^{-}\forall$ must be a Skolem term.

This constraint determines the type of variable, and the variable is then propagated to other nodes by Constraint 2. See Figure 8 for an example.

The variable $\dot{y}$ in Figure 8 is a shorthand for the Skolem term $\dot{y}(x)$. To avoid clutter, I have suppressed the argument list, but it does need to be computed in a complete implementation. One may use a feature `ouv` whose value for a given node ν is the list of **outscoping universal variables**, that is, the list of universal variables whose home position dominates ν. It is straightforward but tedious to write out the feature constraints that determine the correct value for `ouv`; I omit the details.

Figure 8 provides an example with two quantifiers. Note that there are two polarity reversals, both

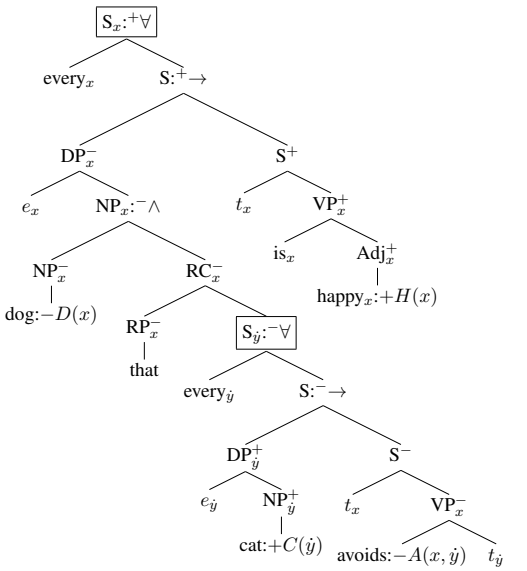

Figure 8: A complex example. The boxed nodes are the home nodes for the variables x and $\dot{y}$.

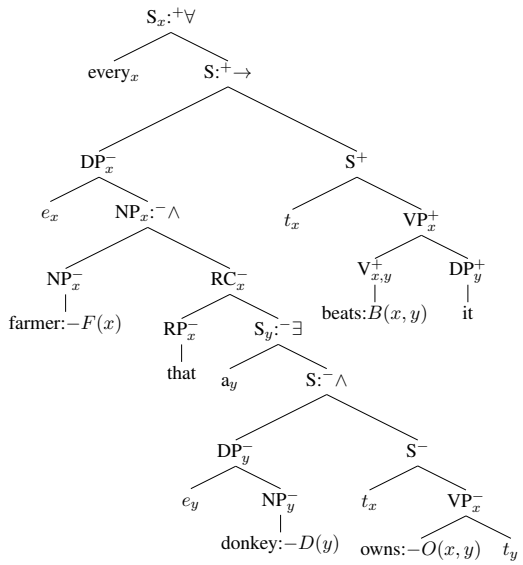

Figure 9: Donkey anaphora is covered without stipulation.

occurring at the first child of a node with operator $\rightarrow$. The boxed nodes are the homes of the two quantifiers. Because the upper one has signed operator $^+\forall$, the variable is a universal, and because the lower one has signed operator $^-\forall$, the variable is a Skolem term.

Reading off the CNF, we obtain the following. For readability, I have again replaced the signed operators with their more familiar unsigned equivalents; I also indicate the Skolem argument lists explicitly.

$$-D(x) \vee [C(\dot{y}(x)) \wedge -A(x, \dot{y}(x))] \vee H(x).$$

In words: either x is not a dog, or else x's $\dot{y}$ is a cat that x fails to avoid, or else x is happy. That is inferentially equivalent to the original sentence *every dog that avoids every cat is happy.*

3.4 Donkey anaphora

A pleasant side effect of the lack of explicit quantifiers in CNF is that donkey anaphora becomes available without any stipulations. The structure of *every farmer that owns a donkey beats it* is essentially the same as that in Figure 8 except for the choice of lower quantifier: see Figure 9. I assume that the pronoun *it* picks up the index of its antecedent *a donkey.* The resulting CNF translation is $-F(x) \vee -D(y) \vee -O(x, y) \vee B(x, y)$, which correctly captures the strong reading.[1]

3.5 Distribution

The final step in the conversion to CNF is distribution of disjunction over conjunction. Distribution unavoidably involves a structural transformation of the tree, so we will not attempt to incorporate it into the LF structure. Although distribution is not uniquely invertible, the degree of ambiguity that arises in inversion is limited. When translating from CNF to English, let us assume the inverse of distribution, which we may call **consolidation**, as a preprocessing step.

To fix ideas, let us consider the following example:

$$\phi = P \vee (Q \wedge R)$$

The result of distribution is ψ:

$$\psi = (P \vee Q) \wedge (P \vee R)$$

In general, whenever distribution is non-trivial, it has the effect of introducing copies of existing atomic formulae, as with P in ψ. Hence inverting distribution (consolidation) involves recombining copies. Consolidation is ambiguous. For example, ϕ is not the only undistributed expression that

[1] I suggest that the weak reading is spurious; the use of the singular "a donkey" presupposes that the set of owned donkeys is a singleton, in which case the strong and weak readings are equivalent.

may give rise to ψ: ψ itself might have been the source. More generally, every way of combining copies produces a form that constitutes a possible undistributed source.

On the other hand, the amount of ambiguity is limited by the number of duplicates in the input (going from CNF to English). Moreover, each possible result of consolidation does give rise to a valid English sentence. The choice among them is not one of well-formedness but of stylistic preference. Since each duplicate atomic formula gives rise to duplicated words, it seems natural to prefer to do as much consolidation as possible, and we may adopt that as a heuristic.

In most cases, there is a unique most-consolidated form, though it is possible to construct examples with multiple distinct maximally-consolidated forms. For example, given the CNF expression $(P \vee Q) \wedge (Q \vee R) \wedge (R \vee P)$, we may eliminate any one of the three duplicate pairs, but only one, leaving us with three different maximally-consolidated forms. This is not likely to be a major problem in practice.

4 Discussion

4.1 Generalized quantifiers

Important questions remain. Perhaps the most urgent is how generalized quantifiers are to be accommodated in the proposed approach. Generalized quantifiers are relations between sets, so the question can be rephrased as accommodating phrases (namely, NPs) that define sets.

One can include sets in a first-order account by reification. That is, introduce a membership predicate M and define a set such as $\dot{s} = \lambda x \,.\, \phi[x]$ as:

$$M(x, \dot{s}) \leftrightarrow \phi[x].$$

The problem is that converting an LF tree that contains $\leftrightarrow$ to CNF involves a substantial structural change. For example, $M(x, \dot{s}) \leftrightarrow F(x) \wedge G(x)$, converted to CNF, expands out as:

$$\begin{array}{l} [-M(x, \dot{s}) \vee F(x)] \wedge \\ [-M(x, \dot{s}) \vee G(x)] \wedge \\ [-F(x) \vee -G(x) \vee M(x, \dot{s})]. \end{array} \quad (1)$$

My proposal relies crucially on the conversion to CNF being structure-preserving, but this is a case in which it emphatically does not preserve structure.

One possibility is to permit $\leftrightarrow$ in the LF tree as a primitive operator, and to handle it much as we handled distribution. Going from the LF tree to the reasoner, an expression containing $\leftrightarrow$ is expanded out as in (1). In the opposite direction, as a preprocessing step one seeks instances of the pattern illustrated in (1) and replaces them with $M(x, \dot{s}) \leftrightarrow F(x) \wedge G(x)$, much as we recognize the repetitions that may be consolidated. A more ambitious alternative is to incorporate the replacement into the reasoner as an inference rule, much as reasoners often include primitive support for an equality predicate and substitution of equals. I leave this as a question for future research.

4.2 Related work

There have been proposals in the literature for reversible grammars, which support both interpretation and generation (Appelt, 1987; de Kok et al, 2011; Copestake et al, 1996; Melamed, 2003; Shieber, 1988; Shieber and Schabes, 1990; Shieber et al, 1990; Strzalkowski, 1991; Strzalkowski, 1994). Reversibility was indeed one of the original motivations for unification grammars (Kay, 1975; Kay, 1996), though the translational target was predicate-argument structure rather than FOPC. The present paper can be seen as extending that work to map bidirectionally between English and CNF using a feature grammar.

For general unification grammars, it was proposed that one define an interpretation relation $I(s, \phi)$ in Prolog: to parse, provide the sentence s and solve for the meaning ϕ, and to generate, provide ϕ and solve for s (Shieber, 1988; Shieber et al, 1990). Unfortunately, solving for s proved to be beyond Prolog's abilities, and much work went into elaborate methods for helping Prolog along (Strzalkowski, 1991; Strzalkowski, 1994). In additional, the usual unification grammars were susceptible to the logical-equivalence problem that Shieber pointed out: the range of ϕ in $I(s, \phi)$ is typically not the entire space of FOPC expressions but only a subset of the space, and given an arbitrary input ψ one must seek a logically equivalent ψ' for which $I(s, \psi')$ is defined; but logical equivalence is not decideable (Shieber, 1993).

These difficulties led to an interest in flat semantic languages, which, one hoped, reduce the number

of logically equivalent expressions corresponding to a given semantic input (Whitelock, 1992; Trujillo, 1995). Perhaps the best known current approach is Minimal Recursion Semantics (MRS) (Copestake et al, 2005). However, MRS expressions are not "flat" in the right way—an MRS expression is actually a meta-logical description of a standard FOPC parse tree—and the use of MRS does not ameliorate the logical equivalence problem. The main attraction of MRS is not that it addresses the problems of interest here, but that it supports a transparent and compact representation of certain ambiguities, particularly quantifier-scope ambiguities.

When genuinely flat semantic languages have been proposed (Whitelock, 1992; Trujillo, 1995), they usually have severely limited expressivity, permitting only conjunctions of ground clauses, and excluding disjunction and quantification. By contrast, CNF represents a flat semantic language with the full expressive power of FOPC. It is flat in the sense that no CNF expression has depth greater than three: the most complex CNF expression is a conjunction of clauses, each clause being a disjunction of literals. There are no explicit quantifiers, but their expressive capacity is preserved via the distinction between universal variables and Skolem terms. The use of CNF for semantic translations does ameliorate the logical equivalence problem. A CNF expression is the normal form for an (infinite) equivalence class of unrestricted FOPC expressions, and the commonest sorts of logically equivalent pairs fall together when we map to CNF.

I build also on a line of inquiry into reversibility that involves simultaneous grammars, in which a single derivation constructs two parse trees. Simultaneous grammars have been used both for machine translation (Melamed, 2003) and for translation between English and FOPC (Shieber and Schabes, 1990). However, previous work has not considered the further conversion from FOPC to CNF. Moreover, the simultaneous grammars considered in this paper are unusually simple: the two syntax trees are homomorphic, allowing them to be treated as a single tree with paired labels.

4.3 Conclusion

I have described a method of translating between English and CNF whose advantages are as follows: it provides a direct connection to automated reasoners; it is fully invertible; it is arguably simpler than simultaneous-tree or standard direct-interpretation approaches; it ameliorates the logical-equivalence problem by virtue of CNF's status as normal form; it is computable using an atomic-valued feature grammar, enabling efficient parsing/generation; and it predicts the existence of donkey anaphora as a side effect of Skolemization, which is an essential step in the conversion to CNF. To the extent that the proposal has merit, it illustrates how considerations of the role of interpretation in the larger cognitive system can influence the form of the semantic account in fundamental ways.

Acknowledgements

I have benefited greatly from discussions with Ezra Keshet and joint work we have done on semantic consequences of using CNF as metalanguage. The paper has also benefited from the comments of anonymous reviewers. Obviously, they bear no responsibility for any remaining shortcomings of the work.

References

James Allen. 1995. *Natural Language Understanding* (Second edition). Benjamin Cummings, Menlo Park, CA.

Douglas E. Appelt. 1987. Bidirectional grammars and the design of natural language generation systems. In: *Theoretical Issues in NLP 3.*

Patrick Blackburn and Johan Bos. 2005. *Representation and Inference for Natural Language.* CSLI Publications, Stanford, CA.

Ann Copestake, Dan Flickinger, Robert Malouf, Susanne Riehemann, and Ivan Sag. 1996. Translation using minimal recursion semantics. *Proceedings of the Sixth International Conference on Theoretical and Methodological Issues in MT.*

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan Sag. 2005. Minimal Recursion Semantics: An introduction. *Research on Language and Computation* 3:281–332.

Daniël de Kok, Barbara Plank, and Gertjan van Noord. 2011. Reversible Stochastic Attribute-Value Grammars. *Proceedings of the Conference of the Association for Computational Linguistics (ACL).*

Vibhav Gogate and Pedro Domingos. 2011. Probabilistic Theorem Proving, *Proceedings of UAI.*

Irene Heim and Angelika Kratzer. 1997. *Semantics in generative grammar.* Blackwell Publishers.

Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing* (2nd edition). Prentice Hall, Upper Saddle River, NJ.

Martin Kay. 1975. Syntactic processing and functional sentence perspective. *Proceedings of TINLAP.*

Martin Kay. 1996. Chart generation. *Proceedings of the Conference of the Association for Computational Linguistcs (ACL).*

John E. Laird. 2012. *The Soar Cognitive Architecture.* The MIT Press, Cambridge, MA and London, England.

William McCune. 2003a. *Mace4 Reference Manual and Guide.* Tech. Memo ANL/MCS-TM-264, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL.

William McCune. 2003b. *Otter 3.3 Reference Manual.* Tech. Memo ANL/MCS-TM-263, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL.

Dan Melamed. 2003. Multitext Grammars and Synchronous Parsers, *Proceedings of NAACL.*

Robert C. Moore. 1989. Unification-based semantic interpretation. *Proceedings of the 27th Meeting of the Association for Computational Linguistics.*

Fernando Pereira and Yves Schabes. 1992. Inside-outside reestimation from partially bracketed corpora. *Proceedings of the Association for Computational Linguistics 30th Annual Meeting,* 128–135. Newark, Delaware.

Stuart J. Russell and Peter Norvig. 2002. *Artificial Intelligence: A Modern Approach* (2nd edition). Prentice Hall, Upper Saddle River, NJ.

Stuart Shieber. 1988. A uniform architecture for parsing and generation. *Proceedings of the 12th Conference on Computational Linguistics (COLING),* vol. 2, pp. 614–619.

Stuart Shieber. 1993. The problem of logical-form equivalence. *Computational Linguistics* 19(1), 179–190.

Stuart Shieber and Yves Schabes. 1990. Synchronous Tree-Adjoining Grammars. *Proceedings of the Conference on Computational Linguistics (COLING).*

Stuart Shieber, Gertjan van Noord, Fernando Pereira, and Robert Moore. 1990. Semantic head-driven generation. *Computational Linguistics* 16(1):30–42.

Tomek Strzalkowski. 1991. A general computational method for grammar inversion. *Proceedings of the ACL Workshop on Reversible Grammar in Natural Language Processing.*

Tomek Strzalkowski (ed.) 1994. *Reversible Grammar in Natural Language Processing.* Kluwer Academic Publishers.

Indalecio Arturo Trujillo. 1995. *Lexicalist Machine Translation of Spatial Prepositions.* PhD dissertation, University of Cambridge.

P. Whitelock. 1992. Shake-and-bake translation. *Proceedings of the Conference on Computational Linguistics (COLING).*