A MINIMALIST IMPLEMENTATION OF VERB SUBCATEGORIZATION

Sourabh Niyogi
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
niyogi@mit.edu

Abstract

Traditional accounts of verb subcategorization, from the classic work of Fillmore on, require either a considerable number of syntactic rules to account for diverse sentence constructions, including cross-language variation, or else complex linking rules mapping the thematic roles of semantic event templates with possible syntactic forms. In this paper we exhibit a third approach: we implement, via an explicit parser and lexicon, the incorporation theory of Hale and Keyser (1993, 1998) to systematically cover most patterns in English Verb Classes and Alternations (Levin 1993), typically using only 1 or 2 lexical entries per verb to subsume a large number of syntactic constructions and also most information typically contained in semantic event templates, and, further, replacing the notion of "thematic roles" with precise structural configurations. The implemented parser uses the merge and move operations formalized by Stabler (1997) in the minimalist framework of Chomsky (2001). As a side benefit, we extend the minimalist recognizer of Harkema (2000) to a full parsing implementation. We summarize the current compactness and coverage of our account and provide this minimalist lexicon and parser online at http://web.mit.edu/niyogi/www/minimal.htm

1 The Problem of Verb Subcategorization

Why do certain verbs undergo particular certain alternations and not others? On some accounts, e.g. Levin (1993), referred to hereafter as EVCA, alternations provide insight into verb subcategorization and hence hooks to parsing, cross-language variation, machine translation, and class based verb learning. However, fully implemented accounts of the phenomena remains an open problem, with at least three alternative models, shown in Figure 1.

Accounts may be solely descriptive – for example, classifying verbs as having an intransitive, a transitive, and/or ditransitive form, as is familiar. Traditional computational accounts (see 1) map these forms into individual grammar rules, (perhaps by macro expansion-like techniques) adding as many rules as necessary to account for naturally' occurring constructions (wh-movement, passive forms, etc.) For each grammatical rule, a separate semantic decomposition is required, typically labeling component phrases with one of several "thematic roles." A richer account provided by lexical semantics (see 2), exemplified in Jackendoff (1983, 1990) and Rappaport Hovav and Levin (1998), is one that hypothesizes semantic templates, but requires linking rules mapping syntactic frames with semantic templates governed by a particular verb. Often these semantic templates are constructed in an ad hoc manner, and the corresponding linking rules are consquently a collection of difficult-to-implement heuristics. In this paper we implement a rather different formalism (Hale and Keyser's Incorporation theory, see 3), wherein fewer lexical entries govern syntactic and semantic behavior, with no appeal to thematic roles or complex linking rules.

0. Verb Subcategorization Phenomena

`* Bob put.`	`Butter was put on the bread.`
`* Bob put butter.`	`What was put on the bread?`
`Bob put butter on the bread.`	`Where was the butter put?`

1. Traditional Account

VP $\rightarrow$ V0 NP PP_{loc}	V0 $\rightarrow$ put
VP $\rightarrow$ was VPass	VPass $\rightarrow$ V0 PP_{loc}
VP/NP $\rightarrow$ V0 NP/NP PP_{loc}	VP/NP $\rightarrow$ V0 NP PP_{loc}/NP
PP_{loc} $\rightarrow$ P_{loc} NP	P_{loc} $\rightarrow$ on | in | ...
PP_{loc}/NP $\rightarrow$ P_{loc} NP/NP	

Exhaustive modelling with a *considerable* number of grammatical rules.
Semantics separate, otherwise unspecified.

2. Lexical Semantics Account

$$\begin{bmatrix} \text{put} \\ \text{V} \\ \text{NP}_j\text{PP}_k \\ \text{CAUSE}\left([BOB]_i, \text{GO}\left([BUTTER]_j, \text{TO}\left([BREAD]_k\right)\right)\right) \end{bmatrix}$$

Syntax handled by numerous argument-fusing “linking rules”, typically difficult to formalize.
Semantic templates mirror alternation patterns, but are ad-hocly constructed.

3. Minimalist/Incorporation Account

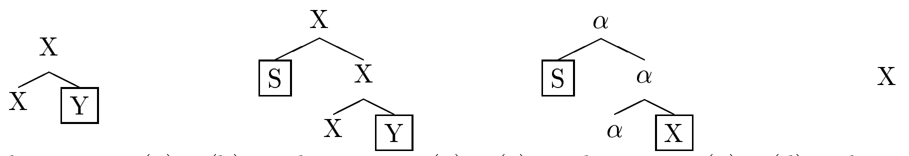
/put/	$=\text{p}_{\text{loc}}$ =d v_{cause}
	$(\lambda(=\text{p}_{\text{loc}})\ (\lambda(=\text{d})\ (=\text{p}_{\text{loc}}\ =\text{d})))$
/on/	=d +k p_{loc}
	$(\lambda(=\text{d})\ (\lambda(\text{x})\ ((\text{go x})\ (\text{path self} =\text{d}))))$
//	$>\text{v}_{\text{cause}}$ +k =d pred
	$(\lambda(>\text{v}_{\text{cause}})\ (\lambda(=\text{d})\ ((\text{cause} >\text{v}_{\text{cause}})\ =\text{d})))$
/-ed/	>pred ++k t
	$(\lambda(>\text{pred})\ (\text{tense} >\text{pred past}))$

Small number of lexical entries handle all syntactic phenomena.
Semantics *directly encoded* in lexical entry.
Entries structurally governed by small number of rules, specifying how N/A/P are related.

Figure 1: Three Different Accounts of Verb Subcategorization

2 Incorporation Theory

At the heart of our new contribution to modeling verb subcategorization is the marriage of Hale and Keyser's (1993, 1998) argument structure theory with Stabler's (1997) ‘minimalist’ structure building rules. In the Hale and Keyser's theory, using the terminology of X-bar syntax, a particular head (labeled X), may or may or may not take a complement (labeled Y) and may or may not project a specifier (labeled S), resulting in 4 possible structural configurations:



(a) -subj, +comp (`V`) (b) +subj, +comp (`P`) (c) +subj, -comp (`A`) (d) -subj, -comp (`N`)

Figure 2: Four fundamental primitives in Hale and Keyser's incorporation theory

The combinatorial possibilities of incorporation with X=`V`, `A`, `N`, `P` heads, plus ‘head movement’, is designed to yield the space of possible syntactic argument structure configurations, presumably across all languages. Notions of agent, patient, instrument, theme, goal, etc. are not ‘primitives’, but are *derived* from positions in structural configurations. In English (but not necessarily in all languages), (a) the category `V` takes a complement but projects no specifier; (b) the category `P` takes both a complement and projects a specifier; (c) the category `A` takes no complement but projects a specifier; (d) the category `N` takes neither complement nor specifier. A particular verbal entry, being of category `V`, may incorporate one or more of these structures as its complement, as shown in Figure 3:

- Nouns incorporated directly into a verbal entry yield structures such as (a): no subject is projected by the `N`. The phonetic material of the noun head *incorporates* (undergoes *head movement*) into the phonetic material of the verb head, which itself may undergo further movement. Verbs such as these are intransitive by nature, generating, e.g., `/The light glow -ed/` but `*/Bob glow -ed the light/`. This argument structure typifies purely internally caused processes.

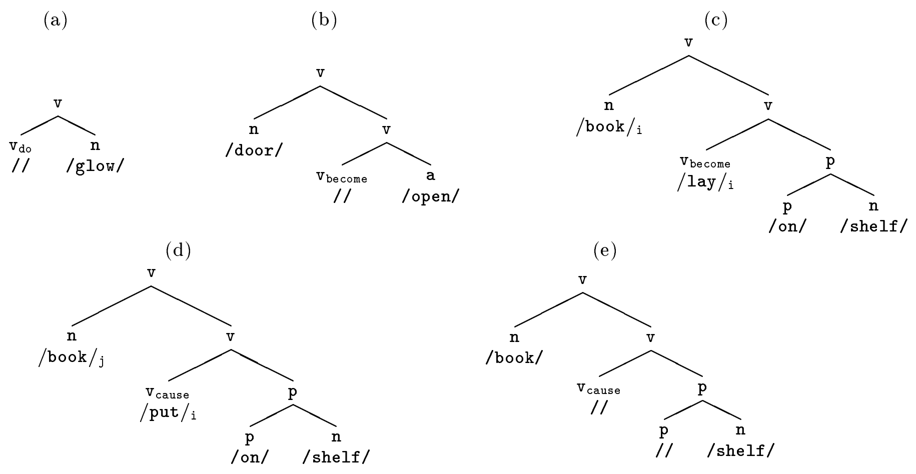


Figure 3: Examples of Structure Building in Hale and Keyser's Incorporation Theory

- Adjectives incorporated into a verbal entry yields structures such as (b): a subject is *projected* by the `A` (i.e. `/the door/`). The phonetic material of the adjective head *incorporates* into the verb head, which again, may undergo further movement. Verbs such as these are transitive by nature, resulting in `/The door open -ed/` and `/Bob open -ed the door/`. This argument structure typifies externally causable state changes.
- Similarly, incorporated prepositions yield fundamentally transitive verbs such as (c), thus both `/The book lay -ed on the shelf/` and `/Bob lay -ed the book on the shelf/` is grammatical.
- To account for why `/The book lay -ed on the shelf/` is grammatical but `*/Bob put -ed on the shelf/` is not, it is hypothesized that either the manner of the external argument (as in `/put/`) or the internal argument (as in `/lay/`) is indexed in the verbal entry, as shown in (d).
- Multiple incorporations are possible, such as in (e), where a preposition is incorporated into a verbal entry, and the preposition itself has a noun incorporated into it (e.g. `/shelf/`) – the preposition *projects* a subject (e.g. `/book/`) through the verbal structure it is incorporated into. This kind of argument structure is common for figure-incorporation, ground-incorporation, and instrument-incorporation.

3 Minimalist Operations

We can now show how one can implement Hale and Keyser's incorporation theory in the framework of the Minimalist Program (Chomsky 2000). In this framework, there are at least 2 fundamental structure-building operations, *Merge* and *Move*. Stabler (1997, 2000) has formalized these into 4 specific structure-building operations for *Merge* and 2 for *Move*. In this model, a lexical entry (a *simple* structure) has the following form:

`/phonetic-content/ feature-list` λ`-expression`

where the `phonetic-content` (possibly null, denoted `//`) is what is actually pronounced, and the `feature-list` is an ordered list of features chosen from a set of licensors (e.g. `>a`, `<a`, `=a`, marking theta role assignment), licensees (e.g. `a`, intuitively, marking an argument needing a theta-role), movement triggers (e.g. `++k`, `+k`, intuitively, case assigners), and movement requirements (e.g. `-k`, intuitively, marking that an argument needs to be assigned case).

Structures can be *simple*, as in the above case, or *complex*, where the operation of *Merge* on two structures A and B (simple or complex):

A the head of a Merge operation, whose feature-list is headed a licensor and whose λ-expression is of the form (λ(=a) exp), whose body exp returns an semantic structure using semantic primitives and the argument =a

B the argument of Merge, whose feature-list is headed by a matching licensee and whose λ-expression is of any form val.

creates a new complex structure (A, B, <, λ-expression) or (B, A, >, λ-expression) – where the > and < symbols denote which piece of the complex structure was the head prior to *Merge*. In this new complex structure, the resulting new internal A and B structures have the licensor-licensee feature pairs deleted, phonetic material may be rearranged, and the λ-expression of the licensor is applied to that of the licensee.

Move, operating on just one structure A, also cancels features (the movement triggers/requirements), but is semantically vacuous: the semantic result of the new complex has the same value as the old complex. To generate a derivation, structures undergo repeated Merge and Move operations, canceling pairs of features from the feature lists until no features remain except a single goal feature c, which specifies that a complete derivation has been constructed. We omit here the clear comparison to categorial grammar and its relatives; see Stabler (1997) and Berwick and Epstein (1995) for additional details. The *Merge* and *Move* rules, summarized from Stabler (1997), are:

OPERATION	EXAMPLE
Simple Merge /h/ =a δ (λ(=a) exp) /c/ a γ val $\rightarrow$ (/h/ δ, /c/ γ, <, ((λ(=a) exp) val)	/the/ =n d -k (λ(=n) =n) /book/ n self $\rightarrow$ (/the/ d -k, /book/, <, ...)
Complex Merge (/h/ =a δ, ..., ..., (λ(=a) exp)) /s/ a γ ... val $\rightarrow$ (/s/ γ ... (/h/ δ, ...), >, ((λ(=a) exp) val))	(/put/ =d v_{cause}, (/on/, (/the/, /shelf/, <), <),<, ...) /what/ d -k -wh (unknown self) $\rightarrow$ (/what/ -k -wh, (/put/ v_{cause}, (/on/, (/the/, /shelf/, <), <), <), >, ...)
Left Incorporate /h/ <a δ ... (λ(<a) exp) /c/ a γ val $\rightarrow$ (/h c/ δ, // γ, <, ((λ(<a) exp) val))	/de-/ <$figure_{removable}$ =d v_{cause} /bone/ $figure_{removable}$ self $\rightarrow$ (/de- bone/ =d v_{cause}, //, <, ...)
Right Incorporate /h/ >a δ ... (λ(>a) exp) /c/ a γ val $\rightarrow$ (/c h/ δ, // γ, <, ((λ(>a) exp) val))	/-s/ >n d -k (λ(>n) (plural >n)) /book/ n self $\rightarrow$ (/book -s/ d -k, //, <, (plural (book)))
Covert Move (... (/h/ +k δ, ... (/c/ -k γ, ...), ...) $\rightarrow$ (... (/h/ δ, ... (/c/ γ, ...), ...)	(/open/ +k =d pred, ((/the/ -k, /door/, <), (//, //, <), >, ...) $\rightarrow$ (/open/ =d pred, ((/the/, /door/, <), (//, //, <), >, ...)
Overt Move (... (/h/ ++k δ, ... (/c/ -k γ, ...), ...) $\rightarrow$ (/c/ γ, (... (/h/ δ, ... (*, ...), ...), >)	(/open -ed/ ++k t, (// pred, ((/the/ -k, /door/, <), (//, //, <), >), <) $\rightarrow$ ((/the/, /door/, <), (/open -ed/ t, (// pred, (*, (//, //, <), >), <), >, ...)

Figure 4: Minimalist Structure-building Rules: *Merge* and *Move*

We illustrate the use of the above structure-building rules with the following lexicon, deriving /Bob put -ed the book on the shelf/:

1 Simple Merge: /the/ =n d -k (λ(=n) =n) and /shelf/ n self $\rightarrow$ (/the/ d -k, /shelf/, <, (shelf))

2 Simple Merge: /on/ =d +k p_{loc} (λ(=d) (λ(x) ((go x) (path self =d)))) and (1) $\rightarrow$
(/on/ +k p_{loc}, (/the/ -k, /shelf/, <), <, (λ(x) ((go x) (path (on) (shelf)))))

3 Covert Move: (2) $\rightarrow$ (/on/ p_{loc}, (/the/, /shelf/, <), <, (...))

4 Simple Merge: /put/ =p_{loc} =d v_{cause} (λ(=p_{loc}) (λ(=d) (=p_{loc} =d))) and (3) $\rightarrow$
(/put/ =d v_{cause},(/on/, (/the/, /shelf/, <), <), <, (λ(=d) ((λ(x) ((go x) (path (on) (shelf)))) =d)))

5 Simple Merge: /the/ =n d -k (λ(=n) =n) and /book/ n self $\rightarrow$ (/the/ d -k, /book/, <, (book))

6 Complex Merge: (4) and (5) $\rightarrow$
((/the/ -k, /book/, <), (/put/ v_{cause}, (/on/, (/the/, /shelf/, <), <), <), >, ((go (book)) (path (on) (shelf))))

7 Right Incorporate: // >v_{cause} +k =d pred (λ(>v_{cause}) (λ(=d) ((cause >v_{cause}) =d))) and (6) →
(/put/ +k =d pred, ((/the/ -k, /book/, <), (//, (/on/, (/the/, /shelf/, <), <), <), >), <,
(λ(=d) ((cause ((go (book)) (path (on) (shelf)))) =d)))

8 Covert Move: (7) → (/put/ =d pred,((/the/, /book/, <), (//, (/on/, (/the/, /shelf/, <), <), <), >), <, (...))

9 Complex Merge: /Bob/ d -k self and (7) →
(/Bob/ -k, (/put/ pred, ((/the/, /book/, <), (//, (/on/, (/the/, /shelf/,<),<),<),>),<),>,
((cause ((go (book)) (path (on) (shelf)))) (Bob)))

10 Right Incorporate: /-ed/ >pred ++k t (λ(>pred) (tense >pred 'past)) and (9) →
(/put -ed/ ++k t,(/Bob/ -k, (//, ((/the/, /book/,<),(//, (/on/, (/the/, /shelf/, <), <), <), >), <), >), <,
(tense ((cause ((go (book)) (path (on) (shelf)))) (Bob)) 'past))

11 Overt Move: (10) →
(/Bob/, (/put -ed/ t, (*, (//, ((/the/, /book/, <),(//, (/on/, (/the/, /shelf/, <),<),<),>),<),>),<),>, (...))

12 Simple Merge: // =t c (λ(=t) =t) and (11) →
(// c, (/Bob/, (/put -ed/, (*, (//, ((/the/, /book/, <), (//, (/on/, (/the/, /shelf/, <),<),<),>),<),>),<),>),<,
(tense ((cause ((go (book)) (path (on) (shelf)))) (Bob)) 'past))

Using semantic-structure building primitives such as:

```
unknown   (λ(x) '(? ,x))
query     (λ(event) '(query :event ,event)))
cause     (λ(event) (λ(agent) '(cause :agent ,agent :effect ,event)))
go        (λ(theme) (λ(path) '(go :theme ,theme :path ,path)))
path      (λ(dir ground) '(path :oper ,dir :terminal+ ,ground))
tense     (λ(event val) (append event (list ':tense val)))
become    (λ(state) (λ(thing) '(become :theme ,thing :goal ,state)))
```

we can reformat the result in any style desired, for example, as in Jackendoff (1983):

```
(cause :agent (bob) :effect (go :theme (book) :path (path :oper (onto) :terminal+ (shelf))) :tense past)
```

Using a small number of additional entries:

/did/ =pred +k t	(λ(=pred) (query (tense =pred 'past)))
/where/ p_{loc} -wh	(λ(x) ((go x) (path () (unknown self))))
// =t ++wh c	(λ(=t) =t)
/what/ d -k -wh	(unknown self)
/who/ d -k -wh	(unknown self)

we can derive `/what did Bob put on the shelf/`:

4 See above → (/put/ =d v_{cause},(/on/, (/the/, /shelf/, <), <), <, (λ(x) ((go x) (path (on) (shelf)))))

5 Complex Merge: /what/ d -k -wh (unknown self) and (4) →
(/what/ -k -wh, (/put/ v_{cause}, (/on/, (/the/, /shelf/, <), <), <), >, ((go (unknown self)) (path (on) (shelf))))

6 Right Incorporate: // >v_{cause} +k =d pred (λ(>v_{cause}) (λ(=d) ((cause >v_{cause}) =d))) and (5) →
(/put/ +k =d pred, (/what/ -k -wh, (//, (/on/, (/the/, /shelf/, <), <), <), >), <,
(λ(=d) ((cause ((go (unknown self)) (path (on) (shelf)))) =d)))

7 Covert Move: (6) →
(/put/ =d pred, (/what/ -wh, (//, (/on/, (/the/, /shelf/,<),<),<),>),<,
(λ(=d) ((cause ((go (unknown self)) (path (on) (shelf)))) =d)))

8 Complex Merge: /Bob/ d -k self and (7) →
(/Bob/ -k, (/put/ pred, (/what/ -wh, (//, (/on/, (/the/, /shelf/, <), <), <), >), <), >,
((cause ((go (unknown self)) (path (on) (shelf)))) (Bob)))

9 Simple Merge: /did/ =pred +k t (λ(=pred) (query (tense =pred 'past))) and (8) →
(/did/ +k t, (/Bob/ -k, (/put/, (/what/ -wh, (//, (/on/, (/the/, /shelf/, <), <), <), >), <), >), <,
(query (tense ((cause ((go (unknown self)) (path (on) (shelf)))) (Bob)) 'past))))

10 Covert Move: (9) →
(/did/ t, (/Bob/, (/put/, (/what/ -wh, (//, (/on/, (/the/, /shelf/,<), <), <), >), <), >), <,(...))))

11 Simple Merge: // =t ++wh c (λ(=t) =t) and (10) →
(// ++wh c, (/did/, (/Bob/, (/put/, (/what/ -wh, (//, (/on/, (/the/, /shelf/, <), <),<),>),<),>),<), <, (...))))

12 Overt Move: (11) →
(/what/, (// c, (/did/, (/Bob/, (/put/, (*, (//, (/on/, (/the/, /shelf/, <), <), <), >), <), >), <), <), >,
(query (tense ((cause ((go (unknown self)) (path (on) (shelf)))) (Bob)) 'past))))

```
⇒ (query :event (cause :agent (bob) :effect (go :theme (? (what))
            :path (path :oper (on) :terminal+ (shelf))) :tense past))
```

It is straightforward to show that we can derive simple 'wh-movement' variations on the above in a comparable number of steps:

```
/What did Bob put the book on/
 ⇒ (query :event (cause :agent (bob) :effect (go :theme (book)
            :path (path :oper (on) :terminal+ (? (what)))) :tense past))
/Where did Bob put the book/
 ⇒ (query :event (cause :agent (bob) :effect (go :theme (book)
            :path (path :oper () :terminal+ (? (where)))) :tense past))
```

Likewise, we derive passive forms with 3 new entries:

/was/ <pred$_p$ ++k t	(λ(<pred$_p$) (tense <pred$_p$ 'past))
/-ed/ >v$_{cause}$ =p$_{by}$? pred$_p$	(λ(>v$_{cause}$) (λ(=p$_{by}$) (=p$_{by}$ >v$_{cause}$)))
/by/ =d +k p$_{by}$	(λ(=d) (λ(event) ((cause event) =d)))

Note how p$_{by}$ is encoded as an optional licensor feature, marked with a ? in the entry for /-ed/. This is *Optional Merge*, where the licensor feature can be cancelled without a corresponding licensee feature. However, the semantic value of the missing licensee is taken from a database of λ-expression applications, one per licensee possibility, generated through an application of what would ordinarily be expected in such a position. For example, for the licensor = p$_{by}$, the semantic value for the missing licensee is ((λ(=d) (λ(event) ((cause event) =d))) 'somebody), i.e. the same merge as /by/ /somebody/. Illustrating the course of the derivation of /the book was put -ed on the shelf/:

6 See above →
((/the/ -k,/book/,<),(/put/ v$_{cause}$, (/on/, (/the/, /shelf/, <), <), <), >, ((go (book)) (path (on) (shelf)))

7 Simple Merge: /-ed/ >v$_{cause}$ =p$_{by}$? pred$_p$ (λ(>v$_{cause}$) (λ(=p$_{by}$) (=p$_{by}$ >v$_{cause}$))) and (6) →
(/put -ed/ =p$_{by}$? pred$_p$, ((/the/ -k, /book/, <), (//, (/on/, (/the/, /shelf/, <), <), <), >), <,
(λ(=p$_{by}$) (=p$_{by}$ ((go (book)) (path (on) (shelf)))))

8 Optional Merge: (7) with ((λ(=d) (λ(event) ((cause event) =d))) 'somebody) →
(/put -ed/ =p$_{by}$? pred$_p$, ((/the/ -k, /book/, <), (//, (/on/, (/the/, /shelf/, <), <), <), >), <,
((cause ((go (book)) (path (on) (shelf))))) 'somebody))

9 Left Incorporate: /was/ <pred$_p$ ++k t (λ(<pred$_p$) (tense <pred$_p$ 'past)) and (8) →
(/was put -ed/ ++k t, (//,((/the/ -k,/book/,<), (//, (/on/, (/the/, /shelf/, <), <), <), >), <), <,
(tense ((cause ((go (book)) (path (on) (shelf))))) 'somebody) 'past))

10 Overt Movement: (9) →
((/the/, /book/, <), (/was put -ed/ t,(//, (*, (//, (/on/, (/the/, /shelf/, <), <), <), >), <), <), >, (...))

11 Simple Merge: // =t c and (10) →
(// c,((/the/, /book/, <), (/was put -ed/, (//,(*, //, (/on/, (/the/, /shelf/, <), <), <), >), <), <), >), <,
(tense ((cause ((go (book)) (path (on) (shelf))))) 'somebody) 'past))

⇒ (cause :agent (somebody) :effect (go :theme (book) :path (path :oper (on) :terminal+ (shelf)))

Using the above rules, we have thus extended the work of Harkema (2000) from a recognizer to a parser: it is straightforward to design a bottom-up chart-based parser that recovers the derivation steps and semantic structure from a given input sentence. See the Appendix for the basic algorithm.

4 Incorporation

We now show how Hale and Keyser's incorporation theory can be implemented with the above minimalist framework, recognizing that other grammatical frameworks, such as lexicalized Tree Adjoining Grammars (e.g. Vijay-Shanker and Weir 1999) or categorial grammars (e.g. Steedman 2000), are likely to be capable of implementing the same theory. Using incorporation theory, we will show how A-incorporation, P-incorporation, and N-incorporation compact grammars to a very small number of entries (1 or 2) per verb.

4.1 A-Incorporation

Adding just 6 new entries to the grammar we have built so far:

Root / Adjective Entry	Verbal Entries
	// >state =d v$_{become}$ (λ(>state) (λ(=d) ((become >state) =d)))
/open/[(45.4)] state self	// >v$_{become}$ +k =d pred (λ(>v$_{become}$) (λ(=d) ((cause >v$_{become}$) =d)))
// >state a (λ(>state) >state))	// >v$_{become}$ pred (λ(>v$_{become}$) >v$_{become}$)
	/-ed/ >v$_{become}$ =p$_{by}$? pred$_p$ (λ(>v$_{become}$) (λ(=p$_{by}$) (=p$_{by}$ >v$_{become}$)))

derives /The door open -ed/:

1 Simple Merge: /the/ =n d -k (λ(=n) =n) and /door/ n self → (/the/ d -k, /door/, <, (door))

2 Right Merge: // >state =d v$_{become}$ (λ(>state) (λ(=d) ((become >state) =d))) and /open/ state self →
(/open/ =d v$_{become}$, //, <, (λ(=d) ((become (open)) =d)))

3 Complex Merge: (1) and (2) → ((/the/ -k, /door/, <), (/open/ v$_{become}$, //, <), >, ((become (open)) (door)))

4 Right Merge: // >v_{become} pred (λ(>v_{become}) >v_{become}) and (3) →
(/open/ pred, ((/the/ -k, /door/, <), (//, //, <), >, (...))

5 Simple Merge: /-ed/ >pred ++k t and (4) →
(/open -ed/ ++k t, (//, ((/the/ -k, /door/, <), (//, //, <), >), <, (tense ((become (open)) (door))) 'past))

6 Overt Move: (5) → ((/the/, /door/,<), (/open -ed/ t, (//, (*, (//, //, <), >), <), >, (...))

7 Simple Merge: // =t c (λ(=t) =t) and (5) →
(// c, ((/the/, /door/, <), (/open -ed/, (//, (*, (//, //, <), >), <), >), <,(...))

⇒ (become :theme (door) :goal (open) :tense past)

Likewise, the derivation of /Bob open -ed the door/ proceeds from step (3) above as follows:

4 Right Merge: // >v_{become} +k =d pred (λ(>v_{become}) (λ(=d) ((cause >v_{become}) =d))) and (3) →
(/open/ +k =d pred,((/the/ -k,/door/,<), (//, //, <),>, (λ(=d) ((cause ((become (open)) (door))) =d)))

5 Covert Move: (4) → (/open/ =d pred, ((/the/, /door/, <), (//, //, <), >, (...))

6 Simple Merge: (5) and /Bob/ d -k self →
(/Bob/ -k, (/open/ pred, ((/the/, /door/, <), (//, //, <), >), >, ((cause ((become (open)) (door))) (Bob)))

7 Simple Merge: /-ed/ >pred ++k t (λ(>pred) (tense >pred 'past)) and (6) →
(/open -ed/ ++k t, (/Bob/ -k, (//, ((/the/, /door/, <), (//, //, <), >), >,
(tense ((cause ((become (open)) (door))) (Bob)) 'past))

8 Overt Move: (7) → (/Bob/, (/open -ed/ t, (*, (// pred, ((/the/, /door/, <), (//, //, <), >), >), >, (...))

9 Simple Merge: // =t c (λ(=t) =t) and (8) →
(// c, (/Bob/, (/open -ed/, (*, (// pred, ((/the/, /door/, <), (//, //, <), >), >), >), <, (...)))

⇒ (cause :agent (bob) :effect (become :theme (door) :goal (open)) :tense past)

We derive passives and questions using the lexical entries above as well:

```
/the door was open -ed/ ⇒ (cause :agent (somebody) :effect (become :theme (door) :goal (open)) :tense past)
/who open -ed the door/ ⇒ (cause :agent (? (who)) :effect (become :theme (door) :goal (open)) :tense past)
/what open -ed/ ⇒ (become :theme (? (what)) :goal (open)) :tense past)
/what was open -ed/ ⇒ (cause :agent (somebody) :effect (become :theme (? (what)) :goal (open)) :tense past)
/did bob open the door/ ⇒
  (query :event (cause :agent (bob) :effect (become :theme (door) :goal (open)) :tense past))
*/Was Bob open -ed the door/
*/Who open the door/
*/What was open (by Bob)/
*/What did open -ed (by Bob)/
```

4.2 P-Incorporation

We have already seen how verbal entries incorporate prepositional entries: /put/ selects p_{loc}, and "locative" prepositions such as /onto/, /on/, /in/, /into/, /below/, etc., have entries of the same form:

// =d +k p_{loc} (λ(=d) (λ(x) ((go x) (path self =d))))

For a verbal entry like /lay/, on the other hand, we require a separate entry:

/lay/ =p_{be-loc} =d v_{become} (λ(=p_{be-loc}) (λ(=d) (=p_{be-loc} =d)))

where "stative locative" prepositions /on/ but not /onto/, /in/ but not /into/, etc. have p_{be-loc} entries:

// =d +k p_{be-loc} (λ(=d) (λ(x) ((be-location x) (place self =d))))

This derives, as desired:

```
/Book -s lay -ed on/*onto the shelf/
  ⇒ (be-location :patient (plural (book)) :location (place :oper (on) :location (shelf)) :tense past)
/Bob lay -ed book -s on/*onto the shelf/
  ⇒ (cause :agent (bob) :effect (be-location :patient (plural (book)))
            :location (place :oper (on) (shelf)) :tense past)
```

As another illustration of preposition incorporation, consider the dative alternation (/Bob give -ed water to Sue/ /Bob give -ed Sue water/). In this case, we have 2 entries for /give/ (c.f. Pinker (1989)), one for the *to*-form and another for the "double object" form, and have similar entries for other "spaces" of location, identity, and information, shown in Figure 5. The /to/ preposition codes the +

	/to/ =d +k p_{goal} (λ(=d) (λ(x) ((go x) (path+ =d))))	// =d =d p_{have} (λ(=d) (λ(=d2) ((have =d) =d2)))
Possession	/give/$^{(13.1)}$ =p_{goal} =d v_{cause} (λ(=p_{goal}) (λ(=d) (space 'poss (=p_{goal} =d)))) /Bob give -ed water to Sue/	/give/$^{(13.1)}$ =p_{have}? +k ++k v_{cause2} (λ(=p_{have}) (space 'poss =p_{have})) /Bob give -ed Sue water/
Location	/send/$^{(11.1)}$ =p_{goal}? =d v_{cause} (λ(=p_{goal}) (λ(=d) (space 'loc (=p_{goal} =d)))) /Bob send -ed a letter to Sue/	/send/$^{(11.1)}$ =p_{have}? +k ++k v_{cause2} (λ(=p_{have}) (space 'loc =p_{have})) /Bob send -ed Sue a letter/
Identity	/turn/$^{(26.6)}$ =p_{goal} =p_{source}? =d v_{become} (λ(=p_{goal}) (λ(=p_{source}) (λ(=d) (space 'ident (combine-paths (=p_{goal} =d) (=p_{source} =d)))))) /Bob turn -ed (from a prince) into a frog/	/appoint/$^{(26.1)}$ =p_{have}? +k ++k v_{cause2} (λ(=p_{have}) (space 'ident =p_{have})) /Sue appoint -ed Bob sheriff/
Information	/read/$^{(37.1)}$ =p_{goal}? =d v_{cause} (λ(=p_{goal}) (λ(=d) (space 'info (=p_{goal} =d)))) /Bob read -ed a story to Sue/	/read/$^{(37.1)}$ =p_{have}? +k ++k v_{cause2} (λ(=p_{have}) (space 'info =p_{have})) /Bob read -ed Sue a story/

Figure 5: Different spaces with P-Incorporation

terminal of a path, and the "space" is marked to differentiate between verbs of transfer. Otherwise the derivation of `/Bob give -ed water to Sue/` is similar to `/Bob put -ed the book on the shelf/`. The dative form is different, and results in a different semantic gloss. Following Baker (1997) and Harley (2000), the double object form derivation is:

1 Simple Merge: // =d =d p_{have} (λ(=d) (λ(=d2) ((have =d) =d2))) and /Sue/ d -k self →
(// =d p_{have}, /Sue/ -k, <, (λ(=d2) ((have (Sue)) =d2)))

2 Complex Merge: (1) and /water/ d -k self → (/water/ -k, (// p_{have}, /Sue/ -k, <), >, ((have (Sue)) (water)))

3 Simple Merge: (2) and /give/ =p_{have} +k ++k v_{cause2} (λ(=p_{have}) (space 'poss =p_{have})) →
(/give/ +k ++k v_{cause2}, (/water/ -k, (//, /Sue/ -k, <), >), <, (space 'poss ((have (Sue))(water))))

4 Covert Move: (3) → (/give/ ++k v_{cause2}, (/water/, (//, /Sue/ -k, <), >), <, (...))

5 Overt Move: (4) → (/Sue/, (/give/ v_{cause2}, (/water/, (//, *, <), >), <), >, (...))

6 Right Incorporate: (5) and // >v_{cause2} =d pred (λ(>v_{become}) (λ(=d) ((cause >v_{become}) =d))) →
(/give/ =d pred, (/Sue/, (//, (/water/, (//, *, <), >), <), >), <,
(λ(=d) ((cause (space 'poss ((have (Sue)) (water)))) =d))))

7 Complex Merge: (6) and /Bob/ d -k self →
(/Bob/ -k, (/give/ pred, (/Sue/, (//, (/water/, (//, *, <), >), <), >), <), >,
((cause (space 'poss ((have (Sue)) (water))) (Bob)))

8 Right Incorporate: (7) and /-ed/ >pred ++k t (λ(>pred) (tense >pred 'past)) →
(/give -ed/ ++k t, (/Bob/ -k, (//, (/Sue/, (//, (/water/, (//, *, <), >), <), >), <), >), >,
(tense ((cause (space 'poss ((have (Sue)) (water))) (Bob)) 'past))

9 Overt Move: (8) → (/Bob/, (/give -ed/ t,(*,(//, (/Sue/, (//, (/water/, (//,*,<),>), <), >), <), >), >), >, (...))

10 Simple Merge: (9) and // =t c →
(// c, (/Bob/, (/give -ed/, (*, (//, (/Sue/, (//, (/water/, (//, *, <), >), <), >), <), >), >), >), >,
(tense ((cause (space 'poss ((have (Sue)) (water))) (Bob)) 'past))

⇒ (cause :agent (bob) :effect (have :possessor (Sue) :theme (water) :space 'poss) :tense past)

4.3 N-Incorporation

Nouns incorporate trivially into verbs, as with verbs like `/glow/`, or into prepositions, which can be incorporated into verbs in turn, as with verbs like `/butter/` (figure), `/shelf/` (ground), and `/shovel/` (instruments):

Considering the derivation of `/Bob shelf -ed the book/` vs. `/Bob butter -ed the bread/`, the core distinction is in how the argument `/the book/` and `/the bread/` are applied to the two primitives p_{loc1} and p_{loc2} that have different orders of selecting "figure" and "ground":

p_{loc1}	(λ(figure) (λ(ground) ((go figure) (path () ground))))
p_{loc2}	(λ(ground) (λ(figure) ((go figure) (path () ground))))

The two derivations proceed identically in form, but results in a different semantic structure as a result of the above figure-ground reversal:

```
/Bob butter -ed the bread/
  ⇒ (cause :agent (bob) :effect (go :theme (butter) :path (path :oper (bread) :terminal+ ())) :tense past)

/Bob shelf -ed the book/
  ⇒ (cause :agent (bob) :effect (go :theme (book) :path (path :oper (shelf) :terminal+ ())) :tense past)
```

Root/Nominal Entry	**Verbal Entry**	**EVCA Sections**
Processes/Activities /glow/$^{(40.2)}$ emission // >emission n identity /a glow/	// >emission v$_{do}$ (λ(>emission) (do >emission)) /The light glow -ed/	
Figures /butter/$^{(9.9)}$ figure$_{loc}$ // >figure$_{loc}$n identity // >figure$_{loc}$d -k identity /the butter/, /butter/	// >figure$_{loc}$ =d v$_{cause}$ (λ(>figure$_{loc}$) (λ(=d) ((p$_{loc1}$ =d) >figure$_{loc}$))) /Bob butter -ed the bread/	/pit/$^{(10.7)}$, /whale/$^{(13.7)}$, /cut/$^{(21.1)}$, /dye/$^{(24)}$, /autograph/$^{(25.3)}$, /calf/$^{(28)}$, /knight/$^{(29.8)}$, /love/$^{(31.2)}$, /whisper/$^{(37.3)}$, /vomit/$^{(40.1.2)}$, /braid/$^{(41.2.2)}$, /smell/$^{(43.3)}$, /fracture/$^{(54.2)}$
Grounds /shelf/$^{(9.10)}$ ground$_{loc}$ // >ground$_{loc}$ n identity /a shelf/	// >ground$_{loc}$ =d v$_{cause}$ (λ(>ground$_{loc}$) (λ(=d) ((p$_{loc2}$ =d) >ground$_{loc}$))) /Bob shelf -ed the book/	/mine/$^{(10.9)}$, /videotape/$^{(25.4)}$, /tutor/$^{(29.8)}$
Instruments /shovel/$^{(9.3)}$ inst$_{loc}$ // >inst$_{loc}$ n identity /the shovel/	// >inst$_{loc}$ =p$_{loc}$? =d v$_{cause}$ (λ(>inst$_{loc}$) (λ(=p$_{loc}$) (λ(=d) ((using >inst$_{loc}$) (=p$_{loc}$ =d))))) /Bob shovel -ed the dirt (onto the truck)/	/mop/$^{(10.4.2)}$, /whip/$^{(8.3)}$, /clamp/$^{(2.4)}$, /pencil/$^{(25.2)}$, /email/$^{(37.4)}$, /ferry/$^{(11.5)}$, /cycle/$^{(51.4.1)}$, /paddle/$^{(51.4.2)}$

Figure 6: Different kinds of N-Incorporation

The same alternation patterns seen in `/butter/`, `/shelf/`, and `/shovel/` can be observed in a variety of other "spaces" in addition to the "location" space - removal, possession, impression, identity, emotion, information, body possession, material possession, and perceptual space.

5 Implementation Analysis

We have modeled all of the verb classes in Levin (1993) through combinations of N-incorporation, A-incorporation, and P-incorporation in verbal entries. Our current lexicon contains a total of 347 entries, where:

1. 199 are verbal entries. Frequently, one entry covers more than 1 EVCA verb class.
2. 51 are pure root entries (e.g. `/glow/ emission`), 37 are nominalizing entries (e.g. `// >emission n`), and 4 are adjectival entries (e.g. `// >state a`)
3. 20 are preposition entries (e.g. /on/ =d +k p$_{loc}$). One entry often covers more than one preposition (e.g. `/on/`, `/in/`)
4. 77 are "other" entries (e.g. `// =t c`), including noun entries.

Of the 199 verbal entries (marked with v$_{do}$, v$_{become}$, v$_{cause}$, etc.), 142 contain 1 or more instances of P-incorporation, 60 contain N-incorporation, and 4 contain A-incorporation. To the extent that the core meaning of the verbs in reflected in the types of structures that are incorporated, this illustrates how prevalent incorporation is. At present, these verbal entries fall into traditional broad classes:

INTRANSITIVES : Not Externally Causable	/The light glow -ed/ */Bob glow -ed the light/
// >v$_{do}$ pred (λ(>v$_{do}$) (>v$_{do}$ =d))	/glow/$^{(40.2)}$ emission self // >emission v$_{do}$ (λ(>v$_{do}$) (do >emission))
INTRANSITIVE/TRANSITIVES : Externally Causable	/The door open -ed/ /Bob open -ed the door/
// >v$_{become}$ +k =d pred (λ(>v$_{become}$) ((cause >v$_{become}$) =d)) // >v$_{become}$ pred (λ(>v$_{become}$) >v$_{become}$) /-ed/ >v$_{become}$ =p$_{by}$? pred$_p$ (λ(=p$_{by}$) (=p$_{by}$ >v$_{become}$))	/open/$^{(45.4)}$ state self // >state v$_{become}$ (λ(>state) (λ(=d) ((become >state) =d)))
TRANSITIVES : Externally Caused	/Bob put -ed the book on the shelf/
// >v$_{cause}$ +k =d pred (λ(>v$_{cause}$) ((cause >v$_{cause}$) =d)) /-ed/ >v$_{cause}$ =p$_{by}$? pred$_p$ (λ(=p$_{by}$) (=p$_{by}$ >v$_{cause}$))	/put/ =p$_{loc}$ =d v$_{cause}$ (λ(=p$_{loc}$) (λ(=d) (=p$_{loc}$ =d)))
DITRANSITIVES : Externally Caused	/Bob give -ed Sue the book/
// >v$_{cause2}$ =d pred (λ(>v$_{cause2}$) ((cause >v$_{cause2}$) =d)) /-ed/ >v$_{cause2}$ =p$_{by}$? pred$_p$ (λ(=p$_{by}$) (=p$_{by}$ >v$_{cause2}$))	/give/ =p$_{have}$ +k ++k v$_{cause2}$ (λ(=p$_{have}$) =p$_{have}$)

Figure 7: Broad verb classes in our implementation

However, the *reason* a particular verb is in a particular verb class requires appealing to notions of whether an event is not externally causable (/glow/ vs. /open/), or whether it must be externally caused (/lay/ vs. /put/). Verbs such as /open/ (A-incorporation) or /lay/ (P-incorporation) are of the $\mathtt{v}_{\mathtt{become}}$ class, and need only one entry to generate 2 alternation patterns, as discussed earlier. Verbs such as /put/, on the other hand, require only one entry because they have only one canonical surface realization, and *must* be externally caused. In some cases, verbs such as /give/ require two entries for each of their canonical surface realizations. A very small number of entries (3) generate all the passive forms for the $\mathtt{v}_{\mathtt{become}}$, $\mathtt{v}_{\mathtt{cause}}$, $\mathtt{v}_{\mathtt{cause2}}$ broad classes : one for each class.

For the 183 verb classes of EVCA, a distributional analysis of entries per class reveals that 141 sections have exactly 1 entry in our lexicon (e.g. the /put/ class, the /lay/ class, the /open/ class), 32 sections have exactly 2 entries in our lexicon (e.g. the /give/ class), and only 10 sections have 3 or more entries in our lexicon (e.g. the /email/ class). Using incorporation theory, we have reduced the vast majority of EVCA sections (77%) to just 1 entry. Only a minority (42/183, 23%) need more than 1 entry, and we suspect that some of these may reduce to 1 entry with further analysis. We should simultaneously stress, however, that at present not all alternations described in Levin (1993) can be currently modeled fully, requiring new operations (selection, adjunction, agreement, reflexives, particles, aspect, etc.) We summarize our present coverage:

ALTERNATIONS MODELED	ALTERNATIONS NOT MODELED
Modeled, does not need 2 entries:	**Requires selection/adjunction:**
1.1.2 Causative	2.5 Reciprocal Alternations
2.4.3/2.4.4 Total Transformation	2.13 Possessor-Attribute Factoring Alternations
5.1 Verbal Passive	3.1 Time Subject Alternation
5.2 Prepositional Passive	3.2 Natural Force Subject Alternation
	3.3 Instrument Subject Alternation
Currently requires 2 or more entries but probably can be reduced to 1:	3.4 Abstract Cause Subject Alternation
1.1.1 Middle (+effect)	3.5 Locatum Subject Alternation
1.3 Conative (+motion, +contact)	3.6 Location Subject Alternation
2.12 Body-Part Possessor Ascension Alternation	3.7 Container Subject Alternation
7.1 Cognate Object Construction	3.8 Raw Material Subject
7.2 Cognate Prepositional Phrase Construction	3.9 Sum of Money Subject Alternation
	3.10 Source Subject Alternation
Modeled, currently needs 2 entries when 2 alternations possible:	7.3 Reaction Object Construction
1.1.3 Substance / Source Alternation	7.4 X's Way Construction
1.2 Unexpressed Object Alternation	7.5 Resultative Construction
1.4. Preposition Drop Alternation	7.8 Direction Phrases with Nondirected Motion
2.1 Dative (give)	8.5 Obligatory Adverb
2.2 Benefactive (carve)	8.6 Obligatory Negative Polarity Element
2.3 Locative Alternation	
2.4.1/2.4.2 Material/Product Alternation	**Requires binding/reflexive operations:**
2.6 Fulfilling Alternation	4.1 Virtual Reflexive Alternation
2.7 Image Impression Alternation	4.2 Reflexive of Appearance
2.8 With/Against Alternation	5.3/5.4 Adjectival Passive
2.9 Through/With Alternation	6.1 There-insertion
2.10 Blame Alternation	7.6 Unintentional Interpretation of Object
2.11 Search Alternation	7.7 Bound Nonreflexive Anaphor as Prepositional Object
2.14 As Alternation	8.1 Obligatory Passive
	8.2 Obligatory Reflexive Object
	8.3 Inalienably Possessed Body-Part
	8.4 Expletive It Object

We can extend our minimalist operations to include Agree (see Chomsky 2001) and Adjoin (Chomsky, forthcoming), or use already well developed theories from earlier formalisms. This is the subject of future work.

Our reduction to one or two entries per verb class is in stark contrast to a typical CFG, which would contain many more entries. Whereas /lay/ =d $=\mathtt{p}_{\mathtt{be-loc}}$ is represented with 1 entry in our implementation, we would expect at least seven grammar rules to handle basic constructions in a typical CFG:

VP $\rightarrow$ V0 NP $\mathrm{PP}_{\mathrm{loc}}$	/He lay -ed the book on the shelf/	VP $\rightarrow$ V0 $\mathrm{PP}_{\mathrm{loc}}$	/The book lay -ed on the shelf/
VPass $\rightarrow$ V0 $\mathrm{PP}_{\mathrm{loc}}$	/The book was lay -ed on the shelf/	VP/NP $\rightarrow$ V0 $\mathrm{PP}_{\mathrm{loc}}$/NP	/What did the book lay on/
VP/NP $\rightarrow$ V0 NP/NP $\mathrm{PP}_{\mathrm{loc}}$	/What was lay -ed on the shelf/	VP/PP $\rightarrow$ V0 $\mathrm{PP}_{\mathrm{loc}}$/PP	/Where did the book lay/
VP/NP $\rightarrow$ V0 NP $\mathrm{PP}_{\mathrm{loc}}$/NP	/Where was the book lay -ed/		

We do not claim that the minimalist implementation presented here is the only account that can reduce the majority of EVCA verb classes to just one entry per verb. It is likely that other frameworks such as lexicalized TAGs or categorial grammars (e.g. Vijay-Shankar and Weir 1999, Steedman 2000) that also compactly handle movement, passivization, etc. can simulate Hale and Keyser incorporation operations present in our implementation, resulting in a more compact grammar/lexicon. The key lesson to be learned is that by implementing Hale and Keyser's incorporation theory in *some* framework, there is enormous compaction, resulting in a grammar that is more easily engineered or learned.

Our parser and lexicon (written in MIT Scheme), and an extensive array of sample derivations and resulting semantic structures is freely available at `http://web.mit.edu/niyogi/www/minimal.htm`

Acknowledgements

I thank Professor Robert C. Berwick for motivating and supporting this work, and for providing many useful comments in improving this paper. Kenneth Hale and Andrew Nevins provided many engaging discussions.

References

[1] Baker, M. (1997). "Thematic roles and syntactic structure." In L. Haegeman (eds.) *Elements of Grammar: Handbook of Generative Syntax.* Dordrecht, Kluwer, pp. 73-137.

[2] Berwick, R. and Epstein, S. (1995). "Computational Minimalism: The Convergence of the Minimalist Syntactic Program and Categorial Grammar." AMILP '95 Workshop.

[3] Chomsky, N. (2000). "Minimalist Inquiries." In R. Martin, D. Michaels and J. Uriagereka (eds.) *Step by Step: Essays on Minimalist Syntax in honor of Howard Lasnik.* MIT Press, Cambridge, MA.

[4] Chomsky, N. (2001). "Derivation by Phase." In M. Kenstowicz (ed.) *Ken Hale: A Life in Language.* MIT Press, Cambridge, MA.

[5] Chomsky, N. (forthcoming). "Beyond Explanatory Adequacy." Ms., MIT.

[6] Harley, H. (2000) "Possession and the double object construction." Ms., University of Arizona.

[7] Hale, K. and Keyser, S. J. (1993). "On Argument Structure and Lexical Expression of Syntactic Relations." In K. Hale and S. J. Keyser (eds). *The View from Building 20*, pp. 53-109. Cambridge, Mass.: MIT Press.

[8] Hale, K. and Keyser, S. J. (1998). "The basic elements of argument structure." In H. Harley, ed., *Papers from the UPenn/MIT Roundtable on Argument Structure and Aspect*, pp. 73-118. MIT Working Papers in Linguistics 32. MITWPL, Department of Linguistics and Philosophy, MIT, Cambridge, Mass.

[9] Harkema, H. (2000) "A recognizer for minimalist grammars." In Sixth International Workshop on Parsing Technologies, IWPT 2000.

[10] Jackendoff, R. S. (1983) *Semantics and Cognition.* MIT Press, Cambridge, MA.

[11] Jackendoff, R. S. (1990) *Semantic Structures.* MIT Press, Cambridge, MA.

[12] Levin, B. (1993) *English Verb Classes and Alternations: A Preliminary Investigation*, University of Chicago Press, Chicago, IL.

[13] Pinker, S. (1989) *Learnability and Cognition.* MIT Press, Cambridge, MA.

[14] Rappaport Hovav, M. and Levin, B. (1998) "Building Verb Meanings." In M. Butt and W. Geuder (eds.), *The Projection of Arguments: Lexical and Compositional Factors*, CSLI Publications, Stanford, CA, 97-134.

[15] Stabler, E. (2000) "Minimalist grammars and Recognition." Manuscript for the SFB340 workshop at Bad Teinach.

[16] Stabler, E. (1997) "Derivational minimalism." Appears in Retore (ed.) *Logical Aspects of Computational Linguistics.* Springer, 1997, pp 68-95.

[17] Steedman, M. (2000) *The Syntactic Process.* MIT Press, Cambridge, MA.

[18] Vijay-Shanker, K., and Weir, D. (1999) "Exploring the Underspecified World of Lexicalized Tree Adjoining Grammars." In Proceedings of 6th Mathematics of Language Conference, Orlando, USA.

Appendix

Below is a definition of an agenda-driven, chart-based parser for minimalist grammars. For a given grammar and input string, there is a set of items, call them axioms, that are taken to represent true grammatical claims. Given these axioms, and the structure-building rules that allow us to make new true grammatical claims, we can design a parser, which, given an input string, determines the truth of the input string. If a structure has a particular set of goal features (i.e. `c`) and phonetic features that match the input, then the input string in is the language defined by our grammar. Our procedure to find all items that are true for a given grammar and input string works as follows:

1. Initialize the chart and the agenda (both modeled as an indexable stack) to be an empty set of items – an item has the form (S, f, i_A, i_B) where the first element S is a simple or complex structure, the second element f is a symbol representing the source of the structure (`Merge`, `Move`, `Optional-Merge`, or `Axiom`), and i_A and i_B are indices into elements in chart which created S. The axioms are pushed onto the agenda, with f = `Axiom`, $i_A = i_B = 0$, and S being a underived simple structure of the form `/phonetic/ feature-list λ-expression`. In our case, the axioms are the union of (1) all phonetically null lexical items and (2) the lexical entry(s) for each word in the input.

2. Repeat the following until the agenda is empty:
   (a) Pop an item off the agenda, call it the trigger.
   (b) Push the trigger onto the chart, if the trigger has not already been placed on the chart.
   (c) If the trigger item was added to the chart in (b), then:
      i. generate all items that can be derived from *Merge* of the trigger item and any items of the chart, pushing each new item onto the agenda with f = `Merge`, and i_A being the index to the licensor item and i_B being the index to the licensee item (one of i_A or i_B being the trigger's index)
      ii. generate all items that can be derived from the trigger item solely (via *Move*, or *Optional Merge*), pushing each new item onto the agenda with f = `Move` (or f = `Optional − Merge`), i_A being the index of the trigger item, i_B =0.

3. When the agenda is empty, scan all items in the chart for structures that contain solely the goal features (a `c` feature). If such a structure exists, then its phonetic content is "spelled-out" – if the phonetic content matches the input string, then we print the derivation recovery and computed semantic structure:
   (a) To print the derivation of an item (S, f, i_A, i_B), we can print the derivations of item i_A and i_B (if non-zero), and then print the resulting structure S.
   (b) To compute the semantics of an item (S, f, i_A, i_B), we condition the result on f:
      - if f =`Merge`, then return the result of applying the semantics of item i_A to that of item i_B
      - if f =`Move`, then return the semantics of item i_A
      - if f =`Optional-Merge`, then return the result of applying the semantics of item i_A to a precomputed `λ-expression` based on the optional feature skipped.
      - if f =`Axiom`, then return the `λ-expression` of the axiom S, guaranteed to be a simple structure