

Making semantics thrive on cheap syntax

In this paper we develop the idea that semantic evaluation comes sufficiently self constrained so as to enforce only appropriate dependencies, including long-distance dependencies, local argument dependencies and discourse dependencies, from a parse tree that has only to present word-class information and constituent structure. We demonstrate this with an evaluation routine that takes an assignment (of sequences of values to variables; e.g., Vermeulen 2000) and a parsed form and returns a predicate logic based translation. Translation provides easy to appraise ‘snapshots’ of the dependencies evaluation establishes, but could be replaced by full interpretation against a model.

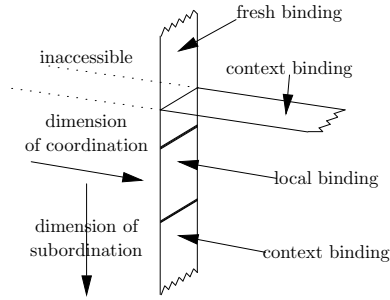
The key to the approach is to limit binding names to fixed roles (grammatical and discourse) which we summarise in Table 1. This works for English, with e.g., person and number agreement coded in the names for subject and nominal bindings, while different languages will require a different inventory 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.

Table 1: Binding roles

fresh bindings: sources for new scopes				local bindings: in-use scopes of current locality			context bindings: scopes of prior localities		
discourse linked	question	existential (quantified)	control	nominal	core arguments		non-core (preposition)	available as antecedents	removed
					subject	object			
"d"	"q"	"e"	"o"	"h3S" "h3P"	"x3S" "x3P" "x2" "x1S" "x1P"	"y"	"with" "near" "behind" "over" "around" ...	"c"	"r"

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



We gain control over bindings and enforce their roles by making the semantic 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 the instance of `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)

```
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)
where
fh = ["e"]; lc = ["x3S", "near"]; a = (λf.(some3S fh 1 "e" 0 [f] nil nil))
another = (λf.(some3S fh 1 "e" 0 [f] ["x3S"] nil))
boy = r fh lc ["h3S"] nil nil "boy"; near = (λf.(λx.(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"
```

(3)

- a. $\exists g: (g, (2a))^\circ = ((\text{boy}(x) \wedge 3S(x)) \wedge \text{smiles}(x))$
- b. $\forall g: (g, (2b))^\circ = *$
- c. $\exists g: (g, (2c))^\circ = ((\text{boy}(y) \wedge 3S(y)) \wedge (\text{smiles} + \text{near}(y, x) \wedge (\text{boy}(x) \wedge 3S(x))))$
- d. $\forall g: (g, (2d))^\circ = *$
- e. $\forall g: (g, (2e))^\circ = *$
- f. $\forall g: (g, (2f))^\circ = *$
- g. $\exists g: (g, (2g))^\circ = ((\text{boy}(y) \wedge 3S(y)) \wedge ((\text{boy}(x) \wedge 3S(x)) \wedge \text{smiles}(x)))$

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 checked 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. This also illustrates `remb`, which provides an embedding relation. 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 evaluation fails (the occurrence of `you` happens to open an "x2" binding and not the required "x3S" binding; signalling second person agreement, but not singular or plural agreement, which are neutralised in all paradigms involving second person in English), while (4d/5d) show how a pronoun is able to pick up a superordinate antecedent when the antecedent is of a different locality.

(4)

- 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)))`
- where
- ```

fh = ["e"]
lc = ["h3S", "x3S", "x2", "y"]
a = (λf.(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"
and = (λf'.(λf.(check "c" fh "∧" [f, f'])))
him = him (T ("c", 0)) fh
thinks = remb fh lc ["x3S"] nil nil "thinks" "" cons
like = r fh lc ["x2", "y"] nil nil "like"

```

(5)

- a.  $\exists g: (g, (4a))^\circ = ((\text{boy}(y) \wedge \exists S(y)) \wedge (\text{likes}(y, x) \wedge x = y \wedge \exists S:M(x)))$
- b.  $\exists g: (g, (4b))^\circ = (((\text{boy}(x) \wedge \exists S(x)) \wedge \text{smiles}(x)) \wedge ((\text{teacher}(z) \wedge \exists S(z)) \wedge (\text{likes}(z, y) \wedge y = x \wedge \exists S:M(y))))$
- c.  $\forall g: (g, (4c))^\circ = *$
- d.  $\exists g: (g, (4d))^\circ = ((\text{boy}(z) \wedge \exists S(z)) \wedge \text{thinks}(z, ((\text{teacher}(y) \wedge \exists S(y)) \wedge (\text{likes}(y, x) \wedge x = z \wedge \exists S:M(x))))$

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 did act as operators that introduce closures: ? brings about the introduction of the existential quantifier in the translation that occurs within the immediate scope of QUEST, and so which has its value under question (via existential disclosure; Dekker 1993); while did leads to the existential quantifier that occurs under EXISTS, which consequently receives an ordinary existential reading (with existential disclosure).

(6)

- a. `(who//did (someone//think smiles))\?`  
 where  
`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" "" cons`  
`smiles = r fh lc ["x3S"] nil nil "smiles"`

(7)

- $\exists g: (g, (6))^\circ = \text{QUEST}(\exists x(\exists S(x) \wedge \text{EXISTS}(\exists y(\exists S(y) \wedge \text{think}(y, \text{smiles}(x)))))$

To sum up, our results suggest that, from inputs limited to word class information and constituent structure, the demands of an appropriately constrained semantics are sufficient to enforce only grammatical dependencies. This obviates the need for more costly inputs to semantics, e.g., logical forms. In addition to offering a new perspective on how grammatical dependencies work, the approach also provides the ability to locate the cause of any aberrant behaviour with diagnostic feedback from an evaluation.

## References

- Dekker, Paul. 1993. Existential Disclosure. *Linguistics and Philosophy* 16:561–587.
- Vermeulen, C. F. M. 2000. Variables as stacks: A case study in dynamic model theory. *Journal of Logic, Language and Information* 9:143–167.