Shallow grammar + constrained semantics = deep grammar

Alastair Butler
Japan Society for the Promotion of Science and Center for the
Advancement of Higher Education, Tohoku University, ajb129@hotmail.com

Kei Yoshimoto
Center for the Advancement of Higher Education, Tohoku University
kyoshimoto@mail.tains.tohoku.ac.jp

Abstract

It has hitherto been a task assigned to deep grammars such as Head-driven Phrase Structure Grammar and Lexical Functional Grammar to build up sufficiently rich meaning representations for sentences, including local argument dependencies, long-distance dependencies and discourse dependencies. In this paper, we propose how to obtain appropriate semantic information from the results of a shallow grammar, i.e., word-class information and the hierarchical syntactic structure of the clause, by developing the dynamic semantics approach to sentence meanings that constrain what dependencies are possible.

1 Introduction

Taking languages to be sets of strings, the purpose of a grammar for a language is to define which strings are possible expressions of the language and which strings are not. In doing this, the grammar will likely tell us about the word class of substrings identified as lexical items together with hierarchical syntactic structures for the valid strings. Let's call the ability to do all this *shallow* grammar. In addition to this, *deep* grammars will relate language expressions to information/interpretations.

Traditionally, deep grammars are hand-crafted, providing outputs with meaning representations, predicate-argument-adjunct structures, logical forms, etc. (e.g., HPSG LingoErg, Copestake and Flickinger 2000; LFG ParGram, Butt et al 1999; Core Language Engine CLE, Alshawi 1992). Wide-coverage deep grammar development is knowledge extensive and expensive. It also remains very hard to scale hand-crafted grammars to unrestricted text. In contrast, approaches limited to developing shallow grammars have been able to employ treebank-based grammar induction/parsing technologies to produce with a low development cost, wide-coverage and robust grammars, only with a limited ability to map strings to meaning representations (see e.g., de Marneffe et al 2006).

Getting appropriate meaning representations is all about providing an adequate input for the independent mechanisms of some semantics. This poses a research question: Could we make do with the outputs of a shallow grammar as the only input to semantics?

To answer this question let's first take a step back and ask why we would need the rich meaning representations of a deep grammar in the first place. The standard answer is that deep grammar provides the constraining guidance semantics needs. Such careful guidance is required when semantics can evaluate any given structure. But if semantics were constrained on its own terms sufficiently, we might be able to get away with providing only the most basic structural guidance.

In this paper we develop this idea to show how it is possible to enforce only appropriate dependencies, including long-distance dependencies, local argument dependencies and discourse dependencies, from a parse tree that only presents word-class information and the hierarchical structure of the clause. We demonstrate this with a semantic system, a generalisation of the

Table 1: Binding roles

fresh bindings: sources for new scopes			local bindings: in-use scopes of current locality				context bindings: scopes of prior localities serving as antecedents
discourse linked	question	existential (quantified)	nominal	core arguments		non-core (preposition)	
				subject	object		
"d"	"q"	"e"	"h3S"	"x3S"	"y"	"with"	"c"
			"h3P"	"x3P"		"near"	
				"x2S"		"behind"	
				"x2P"		"over"	
				"x1S"		"around"	
				"x1P"		...	

Scope Control Theory or SCT of Butler (2007), which implements an evaluation routine that takes an assignment (environment) and a parsed form and returns a predicate logic translation.

2 Introducing bindings

The key to the approach we provide with SCT is to limit binding names to fixed roles (grammatical and discourse) which we summarise in Table 1. This works for English, with e.g., person and number agreement coded in the names for subject and nominal bindings, while different languages will require a different inventory of binding names. By coding sensitivity to binding names, we are able to instrument an expression based on word class information and information from the local syntactic context to enforce requirements on the state of the assignment with respect to which evaluation is made. This in turn will either enforce or relax constraints on possible language expressions, as well as contribute to determine the given interpretation for a valid language expression.

As an evaluation proceeds the assignment can change, with given scopes typically shifting from a binding with one kind of role to a binding with a different kind of role, following the illustration of Fig 1. This shows coordination and subordination as different dimensions. Looking at the dimension of coordination, we see that a scope is first inaccessible, then it gets used, after which it becomes available as a context binding where it can serve as the antecedent to a pronoun. Looking at the dimension of subordination, we see that a scope first appears as a fresh binding, then shifts to a local binding where it must serve as the binder of a main predicate's argument, and then shifts to a context binding when there is garbage collection to end the local binding.

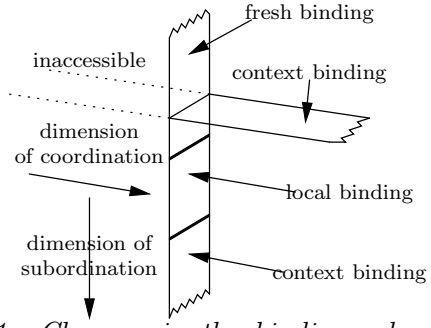

Fig 1: Changes in the binding roles of a scope with subordination and coordination

3 Predicates

We gain control over bindings and enforce their roles by making the SCT evaluation sensitive to what should and should not be present as a binding. This we can do by declaring usage conditions for the grammatical dependencies of a predicate with the form of (1).

(1) `r fh lc args atch ext s`

This takes six parameters. `fh` and `lc` provide 'hot patches' that determine which binding names the predicate is sensitive to. The remaining parameters are specific to the predicate instance: `args` gives the binding names for the required (core) arguments of the predicate; `atch`, binding names for any attached (non-core) arguments; and `ext`,

binding names that are not entered into the argument structure of the predicate, but which nevertheless support corresponding bindings for the predicate instance. Finally, `s` provides the name of the predicate.

In (2) we illustrate three possible forms for a `"smiles"` predicate: `smiles1`, which supports only a single core `"x3S"` binding; `smiles2`, which supports a core `"x3S"` binding plus a `"near"` attachment; and `smiles3`, which supports a core `"x3S"` binding plus an additional vacuous `"x3S"` binding. This also illustrates determiners `a` and `another` as introducing noun phrases that open bindings with third person agreement and nominal restrictions. They differ, with `another` signalling that it must be under an open `"x3S"` binding. The binding name that a determiner opens is determined either by the immediate presence of the infix '`//`' operator, in which case an `"x3S"` binding is opened, indicating the contribution of word order in determining a 'subject' binding; or the immediate presence of a preposition, with `near` contributing the `"near"` binding name. Together with the hot patches that give `"e"` as the possible source for fresh bindings, and `"x3S"` and `"near"` as the possible local bindings, we get the evaluation results of (3).

(2)

```
let
  fh = ["e"]
  lc = ["x3S", "near"]
  a f = some3S fh 1 "e" 0 [f] nil nil
  another f =
    some3S fh 1 "e" 0 [f] ["x3S"] nil
  boy = r fh lc ["h3S"] nil nil "boy"
  near f = f "near"
  smiles1 =
    r fh lc ["x3S"] nil nil "smiles"
  smiles2 =
    r fh lc ["x3S"] ["near"] nil "smiles"
  smiles3 =
    r fh lc ["x3S"] nil ["x3S"] "smiles"
in
  a.  a boy//smiles1
  b.  a boy//(smiles1\near (another boy))
  c.  a boy//(smiles2\near (another boy))
  d.  a boy//(smiles3\near (another boy))
  e.  a boy//(another boy//smiles1)
  f.  a boy//(another boy//smiles2)
  g.  a boy//(another boy//smiles3)
end
```

(3)

a. $\exists g : (g,(2a))^\circ = ((\texttt{boy}(x) \wedge (\texttt{sing}(x) \wedge \texttt{3rd}(x))) \wedge \texttt{smiles}(x))$

b. $\forall g : (g,(2b))^\circ = *$

c. $\exists g : (g,(2c))^\circ = ((\texttt{boy}(y) \wedge (\texttt{sing}(y) \wedge \texttt{3rd}(y))) \wedge (\texttt{smiles +near}(y,x) \wedge (\texttt{boy}(x) \wedge (\texttt{sing}(x) \wedge \texttt{3rd}(x)))))$

d. $\forall g : (g,(2d))^\circ = *$

e. $\forall g : (g,(2e))^\circ = *$

f. $\forall g : (g,(2f))^\circ = *$

g. $\exists g : (g,(2g))^\circ = ((\texttt{boy}(y) \wedge (\texttt{sing}(y) \wedge \texttt{3rd}(y))) \wedge ((\texttt{boy}(x) \wedge (\texttt{sing}(x) \wedge \texttt{3rd}(x))) \wedge \texttt{smiles}(x)))$

With (3a) we see that an evaluation with `smiles1` is possible when an `"x3S"` binding need be the only open binding; (3b) shows that `smiles1` is impossible when there must be an additional `"near"` binding, which provides an environment able to support `smiles2`, (3c), but not `smiles3`, (3d); and (3e) shows that `smiles1` is impossible with an extra `"x3S"` binding, which is able to support `smiles3`, (3g), but not `smiles2`, (3f). From these results we can conclude that, in saying the grammatical dependencies that a predicate supports, we are saying what the bindings are that it must *and* must not get. This is a powerful result, since with this result we don't need syntax to guarantee the dependency of what opens a fixed binding with whatever needs to be bound, as the only licensed dependencies will be the dependency links that we will want to see made.

As a second example, consider (4) and (5), which illustrate how anaphoric dependencies get enforced with evaluations, and, most crucially for our current concerns, with the absence of any coindexing. With (4a/5a) we see an example with a reflexive that forms a link with a local (agreeing) binding. With (4b/5b) we see a pronoun linking with an antecedent across a conjunct. In (4c/5c) we see that when a reflexive lacks a local binding that it can agree with (the occurrence of `you` happens to open an `"x2S"` binding and not an `"x3S"` binding) evaluation fails, while (4d/5d) show how a pronoun is able to pick up a superordinate antecedent when the antecedent is of a different locality.

(4)

```
let
  fh = ["e"]
  lc = ["h3S", "x3S", "x2S", "y"]
  a f = some3S fh 1 "e" 0 [f] nil nil
  boy = r fh lc ["h3S"] nil nil "boy"
  teacher =
    r fh lc ["h3S"] nil nil "teacher"
  likes =
    r fh lc ["x3S", "y"] nil nil "likes"
  himself = him (T ("x3S", 0)) fh
  smiles =
    r fh lc ["x3S"] nil nil "smiles"
  him = him (T ("c", 0)) fh
  thinks =
    remb fh lc ["x3S"] nil nil "thinks"
  you = you2S
  like =
    r fh lc ["x2S", "y"] nil nil "like"
in
  a.  a boy//(likes\\himself)
  b.  (a boy//smiles) and
      (a teacher//(likes\\him))
  c.  a boy//(
        thinks (you//(like\\himself)))
  d.  a boy//(
        thinks (a teacher//(likes\\him)))
end
```

(5)

a. $\exists g : (g,(4a))^\circ = ((\texttt{boy}(y) \wedge (\texttt{sing}(y) \wedge \texttt{3rd}(y))) \wedge (\texttt{likes}(y,x) \wedge x = y \wedge \texttt{masc}(x) \wedge (\texttt{3rd}(x) \wedge \texttt{sing}(x))))$

b. $\exists g : (g,(4b))^\circ = (((\texttt{boy}(x) \wedge (\texttt{sing}(x) \wedge \texttt{3rd}(x))) \wedge \texttt{smiles}(x)) \wedge ((\texttt{teacher}(z) \wedge (\texttt{sing}(z) \wedge \texttt{3rd}(z))) \wedge (\texttt{likes}(z,y) \wedge y = x \wedge \texttt{masc}(y) \wedge (\texttt{3rd}(y) \wedge \texttt{sing}(y)))))$

c. $\forall g : (g,(4c))^\circ = *$

d. $\exists g : (g,(4d))^\circ = ((\texttt{boy}(z) \wedge (\texttt{sing}(z) \wedge \texttt{3rd}(z))) \wedge \texttt{thinks}(z,((\texttt{teacher}(y) \wedge (\texttt{sing}(y) \wedge \texttt{3rd}(y))) \wedge (\texttt{likes}(y,x) \wedge x = z \wedge \texttt{masc}(x) \wedge (\texttt{3rd}(x) \wedge \texttt{sing}(x))))))$

As a final example, consider (6) and (7), which illustrate how a long distance dependency can be established with evaluation, and again without any coindexing. Note that `?` and `can` act as operators that introduce closures: `?` gives rise to the existential quantifier in the translation that occurs within the immediate scope of `QUEST`, and so which consequently has its value under question; while `can` gives rise to the existential quantifier that occurs under `EXISTS`, which consequently receives an existential reading.

(6)

```
let
  fh = ["q", "e"]
  lc = ["h3S", "x3S"]
  who =
    some3S fh 1 "q" 0 nil nil nil
  someone =
    some3S fh 1 "e" 0 nil nil ["x3S"]
  think =
    remb fh lc ["x3S"] nil ["x3S"] "think"
  smiles =
    r fh lc ["x3S"] nil nil "smiles"
in
  (who//can (someone//think smiles)) ?
end
```

(7) $\exists g : (g,(6))^\circ = \texttt{QUEST}(\exists x((\texttt{sing}(x) \wedge \texttt{3rd}(x)) \wedge \texttt{EXISTS}(\exists y((\texttt{sing}(y) \wedge \texttt{3rd}(y)) \wedge \texttt{think}(y,\texttt{smiles}(x))))))$

4 Summary

To sum up, our results suggest that, from the inputs of a shallow grammar, the demands of an appropriately constrained semantics are sufficient to capture the range of grammatical dependencies that deep grammar was thought to be required to enforce.

References

Alshawi, Hiyan, ed. 1992. *The Core Language Engine*. MIT Press.

Butler, Alastair. 2007. Scope control and grammatical dependencies. *Journal of Logic, Language and Information* 16:241–264.

Butt, Miriam, Tracy Holloway King, Maria-Eugenia Nino, and Frederique Segond. 1999. *A Grammar Writer's Cookbook*. Stanford: CSLI Publications.

Copestake, Ann and Dan Flickinger. 2000. An open-source grammar development environment and broad-coverage english grammar using HPSG. In *Proceedings of the Second conference on Language Resources and Evaluation (LREC-2000)*. Athens, Greece.

de Marneffe, Marie-Catherine, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *LREC*.