Extending Lambek grammars: a logical account of minimalist grammars

Alain Lecomte
CLIPS-IMAG
Université Pierre Mendès-France,
BSHM - 1251 Avenue Centrale,
Domaine Universitaire de St Martin d'Hères
BP 47 - 38040 GRENOBLE cedex 9, France
`Alain.Lecomte@upmf-grenoble.fr`

Christian Retoré
IRIN, Université de Nantes
2, rue de la Houssinière BP 92208
44322 Nantes cedex 03, France
`retore@irisa.fr`

Abstract

We provide a logical definition of Minimalist grammars, that are Stabler's formalization of Chomsky's minimalist program. Our logical definition leads to a neat relation to categorial grammar, (yielding a treatment of Montague semantics), a parsing-as-deduction in a resource sensitive logic, and a learning algorithm from structured data (based on a typing-algorithm and type-unification). Here we emphasize the connection to Montague semantics which can be viewed as a formal computation of the logical form.

1 Presentation

The connection between categorial grammars (especially in their logical setting) and minimalist grammars, which has already been observed and discussed (Retoré and Stabler, 1999), deserve a further study: although they both are lexicalized, and resource consumption (or feature checking) is their common base, they differ in various respects. On the one hand, traditional categorial grammar has no *move* operation, and usually have a poor generative capacity unless the good properties of a logical system are damaged, and on the other hand minimalist grammars even though they were provided with a precise formal definition (Stabler, 1997), still lack some computational properties that are crucial both from a theoretical and a practical viewpoint. Regarding applications, one needs parsing, generation or learning algorithms, and, considering more conceptual aspects, such algorithms are needed too to validate or invalidate linguistic claims regarding economy or efficiency. Our claim is that a logical treatment of these grammars leads to a simpler description and well defined computational properties. Of course among these aspects the relation to semantics or logical form is quite important; it is claimed to be a central notion in minimalism, but logical forms are rather obscure, and no computational process from syntax to semantics is suggested. Our logical presentation of minimalist grammar is a first step in this direction: to provide a description of minimalist grammar in a logical setting immediately set up the computational framework regarding parsing, generation and even learning, but also yields some good hints on the computational connection with logical forms.

The logical system we use, a slight extension of (de Groote, 1996), is quite similar to the famous Lambek calculus (Lambek, 1958), which is known to be a neat logical system. This logic has recently shown to have good logical properties like the subformula property which are relevant both to linguistics and computing theory (e.g. for modeling concurrent processes). The logic under consideration is a super-imposition of the Lambek calculus (a non commutative logic) and of intuitionistic multiplicative logic (also known as Lambek calculus with permutation). The context, that is the set of current hypotheses, are endowed with an order, and this order is crucial for obtaining the expected order on pronounced and interpreted features but it can also be relaxed when necessary: that is when its effects have already been recorded (in the labels) and the corresponding hypotheses can therefore be discharged.

Having this logical description of syntactic analyses allows to reduce parsing (and production) to deduction, and to extract logical forms from the proof; we thus obtain a close connection between syntax and semantics as the one between Lambek-style analyses and Montague semantics.

2 The grammatical architecture

The general picture of these logical grammars is as follows. A lexicon maps words (or, more generally, items) onto a logical formula, called the (syntactic) type of the word. Types are defined from syntactic of formal features $\mathcal{P}$ (which are propositional variables from the logical viewpoint):

- categorial features (categories) involved in *merge*: BASE $= \{\mathtt{c}, \mathtt{t}, \mathtt{v}, \mathtt{d}, \mathtt{n}, \ldots\}$
- functional features involved in *move*: FUN $= \{\overline{\mathtt{k}}, \overline{\mathtt{K}}, \overline{\mathtt{wh}}, \ldots\}$

The connectives in the logic for constructing formulae are the Lambek implications (or slashes) $\backslash, /$ together with the commutative product of linear logic $\otimes$. [1]

Once an array of items has been selected, a sentence (or any phrase) is a deduction of IP (or of the phrasal category) under the assumptions provided by the syntactic types of the involved items. This first step works exactly as Lambek grammars, except that the logic and the formulae are richer.

Now, in order to compute word order, we proceed by labeling each formula in the proof. These labels, that are called phonological and semantic features in the transformational tradition, are computed from the proofs and consist of two parts that can be superimposed: a phonological label, denoted by $/word/$, and a semantic label[2] denoted by $(word)$ — the super-imposition of both label being denoted by $word$. The reason for having such a double labeling, is that, as usual in minimalism, semantic and phonological features can move separately. It should be observed that the labels are not some extraneous information; indeed the whole information is encoded in the proof, and the labeling is just a way to extract the phonological form and the logical form from the proof.

We rather use chains or copy theory than movements and traces: once a label or one aspect (semantic or phonological) has been met it should be ignored when it is met again. For instance a label $Peter\ (Mary)\ loves\ Mary$ corresponds to a semantic label $(Peter)\ (Mary)\ (love)$ and to the phonological form $/Peter/\ /loves/\ /Mary/$.

3 Logico-grammatical rules for merge and phrasal movement

Because of the sub-formula property we need not present all the rules of the system, but only the ones that can be used according to the types that appear in the lexicon. Further more, up to now there is no need to use introduction rules (called hypothetical reasoning in the Lambek calculus): so our system looks more like Combinatory Categorial Grammars or classical AB-grammars. Nevertheless some hypotheses can be cancelled during the derivation by the product-elimination rule. This is essential since this rule is the one representing chains or movements.

We also have to specify how the labels are carried out by the rules. At this point some non logical properties can be taken into account, for instance the strength of the features, if we wish to take them into account. They are denoted by lower-case variables. The rules of this system in a Natural Deduction format are:

$$\frac{\Gamma \vdash x : A/B \quad \Delta \vdash y : B}{\Gamma ; \Delta \vdash xy : A}\,[/E]$$

$$\frac{\Delta \vdash y : B \quad \Gamma \vdash x : B\backslash A}{\Delta ; \Gamma \vdash yx : A}\,[\backslash E]$$

$$\frac{\Gamma[(\Delta_1 ; \Delta_2)] \vdash A}{\Gamma[(\Delta_1 , \Delta_2)] \vdash A}\,entropy$$

$$\frac{\Gamma \vdash \alpha : A \otimes B \quad \Delta, x : A, y : B, \Delta' \vdash \gamma : C}{\Delta, \Gamma, \Delta' \vdash \gamma[\alpha/\{x,y\}] : C}\,[\otimes E]$$

[1]The logical system also contains a commutative implication, $\multimap$, and a non commutative product $\bullet$ but they do not appear in the lexicon, and because of the subformula property, they are not needed for the proofs we use.

[2]We prefer *semantic label* to *logical form* not to confuse *logical forms* with the logical formulae present at each node of the proof.

This later rule encodes movement and deserves special attention. The label $\gamma[\alpha/\{x, y\}]$ means the *substitution of* α *to the unordered set* $\{x,\ y\}$ that is the simultaneous substitution of α for both x and y, *no matter the order between* x *and* y *is*. Here some non logical but linguistically motivated distinction can be made. For instance according to the strength of a feature (e.g. weak case $\overline{\texttt{k}}$ versus strong case $\overline{\texttt{K}}$), it is possible to decide that only the semantic part that is (α) is substituted with x.

In the figure 1, the reader is provided with an example of a lexicon and of a derivation. The resulting label is $(a\ book)\ reads\ a\ book$ phonological form is $/reads/\ /a\ book/$ while the resulting logical form is $(a\ book)\ (reads)$.

Notice that language variation from SVO to SOV does not change the analysis. To obtain the SOV word order, one should simply use $\overline{\texttt{K}}$ (strong case feature) instead of $\overline{\texttt{k}}$ (weak case feature) in the lexicon, and use the same analysis. The resulting label would be $a\ book\ reads\ a\ book$ which yields the phonological from $/a\ book/\ /reads/$ and the logical form remains the same $(a\ book)\ (reads)$.

Observe that although entropy which suppresses some order has been used, the labels consist in *ordered* sequences of phonological and logical forms. It is so because when using [/ E] and [\ E], we necessarily order the labels, and this order is then recorded inside the label and is never suppressed, even when using the entropy rule: at this moment, it is only the order *on hypotheses* which is relaxed.

In order to represent the minimalist grammars of (Stabler, 1997), the above subsystem of partially commutative intuitionistic linear logic (de Groote, 1996) is enough and the types appearing in the lexicon also are a strict subset of all possible types:

Definition 1 *$\mathcal{MG}$-proofs contain only three kinds of steps:*

- *implication steps (elimination rules for / and \)*
- *tensor steps (elimination rule for $\otimes$)*
- *entropy steps (entropy rule)*

Definition 2 *A lexical entry consists in an axiom $\vdash w : \mathcal{T}$ where $\mathcal{T}$ is a type:*

$$((F_2\backslash(F_3\backslash ...(F_n\backslash(G_1\otimes G_2\otimes ...\otimes G_m\otimes A))))/F_1)$$

where:

- *m and n can be any number greater than or equal to 0,*
- *$F_1, ..., F_n$ are attractors,*
- *$G_1, ..., G_m$ are features,*
- *A is the resulting category type*

Derivations in this system can be seen as *T-markers* in the Chomskyan sense. [/E] and [\E] steps are merge steps. [$\otimes$E] gives a co-indexation of two nodes that we can see as a move step. For instance in a tree presentation of natural deduction, we shall only keep the coindexation (corresponding to the cancellation of A and B: this is harmless since the conclusion is not modified, and makes our natural deduction *T-markers*).

Such lexical entries, when processed with $\mathcal{MG}$-rules encompass Stabler minimalist grammars; this system nevertheless overgenerates, because some minimalist principles are not yet satisfied: they correspond to constraints on derivations.

3.1 Conditions on derivations

The restriction which is still lacking concerns the way the proofs are built. Observe that this is an algorithmic advantage, since it reduces the search space.

The simplest of these restriction is the following: the attractor F in the label L of the target β locates the *closest* F' in its domain. This simply corresponds to the following restriction.

Definition 3 (Shortest Move) *: A $\mathcal{MG}$-proof is said to respect the shortest move condition if it is such that:*

- *the same formula never occurs twice as a hypothesis of any sequent*
- *every active hypothesis during the proof process is discharged as soon as possible*

The consequences of this definition are the following:

Figure 1: reads a book

$$\begin{array}{lll} reads & ::= & \vdash reads : ((\overline{\mathtt{k}}\backslash \mathtt{vp})/\mathtt{d}) \\ a & ::= & \vdash a : ((\mathtt{d} \otimes \overline{\mathtt{k}})/\mathtt{n}) \\ book & ::= & \vdash book : \mathtt{n} \end{array}$$

$$\dfrac{\dfrac{\vdash a : ((\mathtt{d} \otimes \overline{\mathtt{k}})/\mathtt{n}) \quad \vdash book : \mathtt{n}}{\vdash a\ book : \mathtt{d} \otimes \overline{\mathtt{k}}}[/E] \qquad \dfrac{\dfrac{\dfrac{y : \overline{\mathtt{k}} \vdash y : \overline{\mathtt{k}} \qquad \dfrac{\vdash reads : ((\overline{\mathtt{k}}\backslash \mathtt{vp})/\mathtt{d}) \quad x : \mathtt{d} \vdash x : \mathtt{d}}{x : \mathtt{d} \vdash reads\ x : (\overline{\mathtt{k}}\backslash \mathtt{vp})}[/E]}{y : \overline{\mathtt{k}}; x : \mathtt{d} \vdash y\ reads\ x : \mathtt{vp}}[\backslash E]}{y : \overline{\mathtt{k}}, x : \mathtt{d} \vdash y\ reads\ x : \mathtt{vp}}[entropy]}{\vdash (a\ book)\ reads\ a\ book : \mathtt{vp}}[\otimes E]$$

1. τ_1 ... τ_1 ... τ_2 ... $\vdash$ C is forbidden
2.
   - if there is a sequent ... τ ... $\vdash \tau' \backslash$ C
   - if there is a type τ' such that $\Gamma \vdash \tau' \otimes \tau$ is a (proper or logical) axiom,
   - then a hypothesis τ' must be introduced, rather than any constant τ', in order to discharge τ

We may see an application of this condition in the fact that sentences like:

*Who$_2$ do you know [who$_1$ e$_2$ likes e$_1$]

*Who$_2$ do you know [who$_1$ e$_1$ likes e$_2$]

are ruled out. Let us look at the beginning of their derivation (in a tree-like presentation of natural deduction proofs): at the stage where we stop the deduction on figure 2, we **cannot** introduce a new hypothesis $z_2 : \overline{\mathtt{k}} \otimes \mathtt{d}$ because there is already an active one (z_1), the only possible continuation is to discharge y_2 and x_2 altogether by means of a ”constant”, like $mary$, so that, in contrast:

You know [who$_1$ Mary likes e$_1$]

is correct.

3.2 Extension to head-movement

We have seen above that we are able to account for SVO and SOV orders quite easily. Nevertheless we could not handle this way VSO language. Indeed this order requires head-movement.

In order to handle head-movement, we shall also use the product $\otimes$ but between functor types.

As a first example, let us take the very simple example of: *peter loves mary*. Starting from the following lexicon in figure 3 we can build the tree given in the same figure; it represents a natural deduction in our system, hence a syntactic analysis. The resulting phonological form is $/Peter//loves//Mary/$ while the resulting logical form is $(Peter)(Mary)(loves)$ — the possibility to obtain SOV word order with a $\overline{\mathtt{K}}$ instead of a $\overline{\mathtt{k}}$ also applies here.

4 The interface between syntax and semantics

In categorial grammar (Moortgat, 1996), the production of logical forms is essentially based on the association of pairs $< string, type >$ with lambda terms representing the logical form of the items, and on the application of the Curry-Howard homomorphism: each ($/$ or $\backslash$) - elimination rule translates into application and each introduction step into abstraction. Compositionality assumes that each step in a derivation is associated with a semantical operation.

In generative grammar (Chomsky, 1995), the production of logical forms is in last part of the derivation, performed after the so-called *Spell Out* point, and consists in movements of the semantical features only. Once this is done, two forms can be extracted from the result of the derivation: a phonological form and a logical one.

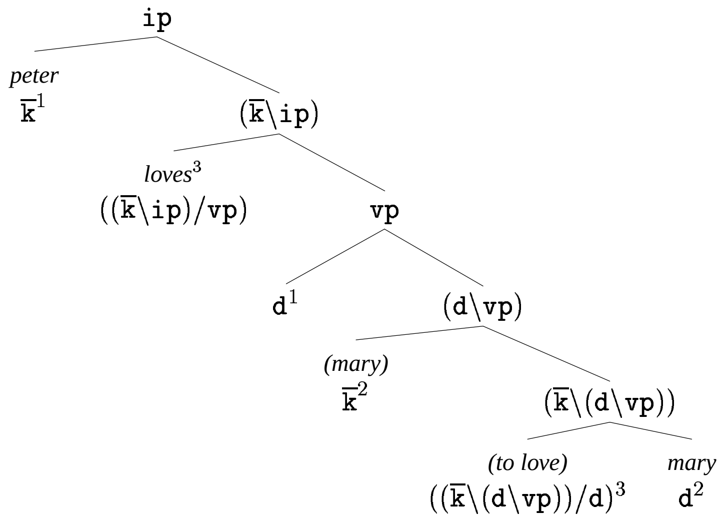
These two approaches are therefore very differ-

Figure 2: Complex NP constraint

$$\dfrac{y_2 : \overline{\mathtt{k}} \qquad \dfrac{((\overline{\mathtt{k}}\backslash\mathtt{t})/\mathtt{vp}) \qquad \dfrac{x_2 : \mathtt{d} \qquad \dfrac{\mathbf{z_1} : \overline{\mathtt{k}} \otimes \mathtt{d} \qquad \dfrac{\cancel{y}_1 : \overline{\mathtt{k}} \qquad \dfrac{likes : ((\overline{\mathtt{k}}\backslash(\mathtt{d}\backslash\mathtt{vp}))/\mathtt{d}) \qquad \cancel{x}_1 : \mathtt{d}}{likes\ x_1 : (\overline{\mathtt{k}}\backslash(\mathtt{d}\backslash\mathtt{vp}))}}{y_1\ likes\ x_1 : (\mathtt{d}\backslash\mathtt{vp})}}{z_1\ likes : (\mathtt{d}\backslash\mathtt{vp})}}{x_2 z_1\ likes : \mathtt{vp}}}{x_2 z_1\ likes : (\overline{\mathtt{k}}\backslash\mathtt{t})}}{y_2 x_2 z_1\ likes : \mathtt{t}}$$

Figure 3: Peter loves Mary

$loves$	$::=$	$\vdash loves : ((\overline{\mathtt{k}}\backslash\mathtt{ip})/\mathtt{vp}) \otimes ((\overline{\mathtt{k}}\backslash(\mathtt{d}\backslash\mathtt{vp}))/\mathtt{d})$
$peter$	$::=$	$\vdash peter : \overline{\mathtt{k}} \otimes \mathtt{d}$
$mary$	$::=$	$\vdash mary : \overline{\mathtt{k}} \otimes \mathtt{d}$



ent, but we can try to make them closer by replacing semantic features by lambda-terms and using some canonical transformations on the derivation trees.

Instead of converting directly the derivation tree obtained by composition of types, something which is not possible in our translation of minimalist grammars, we extract a logical tree from the previous, and use the operations of Curry-Howard on this extracted tree. Actually, this extracted tree is also a deduction tree: it represents the proof we could obtain in the semantic component, by combining the semantic types associated with the syntactic ones (by a homomorphism $\mathcal{H}$ to specify). Such a proof is in fact a proof in implicational intuitionistic linear logic.

4.1 Logical form for example 3

Coindexed nodes refer to ancient hypotheses which have been discharged simultaneously, thus resulting in phonological features and semantical ones at their right place[3].

By extracting the subtree the leaves of which are full of semantic content, we obtain a structure that can be easily seen as a composition:

(peter)((mary)(to_love))

If we replace these "semantic features" by λ-terms, we have:

$$(\lambda u.u(peter), (\lambda u.u(mary), \lambda x.\lambda y.love(y, x)))$$

This shows that necessarily raised constituants in the structure are not only "syntactically" raised but also "semantically" lifted, in the sense that $\lambda u.u(peter)$ is the high order representation of the individual `peter`[4].

4.2 Subject raising

Let us look at now the example: *mary seems to work* From the lexicon in figure 4 we obtain the deduction tree given in the same figure.

This time, it is not so easy to obtain the logical representation:

$$seem(to\_work(mary))$$

The best way to handle this situation consists in assuming that:

- the verbal infinitive head (here *to work*) applies to a variable x which occupies the `d`-position,
- the semantics of the main verb (here *to seem*) applies to the result, in order to obtain $seem(to\_work(x))$,
- the x variable is abstracted in order to obtain $\lambda x.seem(to\_work(x))$ just before the semantic content of the specifier (here the nominative position, occupied by $\lambda u.u(mary)$) applies.

This shows that the semantic tree we want to extract from the derivation tree in types logic is not simply the subtree the leaves of which are semantically full. We need in fact some transformation which is simply the *stretching* of some nodes. These stretchings correspond to $\rightarrow$-introduction steps in a Natural deduction tree. They are allowed each time a variable has been used before, which is not yet discharged and they necessarily occur just before a semantically full content of a specifier node (that means in fact a node labelled by a functional feature) applies.

Actually, if we say that the tree so obtained represents a deduction in a natural deduction format, we have to specify which formulae it uses and what is the conclusion formula. We must therefore define a homomorphism between syntactic and semantic types.

Let $\mathcal{H}$ be this homomorphism.

We shall assume:

- $\mathcal{H}(\mathtt{ip})=\mathbf{t}$, $\mathcal{H}(\mathtt{vp})\in\{\mathbf{t},(\mathbf{e} \rightarrow \mathbf{t})\}$, $\mathcal{H}(\mathtt{d})=\mathbf{e}$,
- $\mathcal{H}(\mathtt{a}\backslash\mathtt{b})=\mathcal{H}(\mathtt{b}/\mathtt{a})= (\mathbf{H}(\mathtt{a}) \rightarrow \mathbf{H}(\mathtt{b}))$,
- $\forall \bar{\mathtt{f}}\ \mathbf{H}(\bar{\mathtt{f}}) \in \{((\mathbf{e} \rightarrow \mathbf{X}) \rightarrow \mathbf{X}), (\mathbf{X} \rightarrow \mathbf{X})\}$ [5]

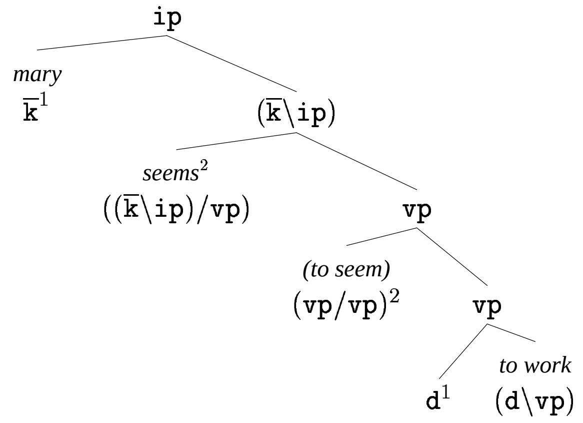
[3]For the time being, we make abstraction of the representation of time, mode, aspect... that would be supported by the inflection category.

[4]It is important to notice that if we consider $\lambda u.u(mary)$ a typed lambda term, we must only assume it is of some type freely raised from $\mathbf{e}$, something we can represent by $((\mathbf{e} \rightarrow \mathbf{X}) \rightarrow \mathbf{X})$, where X is a type-variable, here X = $(\mathbf{e} \rightarrow \mathbf{t})$ because $\lambda x.\lambda y.love(y, x)$ has type $(\mathbf{e} \rightarrow (\mathbf{e} \rightarrow \mathbf{t}))$

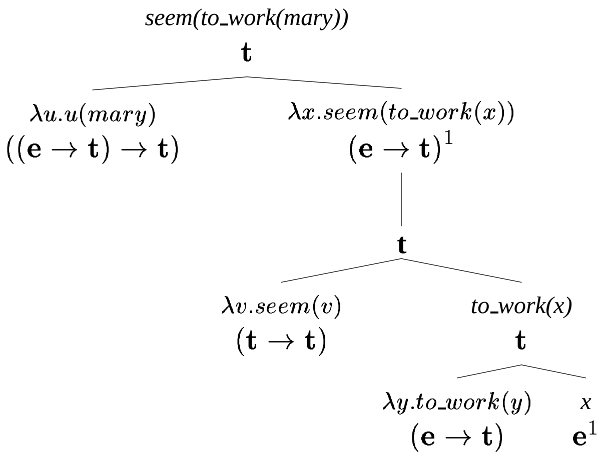
[5]X is a variable of type. This may appear as non-determinism but the instantiation of X is always unique. Moreover, when $\bar{\mathtt{f}}$ is of type $(X \rightarrow X)$, it is in fact endowed with the **identity** function, something which happens everytime $\bar{\mathtt{f}}$ is linked by a chain to a higher node.

Figure 4: Mary seems to work

$$\begin{array}{lll} \mathit{seems} & ::= & \vdash \mathit{seems} : ((\overline{\mathtt{k}}\backslash\mathtt{ip})/\mathtt{vp}) \otimes (\mathtt{vp}/\mathtt{vp}) \\ \mathit{mary} & ::= & \vdash \mathit{mary} : \mathtt{d} \otimes \overline{\mathtt{k}} \\ \mathit{to\ work} & ::= & \vdash \mathit{to\ work} : (\mathtt{d}\backslash\mathtt{vp}) \end{array}$$



With this homomorphism of labels, the transformation of trees consisting in stretching ”intermediary projection nodes” and erasing leaves without semantic content, we obtain from the derivation tree of the second example, the following ”semantic” tree:



where coindexed nodes are linked by the discharging relation.
Let us notice that the characteristic weak or strong of the features may often be encoded in the lexical entries. For instance, Head-movement from `V` to `I` is expressed by the fact that tensed verbs are such that:

- the full phonology is associated with the inflection component,
- the empty phonology and the semantics are associated with the second one,
- the empty semantics occupies the first one[6]

Unfortunately, such rigid assignment does not always work. For instance, for phrasal movement (say of a $\mathtt{d}$ to a $\overline{\mathtt{k}}$) that depends of course on the particular $\overline{\mathtt{k}}$-node in the tree (for instance the situation is not necessary the same for nominative and for accusative case). In such cases, we may assume that **multisets** are associated with lexical entries instead of vectors.

4.3 Reflexives

Let us try now to enrich this lexicon by considering other phenomena, like reflexive pronouns. The assignment for `himself` is given in figure 5 — where the semantical type of `himself` is assumed to be $((e \rightarrow (e \rightarrow t)) \rightarrow (e \rightarrow t))$. We obtain for `paul shaves himself` as the syntactical tree something similar to the tree obtained for our first little example (`peter loves mary`), and the semantic tree is given in figure 5.

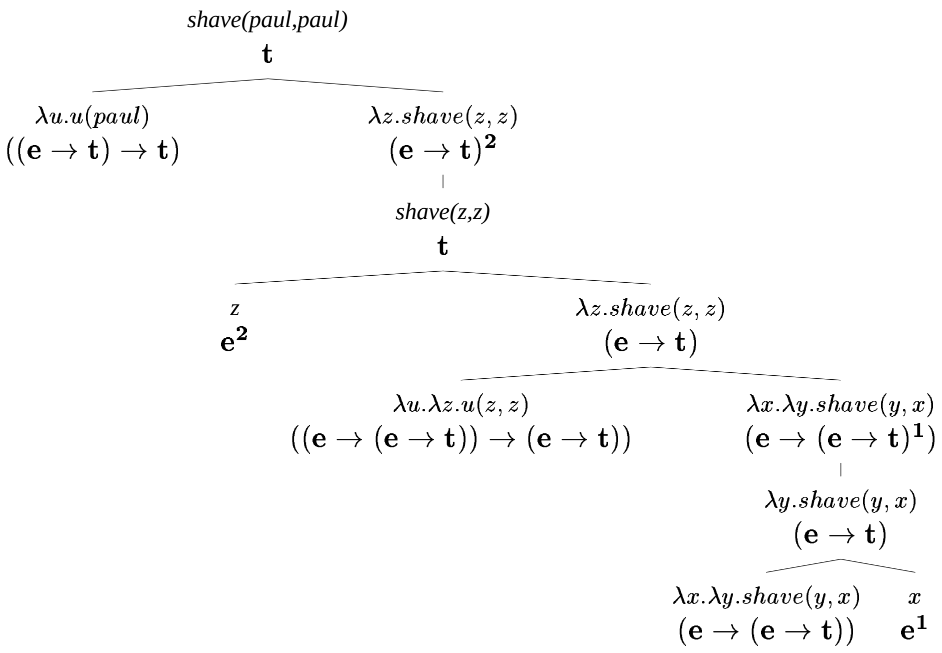
5 Remarks on parsing and learning

In our setting, parsing is reduced to proof search, it is even optimized proof-search: indeed the re-

[6]as long we don't take a semantical representation of tense and aspect in consideration.

Figure 5: Computing a semantic recipe: shave himself

$$shaves \quad ::= \quad [shaves : \emptyset : ((\overline{\mathtt{k}}\backslash\mathtt{ip})/\mathtt{vp})] \otimes [\epsilon : \lambda x.\lambda y.shave(y,x) : ((\overline{\mathtt{k}}\backslash(\mathtt{d}\backslash\mathtt{vp}))/\mathtt{d})]$$
$$himself \quad ::= \quad [\epsilon : \lambda u.\lambda z.u(z,z) : \overline{\mathtt{k}}] \otimes [himself : x : \mathtt{d}]$$



shave(paul,paul)
$\mathbf{t}$
$\lambda u.u(paul)$
$((\mathbf{e} \to \mathbf{t}) \to \mathbf{t})$
$\lambda z.shave(z,z)$
$(\mathbf{e} \to \mathbf{t})^{\mathbf{2}}$
shave(z,z)
$\mathbf{t}$
z
$\mathbf{e}^{\mathbf{2}}$
$\lambda z.shave(z,z)$
$(\mathbf{e} \to \mathbf{t})$
$\lambda u.\lambda z.u(z,z)$
$((\mathbf{e} \to (\mathbf{e} \to \mathbf{t})) \to (\mathbf{e} \to \mathbf{t}))$
$\lambda x.\lambda y.shave(y,x)$
$(\mathbf{e} \to (\mathbf{e} \to \mathbf{t})^{\mathbf{1}})$
$\lambda y.shave(y,x)$
$(\mathbf{e} \to \mathbf{t})$
$\lambda x.\lambda y.shave(y,x)$
$(\mathbf{e} \to (\mathbf{e} \to \mathbf{t}))$
x
$\mathbf{e}^{\mathbf{1}}$

striction on types, and on the structure of proof imposed by the *shortest move* principle and the absence of introduction rules considerably reduce the search space, and yields a polynomial algorithm. Nevertheless this is so when traces are known: otherwise one has to explore the possible places of theses traces.

Here we did focus on the interface with semantics. Another excellent property of categorial grammars is that they allow — especially when there are no introduction rules — for learning algorithms, which are quite efficient when applied to structured data. This kind of algorithm applies here as well when the input of the algorithm are derivations.

6 Conclusion

In this paper, we have tried to bridge a gap between minimalist program and the logical view of categorial grammar. We thus obtained a description of minimalist grammars which is quite formal and allows for a better interface with semantics, and some usual algorithms for parsing and learning.

References

Noam Chomsky. 1995. *The minimalist program*. MIT Press, Cambridge, MA.

Philippe de Groote. 1996. Partially commutative linear logic. In M. Abrusci and C. Casadio, editors, *Third Roma Workshop: Proofs and Linguistics Categories*, pages 199–208. Bologna:CLUEB.

Joachim Lambek. 1958. The mathematics of sentence structure. *American mathematical monthly*, 65:154–169.

Michael Moortgat. 1996. Categorial type logic. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, chapter 2, pages 93–177. North-Holland Elsevier, Amsterdam.

Christian Retoré and Edward Stabler. 1999. Resource logics and minimalist grammars: introduction to the ESSLLI workshop. To appear in *Language and Computation* RR-3780 http://www.inria.fr/RRRT/publications-eng.html.

Edward Stabler. 1997. Derivational minimalism. In Christian Retoré, editor, *LACL'96*, volume 1328 of *LNCS/LNAI*, pages 68–95. Springer-Verlag.