A Constrained Semantics for Parsed English Sentences

Alastair Butler[*][†] **Yusuke Miyao**[‡] **Kei Yoshimoto**[†] **Jun'ichi Tsujii**[‡]
[*]**Japan Society for the Promotion of Science**
[†]**Center for the Advancement of Higher Education, Tohoku University**
[‡]**Graduate School of Information Science and Technology, Tokyo University**

Abstract A report is given on an experimental semantics-based system that carries out a recursive semantic evaluation on parsed forms of natural language. The system derives from a static reformulation of the results of dynamic semantics that works by placing instrumented constraints on possible dependencies. We describe our experience with a specific implementation of the proposal that takes parsed forms of English and outputs predicate logic formulae as semantic representations. This has proven to be robust, efficient, and to offer a wide coverage.

1 Introduction

The last two decades has seen the development of statistical approaches to natural language processing. While this has led to high-speed and robust parsing technologies for a great variety of linguistic data, it has generally been abandoned to automatically understand sentences as deeply as humans do. In this paper, we give an account of our experiment that combines a dynamic semantics-based system for semantic evaluation with a probabilistic HPSG parser. We show that it robustly translates a wide coverage of English sentences into predicate logic formulae with possible linkage to the context.

The remainder of this paper is structured as follows. Section 2 sketches the semantic framework we rely on, Scope Control Theory, and the HPSG parser Enju. Section 3 depicts how the two independently developed systems, the syntactic parser and the constrained semantics-based system, have been integrated, and illustrates the results of our English sentence processing experiments. Section 4 concludes the paper, summarising our research and explaining its significance.

2 SCT and Enju

2.1 SCT

Scope Control Theory or SCT (Butler 2007, forthcoming) is a small logical language that approximates the dependency structures in natural languages by fine-grained and restricted scope management. This is realised by combining the Sequence Semantics of Vermeulen (1993) with static versions of Dynamic Semantics by Dekker (2002) and Cresswell (2002). Unlike the standard Dynamic Semantic theories, however, SCT is setup for the processing of parsed forms of natural language, with the runtime of evaluations being used to output predicate logic translations.

In SCT, everything that is standardly achieved with grammatical rules in existing syntactic theories is managed with the manipulation of scopes, captured with the states of an assignment function that assigns sequence values to binding names. Binding names can, for example, correspond to grammatical functions like *subject* and *object*.

The following basic operators are used as building blocks to construct SCT formulae:

- The `T` operator takes a binding name as argument and provides its value (i.e., a term), which is usually the topmost one (the `0`th member) in the sequence of values for the binding name. Successful translation of a term returns a predicate logic variable which, for example, fills an argument slot in a predicate.

- `Use` marks a domain called *locality* in which a scope is active. The operators form countable entities with usage counts that signal the scope requirements of an expression. `Use` can tell `Close` how many scopes to create (creation support), and can tell `Rel` what scopes are to be made available for given names of a given argument (binding support).

- The `Hide` operator makes the marking by `Use` invisible from an embedding environment to avoid interference. Customarily located directly above `Close` for a particular binding name, it prevents an embedded `Use` from adding to the usage counts of higher `Close` or `Rel` operators for the same name. `Hide` operators can be taken to mark localities that enable the reuse of binding names, which is necessary for embeddings.

- `Close` provides an environment where new scopes can be opened. It creates new scopes in the assignment corresponding to a usage count for the number of `Use` that are not occurring under `Hide`, for a particular binding name. When the usage count is greater than zero, `Close` operators translate into quantifiers introducing new scopes.

- `Lam` shifts one scope from the front of a named sequence to the front of another named sequence.
- `Rel` plays two important roles on translation. Firstly, it builds relations including not only predicates, but all other relations like coordinating relations. Secondly, working together with `Use`, it distributes the correct scopes to each argument of the relations and, based on the assignments, translates its subexpressions to construct each argument of the resulting relation.

On the basis of the basic operators given above, syntactic sugar is defined to develop a grammar. Central to this is `r`, which lays the foundation for defining predicates. `r` takes six parameters, `lc` for local bindings, `fh` for fresh bindings, `args` for the binding names for the required (core) arguments, `atch` for the binding names for any attached (non-core) arguments, and `ext` that is used for unbounded dependencies. These binding names undergo the checksum calculation carried out by the `Rel` operator so that they may be neither more or less than necessitated in any part of the sentence.

(1b) is the SCT expression corresponding to example (1a). (2a,b) provide the condition under which (1b) is evaluated: `"e"` is the possible source for fresh bindings, and `"h"` and `"x"` are possible local bindings. (2c,d) show how `smiles` and `girl` occurring in (1b) are defined by means of `r`. `some` used by (2e) essentially constructs a framework for an existentially quantified sentence by shifting a fresh binding (`"e"` in this case) to a local binding, and checking the fresh and local bindings for the noun phrase restriction. The checksum is made against every relevant subtree so that it may be given the exact numbers of local bindings by the assignment which are required for the evaluation of the predicates. '//' is an infix operator that combines the left-hand constituent for the subject NP with the one for the VP to the right. Figure 1 outlines the process of evaluation of (1b) that successfully outputs (3) as predicate logic translation. Note that the existential closure meant by (1a) corresponds to there being some assignment that can assign values to the free variables.

(1) a. A girl smiles.

b. `a girl // smiles`

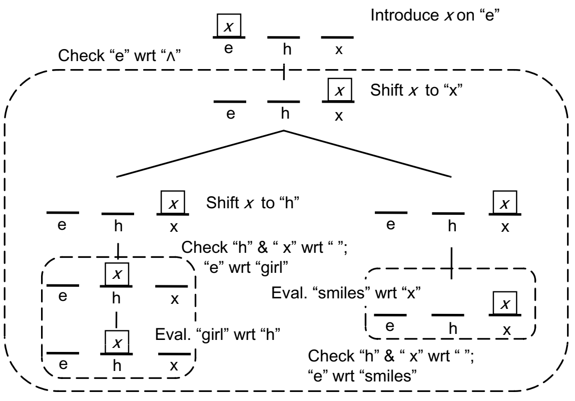

Figure 1: Outline of (1b) evaluation

(2) a. `lc = ["h", "x"]`

b. `fh = ["e"]`

c. `smiles = r lc fh ["x"] nil nil "smiles"`

d. `girl = r lc fh ["h"] nil nil "girl"`

e. `a =` λ`f.some lc fh "e" [f]`

(3) $\exists g$: $(g, (2a))^\circ = \mathrm{girl}(x) \wedge \mathrm{smiles}(x)$

The examples below illustrate how anaphoric dependencies can be established without any coindexing with SCT. (4a) is correctly evaluated as (6a), since the reflexive pronoun `himself` is given the specification that it is identical to the topmost value of the binding name for the subject (see `T ("x", 0)` in (5g)). `him` in (4b) is successfully identified with `a boy`, since this pronoun refers to the topmost value of `"c"` (see `T ("c", 0)` in (5h)) and crossing into the embedding of `thinks` brings the consequence of shifting what is the `"x"` binding in the matrix clause to become the topmost `"c"` binding in the embedded clause. '\\' is an operator that combines the verb to the left with the right-hand object NP.

(4) a. `a boy // (likes\\himself)`

b. `a boy // (thinks (a teacher // (likes\\him)))`

(5) a. `fh = e`

b. `lc = ["h", "x", "y"]`

c. `a =` λ`f.some lc fh "e" [f]`

d. `boy = r fh lc ["h"] nil nil "boy"`

e. `teacher = r fh lc ["h"] nil nil "teacher"`

f. `likes = r fh lc ["x", "y"] nil nil "likes"`

g. `himself = pronoun (T ("x", 0)) fh`

h. `him = pronoun (T ("c", 0)) fh`

i. `thinks = remb fh lc ["x"] nil nil "thinks"`

(6) a. $\exists g$: $(g, \text{(5a)})^{\circ} = \text{boy}(x) \wedge \text{likes}(x,y) \wedge x = y$

b. $\exists g$: $(g, \text{(5c)})^{\circ} = \text{boy}(x) \wedge \text{thinks}(x, \text{teacher}(y) \wedge \text{likes}(y,z) \wedge x = z)$

In this manner it has been shown that SCT can capture a variety of grammatical dependencies including the obligatory case NP – verb relationship, adverbial adjunction, relative clause, voices, WHs both fronted and in situ, topic-subject coreference, tense interpretation, among others, in English, Japanese, French, Tukang Besi (Indonesia), and other languages (see e.g. Butler 2007, forthcoming).

2.2 Enju

Enju is a syntactic parser for English that is based on probabilistic modeling of the lexicalized grammar HPSG (Miyao and Tsujii 2008). While SCT requires grammatical categories of the words and the hierarchical structure of the sentence as input, Enju can provide both rich lexical information, parts of speech and subcatagorisation frames among others, and detailed information on constituent structures including that for unbounded dependencies. Furthermore, it has been proven to be robust: it can produce complete parsed forms for 99.7% of news-wire texts. It assigns correct predicate-argument relations to about 83 to 90 % of their test set sentences with less than 100 words, depending on the reference model adopted (Miyao and Tsujii 2008).

3 Integrating the systems

We have developed an experimental system that integrates both Enju and the SCT engine implemented with Standard ML.[1] Whereas Enju can provide SCT with sufficient parsed information, its output is not directly applicable to the latter. Therefore, a translation process is indispensable that converts the XML format output of parsing into the style acceptable by SCT (see Figure 2, an outline of an XML output by Enju; and Figure 3, which exemplifies an SCT formula inputted to the SCT engine). This includes both conversion of terminal nodes into SCT lexical entries and that of constituent structures into SCT subformulae.

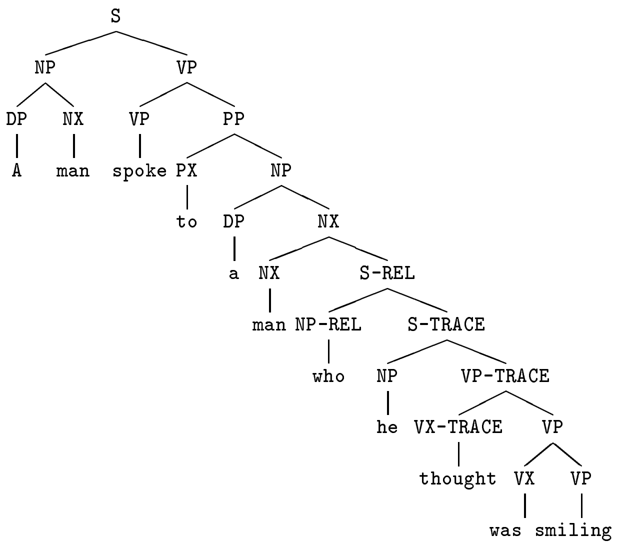

Figure 2: Enju XML output in a graphical view

The following are examples of word conversion:

- A common noun
  man ⇒ `r lc fh ["h"] [] nil "man"`
- A preposition *to* ⇒ `prep "to"`
- An intransitive verb
  smile ⇒ `r lc fh ["x"] [] nil "smile"`

```
some lc fh "e"
 [r lc fh ["h"] [] nil "man"] //
  (r lc fh ["x"] ["to"] nil "speak-past" \\
   prep "to"
    (some lc fh "e"
     [r lc fh ["h"] [] nil "man",
      relc_base "x" lc fh
       comp
        (pronoun (T ("c", 0)) fh //
         (remb lc fh ["x"] [] ["x"]
          "think-past" \\
          (comp r lc fh ["x"] [] nil
           "smile-prog-past")))]))
```

Figure 3: Input to SCT

[1]A demonstration of the enju+SCT system is given at the following site:
http://www-tsujii.is.s.u-tokyo.ac.jp/enju/enju_sct/demo.cgi

- An intransitive verb with an embedded sentence
  think ⇒ `remb lc fh ["x"] [] nil "think"`
- An indefinite article *a* ⇒ `some lc fh "e"`
- A pronoun *he* ⇒ `pronoun (T ("c", 0)) fh`

The constituent structures obtained by Enju are transformed to fit the SCT formalism in the following manner:

- Subject-head construction ⇒ *`Subject_NP`* // *VP*
- Verb-headed head-complement construction
  ⇒ *V* \\ *`object_NP`*
- Relative clause construction ⇒ *`nominal_head`*, `relc_base` *X* `lc fh` *`rel_clause`*, where *X* is the binding name for the relativized argument.

The output from Enju and the required input to the SCT engine are systematically different in some respects, specifically in modification and unbounded dependencies. In distinction from the HPSG-based treatment of modification according to which the modifier selects the modifiee, in SCT the lexical entry for the head provides information on the modifiee. For instance, to process the example in Figure 2, SCT extends the verb *speak* to include an additional binding name `to` as its second argument to incorporate information on the addressee. For this purpose, we need pre-processing of modifiers for each word to create SCT lexical entries. Relative clauses are dealt with similarly. Since the predicate within the clause needs information on the binding name of the relative pronoun, we carry out extra processing to obtain this information.

We have found that the SCT engine can produce accurate evaluations from a large majority of correct parsings from Enju. They cover personal and reflexive pronouns, passive voice, an interrogative WH *fronted* from an embedded clause, relative clauses, subordinate clauses, and donkey sentences. Furthermore, it successfully processes a short discourse of sentences with the occurring pronouns properly bound.

4 Conclusions

We have given an account of an alternative approach to the computational understanding of sentence meanings. By contrast to the standard syntax-driven methods that lead to uncontrollability of an enormous amount of grammatical rules, it gets around the difficulty by semantic evaluation of syntactically parsed results. It is specifically advantageous in that it can be easily scaled up, and requires no rich lexicon owing to its ability to build/adjust lexical entries depending on constituent context. One of the possible developments of the theory is to integrate information both from the sentence and the context. This can be easily achieved with SCT, since both kinds of information are dealt with equally as scopes.

References

Butler, A. 2007. Scope control and grammatical dependencies. *Journal of Logic, Language and Information* 16, 241-264.

Butler, A. forthcoming. *The Semantics of Grammatical Dependencies.* Emerald, Bingley.

Cresswell, M. J. 2002. Static semantics for dynamic discourse. *Linguistics and Philosophy* 25, 545-571.

Dekker, P. (2002) Meaning and use of indefinite expressions. *Journal of Logic, Language and Information* 11, 141-194.

Miyao, Y., and J. Tsujii. 2008. Feature forest models for probabilistic HPSG parsing. *Computational Linguistics* 34:1, 35-80.

Vermeulen, C. F. M. 1993. Sequence semantics for dynamic predicate logic. *Journal of Logic, Language and Information* 2, 217-254.

Contact email addresses
Alastair Butler: ajb129@hotmail.com
Yusuke Miyao: yusuke@is.s.u-tokyo.ac.jp