LAMBEK CALCULI WITH **0** AND TEST–FAILURE IN DPL

SEBASTIAN SEQUOIAH-GRAYSON*

Formal Epistemology Project
University of Leuven

Abstract

In Sequoiah-Grayson (2009a), the author gave a basic dynamic semantics the Lambek Calculi, to which a dynamic negation was added. This dynamic negation was interpreted as procedural prohibition, or process exclusion. The resulting framework suggested connections with the analysis of negation as test–failure in the Dynamic Predicate Logic (DPL) due originally to Groenendijk and Stokhof (1991), and developed in van Benthem (1996). The aim of the present article is to explore this connection in detail.

keywords: Lambek, negation, test-failure, DPL.

1 Introduction

The Dynamic Predicate Logic (DPL) of Groenendijk and Stokhof (1991) was instrumental in its understanding of formulas as *programs*. In contrast to the familiar dynamic logic *while–do* style operations on static (propositional or predicate) logical formulas, for which see Harel *et. al.* (2000), DPL places the dynamic operations *inside* the predicate formulas themselves via a dynamic semantics. It is in this sense that formulas acquire the status of programs, or *tests*. Programs are tests in the sense that they test their inputs for "processability", where this processability is a function of the constraints imposed by the program, hence programs test their inputs. In this context, it is natural to interpret negation as test *failure* in the sense that negated formulas are programs that tell you which inputs are ruled out. In other words, they tell you which tests are guaranteed to fail.

Recently, and independently of DPL, Sequoiah-Grayson (2009a) examined a dynamic propositional negation in the context of non–commutative logics,

*Postdoctoral Research Fellow, Centre for Logic and Analytical Philosophy, *University of Leuven* - Belgium. Senior Research Associate, IEG - Computing Laboratory, *University of Oxford*. Research Associate, GPI - *University of Hertfordshire*.

including both the associative and non–associative Lambek Calculi, Lambek (1958) and Lambek (1961) respectively. The dynamics were imposed from inside via an operational semantics, in much the same spirit as in Groenendijk and Stokhof (1991). The resulting negation was a split pair $\langle \sim, \neg \rangle$ on account of commutation–failure generating a double negation pair $\langle \rightarrow, \leftarrow \rangle$ and negation being defined in terms of the implication of bottom, $\mathbf{0}$. Rather than examine any particular logic in detail, the goal there was to examine the dynamic properties that should hold for *any* logic with such a negation pair, given the operational semantics. The central move was to operationalise the ternary relation, and understand the points in the semantic frame as information states. Negation was then understood as the exclusion of particular directional information processing operations.

The conceptual connection between test–failure and process exclusion is too suggestive to ignore. The point of the present article is to explore the extent of this connection in detail. In section 2 the details of Sequoiah-Grayson (2009a) are sketched in enough detail for the basic ideas to be clear. In section 3 the notions of formulas as programs and negation as test–failure from Groenendijk and Stokhof (1991) are explained. The purpose of section 4 is to examine the connections that exist between the two dynamic negations, despite the former system(s) being propositional and the latter being predicate. The key moves are to understand a lexicon as a particular instance of a *database*, as well to note that an information–state based operational semantics has a natural functional interpretation. In section 5, we will explore a range of examples of directional (split) and non–directional negation in terms of the exclusion of particular information processing operations. Some of the databases in the examples are lexical, whilst others involve deductive information processing. The motivation is to respond to a general suspicion of the concrete interpretability of negation in terms of the implication of $\mathbf{0}$ (especially in lexical contexts) such as is expressed in Dosen (1993) and Buszkowski (1996).

2 Lambek Calculi with 0: Process Exclusion

We start with a non-commuting information frame $\mathbf{F} := \langle S, \sqsubseteq, \bullet \rangle$, where S is a set of information states, $\sqsubseteq$ is a partial–order of informational inclusion, or informational development on members of S, and $\bullet$ is a non–commuting combination operator on members of S.

A model $\mathbf{M} := \langle \mathbf{F}, \Vdash \rangle$ is an ordered pair $\mathbf{F}$ $\langle S, \sqsubseteq, \bullet \rangle$ and $\Vdash$, such that $\Vdash$ is an evaluation relation that holds between members of S and formulas constructed out of the binary connectives $\otimes, \rightarrow, \leftarrow$, as well as the constant $\mathbf{0}$. In what follows, we will often write $x, y, \ldots \in \mathbf{F}$ as shorthand for $x, y, \ldots \in S$ where $S \in F$. Where A is a propositional formula, and $x, y, z \in \mathbf{F}$, $\Vdash$ obeys the heredity condition:

$$\text{For all } A, \text{if } x \Vdash A \text{ and } x \sqsubseteq y, \text{then } y \Vdash A, \tag{1}$$

And also obeys the following conditions for each of our connectives:

$$x \Vdash A \otimes B \textit{ iff} \text{ for some } y, z, \in \mathbf{F} \text{ s.t. } y \bullet z \sqsubseteq x, y \Vdash A \text{ and } z \Vdash B. \tag{2}$$

$$x \Vdash A \rightarrow B \textit{ iff} \text{ for all } y, z \in \mathbf{F} \text{ s.t. } x \bullet y \sqsubseteq z, \text{if } y \Vdash A \text{ then } z \Vdash B. \tag{3}$$

$$x \Vdash A \leftarrow B \textit{ iff} \text{ for all } y, z \in \mathbf{F} \text{ s.t. } y \bullet x \sqsubseteq z, \text{if } y \Vdash A \text{ then } z \Vdash B. \tag{4}$$

$$x \Vdash \mathbf{0} \text{ for no } x \in \mathbf{F}. \tag{5}$$

We have the usual residuation conditions:

$$A \otimes B \vdash C \textit{ iff } B \vdash A \rightarrow C \tag{6}$$

$$A \otimes B \vdash C \textit{ iff } A \vdash C \leftarrow B \tag{7}$$

We read the turnstile '$\vdash$' in processing terms, so that '$X \vdash A$' comes out as "processing the information in X generates the information in A". What it is exactly that such processing amounts to depends on the constraints imposed upon the database, with these constraints specified by the presence or absence of particular structural rules, see Gabbay (1996).

We define a split negation pair in terms of double implication:

$$\sim A := A \rightarrow \mathbf{0} \tag{8}$$

$$\neg A := \mathbf{0} \leftarrow A \tag{9}$$

Hence the frame conditions for $\sim A$ and $\neg A$ are cashed out in explicit informational terms as follows:

$$x \Vdash \sim A \, [A \rightarrow \mathbf{0}] \textit{ iff} \text{ for all } y, z \text{ s.t. } x \bullet y \sqsubseteq z, \text{ if } y \Vdash A \text{ then } z \Vdash \mathbf{0} \tag{10}$$

$$x \Vdash \neg A \, [\mathbf{0} \leftarrow A] \textit{ iff} \text{ for all } y, z \text{ s.t. } y \bullet x \sqsubseteq z, \text{ if } y \Vdash A \text{ then } z \Vdash \mathbf{0} \tag{11}$$

We interpret $\sim A$ as the body of information that cannot be applied to bodies of information of type A, and we interpret $\neg A$ as the body of information that cannot have bodies of information of type A applied to it. The interpretation is supported by the model theory; by the information states supporting $\sim A, \neg A$, and A. If x supports $\sim A$ (x carries the information that $\sim A$) and y supports A, x cannot be applied to y. Similarly, if x supports $\neg A$ and y supports A, then y cannot be applied to x. This is not because such an application will cause an explosion of information, but because it does not generate any information.

The frame conditions in informational terms for $\sim A$ laid out in (10) above tell us that there is no information resulting from the application of x to y where $x \Vdash \sim A$ and $y \Vdash A$, since $x \bullet y \sqsubseteq z$ and $z \Vdash \mathbf{0}$, and via (5) $z \Vdash \mathbf{0}$ nowhere. Suppose though that we were to attempt to apply $\sim A$ to A, in other words to attempt $\sim A \otimes A$. In informational terms, the frame conditions for fusion (2) tell us that an information state x will support $\sim A \otimes A$ iff for some information states y and z such that x is an informational development of the application

of the information in y to the information in z, y supports $\sim A$ and z supports A. However, we know from our definition of $\sim A$ in terms of $A \to \mathbf{0}$, that there is no state x such that it supports the application of $\sim A$ to A, this is simply what (10) tells us. Support for the ruling out conditions on $\neg A$ from the frame conditions for $\neg A$ works similarly, and involves only a directional change.

Given the non-gerrymandered nature of the interpretation of split negation in terms of procedural prohibition, we should be able to give a natural interpretation of general proof-theoretic, hence information-processing properties of split negation in such terms. For any split negation, independently of which structural rules are present, the following (12)–(15) hold:

$$\frac{A \vdash B}{\sim B \vdash \sim A} \qquad \frac{A \vdash B}{\neg B \vdash \neg A} \tag{12}$$

$$A \vdash \sim B \textit{ iff } B \vdash \neg A \tag{13}$$

$$A \vdash \neg\sim A \tag{14}$$

$$A \vdash \sim\neg A \tag{15}$$

We can give sensible information processing interpretations of (12)–(15), and full details may be found in Sequoiah-Grayson (2009a).

By the residuation conditions laid out in (6) and (7) above, along with the definition of split negation in terms of arrows, we already know something of the relationship between intensional structure and negation. Since our split negation is defined intensionally from the start, this is as we would expect. We can specify some further relationships between our negation connectives and intensional structure under a procedural interpretation. In (16)–(19) below, the formulas on the right–hand–side denote the prohibited procedures underpinning the sequents on the left–hand–side (again, full details may be found in Sequoiah-Grayson (2009a)):

$$A \to \neg B \vdash B \to \sim A \qquad \longmapsto \qquad A \otimes (B \otimes (A \to \neg B)) \tag{16}$$

$$A \to \sim B \vdash B \to \neg A \qquad \longmapsto \qquad (B \otimes (A \to \sim B)) \otimes A \tag{17}$$

$$A \to \neg B \vdash \sim A \leftarrow B \qquad \longmapsto \qquad A \otimes ((A \to \neg B) \otimes B) \tag{18}$$

$$A \to \sim B \vdash \neg A \leftarrow B \qquad \longmapsto \qquad ((A \to \sim B) \otimes B) \otimes A \tag{19}$$

Take (17) as a running example. Note that with the procedure prohibited by (17), B is applied to the left of the formula on the left hand side of the sequent in (17), whilst$A$ is applied to the right. This is because whilst B occurs of the left hand side of its arrow on the right hand side of (17), A occurs on the right hand side of its arrow via (9) (i.e., the expanded form of the right hand side of (17) is $B \to (\mathbf{0} \leftarrow A)$). Taking X to stand for the structured process $B \otimes (A \to \sim B)$, what we learn from the excluded process extraction from (17) (as well as the others) is that if from X we can we can get the body of information that can

never have bodies of information of type A applied to it, then it is also the case that we cannot apply X to A. In other words, if it is the case that from processing on X we can get the body of information of type $\neg A$, then X itself must be of type $\sim A$. That is, we know the relationship between the types of procedural prohibition:

$$X \vdash \neg A \quad \Rightarrow \quad X : \sim A \tag{20}$$

$$X \vdash \sim A \quad \Rightarrow \quad X : \neg A \tag{21}$$

The moral is this: when we are searching for the excluded process underpinning sequents of conditionalised bodies of information, we do not need to go through the inferential steps (which might well be lengthly). We simply apply the relevant informational antecedents to the left and/or right hand side(s) of the formula on the left hand side of the original sequent until the only thing left on the right hand side is **0**, then we have got it.

Nothing said here with respect to process exclusion depends on the associativity or otherwise of the underlying logic. As such, negation as process exclusion is compatible with either the associative or non-associative versions of the Lambek Calculi, **L** and **NL** respectively. In 4, we will how such a directional process exclusion survives a non-directional transformation via the addition of commutation. In this case, negation as process exclusion is compatible with the permuting versions of **L** and **NL**, **LP** and **NLP** respectively. Before we do this, we need to get clear on negation as test–failure in DPL.

3 Test–Failure in DPL

In DPL, the dynamic interpretation of implication takes implication to have the nature of a *test*. The evaluation conditions for DPL–implication are as follows:

$$[\![A \rightarrow B]\!] = \{\langle x, x\rangle \mid \forall y : \langle x, y\rangle \in [\![A]\!] \Rightarrow \exists z : \langle y, z\rangle \in [\![B]\!]\} \tag{22}$$

'$[\![\ ]\!]$' is simply the function that assigns semantic values. $\langle x, y\rangle$ is an ordered pair of input-output assignments. Taking its cue from the denotational semantic of programming languages, DPL "dynamifies" logic by interpreting formulas as programs. If program X has $\langle x, y\rangle$ in its interpretation, then the execution X in state x has state y as a possible output. In this case, although the formalism is a little baroque, we can see that (22) states that $A \rightarrow B$ successfully inputs x iff for all outputs y of A with respect to this input x, if we input this output y of A to B, then this will successfully result in an output z of B. Otherwise, $A \rightarrow B$ will reject the input x. It is in this sense that implication is analysed as a test by DPL. It should come as no surprise that DPL was inspired by the situation semantics analysis of implications as *constraints*, for which see Barwise (1987).

Why is it that the implications have identical inputs and outputs? That is, why is $A \rightarrow B$ evaluated with respect to $\langle x, x\rangle$, as opposed to $\langle x, y\rangle$ for some y such that $y \neq x$? To see what is going on here, note that the purpose of

DPL is linguistic analysis, in particular to provide a compositional semantics for natural language discourse. Normally, an implication as a whole will not pass on values assigned within the implication itself, to sentences in the future of the discourse. Consider the (slightly adapted) example from Groenendijk and Stokhof (1991, 49):

If a farmer owns a donkey, he vaccinates it. He likes it. (23)

In (23), the pronouns occurring in the second sentence ("he" and "it"), cannot be linked anaphorically to the indefinite terms in the first sentence. Recalling that DPL is a predicate system, the authors conclude that generally, variables outside the implication cannot be bound by quantifiers occurring inside the implication, irrespectively of whether or not they occur in the antecedent or the consequent. In DPL–ease: although implication is *internally dynamic* in the sense captured by (22), it is not *externally dynamic*, in contrast with conjunction. Consider:

A farmer owns a donkey and a tractor. He likes them. (24)

With conjunction, pronouns *can* be linked anaphorically to the indefinite terms in the first sentence. It is in this sense that conjunction is both externally dynamic, as well as internally dynamic. This is marked in DPL via the evaluation of conjunction being carried out with respect to non–identical input–output pairs:

$$[\![A \wedge B]\!] = \langle x, y\rangle |\ \exists z\colon \langle x, z\rangle \in [\![A]\!] \text{ and } \langle z, y\rangle \in [\![B]\!] \tag{25}$$

Negation in DPL is evaluated with respect to *identical* input–output pairs, hence it is internally, but not externally, dynamic in the sense captured by DPL. This is because, similarly to implication but non–similarly to conjunction, negation usually blocks anaphoric links between indefinite terms occurring in its scope, and pronouns occurring outside of it. Consider:

It is not the case that a man is drinking a whisky. He skips. (26)

This is enough, hopefully, to get the general flavour of things, as it is not our brief to examine the compositional analysis of discourse via DPL in any detail. We are concerned principally with the higher level of abstraction at which DPL analyses negation in terms of test–failure. The evaluation conditions for DPL negation are as follows:

$$[\![-A]\!] = \{\langle x, x\rangle |\ -\exists y\colon \langle x, y\rangle \in [\![A]\!]\} \tag{27}$$

$-A$ is interpreted as test–failure in the following sense: $-A$ will return its input x as its output, iff A *cannot* be successfully processed with respect to its input. If A *can* be successfully processed with respect to x as an input, then x is blocked by $-A$. That is, $-A$ is ruling out the processing of A. The input/output x of $-A$ cannot be input to A, as there is no possible output of A from this input x of $-A$. If something can be successfully input to A, i.e., A delivers an output with respect to this input, then, whatever it is, this input will be blocked by $-A$ (blocked from being an input to $-A$).

With DPL style negation as test–failure now out in the open, we can move on to connecting it with negation as process exclusion.

4 Connecting Process Exclusion with Test–Failure

The first step in connecting process exclusion with test–failure involves unifying the notions of information combination and program inputs/outputs. To unify the notions of information combination on the one hand, and program inputs/outputs on the other, we first generalize across left/right composition of (3) and (4) to get:

$$x \Vdash A \multimap B \textit{ iff for all } y, z \in \mathbf{F} \text{ s.t. } x \bullet y \sqsubseteq z, \text{if } y \Vdash A \text{ then } z \Vdash B. \tag{28}$$

In (28), $\bullet$ is no longer directional, so now $x \bullet y = y \bullet z$. Now we define the resulting generalised negation as:

$$A^{\perp} := A \multimap \mathbf{0} \tag{29}$$

That is:

$$x \Vdash A^{\perp} \; [A \multimap \mathbf{0}] \textit{ iff } \text{for each } y, z \text{ s.t. } x \bullet y \sqsubseteq z, \text{ if } y \Vdash A \text{ then } z \Vdash \mathbf{0} \tag{30}$$

Now we are getting somewhere. Recall that $\bullet$ is information–combination, the information–state–operation corresponding to fusion. Functions themselves are information, of a conditional kind. They tell you what information you will get, if you feed them the correct information as an input. In the case of (28) (and (4) and (3)), $\bullet$ combines functional information with input information. The relation of information development, $\sqsubseteq$, has a natural reading as the information output resulting from the combination of functional information with input information. That is, the information states on the left hand side of $\sqsubseteq$ the functional and input information, and the state immediately on the right hand side of $\sqsubseteq$ is the output. The clause $x \bullet y \sqsubseteq z$ is then read *take the functional information x and feed it the input information y, and it outputs the information z.*

Now we are here. (30) states that feeding y to x will *never* result in an output. Why? Simply because the functional information carried by x is $A \multimap \mathbf{0}$, and the input information carried by y is A. In this case, there is no output information carried by z, since $z \Vdash \mathbf{0}$ and $z \Vdash \mathbf{0}$ nowhere via (5). This is just to say that since $y \Vdash A$, y is blocked by x since $x \Vdash A \multimap \mathbf{0}$. In other words, the process of $x \bullet y$ is excluded by $A \multimap \mathbf{0}$, since the result is informationally impossible, as $x \Vdash \mathbf{0}$ nowhere.

Since $\bullet$ is commutative here, or non–directional, information–feed is simply information–combination, as opposed to left/right application/attachement. In the following section, we will explore concrete examples of both directional and non-directional process exclusion. Before this, we need to resolve one remaining issue for connecting process exclusion with test–failure.

DPL is a a predicate system, yet the logics underpinning **L** and **NL** with negation as process exclusion expounded here are propositional. This is a distinction with a difference if anything is. However, at the conceptual level that we are working at, which is negation in a dynamic setting, this is a distinction

without a difference. How so? Note again the DPL interpretation of formulas as programs, or tests. Programs are tests in the sense that they test their inputs for "processability", where this processability is a function of the constraints imposed by the program, hence programs test their inputs. Under such an interpretation, the difference between a predicate and a propositional system is *not* whether or not the relevant formulas can be given program–interpretations. Indeed, such a concern might well prove to be fatal. Rather, all that this difference turns on is the type of information that is taken as an input/output of the formula(s). Or to put the same point differently, the difference turns on what sort of information is tested by the formula.

The variables in DPL are the usual individual variables of first–order logic, hence the input/ouput pairs are individuals. In other words, the information tested is the information that certain individuals possess certain properties. By contrast, the information tested by formulas in propositionalised versions of the Lambek Calculi is propositional information. So, the difference comes down to the type of information being tested, not the plausibility of the information–testing interpretation itself. It is in the sense exactly that the propositional/predicate distinction is, for us, a distinction without a difference. Of course, Lambek Calculi are not *restricted* to propositional interpretations. In the following section, we will see examples of formulas–as–tests in operationalised Lambek Calculi from both propositional and non–propositional perspectives.

In a dynamic setting, be it propositional or predicate, negation will not be understood as the negation of a proposition. In a dynamic setting, it is natural to take negation as the exclusion of a process, or the prohibition of a procedure. In such a setting, instead of telling you what information *can* be taken as an input, negations tell which information can *never* be taken as an input, on account of it never giving you an output. Negations tell you which tests are guaranteed to fail. In the following section, we examine some examples of just this phenomenon.

5 Examples

DPL is concerned with natural language semantics. One of the insights of DPL was to understand a lexicon as, in a sense, a particular type of *database*. We may understand the operationalised Lambek structure in section 2 as a[n independently] generalised extension of this idea, generalised in virtue of there not being any obvious restriction on the type of database under consideration. Given this, we should expect to find a range of examples of process exclusion across database types. This is in fact the case.

Taking natural language as a natural starting point, we can locate quickly cases where some information is of type $\sim A$ but not of type $\neg A$, and others where some information is of type $\neg\, A$ but not of type $\sim A$.

For an example that is of type $\sim A$ but not of type $\neg A$, take the adjective 'happy'. Where nouns are of type n, 'happy' is of type $n \leftarrow n$, since it is

the case that when 'happy' is applied to the left of another noun, the result is another (complex) noun (phrase): 'Happy Friederike'. So we know that that the adjective 'happy' is *not* of type $\neg\, n$, since $\neg\, n$ is just $\mathbf{0} \leftarrow n$. However, 'happy' *is* of type $\sim n$, or $n \rightarrow \mathbf{0}$ (as is standard in the categorical grammar literature, we are allowing strings to be of multiple types). 'happy' is of type $n \rightarrow \mathbf{0}$, because it is the case that when the adjective 'happy' is applied to the *right* of a noun, the result, 'Frederike happy', is simply not well formed. In database terms, 'Frederike happy' cannot be processed, as we cannot get anywhere informationally. In terms of our operational semantics from section 2, in particular the frame conditions laid out for $\sim A$ in (10), there is no information state x carrying the information 'Frederike happy'. In fact there *cannot* be one (for any noun), as to reiterate, 'happy' is of type $n \rightarrow \mathbf{0}$.

In explicit information processing terms, the types corresponding to 'happy' have the following frame conditions:

$$x \Vdash \sim n\ [n \rightarrow \mathbf{0}]\ \textit{iff}\ \text{for all } y, z \text{ s.t. } x \bullet y \sqsubseteq z,\ \text{if } y \Vdash n \text{ then } z \Vdash \mathbf{0} \tag{31}$$

$$x \Vdash n \leftarrow n\ \textit{iff}\ \text{for all } y, z \in \mathbf{F} \text{ s.t. } y \bullet x \sqsubseteq z, \text{if } y \Vdash n \text{ then } z \Vdash n \tag{32}$$

Note that the direction of the information states around $\bullet$ correspond directly with the direction of information application stipulated via the various types. In (31), the type $\sim A$ takes its input, information of type n on the right hand side, a fact marked by the right–pointing arrow. Correspondingly, the information state y carrying the input information n is on the right hand side of $\bullet$. Similarly, in (32), the type $n \leftarrow n$ takes input information of type n on its left hand side, a fact marked by the left–pointing arrow. Correspondingly, the information state y carrying the input information n is on the left hand side of $\bullet$.

For an example that is of type $\neg A$ but not of type $\sim A$, take the intransitive verb 'peddles'. 'peddles' is of type $n \rightarrow s$ (where sentences are of type s), since in is the case that when 'peddles' is applied to the right of a noun, the result is a sentence: 'Frederike peddles'. In this case, we know that 'peddles' is *not* of type $\sim n$, since $\sim n$ is $n \rightarrow \mathbf{0}$. However, 'peddles' *is* of type $\neg A$, or $\mathbf{0} \leftarrow n$. This is because the result of applying 'peddles' to the left of a noun, 'Peddles Frederike', is not well formed. In database terms, 'Peddles Frederike' cannot be processed, as it is not information, at least not in any useful *processable* sense. It terms of our operational semantics, and in particular the frame condition in (11) in section 2, there is no information state x carrying the information 'Peddles Frederike'. Here too, there cannot be such a state, as to reiterate, 'peddles' is of type $\mathbf{0} \leftarrow n$.

The types corresponding to 'peddles' have the following frame conditions:

$$x \Vdash \neg\, n\ [\mathbf{0} \leftarrow n]\ \textit{iff}\ \text{for all } y, z \text{ s.t. } y \bullet x \sqsubseteq z,\ \text{if } y \Vdash n \text{ then } z \Vdash \mathbf{0} \tag{33}$$

$$x \Vdash n \rightarrow s\ \textit{iff}\ \text{for all } y, z \in \mathbf{F} \text{ s.t. } x \bullet y \sqsubseteq z, \text{if } y \Vdash n \text{ then } z \Vdash s \tag{34}$$

Similarly to (31) and (32) above, the direction of the information states around $\bullet$ in (33) and (34) correspond directly with the direction of information application

stipulated via the various types. In (33), the type $x \Vdash \neg\, n$ takes its input, information of type n, on the left hand side, a fact marked by the left–pointing arrow. Correspondingly, the information state y carrying the information n is on the left hand side of $\bullet$. Similarly, in (34), the type $x \Vdash n \rightarrow s$ takes input information of type n on its right hand side, a fact marked by the right–pointing arrow. Correspondingly, the information state y carrying the input information n is on the right hand side of $\bullet$. Of course all of these facts concerning left/right correspondence around $\bullet$ are there to see in the abstract frame conditions in section 2, however they are much easier to appreciate in the context of concrete examples. Such left/right correspondence drops away in the context of non–directional applications.

For a non-direcitonal example, we move from natural language semantics databases (i.e. natural language lexicons), and briefly consider databases structured for deductive information processing. If we set things up so that $\phi \Rightarrow \psi : A$, and $\sigma : B$, it will be the case that $\phi \Rightarrow \psi : B^{\perp}$, that is $\phi \Rightarrow \psi : B \multimap \mathbf{0}$. This is because there is no information state x carrying the result of combining the information A with the information B. Such a procedure is informationally redundant. We forgo laying out the frame conditions for this example, as they should be obvious by this stage. In this example, the database may be understood as type of "cognitive lexicon" for agent–based deductive reasoning (see Sequoiah-Grayson (2010) for a full development).

For significant fragments of such reasoning, commutation will be harmless, despite association destroying the "well–formed" nature of the inferential sentences. In such cases we will be working with a permutative but nonassociative Lambek Calculus, **NLP**, or the associating version, **LP**, see Moortgat (1995). It is the presence or absence of commutation, or permutation, that makes the difference for our negation, and hence for the properties of our process exclusion. Directional cases for deductive information processing are found in interpreted inferential settings where fully sequential reasoning (*the agent opens the fridge, and the agent retrieves a beer* etc.) is concerned, see Sequoiah-Grayson (2009b).

The point here is not to delve deeply into the any particular application from which such examples are drawn. Instead, the point is to draw attention to the common factor across a diverse range of cases. In all of the examples above, the factor in common is that particular processes are being excluded. The greater point here, is the rich array of applications that the addition of $\mathbf{0}$ to various versions of the Lambek Calculi has in information processing contexts. Such applications are clearly not restricted to natural language semantics, although the canonically natural language focused DPL was a significant proto–manifestation of such an approach.

6 Conclusion

The DPL understanding of formulas as programs has a longer reach than the original authors perhaps realised. DPL was primarily concerned with discourse analysis. A setting such as a discourse imposes quickly its dynamic properties

upon any putative analysis, as a discourse is a dynamic phenomenon if anything is. In an informational setting, a move to dynamic analysis means a move from an analysis of bodies of information, to the analysis of the manipulation and generation, or *processing of* such bodies of information. Taking the information processing perspective a step further, we can understand discourse as distributed information processing, with the agents in the discourse acting as the processing nodes in the distributed computation itself. As noted by the designers of DPL, "The utterance of a sentence brings us from a certain state of information to another one", Groenendijk and Stokhof (1991).

Of course, acts of reasoning by a single agent are information processing operations themselves. It is in precisely this sense that an information processing perspective returns from (massively–) multi–agent settings to mono–agent ones. It is unsurprising that resources from the Lambek Calculi, under a suitable operational interpretation, offer such a rich starting point for the dynamic aspects of mono–agent reasoning. Reasoning to oneself is not unlike talking to or communicating with oneself after all.

In a classical static propositional setting, propositional falsity is important, since we need to be able to represent truth failure. In a dynamic setting, process exclusion is equally important, as we need to be able to represent process failure. Such dynamic settings are not restricted to simple single or multi–agent communicative settings. The advent of information communications technologies, in particular the Web, have dramatically altered the way that we communicate with (transmit and send information to and from) each other, as well as the way in which we store such information. Such phenomena are an increasingly important emerging aspect of the natural world in which we operate. In order to analyse and understand the computational properties of such phenomena, we require increasingly sophisticated models of information processing. Along with these, we will require increasingly sophisticated models of process exclusion.[1]

References

Barwise, J. (1987): Noun Phrases, Generalized Quantifiers and Anaphora, in P. Gardenfors (ed.): *Generalized Quantifiers*, Reidel, Dordrecht, pp. 1–29.

van Benthem, J. (1996): *Exploring Logical Dynamics* CSLI Publications, Stanford, and FoLLI.

Buszkowski, W. (1995): Categorical Grammars with Negative Information, in H. Wansing (ed.): *Negation, A Notion in Focus*, Gruyter, Berlin, pp. 107–126.

Dosen, K. (1993): A Historical Introduction to Substructural Logics, in P. Schroeder-Heister and K. Dosen (eds.), *Substructural Logics, Studies in Logic*

[1]I am indebted to Johan van Benthem for informing me of the process exclusion similarities apparent in the test–failure of DPL. I would also like the thank David Willingham at *Linguistic Analysis* for his endless logistic management and assistance. Most of all, I would like to thank Jim Lambek, who has given all of us so very much.

and Computation no. 2, Oxford Science Publications, Clarendon Press, Oxford: 1–30.

Gabbay, D. M., (1996): *Labelled Deductive Systems*, (Vol. 1) Oxford Logic Guides No. 33, Clarendon, Oxford, Oxford University Press.

Groenendijk, J., and Stokof, M. (1991): Dynamic Predicate Logic, *Linguistics and Philosophy* **14**: 33–100.

Harel, D., Kozen, D, Tiuryn, J. (2000): *Dynamic Logic*, MIT Press.

Lambek, J. (1958): The Mathematics of Sentence Structure, *American Mathematical Monthly*, 65, 154–170.

Lambek, J. (1961): On the Calculus of Syntactic Types, in R. Jakobson (ed.): *Structure of Language and its Mathematical Aspects*, American Mathematical Society, Providence, 166–178.

Moortgat, M. (1995): Residuation in Mixed Lambek Systems, Research Transcript no. 10, Research Institute for Language and Speech (OTS), Utrect.

Sequoiah-Grayson, S. (2010): Epistemic Closure and Weakly–Commuting Interaction Models, forthcoming in *Synthese*.

Sequoiah-Grayson, S. (2009a): Dynamic Negation and Negative Information, *Review of Symbolic Logic*: 2.1, 233–248.

Sequoiah-Grayson, S. (2009b): A Positive Information Logic for Inferential Information, *Synthese*, **167**: 2, pp. 409–431.