Meaning representations from treebank annotation

Alastair BUTLER[*†]　Kei YOSHIMOTO[†]
[*]PRESTO, Japan Science and Technology Agency
[†]Center for the Advancement of Higher Education, Tohoku University

Abstract

This paper describes an implemented system that directly obtains meaning representations of quality from treebank annotations. The central component is a system of evaluation for a small formal language with respect to a structured information state. Inputs to the system are expressions of the formal language obtained from the conversion of parsed treebank data. Outputs are Davidsonian (higher order) predicate logic meaning representations. Having a system of formal evaluation as the basis for generating meaning representations makes possible accepting input with minimal conversion from existing treebanks and treebank parsers.

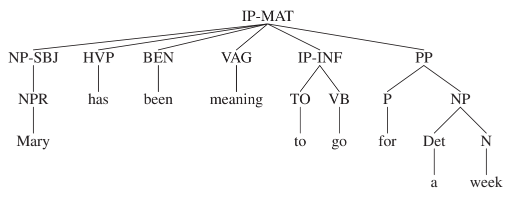
1 Introduction

This paper describes an implemented system that offers a way to obtain meaning representations for natural language. The method involves conversion of parsed constituent tree annotations into expressions of a small formal language (Scope Control Theory or SCT; Butler 2010) which can be subsequently processed with respect to a sequence based information state (cf. Vermeulen 2000, Dekker 2012) to return predicate logic based meaning representations. The method is of interest since, without requiring explicit indexing to be coded with the input phrase structure data, the output meaning representations obey a wide range of valid and cross-linguistically robust binding dependency patterns, including scope effects, locality effects, control effects, island effects, intervention effects, circumstances for long-distance dependencies and accessibility of anaphoric referents. On the practical side, this work gives a foundation for automatically creating meaning representations of high quality and with binding dependencies correctly resolved when unambiguous, on the back of existing treebank annotations and treebank parsers.

2 Annotation

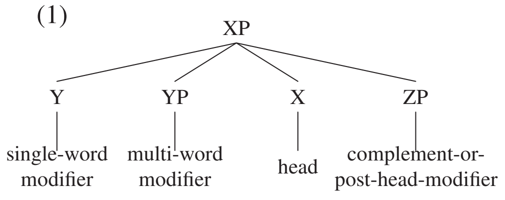
The implemented system is currently tuned to accept parsed data that conforms to the *Annotation manual for the Penn Historical Corpora and the PCEEC* (Santorini 2010), hereafter referred to as the annotation system. This is a widely and diversely applied annotation scheme, forming the basis for annotations of English, French, Icelandic, Portuguese, Ancient Greek, Yiddish, Japanese, among others.

With the annotation system constituent structure is represented with labelled bracketing and augmented with grammatical functions and notation for recovering discontinuous constituents. A typical parse in tree form looks like:



Every word has a word level part-of-speech label. Phrasal nodes (NP, PP, ADJP, etc.) immediately dominate the phrase head (N, P, ADJ, etc.), so that the phrase head has as sisters both modifiers and complements following the scheme of (1).

(1)



Modifiers and complements are distinguished because there are extended phrase labels to mark function (e.g., -INF above encodes that the clause *to go* is a complement of the matrix clause head *meaning*). All noun phrases immediately dominated by IP are marked for function (NP-SBJ=subject, NP-OB1=direct object, NP-TMP=temporal NP, etc.). All clauses have extended labels to mark function (IP-MAT=matrix clause, CP-ADV=adverbial clause, CP-REL=relative clause, etc.).

3 Meaning representations

To automatically build meaning representations, the first step is to convert a labelled bracketed tree into an expression that can serve as input to the SCT evaluation system. As a demonstration, consider the opening tree of section 2, here given with bracketed notation:

```
(IP-MAT (NP-SBJ (NPR Mary))
        (HVP has)
        (BEN been)
        (VAG meaning)
        (IP-INF (TO to)
                (VB go))
        (PP (P for)
            (NP (D a)
                (N week))))
```

An SCT expression is built by exploiting the phrase structure, which adheres to the scheme of (1), by locating any complement for the phrase head to scope over, and adding modifiers as elements that scope above the head, with, for example, the following intermediate results:

```
NP-SBJ-in: (NPR mary)
NP-SBJ-out: (NP-SBJ npr "entity"
"mary"__LOCAL__)

IP-INF-in: (TO to)@IP-INF@@(VB go)
IP-INF-out: (IP-INF-to verb lc fh "event"
[] "go"__LOCAL__)

NP-in: (D a)@NP@@(N week)
NP-out: (NP some lc fh "time" (nn lc fh
"week")__LOCAL__)

PP-in: (P for)@PP@@(NP some lc fh "time"
(nn lc fh "week")__LOCAL__)
PP-out: (PP-NP (some lc fh "time"
(nn lc fh "week")) "for"__LOCAL__"for")

IP-MAT-in: (NP-SBJ npr "entity" "mary"
__LOCAL__)@IP-MAT@@(HVP has)@IP-MAT@@
(BEN been)@IP-MAT@@(VAG meaning)@IP-MAT@@
(IP-INF-to verb lc fh "event" [] "go"
__LOCAL__)@IP-MAT@@(PP-NP (some lc fh
"time" (nn lc fh "week")) "for"
__LOCAL__"for")
IP-MAT-out: (IP-MAT-fact ((npr "entity"
"mary") "arg0") (((some lc fh "time" (nn
lc fh "week")) "for") (embVerb lc fh
"event" ["arg0", "for"] "has_been_meaning"
toComp (verb lc fh "event" [] "go")))
__LOCAL__"toComp"@NAME@"for"@NAME@"arg0")
```

With inclusion of information about the possible binding names of the expression (integrated with `fn fh =>` and `fn lc =>`), the overall output from conversion is as follows:

```
val ex1 =
( fn fh =>
  ( fn lc =>
    ( ( ( npr "entity" "mary")
        "arg0")
      ( ( ( some lc fh "time"
            ( nn lc fh "week"))
          "for")
        ( embVerb lc fh "event" ["arg0", "for"]
            "has_been_meaning" toComp
          ( verb lc fh "event" nil "go")))))
  ["toComp", "for", "arg0", "arg1", "h"])
["constant", "event", "time"]
```

This conversion to `ex1` notably transforms into operations the part of speech tags given by the nodes immediately dominating the terminals of the input constituent tree (`npr` (proper name), `some` (indefinite determiner) `nn` (noun), `embVerb` (verb that takes an embedding), `verb` (verb without an embedding), etc.). Conversion also adds information about binding names (`"arg0"` (grammatical subject role), `"arg1"` (grammatical object role), `"toComp"` (complement role), `"for"` ("for" role), `"h"` (generic role), `"constant"`, `"event"` and `"time"`). The created operations further reduce to primitives of the SCT language as demonstrated with (2).

(2)
```
Hide ("constant",
 CClose ("constant",
  Hide ("time",
   Close ("∃", ("time","time"),
     ["event", "time"],
    Hide ("event",
     Close ("∃", ("event",
       "event"),
       ["event", "time"],
      Clean (0, ["arg0"], "c",
       CUse (C ("mary", "entity"),
         "constant",
        Lam ("constant", "arg0",
         Clean (0, ["for"], "c",
          Use ("time",
           Lam ("time", "for",
            Rel (["constant",
               "event", "time"],
               ["c", "c", "c"],
               "CHECK", [
             Throw ("time",
              Lam ("for", "h",
               Clean (0, ["toComp",
                  "for", "arg0",
                  "arg1"], "c",
                Clean (1, ["h"],
                  "REMOVE",
                 If (fn,
                  If (fn, ...
```

The SCT language primitives (`Use`, `Hide`, `At`, `Close`, `Rel`, `If`, `Lam`, `Clean`, among others) access and possibly alter the content of a sequence based information state that serves to retain binding information by assigning (possibly empty) sequences of values to binding names. Evaluation of the resulting SCT expression conspires to bring about the enforcement of fixed roles on the binding names from the conversion of the parsed constituent tree annotation (`"arg0"`, `"arg1"`, `"for"`, etc.).

When binding requirements are specified (with combinations of `Use` and `Hide`) evaluation is constrained to accept only certain ‘grammatical’ usage; and when binding requirements are un(der)specified, evaluation itself provides guidance for determining *when*, *where* and *how* binding dependencies are established by governing the release and subsequent accessibility of bindings. The practical consequence is the automatic production of meaning representations of high quality with binding dependencies correctly resolved when unambiguous. Thus following an evaluation of (2) (see Butler 2010 for exact details) the meaning representation (3) is returned.

(3) $\exists t_1 e_2 e_3\,(\text{week}\,(t_1) \;\wedge\; \text{has\_been\_meaning}\,(e_3,\; \text{mary},\; \text{go}\,(e_2,\; \text{mary}))\; \wedge\; \text{for}\,(e_3) = t_1)$

Triggered by the IP-INF node of the source annotation, the presence of `toComp` in `ex1` has the consequence of establishing a control relationship in which the subject of the matrix clause, *mary*, is also the subject of the infinitive embedding. Also note how `"for"` in `ex1` is given as an expected argument of `embVerb` (together with `"arg0"`) to make the *week* that is t_1 a *for* modifier of the *has_been_meaning* event e_3. This assumes a Davidsonian theory (Davidson 1967) in which verbs are encoded with minimally an implicit event argument which is existentially quantified over and may be further modified. Such a meaning representation encodes truth-conditional content and could be used (with post processing) to feed theorem provers and model builders (see e.g., Blackburn and Bos 2003).

4 Examples

As a testing ground for the system and a basis for experimenting with annotation, a partially parallel corpus has been prepared with parsed constituent trees and meaning representations automatically generated by the system and then human checked for 4,120 sentences of English and 4,155 sentences of Japanese (`http://www.compling.jp/ts`). For reasons of space we limit attention to results from two annotated sentences, as demonstrations of the level of detail the generated meaning representations achieve.

4.1 Binding and covaluation with quantification

Heim (1993) observes that (4) has the possibility of being construed either with a bound reading where every wife has the thought ‘No one else respects their own husband!’, or with a covaluation reading where every wife has the thought ‘No one else respects my husband!’.

(4) Every wife thinks that only she respects her husband.

Example (4) is annotated:

```
(IP-MAT (NP-SBJ (Q Every) (N wife))
        (VBP thinks)
        (CP-THT (C that)
                (IP-SUB (NP-SBJ (FP only)
                                (PRO she))
                        (VBP respects)
                        (NP-OB1 (PRO$ her)
                                (N husband))))
        (. .))
```

Conversion arrives at `ex2`:

```
val ex2 =
( fn fh =>
  ( fn lc =>
    ( ( ( every lc fh ("entity", "entity")
          ( nn lc fh "wife"))
        "arg0")
      ( embVerb lc fh "event" ["arg0"]
          "thinks" that
        ( ( ( focusParticle fh ("entity",
                "entity") "ONLY" "="
              ( pro ["c"] fh ["entity"]
                  ("entity", "entity")
                  "she"))
            "arg0")
          ( ( ( some lc fh "entity"
                ( ( ( pro ["c"] fh
                       ["entity"]
                       ("entity",
                       "entity") "her")
                    "of")
                  ( nn lc fh "husband")))
              "arg1")
            ( verb lc fh "event"
                ["arg0", "arg1"]
                "respects"))))))
    ["of", "arg1", "arg0", "that", "h"])
["entity", "event"]
```

An evaluation of `ex2` produces:

(5) $\forall x_1\,(\text{wife}\,(x_1) \;\rightarrow\; \exists x_3 e_2\,(x_3 = \text{she}\{x_1\} \;\wedge\; \text{thinks}\,(e_2,\; x_1,\; \text{ONLY} x_4\,(\exists x_7 x_5 e_6\,(x_7 = \text{her}\{x_4,\; x_1\} \;\wedge\; \text{is\_husband\_of}\,(x_5,\; x_7) \;\wedge\; \text{respects}\,(e_6,\; x_4,\; x_5)),\; x_3 = x_4))))$

While x_3 should be resolved to have x_1 as antecedent, for x_7 the system leaves a choice: resolving to x_4 results in the bound reading, while resolving to x_1 brings about the covaluation reading.

4.2 Conditionals

Next consider an example of a conditional:

(6) 1万円出していたら、足りただろう。
Had I withdrawn 10,000 yen, it would have been enough.

Such a conditional can be annotated as follows, with the `(CND *)` disambiguation information to trigger the conditional interpretation.

```
(IP-MAT (PP (IP-ADV (NP-SBJ *speaker*)
                    (NP-OB1 (NUMCLP (CARD 1)
                                    (CARD 万)
                                    (NUMCL 円)))
                    (VB 出し)
                    (P て)
                    (VB2 い))
                (P たら))
        (CND *)
        (PU 、)
        (NP-SBJ *pro*)
        (VB 足り)
        (AXD た)
        (MD だろう)
        (PU 。))
```

Conversion arrives at `ex3`, with the dependent clause of the conditional integrated with `cond`:

```
val ex3 =
( fn fh =>
  ( fn lc =>
    ( ( cond fh "entity" "たら"
        ( clause lc nil
          ( ( fn lc =>
              ( ( ( pro ["personalc"]
                    fh ["entity"]
                    ( "ENTITY",
                      "personalentity")
                    "speaker")
                  "arg0")
                ( ( ( card lc fh "1_万_円"
                        "group"
                      ( nn lc fh "xxx"))
                    "arg1")
                  ( verb lc fh "event"
                      ["arg0", "arg1"]
                      "出し_て_い"))))
            ["arg1", "arg0", "h"])))
      ( ( ( pro ["c"]
            fh ["entity", "group"]
            ( "entity",
              "entity")
            "pro")
          "arg0")
        ( ( md fh "だろう")
          ( past "event"
            ( verb lc fh "event"
                ["arg0"] "足り"))))))
  ["arg0", "arg1", "h"])
["ENTITY", "group", "event", "entity"]
```

The following is the ouput from an evaluation of `ex3`, with たら essentially acting as the conditional operator ('→'):

(7)
$$\begin{array}{l}
\exists z_3\,(\\
\quad z_3 = \text{speaker} \wedge \\
\quad\ \forall X_1 e_2 \text{たら}\,(\\
\qquad 1\_\text{万}\_\text{円}\,(X_1) \wedge \\
\qquad \text{出し\_て\_い}\,(e_2,\ z_3,\ X_1), \\
\qquad \exists x_4\,(\\
\qquad\ x_4 = \text{pro}\{X_1\} \wedge \\
\qquad\quad \text{だろう}\,(\exists e_5\,(\text{足り}\,(e_5,\ x_4))) \\
\qquad)))
\end{array}$$

Notably X_1 receives universal quantification (from the closure brought about by `cond` of `ex3`) and serves as accessible antecedent for the subject zero pronoun of the matrix clause given in the source annotation as `(NP-SBJ *pro*)`. This demonstrates how SCT evaluation captures an archetypal donkey anaphora dependency (Kamp 1981).

5 Conclusion

To sum up this paper has described a system that takes constituent tree annotations as input and outputs predicate logic based meaning representations. The system has been tuned to accept annotations of a specific scheme, with clause level functional annotation that makes it readily possible to build meaning representations beyond the predicate-argument structure level. The system could be tuned to an alternative annotation scheme, but in future work we plan to experiment with converting annotations of different schemes to the assumed scheme, with a view to obtaining useful meaning representations from a wide range of syntactic annotations and parsing systems.

References

Blackburn, Patrick and Johan Bos. 2003. Computational semantics. *Theoria* 13:27–45.

Butler, Alastair. 2010. *The Semantics of Grammatical Dependencies*, vol. 23 of *Current Research in the Semantics/Pragmatics Interface*. Bingley: Emerald.

Heim, Irene. 1993. Anaphora and semantic interpretation: A reinterpretation of Reinhart's approach. Tech. Rep. SfS-Report-07-93, University of Tübingen.

Santorini, Beatrice. 2010. Annotation manual for the Penn Historical Corpora and the PCEEC (Release 2). Tech. rep., Department of Computer and Information Science, University of Pennsylvania, Philadelphia.