Sense and the Computation of Reference

Reinhard Muskens (r.a.muskens@kub.nl)
Department of Linguistics
Tilburg University
P.O. Box 90153
5000 LE Tilburg, The Netherlands

Abstract. The paper shows how ideas that explain the sense of an expression as a method or algorithm for finding its reference, preshadowed in Frege's dictum that sense is the way in which a referent is given, can be formalized on the basis of the ideas in Thomason (1980). To this end, the function that sends propositions to truth values or sets of possible worlds in Thomason (1980) must be replaced by a relation and the meaning postulates governing the behaviour of this relation must be given in the form of a *logic program.* The resulting system does not only throw light on the properties of sense and their relation to computation, but also shows circular behaviour if some ingredients of the Liar Paradox are added. The connection is natural, as algorithms can be inherently circular and the Liar is explained as expressing one of those. Many ideas in the present paper are closely related to those in Moschovakis (1994), but receive a considerably lighter formalization.

1. Introduction

In this paper I will pursue the Fregean idea that the sense of an expression essentially is a method or algorithm to get at its reference. I will argue that this idea can be formalized in a simple way and that an existing account of linguistic semantics (the one in Thomason, 1980) in fact already goes halfway in capturing it, although it presumably was never intended to do so. The view has repercussions in at least two areas of foundational difficulty, as it sheds light on problems of intensionality, but also on the question of circular propositions such as the famous Liar Paradox. That algorithms can be inherently circular and their execution diverging explains why the Liar arises Moschovakis (1994) and the reason that the move also illuminates problems of intensionality is that once we assume that senses are recipes for finding referents it is not only easy to see how senses that lead to the same referent can be different, but we also actually get some grip on their identity criteria. If senses are a certain kind of algorithms, then two senses are identical if the corresponding algorithms are. While identity of algorithms itself is a non-trivial problem, this at least gives something to start with.

The idea that senses are procedures that can be used to compute reference is an old one, attributed to Frege in Dummett (1978) and Frege's famous explanation of sense as the *Art des Gegebenseins* of a referent (the way the referent is given[1]) certainly can be read as expressing something closely akin to this point of view. The idea was provided with extensive philosophical justification in Tichý (1988),[2] where it lies at the heart of a system of intensional logic. It received its most rigorous formalization in Moschovakis (1994), where a system called the *Lower Predicate Calculus with Reflection* (LPCR) is used to capture it. Conceptually, LPCR is based on Kripke's formalism to tackle the Liar (Kripke, 1975) and the system is obtained by adding a form of recursion to first-order logic.[3] The result is rather heavy artillery and van Lambalgen and Hamm (2003) propose to study the idea of senses-as-algorithms in an alternative context that likewise combines the declarativity of semantics with a form of procedurality: the paradigm of *logic programming*. In logic programming existing logics get a procedural interpretation and from the point of view of linguistic theory Van Lambalgen and Hamm's move has the great advantage that procedural aspects of semantics can be studied without the need of additional logical overhead. I will follow this move of taking a logic programming perspective on natural language semantics, but while Van Lambalgen and Hamm apply their insights to the area of lexical semantics, in particular the semantics of temporal operators, I will put them to use in the compositional semantics of phrases, with intensionality and Liar-like phenomena remaining the main focus of interest.

The starting point of this paper will be the Intentional Logic of Thomason (1980), which is essentially a Montague-like system with an extra type p for *propositions*, which are thus treated as objects in their own right, not constructed from other objects such as possible worlds. Natural language expressions in this system are rendered as terms of type p or terms of types derived from p and there is a function (denoted with $^{\cup}$) sending propositions to their extensions. The behaviour of this function is constrained by meaning postulates and this is where the connection with logic programming will be made. I will argue that, after some slight alterations of Thomason's system, these meaning postulates can not only be cast rather naturally in the form of a logic program, but also that, if this is done, they provide an axiomatization that allows interpreting the elements of the type p domain as consisting of certain algorithms. In particular, type p objects definable in the system can naturally be associated with certain *queries*.

[1] Sometimes translated as 'mode of presentation', but the present more pedestrian translation seems closer.

[2] I take it that Tichý's notion of senses as *constructions* esentially captures the same idea.

[3] Moschovakis (2003) gives a highly interesting higher-order Montague-like system, which unlike LPCR cannot capture Liar-like phenomena however.

The set-up of the remainder of the paper is as follows. In the next section we briefly review Thomason's Intentional Logic; we will get rid of some minor redundancies in section 3 and give a streamlined version that also contains possible worlds in section 4. After a short comparison of the theory with other approaches to hyperintensionality (section 5) this version is then altered in section 6 by replacing the function sending propositions to their associated sets of possible worlds (two notions that should be distinguished on the present account) by a family of relations and by replacing the meaning postulates governing the behaviour of the original function by meaning postulates for these relations. The latter form a logic program and it is shown how finding the set of worlds associated with a given proposition corresponds to the execution of a certain query. In section 7 we consider circular propositions and show how the addition of a rudimentary system of anaphoric reference and a trivial notion of truth leads to divergence (infinite computation) in certain cases, while it works in a perfectly straightforward manner in other, 'innocent', ones. A brief conclusion ends the paper.

2. Thomason's Intentional Logic

Thomason (1980) gives a simple and elegant theory of intentionality[4] which follows Montague (1973) in translating a fragment of natural language into a version of type logic (Church, 1940). Intentionality is obtained by introducing a special basic type p for *propositions* (which are conceived of as primitive entities). The usual logical constants $\neg$, $\wedge$, and $\rightarrow$ are mirrored by a set of newly introduced ones $\sim_{pp}$, $\cap_{p(pp)}$, and $\supset_{p(pp)}$. Thus if Φ and Ψ are propositions, then $\sim\Phi$, $\cap\Phi\Psi$, and $\supset\Phi\Psi$ are too, the last two typically written as $\Phi\cap\Psi$ and $\Phi\supset\Psi$. Similarly $\forall$, $\exists$ and $=$ are mirrored by logical constants $\bigcap_{(\alpha p)p}$, $\bigcup_{(\alpha p)p}$, and $\approx_{\alpha(\alpha p)}$, for each type α. With some notational sugaring this means that we now also have proposition denoting terms of the forms $\bigcap X_\alpha\Phi$, $\bigcup X\Phi$, and $A_\alpha \approx B_\alpha$, mirroring the usual quantificational and identity statements. In short, the whole usual logical apparatus, which, loosely speaking,

[4] From this point on I will follow Thomason in using the word 'intention' instead of 'intension' in a context where senses can be distinguished with much finer grain as is possible in their usual possible worlds treatment. The notion of intentionality has a long and respectable history, whose modern part is summed up as follows by Tichý (1988, pp. 144–145):

> In modern literature the notion first appears in Brentano's thesis of intentional inexistence, and can then be traced through Twardowski's notion of content and Meinong's notion of pure object to its clearest manifestation in Frege's notion of mode of presentation.

Table I. Typographical convention. Variables of the form indicated will have the associated type. The convention will hold throughout this paper.

Variables	x, y, z	τ	i, j	P	π	$\mathcal{P}$
Types	e	st	s	$e(st)$	p	ep

essentially operates at the level of truth values (type t), is mirrored at the level of propositions (type p).

Having set up these operations on the propositional level, Thomason introduces a function $^{\cup}$ of type pt and imposes the following homomorphism constraints. (Here π and π' are variables of type p and x is of type e. See also the convention in Table I.)

(1) a. $\forall\pi[^{\cup}{\sim}\pi = \neg{}^{\cup}\pi]$
b. $\forall\pi\forall\pi'[^{\cup}[\pi \cap \pi'] = {}^{\cup}\pi \wedge {}^{\cup}\pi']$
c. $\forall\pi\forall\pi'[^{\cup}[\pi \supset \pi'] = {}^{\cup}\pi \rightarrow {}^{\cup}\pi']$
d. $^{\cup}\bigcap x\Phi = \forall x{}^{\cup}\Phi$
e. $^{\cup}\bigcup x\Phi = \exists x{}^{\cup}\Phi$
f. $^{\cup}[A \approx B] = [^{\cup}A = {}^{\cup}B]$

The idea is that if Φ is a proposition, then $^{\cup}\Phi$ is its reference. Note that while these meaning postulates connect the p and t domains, they leave open the possibility that the identity relation on the first of these domains has a much finer grain than the identity relation on the second. For example, we have that $^{\cup}{\sim}\bigcap x{\sim}\Phi = \neg\forall x\neg{}^{\cup}\Phi = \exists x{}^{\cup}\Phi = {}^{\cup}\bigcup x\Phi$. But there will be many models in which ${\sim}\bigcap x{\sim}\Phi = \bigcup x\,\Phi$ fails to hold. In general, while $^{\cup}$ is required to be a homomorphism, it need not be an isomorphism.

A natural language such as English can now be provided with a semantics by associating each sentence with a proposition. For example, one could associate common nouns and intransitive verbs such as 'woodchuck', 'groundhog', 'unicorn', and 'walk' with translations woodchuck, groundhog, unicorn, walk, etc. of type ep[5] and determiners such as 'all' and 'a' with translations all and a of type $(ep)((ep)p)$. Then 'a woodchuck is walking' would be associated with the type p

[5] I am presenting a streamlined version of Thomason's theory here and will deviate in non-essential ways from the original. For example, the translation of the intransitive verb 'walk' in Thomason (1980) is is a constant **walk**$'$ of type $((ep)p)p$, which is then systematically related to a constant **walk**† of type ep by a meaning postulate of the type shifting variety. My walk corresponds to the latter. A similar remark can be made about the translation of 'believes' below.

term ((a woodchuck)walk) and 'all woodchucks are groundhogs' should be translated as ((all woodchuck)groundhog).[6]

In order for this to work and to connect such translations to their usual truth values, we must impose some requirements, as in (2).

(2) a. $\forall x[{}^{\cup}(\mathsf{woodchuck}\ x) = \mathit{woodchuck}_{et}\ x]$,
and similar for groundhog, unicorn, walk, ...
b. $\forall \mathcal{P}'\forall\mathcal{P}[\mathsf{all}\,\mathcal{P}'\mathcal{P} = \bigcap x[\mathcal{P}'x \supset \mathcal{P}x]]$
c. $\forall \mathcal{P}'\forall\mathcal{P}[\mathsf{a}\,\mathcal{P}'\mathcal{P} = \bigcup x[\mathcal{P}'x \cap \mathcal{P}x]]$

The first of these requirements associates the translations that were introduced for common nouns and intransitive verbs with their usual extensions in type *et*.[7] The second two connect the translations of 'all' and 'a' with the newly introduced constants $\bigcap$ and $\bigcup$. The $^{\cup}$ function will send these to $\forall$ and $\exists$.

Again, that what is identified at the level of truth values may be kept distinct at the level of propositions. For example, we may well want to require (3a), which in view of (2a) is equivalent to (3b). But this requirement is fully compatible with (3c). Although all woodchucks are groundhogs and vice versa, the *sense* of '*a* is a woodchuck' may be distinct from the *sense* of '*a* is a groundhog' for some *a*. Indeed, it may consistently be assumed that this is the case for all *a*.[8]

(3) a. $\forall x[{}^{\cup}(\mathsf{woodchuck}\ x) = {}^{\cup}(\mathsf{groundhog}\ x)]$
b. $\forall x[\mathit{woodchuck}\,x = \mathit{groundhog}\,x]$
c. $\neg\forall x[\mathsf{woodchuck}\ x = \mathsf{groundhog}\ x]$

With the meaning postulates in (1) and (2) in place, translations in our mini-fragment of English are connected to their truth conditions via the $^{\cup}$ function. For example, the following short derivation shows that the proposition ((all woodchuck)groundhog) is sent to the usual truth conditions.

(4)
$$\begin{aligned} {}^{\cup}((\mathsf{all\ woodchuck})\mathsf{groundhog}) &= (2b)\\ {}^{\cup}\textstyle\bigcap x[\mathsf{woodchuck}\ x \supset \mathsf{groundhog}\ x] &= (1d)\\ \forall x^{\cup}[\mathsf{woodchuck}\ x \supset \mathsf{groundhog}\ x] &= (1c)\\ \forall x[{}^{\cup}(\mathsf{woodchuck}\ x) \to {}^{\cup}(\mathsf{groundhog}\ x)] &= (2a)\\ \forall x[\mathit{woodchuck}\,x \to \mathit{groundhog}\,x] & \end{aligned}$$

[6] The result of applying A to B is written as (AB). Parentheses may be omitted on the understanding that association is to the left; e.g. ABC is $((AB)C)$.

[7] Note that (2a) can be read as an abbreviatory convention that allows us to write *woodchuck* for $\lambda x.{}^{\cup}(\mathsf{woodchuck}\ x)$. In other words, (2a) constrains the class of models only in an inessential way.

[8] Thomason (1980) provides a model in which both (3a) and $\forall x\neg[\mathsf{woodchuck}\ x = \mathsf{groundhog}\ x]$ hold.

As long as only *extensional* constructions are considered, translations will not differ from those in the usual set-up. But as soon as intentional constructs are taken into account, things begin to change. Consider, for example, the translation $\mathsf{believe}$ of 'believes'. We may take this translation to be of type $p(ep)$, as it combines with a proposition and a subject to form a proposition. The sentence 'John believes that a unicorn walks' could, on the reading where 'a unicorn' takes wide scope, be translated as $((\mathsf{a\ unicorn})\lambda x(\mathsf{believe}(\mathsf{walk}\,x)\mathit{john}))$. The following computation gives the truth conditions.

(5)
$$\begin{array}{rcl}
{}^{\cup}((\mathsf{a\ unicorn})\lambda x(\mathsf{believe}(\mathsf{walk}\ x)\mathit{john})) & = & (2c)\\
{}^{\cup}\bigcup x[\mathsf{unicorn}\ x \cap (\mathsf{believe}(\mathsf{walk}\ x)\mathit{john})] & = & (1e)\\
\exists x^{\cup}[\mathsf{unicorn}\ x \cap (\mathsf{believe}(\mathsf{walk}\ x)\mathit{john})] & = & (1b)\\
\exists x[{}^{\cup}(\mathsf{unicorn}\ x) \wedge {}^{\cup}(\mathsf{believe}(\mathsf{walk}\ x)\mathit{john})] & = & (2a)\\
\exists x[\mathit{unicorn}\ x \wedge {}^{\cup}(\mathsf{believe}(\mathsf{walk}\ x)\mathit{john})] & &
\end{array}$$

This may be reduced a bit further if the postulate in (6a) is adopted (where *believe* is a constant of type $p(et)$). We then arrive at (6b).

(6) a. $\forall\pi\forall x[{}^{\cup}(\mathsf{believe}\,\pi x) = \mathit{believe}\,\pi x]$
b. $\exists x[\mathit{unicorn}\,x \wedge \mathit{believe}(\mathsf{walk}\,x)\mathit{john}]$

But here the process stops and, crucially, there is no reduction of the embedded proposition $\mathsf{walk}\,x$ to its denotation. This is as it should be, as belief is a relation to the sense of an expression, not to its reference. Consider $((\mathsf{all\ woodchuck})\mathsf{woodchuck})$ and $((\mathsf{all\ woodchuck})\mathsf{groundhog})$. In the presence of (3a) these will get equivalent denotations (${}^{\cup}$ sends both to truth). Still, they may denote different propositions. As a consequence (7a) and (7b) are not equivalent. In particular, the first statement can be true while the second is false, even with (3a) in force.

(7) a. $\mathit{believe}\,((\mathsf{all\ woodchuck})\mathsf{woodchuck})\,\mathit{john}$
b. $\mathit{believe}\,((\mathsf{all\ woodchuck})\mathsf{groundhog})\,\mathit{john}$

3. Ironing out a Redundancy

Thomason's treatment of intentionality is so simple that the introductory paragraphs of Thomason (1980) even express a fear that its simplicity may be mistaken for triviality. Nevertheless, the theory can be simplified a bit further, as it turns out that the non-standard logical constants $\sim$, $\cap$, $\supset$, $\bigcap$, $\bigcup$, and $\approx$ can be gotten rid of. A closer look at the workings of the theory shows that they are superfluous. The essential function of Thomason's meaning postulates is to connect the sense of an expression with its denotation. This succeeds, but the non-standard logical constants only play an intermediate role in this. With the exception of (2a), the meaning postulates in (1) and (2) can be given up in favour of the more direct ones in (8).

(8) a. $\forall\mathcal{P}'\forall\mathcal{P}[{}^{\cup}(\mathsf{all}\,\mathcal{P}'\mathcal{P}) = \forall x[{}^{\cup}(\mathcal{P}'x) \rightarrow {}^{\cup}(\mathcal{P}x)]]$

b. $\forall\mathcal{P}'\forall\mathcal{P}[{}^{\cup}(\mathsf{a}\,\mathcal{P}'\mathcal{P}) = \exists x[{}^{\cup}(\mathcal{P}'x) \wedge {}^{\cup}(\mathcal{P}x)]]$

This short-circuits the theory in a way that will not effect the sense-reference function. The derivation (4), for example, can now be replaced by the slightly shorter (9).

(9)
$$\begin{aligned} {}^{\cup}((\mathsf{all\ woodchuck})\mathsf{groundhog}) &= (8a)\\ \forall x[{}^{\cup}(\mathsf{woodchuck}\ x) \rightarrow {}^{\cup}(\mathsf{groundhog}\ x)] &= (2a)\\ \forall x[\mathit{woodchuck}\ x \rightarrow \mathit{groundhog}\ x] \end{aligned}$$

As far as I can see, such a streamlining has no negative impact upon Thomason's theory and in any case the streamlined theory will do as a starting point for this paper.

There is an advantage to this that goes beyond simplification of an already simple theory. The logical constants that were dispensed with had a property that is not normally associated with *logical* constants: their interpretations could vary with each model. One of the usual criteria that come with the notion of logicality is that the interpretation of a logical constant in a given model can only depend on the domain(s) of that model. In Intentional Logic this is not so. For example, it is possible to have models M, M', based on the same frame (i.e. with all domains identical), such that, for some Φ, Ψ, $[\![\Phi \cap \Psi]\!]^M \neq [\![\Phi \cap \Psi]\!]^{M'}$, even while $[\![\Phi]\!]^M = [\![\Phi]\!]^{M'}$ and $[\![\Psi]\!]^M = [\![\Psi]\!]^{M'}$. Doing away with $\sim$, $\cap$, $\supset$, $\bigcap$, $\bigcup$, and $\approx$ rids us of a set of logical constants with a decidedly non-logical behaviour. We still have ${}^{\cup}$, but since considerations about logicality similar to those above hold for this constant as well, we will change it into a non-logical constant, receiving its interpretation from the usual interpretation function.

4. A Larger Fragment with Possible Worlds

All special logical constants of Intentional Logic have at this point either been dispensed with, or, in the case of ${}^{\cup}$, been re-interpreted as ordinary non-logical constants. Technically this boils down to a move from a specialized type logic to a standard one. In particular, we can use the three-sorted Church-Henkin type logic $\mathbf{TY}_3$ in this paper. Ground types will be e, t, s, and p, for entities, truth-values, possible worlds, and propositions respectively. We have assumed that the range of the function $^\cup$ is the set of truth values and this has given a purely extensional logic on formulas $^\cup\Phi$, but, as was already observed in Thomason's paper, an attractive alternative is to also consider possible worlds. This will allow us to combine intentionality with the advantages of possible worlds semantics. Accordingly, we will now replace $^\cup$ with a non-logical constant r of type $p(st)$, which sends propositions to sets of possible worlds.[9]

In Table II some non-logical constants are given of which the ones in sans serif will be used to directly translate words of English and the ones in *italic* will play a role in our meaning postulates.[10] This means that the expressions in (10), for example, are of type p.

(10) a. $((\mathsf{a\ woman})\mathsf{walk})$
b. $((\mathsf{no\ man})\mathsf{talk})$
c. $(\mathsf{hesperus}\ \lambda x((\mathsf{a\ planet})(\mathsf{is}\ x)))$
d. $((\mathsf{if}((\mathsf{a\ woman})\mathsf{walk}))((\mathsf{no\ man})\mathsf{talk}))$
e. $((\mathsf{if}((\mathsf{a\ man})\mathsf{talk}))((\mathsf{no\ woman})\mathsf{walk}))$
f. $(\mathsf{mary}(\mathsf{aware}((\mathsf{if}((\mathsf{a\ woman})\mathsf{walk}))((\mathsf{no\ man})\mathsf{talk}))))$
g. $(\mathsf{mary}(\mathsf{aware}((\mathsf{if}((\mathsf{a\ man})\mathsf{talk}))((\mathsf{no\ woman})\mathsf{walk}))))$
h. $((\mathsf{a\ woman})\lambda x(\mathsf{mary}(\mathsf{aware}((\mathsf{if}(\mathsf{walk}\ x))((\mathsf{no\ man})\mathsf{talk})))))$
i. $(\mathsf{mary}\ \lambda x((\mathsf{try}(\mathsf{run}\ x))\ x))$

It should be emphasized that these expressions contain no logical constants (unless λ is regarded as such). The constants some, no etc. are non-logical, but will be connected to their expected logical interpretation shortly. Also, although there is a clear and intended analogy with natural language, these are logical terms, amenable to logical manipulation. Let us write $\mathcal{T}_{\mathrm{LF}}$ for the smallest set of terms that a) contains all variables of type e, b) contains all constants for which we have used sans serif type in Table II, and c) is closed under application and abstraction over variables of type e. The examples in (10) are closed $\mathcal{T}_{\mathrm{LF}}$ terms of type p. Note that closed $\mathcal{T}_{\mathrm{LF}}$ terms are virtually

[9] Clearly, on the present account propositions cannot be identified with sets of possible worlds (although they still *determine* sets of possible worlds).

[10] The type $s(st)$ constant *acc* in Table II will be used in the meaning postulates for necessarily and possibly. Its denotation plays the role of the usual accessibility relation. Additional meaning postulates may further constrain the behaviour of *acc*, requiring reflexivity, transitivity, etc. Details are left to the reader.

Table II. Some non-logical constants

Non-logical Constants	Type	Non-logical Constants	Type
not	pp	*hesperus, mary, . . .*	e
and, or, if	$p(pp)$	*love, kiss, . . .*	$e(e(st))$
every, a, no, the	$(ep)((ep)p)$	*planet, man, run, . . .*	$e(st)$
is, love, kiss, . . .	$e(ep)$	*acc*	$s(st)$
hesperus, mary, . . .	$(ep)p$	*try, wish*	$p(e(st))$
planet, man, run, . . .	ep	*believe, know, aware*	$p(e(st))$
necessarily, possibly	pp		
try, wish	$p(ep)$		
believe, know, aware	$p(ep)$		

identical to the 'Logical Forms' we find in generative syntax, whence the subscript.

The following are meaning postulates connecting the p and st levels (initial universal quantifiers are suppressed, so free variables get a universal interpretation).

(11) a. $r(\mathsf{not}\ \pi) = \lambda i.\neg r\pi i$
b. $r(\mathsf{and}\ \pi\pi') = \lambda i.r\pi i \wedge r\pi' i$
c. $r(\mathsf{or}\ \pi\pi') = \lambda i.r\pi i \vee r\pi' i$
d. $r(\mathsf{if}\ \pi\pi') = \lambda i.r\pi i \rightarrow r\pi' i$
e. $r(\mathsf{every}\ \mathcal{P}'\mathcal{P}) = \lambda i.\forall x[r(\mathcal{P}'x)i \rightarrow r(\mathcal{P}x)i]$
f. $r(\mathsf{a}\ \mathcal{P}'\mathcal{P}) = \lambda i.\exists x[r(\mathcal{P}'x)i \wedge r(\mathcal{P}x)i]$
g. $r(\mathsf{no}\ \mathcal{P}'\mathcal{P}) = \lambda i.\neg\exists x[r(\mathcal{P}'x)i \wedge r(\mathcal{P}x)i]$
h. $r(\mathsf{necessarily}\ \pi) = \lambda i.\forall j[acc\ ij \rightarrow r\pi j]$
i. $r(\mathsf{possibly}\ \pi) = \lambda i.\exists j[acc\ ij \wedge r\pi j]$
j. $r(\mathsf{mary}\ \mathcal{P}) = r(\mathcal{P}\ mary)$ (and similarly for hesperus etc.)
k. $r(\mathsf{is}\ xy) = \lambda i.(x = y)$
l. $r(\mathsf{love}\ xy) = love\ xy$ (similarly for kiss, . . .)
m. $r(\mathsf{planet}\ x) = planet\ x$ (similarly for man, woman, . . .)
n. $r(\mathsf{believe}\ \pi x) = believe\,\pi x$ (similarly for try, wish, know, aware)

Again, these postulates can be used to compute the truth conditions associated with proposition denoting terms. In (12), for example, it is shown how (10d) unfolds.

(12)
$$\begin{array}{rcl}
r((\mathsf{if}((\mathsf{a\ woman})\mathsf{walk}))((\mathsf{no\ man})\mathsf{talk})) & = & (11d)\\
\lambda i.r((\mathsf{a\ woman})\mathsf{walk})i \to r((\mathsf{no\ man})\mathsf{talk})i & = & (11f)\\
\lambda i.\exists x[r(\mathsf{woman}\ x)i \wedge r(\mathsf{walk}\ x)i] \to r((\mathsf{no\ man})\mathsf{talk})i & = & (11g)\\
\lambda i.\exists x[r(\mathsf{woman}\ x)i \wedge r(\mathsf{walk}\ x)i] \to \neg\exists x[r(\mathsf{man}\ x)i \wedge r(\mathsf{talk}\ x)i] & = & (11m)\\
\lambda i.\exists x[\mathit{woman}\ xi \wedge \mathit{walk}\ xi] \to \neg\exists x[\mathit{man}\ xi \wedge \mathit{talk}\ xi] & &
\end{array}$$

Let w be some fixed constant of type s (which we can think of as denoting the actual world). Type p terms $A_1, \ldots, A_n$ will be said to *entail* a type p term B if

$$(11),\ rA_1w, \ldots, rA_nw \models rBw\ ,$$

i.e. if, given the meaning postulates in (11), truth of $A_1, \ldots, A_n$ in the actual world implies truth of B in the actual world, or, equivalently, if the intersection of $rA_1, \ldots, rA_n$ must denote a subset of rB, given the meaning postulates.

The truth conditions of (10e) can be found using a computation very similar to the one in (12) and clearly (10d) and (10e) co-entail. On the other hand, it is consistent to assume that (10d) $\neq$ (10e) and equivalence of (10f) and (10g) cannot be derived. The truth conditions of (10f) are given in (13a), those of (10g) in (13b). Since the embedded type p terms need not corefer, (13b) could be true in some world where (13b) is false. The treatment therefore is hyperintensional and does not suffer from the logical omniscience problem.

(13) a. $\mathit{aware}((\mathsf{if}((\mathsf{a\ woman})\mathsf{walk}))((\mathsf{no\ man})\mathsf{talk}))\,\mathit{mary}$
b. $\mathit{aware}((\mathsf{if}((\mathsf{a\ man})\mathsf{talk}))((\mathsf{no\ woman})\mathsf{walk}))\,\mathit{mary}$

5. A Short Comparison with Other Approaches

There is a sense in which Thomason's is an almost *minimal* theory of intentionality: the few ingredients present in it must well-nigh be present in any other theory of hyperintensional phenomena as well. If a theory of intentionality is to say anything about the logical relations we find among expressions it should contain some algebraic domain $\mathcal{L}$ where we find notions such as entailment, conjunction, disjunction and the like. It should presumably also contain some domain $\mathcal{P}$ of propositions. If $\mathcal{L}$ is anything like the usual algebras of truth-values, possible worlds etc., it must be distinct from $\mathcal{P}$, as identifying $\mathcal{L}$ and $\mathcal{P}$ in that case will immediately lead to the very problems any theory of hyperintensionality is set out to solve. There must also be a connection $\mathcal{C}$ associating the $\mathcal{P}$ and $\mathcal{L}$ domains, as it is uncontroversial that propositions (often) do have truth-conditions and do enter into logical relationships. In the previous section $\mathcal{L}$ was the domain of type st, $\mathcal{P}$ the domain of type p and $\mathcal{C}$ was the function r, whose behaviour was axiomatized in (11).

While mentioning these elements almost sums up Thomason's theory, it seems that very similar ones must also be present in competing approaches, and indeed if we analyse some of these it is not difficult to recognize the $\mathcal{L}$, $\mathcal{P}$ and $\mathcal{C}$ ingredients. Take, for example, the theory of *structured meanings*, which took its origin in Carnap (1947) and was revived in Lewis (1972) and Cresswell (1985). On Lewis' account the *meaning* of an expression is a finite ordered tree having at each node a category and an appropriate intension. The intension associated with the expression is the one found at the root of that tree. The role of the domain $\mathcal{P}$ is therefore played by a set of trees labelled with categories and intensions; the domain $\mathcal{L}$ is a domain of intensions, much as in our previous section; and the connection $\mathcal{C}$ is given by simply taking the intension found at the root node of any meaning.[11] Since the identity criteria of trees of intensions are more fine-grained than those of these intensions themselves, most unwanted equivalences are blocked.

Another example of an approach in which the essential ingredients of Thomason's theory can easily be recognised is the theory of *impossible worlds*. The idea behind this line of thought is that if the usual set of possible worlds is not large enough to make enough distinctions between semantic values extra worlds, impossible ones, should be added. A key point is that the logical operators need *not* have their usual meaning at these points of reference and that logical validities will therefore cease to hold throughout the set of all worlds. The name "impossible (possible) world" derives from Hintikka (1975), but the idea was also present in Montague (1970) and Cresswell (1972) and has been followed up in Rantala (1982), Barwise (1997), and Zalta (1997), to mention but a few. The set $\mathcal{P}$ here consists of sets of possible and impossible worlds; $\mathcal{L}$ contains only sets of possible worlds and $\mathcal{C}$ is the function that, given any set of worlds, returns the subset that contains just the possible ones, i.e. those in which the logical operators do have their standard meaning. For a formulation of impossible worlds semantics very much along these lines but within a standard logic, see Muskens (1991).

For a third approach in which the Thomasonian ingredients are manifest, let us look at Property Theory (Turner, 1987; Chierchia and Turner, 1988; Fox and Lappin, 2001). Property Theory does not only give an account of hyperintensionality, but also aims to be a theory

[11] In fact the construction of meanings in Lewis (1972) is such that postulates closely akin to (11) will be satisfied by this $\mathcal{C}$.

of *self-predication* (as in *having fun is fun*). This makes it harder to compare the approach with the current one, but if one looks at models for the theory, as given in Chierchia and Turner (1988), one finds a homomorphism T sending an algebra of 'information units' $\mathbf{I}$ to a boolean algebra $\mathbf{P}$. These are essentially our $\mathcal{C}$, $\mathcal{P}$ and $\mathcal{L}$ ingredients respectively (see also Lappin and Pollard, 2000 for this view), and the basic picture reemerges.

When we claimed minimality for Thomason's theory above, we were careful to hedge our claim by calling it an *almost* minimal theory of intentionality. There was a good reason for this hedge, as in fact one can do without the $\mathcal{L}$ and $\mathcal{C}$ ingredients. Consider the entailment relation on propositions defined in the previous section. It is easy to see that this relation is reflexive and transitive but that it is not necessarily antisymmetric. It is therefore a *preorder* and using the interpretations of and, or, if and not one easily sees that our p domains form a *boolean prelattice* (as defined in Fox et al., 2002). Of course, this notion can also be axiomatized independently and one then can do without a domain of truth values. This is the strategy followed in Fox et al. (2002) who also construct worlds as ultrafilters on their basic boolean prelattice. The paper describes a higher-order logic (FIL) that is close to Church's simple theory of types and to the approach we have described above. However, in the following section we shall argue that the trajectory from propositions to truth-conditions is of independent interest and admits of a computational interpretation. A move to discard the $\mathcal{L}$ and $\mathcal{C}$ elements (or to identify $\mathcal{P}$ and $\mathcal{L}$, another way to view the matter) therefore does not suit our purposes.

For a last comparison, let us consider an area where nonextensionality is studied for reasons that differ from our present concerns, the proof theory and model theory of higher order logic. It has long been known that the axiom of extensionality is an obstacle to proving cut-elimination for higher order logic. However, Takahashi (1967) and Prawitz (1968) manage to prove cut-elimination by considering a wider class of structures, not necessarily extensional (the extensionality axiom can be added again after the theorem has been proved). Such nonextensional structures have also been used in Andrews (1971), which lies at the basis of much work in computational higher order logic, but, surprisingly, an explicit modeltheoretic use of them had to await more recent times (Fitting, 2002; Benzmüller et al., 2004; Muskens, 2005). In Fitting (2002) one finds a construction in which type logical expressions of type t are mapped to domains $\mathcal{H}(t)$ of that type. The elements of these domains need not be sets, not even if $t = \langle t_1, \ldots, t_n \rangle$ is complex (Fitting uses relational, not functional, types). A special *extension function* $\mathcal{E}$ sends objects in domains $\mathcal{H}(\langle t_1, \ldots, t_n \rangle)$ to subsets of $\mathcal{H}(t_1) \times \cdots \times \mathcal{H}(t_n)$. Again, we see the basic picture emerge, be it on the level of the metatheory of the logic this time, not on its object level. In fact, our domain of type p now corresponds to $\mathcal{H}(\langle s \rangle)$ and the extension function $\mathcal{E}$ restricted to that domain resembles our r. This is an extremely interesting route to follow, as it proceeds by generalizing existing type logic, not adding to it, with nice complete tableau systems to boot. However, the move of pushing the connection between intensions and extensions to the metalevel does not suit our present purposes for a reason just touched upon: we want to explicitly have this connection at the object level in order to be able to give it a computational interpretation.

In this section I have mentioned a series of approaches to hyperintensionality that are alternatives to the theory this paper is based upon. Lack of space has prevented me to do any of the works mentioned the justice it deserves and I must refer the reader to the originals for details. However, what I do think has been established is the recurrence within all theories of intentionality of a certain pattern in which some function $\mathcal{C}$ sends a domain of propositions $\mathcal{P}$ to a logical algebra $\mathcal{L}$. Since Thomason's theory is a particularly straightforward and lean formalization of this idea, it seems a good idea to study it further. In the next section we will do so, giving a computational interpretation to the connection $\mathcal{C}$.

6. Senses as Queries

What exactly *are* propositions? We have treated propositions as primitive objects in our theory and will continue to do so, but this does not preclude a further investigation of their character. There are many theories in which objects that, from a model-theoretic point of view, are considered to be primitives obtain considerable structure when looked at from a theory internal perspective. In axiomatic set theories, for example, the elements of a models' domain are primitive but they are interpreted as sets from the viewpoint of the theory. This gives them a rich structure. Likewise, a model of second-order Peano Arithmetic can have any kind of objects in its domain, but the theory will interpret them as indistinguishable from the natural numbers, so that they will enter in all kinds of arithmetical relations. In both cases it is the axioms that impose this structure and since our theory also contains axioms, in the form of meaning postulates, we can ask ourselves what structure they impose on the p domain and what structure the objects in this domain get.

Let us take a closer look at (12). This derivation consists of a series of equations, but clearly also allows for a computational interpretation in which we progress from top to bottom. We can therefore interpret our meaning postulates as providing an algorithm that, when given a $\mathcal{T}_{\mathrm{LF}}$ term of type p as input, returns a term denoting a set of possible worlds. In fact we can also interpret the $\mathcal{T}_{\mathrm{LF}}$ terms themselves as programs. In such a view the meaning postulates act as a kind of interpreter for the $\mathcal{T}_{\mathrm{LF}}$ language and the term ((if((a woman)walk))((no man)talk)), for example, is a small program that will lead to the computation in (12).

All this happens on the syntactic level of the logic, not on the semantic level. But taking into consideration that it is really the *essential* function of, say, ((if((a woman)walk))((no man)talk)) that it should lead to (12), it becomes reasonable to start thinking about the proposition that is the denotation of this type p term as if it were an algorithm itself. When it is run, it calls the algorithms for ((a woman)walk) and ((no man)talk) and uses the output of these to arrive at its own output. In what follows nothing will hinge on this interpretation, at least not formally, but we will let ourselves be inspired by this view when it is necessary to make choices of design. The choice that will fit better into the propositions-as-algorithms picture will be the one that is preferred.

The theory thus leads to a computational interpretation but this computational interpretation in its turn naturally leads to a revision of the theory. Since $r(\pi) = \tau$ should now be read as 'algorithm π will output τ' we may well wonder what happens when an algorithm does not lead to an output, as algorithms sometimes do. As things stand the formalization is not able to capture this possibility and we therefore move to a somewhat different set-up. Let d be a constant of type $p((st)t)$. $d\pi\tau$, written $d(\pi, \tau)$, is to be read as 'τ is an output of algorithm π'. The possibility that there is no τ for a given π such that $d(\pi, \tau)$ is left open, as computations may fail or diverge. On the other hand, we will not allow any sense to determine more than one set of worlds and so we will adopt the following functionality requirement as a meaning postulate.

(14) $d(\pi, \tau) \wedge d(\pi, \tau') \rightarrow \tau = \tau'$

From now on statements using d (and its generalizations d^n—see below) will replace statements that make use of r.

We could now proceed with giving meaning postulates such as those in (15), where (11b) is replaced by *two* statements, but in fact a further revision of the theory suggests itself. If our meaning postulates are to capture the computation that is needed when progressing from sense to reference (or from sense to reference-in-each-possible-world), then (15a) is needed, but (15b) seems superfluous. In order to find a value τ

such that $d(\mathsf{and}\ \pi\pi', \tau)$ we need to find τ' and τ'' such that $d(\pi, \tau')$ and $d(\pi', \tau'')$ and set $\tau = \lambda i.\tau' i \wedge \tau'' i$. For going in the opposite direction there is no need.

(15) a. $d(\pi, \tau) \wedge d(\pi', \tau') \to d(\mathsf{and}\ \pi\pi', \lambda i.\tau i \wedge \tau' i)$
b. $d(\mathsf{and}\ \pi\pi', \tau) \to \exists \tau'\tau''[d(\pi, \tau') \wedge d(\pi', \tau'') \wedge \tau = \lambda i.\tau' i \wedge \tau'' i]$

Readers familiar with logic programming[12] will recognize (15a) as (a variant of) a *definite clause*. If we can frame the other meaning postulates as definite clauses as well, the set of meaning postulates will be a logic program! This can be done, but a generalization is needed. Using (15a) the statement $d(\mathsf{if}\ (\mathsf{john\ walk})(\mathsf{john\ talk}), \lambda i.\mathit{walk\ john\ i} \to \mathit{talk\ john\ i})$ can be concluded from the simpler $d(\mathsf{john\ walk}, \mathit{walk\ john})$ and $d(\mathsf{john\ talk}, \mathit{talk\ john})$, for example, and this in general suggests finding the st term associated with a $\mathcal{T}_{\mathrm{LF}}$ term of type p by progressively breaking down the latter. But meaning postulates such as (15a) are insufficiently general to carry out this plan. Consider $(\mathsf{a\ unicorn})\lambda x.\mathsf{and}(\mathsf{walk}\ x)(\mathsf{talk}\ x)$, for example. Decomposing this term will lead to $\lambda x.\mathsf{and}(\mathsf{walk}\ x)(\mathsf{talk}\ x)$, but the rule in (15a) cannot be used for further decomposition in view of the initial abstraction. What is needed is a general way to associate terms of type $e^n p$[13] with terms of type $e^n(st)$. In order to obtain this we will assume the existence of constants d^n, where, for each natural number n, d^n is of type $(e^n p)((e^n(st))t)$. The original d will be d^0, but we will continue to write it simply as d.

The following postulates are schemas; a concrete postulate can be obtained by instantiating n to any natural number, $\vec{z}$ to a sequence of pairwise distinct variables $z_1 \ldots z_n$, and x and y to distinct variables of type e that are also distinct from all $\vec{z}$. $\vec{u}$ must be a sequence of variables of type e of length $n+2$ and $\vec{v}$ a similar sequence of length $n+1$. The notation $d^k(\mathcal{R}, R)$ of course entails that $\mathcal{R}$ is of type $e^k p$ and R is of type $e^k(st)$.

(16) a. $d^n(\mathcal{R}, R) \to d^n(\lambda\vec{z}.\mathsf{not}\ \mathcal{R}\vec{z}, \lambda\vec{z}\lambda i.\neg R\vec{z}i)$
b. $d^n(\mathcal{R}, R) \wedge d^n(\mathcal{R}', R') \to d^n(\lambda\vec{z}.\mathsf{and}(\mathcal{R}\vec{z})(\mathcal{R}'\vec{z}), \lambda\vec{z}\lambda i.R\vec{z}i \wedge R'\vec{z}i)$
c. $d^n(\mathcal{R}, R) \wedge d^n(\mathcal{R}', R') \to d^n(\lambda\vec{z}.\mathsf{or}(\mathcal{R}\vec{z})(\mathcal{R}'\vec{z}), \lambda\vec{z}\lambda i.R\vec{z}i \vee R'\vec{z}i)$
d. $d^n(\mathcal{R}, R) \wedge d^n(\mathcal{R}', R') \to d^n(\lambda\vec{z}.\mathsf{if}(\mathcal{R}\vec{z})(\mathcal{R}'\vec{z}), \lambda\vec{z}\lambda i.R\vec{z}i \to R'\vec{z}i)$
e. $d^{n+1}(\mathcal{R}, R) \wedge d^{n+1}(\mathcal{R}', R') \to$
$d^n(\lambda\vec{z}.\mathsf{every}(\mathcal{R}'\vec{z})(\mathcal{R}\vec{z}), \lambda\vec{z}\lambda i\forall x[R'\vec{z}xi \to R\vec{z}xi])$
f. $d^{n+1}(\mathcal{R}, R) \wedge d^{n+1}(\mathcal{R}', R') \to$
$d^n(\lambda\vec{z}.\mathsf{a}(\mathcal{R}'\vec{z})(\mathcal{R}\vec{z}), \lambda\vec{z}\lambda i.\exists x[R'\vec{z}xi \wedge R\vec{z}xi])$

[12] See e.g. (Apt, 1990) for the theory; (Blackburn et al., 2001) is a nice introduction to the programming language Prolog based on this theory.
[13] For any types α and β, we define $\alpha^0\beta = \beta$ and $\alpha^{n+1}\beta = \alpha(\alpha^n\beta)$.

g. $d^{n+1}(\mathcal{R}, R) \wedge d^{n+1}(\mathcal{R}', R') \rightarrow$
$\qquad d^n(\lambda\vec{z}.\mathsf{no}(\mathcal{R}'\vec{z})(\mathcal{R}\vec{z}), \lambda\vec{z}\lambda i.\neg\exists x[R'\vec{z}xi \wedge R\vec{z}xi])$

h. $d^{n+1}(\mathcal{R}, R) \rightarrow d^n(\lambda\vec{z}.\mathsf{mary}(\mathcal{R}\vec{z}), \lambda\vec{z}\lambda i.\exists x[x = \mathit{mary} \wedge R\vec{z}xi])$

i. $d^n(\mathcal{R}, R) \rightarrow d^n(\lambda\vec{z}.\mathsf{necessarily}(\mathcal{R}\vec{z}), \lambda\vec{z}\lambda i.\forall j[\mathit{acc}\, ij \rightarrow R\vec{z}j])$

j. $d^n(\mathcal{R}, R) \rightarrow d^n(\lambda\vec{z}.\mathsf{possibly}\,(\mathcal{R}\vec{z}), \lambda\vec{z}\lambda i.\exists j[\mathit{acc}\, ij \wedge R\vec{z}j])$

k. $d^{n+2}(\lambda\vec{u}.\mathsf{is}\; xy, \lambda\vec{u}\lambda i.x = y)$, where $\vec{u}$ contains x and y

l. $d^{n+2}(\lambda\vec{u}.\mathsf{love}\; xy, \lambda\vec{u}.\mathit{love}\; xy)$, where $\vec{u}$ contains x and y

m. $d^{n+1}(\lambda\vec{v}.\mathsf{planet}\; x, \lambda\vec{v}.\mathit{planet}\; x)$, where x is among the $\vec{v}$

n. $d^{n+1}(\lambda\vec{z}.\mathsf{believe}\,(\mathcal{R}\vec{z}), \lambda\vec{z}.\mathit{believe}\,(\mathcal{R}\vec{z}))$

These schemas are to be interpreted paradigmatically (so, say, (16h) will not only govern the behaviour of mary, but also that of bill, john, hesperus and the like) and can be used to break down any closed $\mathcal{T}_{\mathrm{LF}}$ term into its constituent parts and compute the value that it determines in terms of the values of the latter. The $\vec{z}$ in (16) take care of *variable management* and make sure that the bound variables that are encountered in the process of breaking down a $\mathcal{T}_{\mathrm{LF}}$ term are handled correctly. Here is a derivation using (16) that illustrates the process.

(17)

$$\dfrac{\dfrac{}{d^1(\lambda x.\mathsf{girl}\; x, \lambda x.\mathit{girl}\; x)} \qquad \dfrac{\dfrac{}{d^2(\lambda xy.\mathsf{boy}\; y, \lambda xy.\mathit{boy}\; y)} \quad \dfrac{}{d^2(\lambda xy.\mathsf{kiss}\; yx, \lambda xy.\mathit{kiss}\; yx)}}{d^1(\lambda x.(\mathsf{every\ boy})\lambda y.\mathsf{kiss}\; yx, \lambda x\forall y.\mathit{boy}\; yi \rightarrow \mathit{kiss}\; yxi)}}{d((\mathsf{a\ girl})\lambda x.(\mathsf{every\ boy})\lambda y.\mathsf{kiss}\; yx, \lambda i\exists x[\mathit{girl}\; xi \wedge \forall y[\mathit{boy}\; yi \rightarrow \mathit{kiss}\; yxi]])}$$

Reading this derivation from bottom to top, note how superscripts on d increase with each quantifier that is met and how the arguments of the d^n come with an initial prefix of lambdas of length n. Not all abstractors in this prefix need to actually bind a variable (e.g. the λx in $\lambda xy.\mathsf{boy}\, y$ does not). The idea is that initial abstractors $\lambda\vec{z}$ contain all variables that *potentially* occur free in the rest of the term. Note also that the variables in initial abstractors come in the order in which quantification has taken place; this may not be the order in which operators take their arguments. Meaning postulates such as (16l) provide for this possibility by just requiring that the variables filling the argument slots of a given argument-taking expression should be among the variables that are abstracted over, without prescribing order.

Derivations such as the one in (17) have a conventional form. They *derive* statements of the form $d(\pi, \tau)$; they do not *find* a τ such that $d(\pi, \tau)$, given some π. But since (16) is in fact a *logic program* it is possible to interpret it as providing an algorithm that does just that. This squares well with the view of senses as recipes for finding referents, and therefore we now turn to this interpretation.

First, let us introduce some technicalities. Logic programming involves a notion of *unification*, but while in the standard theory this notion is taken to involve first-order terms only, we need a generalization, as the terms in (16) contain second-order variables. We must therefore introduce some notions from the theory of higher-order unification (for a survey of higher-order unification theory see Dowek, 2001). A *substitution* σ is defined as a finite function from variables to terms, such that $\sigma(X)$ and X always have the same type. If $dom(\sigma) = \{X_1, \ldots, X_n\}$ and $\sigma(X_i) = M_i$ we may use $[X_1 := M_1, \ldots, X_n := M_n]$ to denote σ. If M is a term and $\sigma = [X_1 := M_1, \ldots, X_n := M_n]$ is a substitution, then $M\sigma$ is the term obtained from M by simultaneously substituting each X_i in M by M_i, with possible renaming of bound variables in order to avoid variable clashes. The *composition* of the substitutions σ_1 and σ_2 is the substitution $\sigma_1\sigma_2$ such that, for all variables X, $\sigma_1\sigma_2(X) = X\sigma_2\sigma_1$ if $X \neq X\sigma_2\sigma_1$ and $\sigma_1\sigma_2(X)$ is undefined otherwise. A *unifier* of M_1 and M_2 is a substitution σ such that $M_1\sigma$ and $M_2\sigma$ have the same $\beta\eta$ normal form and a *most general unifier* of M_1 and M_2 is a unifier σ of M_1 and M_2 such that for all unifiers ϑ of M_1 and M_2 there is an η such that $\vartheta = \eta\sigma$.

There is no general algorithm that, given two terms M_1 and M_2, decides whether M_1 and M_2 have a unifier (Huet, 1973). Huet's proof uses third-order terms while the terms in (16) are second-order, but second-order unification is also undecidable (Goldfarb, 1981) and therefore unattractive for our purposes. Fortunately there is a notion of unification, *higher-order patterns unification* (Miller, 1991), that fits our bill. A *pattern* is a term M such that for every subterm of the form $XM_1 \ldots M_n$, where X is a free variable, the terms $M_1, \ldots, M_n$ are distinct variables bound in M. Note that the terms $d^k(M, M')$ in (16) are all of this form. Unification of patterns is well-behaved: it is decidable in polynomial time and when a unification problem has a unifier, it has a most general unifier (Miller, 1991). We will adopt higher-order patterns unification as our notion of unification here.[14]

With the right concept of unification in place we can now consider logic programming proper. If $\alpha = (\alpha_1 \ldots (\alpha_{n-1}(\alpha_n\,\beta)) \ldots)$, for some type α, where β is a basic type, β is called the *target type* of α. Fix a

[14] One clarification may be necessary: Higher-order unification is normally defined for languages that are built up from variables and constants, using λ-abstraction and application only. This means that, since we want the standard theory to apply to our higher-order logic, logical operators such as $=, \forall, \exists, \wedge, \ldots$ must be treated with the help of these. In order to obtain the universal quantifier, for example, we will assume that, for each type α, the initial vocabulary contains a constant $\Pi_{(\alpha t)t}$, and that $\forall X_\alpha\,\varphi$ is syntactic sugaring for $\Pi_{(\alpha t)t}\lambda X_\alpha\,\varphi$. The other logical constants are assumed to be treated in similar ways.

set of constants $\mathcal{A}$ all of whose types have target type t and such that $d^k \in \mathcal{A}$ for all k. *Atoms* will be terms of type t of the form $aM_1 \ldots M_m$, where $a \in \mathcal{A}$. *Literals* will be atoms (positive literals) or the negations of atoms (negative literals), while *clauses* are disjunctions of literals. Note that a clause with positive literals $B_1, \ldots, B_m$ and negative literals $\neg A_1, \ldots, \neg A_n$ can equivalently be written as $A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \vee \ldots \vee B_m$. In the logic programming literature this is often written in the *clausal form* $B_1, \ldots, B_m \leftarrow A_1, \ldots, A_n$, even when $m = 0$, in which case the disjunction is identified with $\bot$, or when $n = 0$ and the conjunction reduces to $\top$. The free variables in a clause are interpreted universally, so that a clause is interpreted as its universal closure. A *variant* of a clause is the result of renaming the free variables of that clause. The variant is *fresh* (to some proof) if the new free variables have not been used before (in that proof). A clause is *Horn* if it has at most one positive literal; if it has exactly one, it is called *definite*; if it has none, it is called a *goal* or *query*. In a definite clause $A_1 \wedge \ldots \wedge A_n \rightarrow B$, the atom B is the *head* and $A_1 \wedge \ldots \wedge A_n$ is the *body*. A *(logic) program* or *database* is a set of definite clauses. Note that the meaning postulates in (16) form a program.[15]

In logic programming there is a single rule of inference, called the *resolution rule*, which can be formulated as follows.

(18) **Resolution Rule.** Let Π be some logic program and let

$$B_1 \wedge \ldots \wedge B_m \rightarrow B$$

be a fresh variant of a definite clause in Π. From

$$A_1 \wedge \ldots \wedge A_i \wedge \ldots \wedge A_n \rightarrow \bot$$

to infer

$$[A_1 \wedge \ldots \wedge A_{i-1} \wedge B_1 \wedge \ldots \wedge B_m \wedge A_{i+1} \wedge \ldots \wedge A_n \rightarrow \bot]\sigma \ ,$$

where σ is the most general unifier of B and A_i. The atom A_i is called the *selected* atom.

This resolution rule can be used in the following manner. Suppose that, given the program in (16), you want to find a τ such that

(19) $d((\mathsf{if}((\mathsf{a\ woman})\mathsf{walk}))((\mathsf{no\ man})\mathsf{talk}), \tau)$

[15] Programs are usually defined to be *finite* sets of definite clauses, for obvious practical reasons. But since the meaning postulates in (16) are schemas which have infinitely many instantiations, we will allow programs to be infinite. Even so, our derivations have the property that, at each inference step, only finitely many (and in fact just one singular) definite clause needs be considered.

holds. This amounts to finding a constructive proof for

(20) $\exists\tau\, d((\mathsf{if}((\mathsf{a\ woman})\mathsf{walk}))((\mathsf{no\ man})\mathsf{talk}), \tau)$,

i.e. a proof that does not only affirm the existence of such a τ, but also gives one. In order to find this constructive proof, you can form the query

(21) $\leftarrow d((\mathsf{if}((\mathsf{a\ woman})\mathsf{walk}))((\mathsf{no\ man})\mathsf{talk}), \tau)$

and try to refute it. This will work because the query is equivalent to the negation of the sentence you want to prove. There is exactly one clause in (16) whose head unifies with the only atom in this goal, namely (16d), the rule for if, with $n = 0$. Here is a fresh variant of that rule, in claual form.

(22) $d(\mathsf{if}\,\pi_1\pi_2, \lambda i.\tau_1 i \to \tau_2 i) \leftarrow d(\pi_1, \tau_1), d(\pi_2, \tau_2)$

The head of this rule unifies with your query, with most general unifier

(23) $[\pi_1 := ((\mathsf{a\ woman})\mathsf{walk}), \pi_2 := ((\mathsf{no\ man})\mathsf{talk}), \tau := \lambda i.\tau_1 i \to \tau_2 i]$

The value of τ has now been expressed in terms of the new variables τ_1 and τ_2 and you can proceed with trying to find values for these, continuing with the new goal

(24) $\leftarrow d(((\mathsf{a\ woman})\mathsf{walk}), \tau_1),\ d(((\mathsf{no\ man})\mathsf{talk}), \tau_2)$.

The rest of the search is depicted in Figure 1, where only those parts of the relevant substitutions are given that matter for the final answer. Note that the computation ends with $\leftarrow$, which is short for the absurdity $\bot \leftarrow \top$, and the original query has thus been refuted. The conclusion is that a τ with the given specifications indeed exists and composition of the substitutions that were found gives it the value

(25) $\lambda i.\exists x[\mathit{woman}\ xi \wedge \mathit{walk}\ xi] \to \neg\exists x[\mathit{man}\ xi \wedge \mathit{talk}\ xi]$

Figure 2 gives a similar refutation for a query that is slightly more complex than (21), with more embedding of operators. The reader will have no difficulty in constructing more examples. Figure 3 goes the other way round; it takes (25) and then starts out finding the corresponding π, which clearly must lead to the value

$$\pi = (\mathsf{if}((\mathsf{a\ woman})\mathsf{walk}))((\mathsf{no\ man})\mathsf{talk}) .$$

The query in a sense is the *reverse* of (21). Note however that in general the reversibility of computations is imperfect in the following sense. For any closed π in $\mathcal{T}_{\mathrm{LF}}$ it is possible to compute a τ such that $d(\pi, \tau)$.

$$\begin{array}{c}
\leftarrow \underline{d((\mathsf{if}((\mathsf{a\ woman})\mathsf{walk}))((\mathsf{no\ man})\mathsf{talk}), \tau)} \\
(16\mathrm{d}) \downharpoonleft \tau := \lambda i.\tau_1 i \rightarrow \tau_2 i \\
\leftarrow d(((\mathsf{a\ woman})\mathsf{walk}), \tau_1),\ \underline{d(((\mathsf{no\ man})\mathsf{talk}), \tau_2)} \\
(16\mathrm{g}) \downharpoonleft \tau_2 := \lambda i.\neg\exists x[P_1 xi \wedge P_2 xi] \\
\leftarrow d(((\mathsf{a\ woman})\mathsf{walk}), \tau_1),\ \underline{d^1(\mathsf{man}, P_1)},\ d^1(\mathsf{talk}, P_2) \\
(16\mathrm{m}) \downharpoonleft P_1 := \mathit{man} \\
\leftarrow \underline{d(((\mathsf{a\ woman})\mathsf{walk}), \tau_1)},\ d^1(\mathsf{talk}, P_2) \\
(16\mathrm{f}) \downharpoonleft \tau_1 := \lambda i.\exists y[P_3 yi \wedge P_4 yi] \\
\leftarrow \underline{d^1(\mathsf{woman}, P_3)},\ d^1(\mathsf{walk}, P_4),\ d^1(\mathsf{talk}, P_2) \\
(16\mathrm{m}) \downharpoonleft P_3 := \mathit{woman} \\
\leftarrow d^1(\mathsf{walk}, P_4),\ \underline{d^1(\mathsf{talk}, P_2)} \\
(16\mathrm{m}) \downharpoonleft P_2 := \mathit{talk} \\
\leftarrow \underline{d^1(\mathsf{walk}, P_4)} \\
(16\mathrm{m}) \downharpoonleft P_4 := \mathit{walk} \\
\leftarrow
\end{array}$$

Figure 1. A refutation of $\leftarrow d((\mathsf{if}((\mathsf{a\ woman})\mathsf{walk}))((\mathsf{no\ man})\mathsf{talk}), \tau)$. Selected atoms are underlined. Composition of the substitutions that are found gives the value $\tau = \lambda i.\exists x[\mathit{woman}\, xi \wedge \mathit{walk}\, xi] \rightarrow \neg\exists x[\mathit{man}\, xi \wedge \mathit{talk}\, xi]$.

Moreover this τ will be equivalent to any τ' such that (16) $\models d(\pi, \tau')$. But the reverse is not the case. On the one hand, some closed st terms τ may lack a π such that $d(\pi, \tau)$ will be computed, and, on the other, equivalent τ and τ' may lead to nonequivalent π and π' such that $d(\pi, \tau)$ and $d(\pi', \tau')$. This is because the equivalence relation on terms of type p has much finer grain than that on terms of type st.

We now come back to the question asked at the beginning of this section and to our decision to interpret propositions as algorithms, at least informally. Clearly, the theory in itself does not force us to make any identification of the model-theoretic object denoted by a given term Φ of type p and the query $\leftarrow d(\Phi, \tau)$, but we may decide that this is the intended interpretation. More generally, we can identify the denotation of any closed $\mathcal{T}_{\mathrm{LF}}$ term Θ of type $e^n p$ with the query $\leftarrow d^n(\Theta, R)$, where R is a variable of type $e^n(st)$. Such a computational interpretation answers some questions about the identity relation on senses. For example, the question whether, in general, $\mathsf{not\ not}\,\Phi = \Phi$ has an immediate negative answer. The first leg of this equation leads to the computation in (26), while the second does not.

(26)
$$\begin{array}{l}
\leftarrow d(\mathsf{not\ not}\,\Phi, \tau) \\
\qquad \downharpoonleft \tau := \lambda i.\neg\tau_1 i \\
\leftarrow d(\mathsf{not}\,\Phi, \tau_1) \\
\qquad \downharpoonleft \tau_1 := \lambda i.\neg\tau_2 i \\
\leftarrow d(\Phi, \tau_2) \\
\qquad \vdots
\end{array}$$

$$\begin{array}{c}
\leftarrow d((\mathsf{a\ man})(\lambda x.\mathsf{necessarily}((\mathsf{every\ unicorn})(\lambda y.\mathsf{kiss}\ yx))),\tau) \\
\downarrow \tau := \lambda i\exists x[P_1xi \wedge P_2xi] \\
\leftarrow d^1(\mathsf{man},P_1),\ d^1(\lambda x.\mathsf{necessarily}((\mathsf{every\ unicorn})(\lambda y.\mathsf{kiss}\ yx)),P_2) \\
\downarrow P_1 := man \\
\leftarrow d^1(\lambda x.\mathsf{necessarily}((\mathsf{every\ unicorn})(\lambda y.\mathsf{kiss}\ yx)),P_2) \\
\downarrow P_2 := \lambda x\lambda i\forall j[acc\, ij \to P_3xj] \\
\leftarrow d^1(\lambda x.(\mathsf{every\ unicorn})(\lambda y.\mathsf{kiss}\ yx),P_3) \qquad \text{n} \\
\downarrow P_3 := \lambda x\lambda i\forall y[R_1xyi \to R_2xyi] \\
\leftarrow d^2(\lambda x.\mathsf{unicorn},R_1),\ d^2(\lambda xy.\mathsf{kiss}\ yx,R_2) \\
\downarrow R_1 := \lambda x.unicorn \\
\leftarrow d^2(\lambda xy.\mathsf{kiss}\ yx,R_2) \\
\downarrow R_2 := \lambda xy.kiss\ yx \\
\leftarrow
\end{array}$$

Figure 2. $\leftarrow d((\mathsf{a\ man})(\lambda x.\mathsf{necessarily}((\mathsf{every\ unicorn})(\lambda y.\mathsf{kiss}\ \ yx))),\tau)$ leads to a refutation as depicted here. In each case the first atom is selected. Composition of subsitutions gives $\tau = \lambda i\exists x[man\ xi \wedge \forall j[acc\ ij \to \forall y[unicorn\ yj \to kiss\ yxj]]]$.

What about conjunctions? Will the interpretation of senses as queries force us to identify $\mathsf{and}\,\Phi_1\Phi_2$ and $\mathsf{and}\,\Phi_2\Phi_1$, for arbitrary Φ_1 and Φ_2? This depends on our identification criteria for queries.

(27)
$$\begin{array}{cc}
\leftarrow d(\mathsf{and}\,\Phi_1\Phi_2,\tau) & \leftarrow d(\mathsf{and}\,\Phi_2\Phi_1,\tau) \\
\downarrow \tau := \lambda i.\tau_1 i \wedge \tau_2 i & \downarrow \tau := \lambda i.\tau_2 i \wedge \tau_1 i \\
\leftarrow d(\Phi_1,\tau_1),\ d(\Phi_2,\tau_2) & \leftarrow d(\Phi_2,\tau_2),\ d(\Phi_1,\tau_1) \\
\vdots & \vdots
\end{array}$$

In (27) it is shown how the first proposition under consideration leads to $\leftarrow d(\Phi_1,\tau_1),\ d(\Phi_2,\tau_2)$, while the second leads to $\leftarrow d(\Phi_2,\tau_2),\ d(\Phi_1,\tau_1)$. These queries are certainly equivalent and that would warrant the conclusion that the two propositions are identical.[16] On the other hand, while in computations that arise from (16) atoms can always be selected in any order, it is evident that as soon as things are scaled up and anaphora and presuppositions are taken into account, dependencies will arise that will make some form of *control* imperative (the next section will give an example). In fact, it seems almost unavoidable to assume that natural language has a control regime that computes subgoals more or less strictly in the left-to-right order in which they appear in syntax (either surface syntax or LF). If control is factored in into the identity criteria for queries in some way, the propositions $\mathsf{and}\,\Phi_1\Phi_2$ and $\mathsf{and}\,\Phi_2\Phi_1$ may well be distinguished.

At the start of this section we decided to replace the function r, that was sending propositions to their corresponding sets of possible worlds,

[16] This conclusion agrees with the one reached in Moschovakis (1994).

$$\begin{array}{c}
\leftarrow \underline{d(\pi, \lambda i.\exists x[woman\ xi \wedge walk\ xi] \rightarrow \neg\exists x[man\ xi \wedge talk\ xi])} \\
\downarrow \pi := \mathsf{if}\,\pi_1\pi_2 \\
\leftarrow d(\pi_1, \lambda i.\exists x[woman\ xi \wedge walk\ xi]),\ \underline{d(\pi_2, \lambda i.\neg\exists x[man\ xi \wedge talk\ xi])} \\
\downarrow \pi_2 := \mathsf{no}\,\mathcal{P}_1\mathcal{P}_2 \\
\leftarrow d(\pi_1, \lambda i.\exists x[woman\ xi \wedge walk\ xi]),\ \underline{d^1(\mathcal{P}_1, man)},\ d^1(\mathcal{P}_2, talk) \\
\downarrow \mathcal{P}_1 := \mathsf{man} \\
\leftarrow \underline{d(\pi_1, \lambda i.\exists x[woman\ xi \wedge walk\ xi])},\ d^1(\mathcal{P}_2, talk) \\
\downarrow \pi_1 := \mathsf{a}\,\mathcal{P}_3\mathcal{P}_4 \\
\leftarrow \underline{d^1(\mathcal{P}_3, woman)},\ d^1(\mathcal{P}_4, walk),\ d^1(\mathcal{P}_2, talk) \\
\downarrow \mathcal{P}_3 := \mathsf{woman} \\
\leftarrow d^1(\mathcal{P}_4, walk),\ \underline{d^1(\mathcal{P}_2, talk)} \\
\downarrow \mathcal{P}_2 := \mathsf{talk} \\
\leftarrow \underline{d^1(\mathcal{P}_4, walk)} \\
\downarrow \mathcal{P}_4 := \mathsf{walk} \\
\leftarrow
\end{array}$$

Figure 3. A refutation of $\leftarrow d(\pi, \lambda i.\exists x[woman\ xi \wedge walk\ xi] \rightarrow \neg\exists x[man\ xi \wedge talk\ xi])$. Selected literals are underlined. $\pi = \mathsf{(if((a\ woman)walk))((no\ man)talk)}$ results if substitutions are composed.

by a relation d (or rather a family of relations d^n). This necessitates a redefinition of the notion of *entailment* between propositions, as the definition in section 4 depended on r. Although it is true that for every closed term $\Phi \in \mathcal{T}_{\mathrm{LF}}$ of type p there is a φ such that $d(\Phi, \varphi)$ is derivable from the meaning postulates, this property will not be retained in the following section. The definition of entailment between propositions must therefore take into account the possibility that propositions may fail to determine a set of worlds. (28) gives two notions that seem reasonable, one of *strict entailment*, where all propositions involved are required to provably determine a set of worlds, and a derived notion of entailment that leaves room for non-determining propositions, as long as these are redundant to the argument.

(28) **Entailment between Propositions.** Let $\Phi_1, \ldots, \Phi_n$ and Ψ be terms of type p and let MP be a set of meaning postulates. We say that $\Phi_1, \ldots, \Phi_n$ *strictly entail* Ψ given MP if

1. MP $\models \exists\tau\, d(\Xi, \tau)$ holds for all $\Xi \in \{\Phi_1, \ldots, \Phi_n, \Psi\}$, and
2. MP, $d(\Phi_1, \varphi_1) \wedge \ldots \wedge d(\Phi_n, \varphi_n) \wedge d(\Psi, \psi) \wedge \varphi_1 w \wedge \ldots \wedge \varphi_n w \models \psi w$, where the φ_k and ψ are any terms of type st and w is any constant of type s

If Γ is a set of type p terms, Γ is said to *entail* Ψ given MP if there are $\Phi_1, \ldots, \Phi_n \in \Gamma$ such that $\Phi_1, \ldots, \Phi_n$ strictly entail Ψ given MP.

The fact that some propositions may fail to determine a set of worlds in extensions of our system also creates room for variation where meaning postulates are concerned. Consider the treatment of and, or and if. If the meaning postulates in (16) are all we can go by, a term $\mathsf{if}\,\Phi_1\Phi_2$ can only provably determine a set of worlds if both Φ_1 and Φ_2 do. Similar remarks can be made about and and or. The situation resembles that of having a Weak Kleene evaluation scheme in a logic with truth value gaps, where a complex formula will be undefined if one of its constituent parts is. Additional meaning postulates can be adopted and then lead to behaviour reminiscent of stronger evaluation schemes. Adding the following set, for example, gives a Strong Kleene way of evaluating:[17]

(29) a. $d(\pi_1, \lambda i.\bot) \to d(\mathsf{and}\,\pi_1\pi_2, \lambda i.\bot)$
b. $d(\pi_2, \lambda i.\bot) \to d(\mathsf{and}\,\pi_1\pi_2, \lambda i.\bot)$
c. $d(\pi_1, \lambda i.\top) \to d(\mathsf{or}\,\pi_1\pi_2, \lambda i.\top)$
d. $d(\pi_2, \lambda i.\top) \to d(\mathsf{or}\,\pi_1\pi_2, \lambda i.\top)$
e. $d(\pi_1, \lambda i.\bot) \to d(\mathsf{if}\,\pi_1\pi_2, \lambda i.\top)$
f. $d(\pi_2, \lambda i.\top) \to d(\mathsf{if}\,\pi_1\pi_2, \lambda i.\top)$

Whether such extra meaning postulates need be adopted may be a matter of empirical investigation. But it is curious that one can, within one logic, have options that are strongly reminiscent of evaluation schemes that in the usual setting are constitutive of different logics.

7. Circular Propositions

‘A logical theory,’ Russell famously wrote in *On Denoting*, ‘may be tested by its capacity for dealing with puzzles … ’ The puzzles we are turning to in this section are the Liar and friends. Linguistic semantics has an obligation to say something about these puzzles for the simple reason that they can be formulated very easily in natural languages, with the help of mechanisms that seem central to the workings of language itself and that, in a vast majority of cases, present no puzzle at all. Semantic theory must explain the workings of these mechanisms, and if it does, the behaviour of the Liar should follow from this explanation as a corollary.

The ingredients of the Liar are well-known: self-reference and the capacity to talk about truth and falsity. But, as Kripke (1975) has argued convincingly, the property of self-reference, although it needs

[17] The formulation in (29) can easily be extended to deal with d^n for arbitrary n.

to be present if a sentence[18] or set of sentences is to be circular, need not be evident from a Liar-like sentence at all, and even a seemingly innocent statement like (30), may, under unfavourable circumstances, turn out to be paradoxical.

(30) Most of Blair's assertions about Iraq are false

A set of unfavourable circumstances which turn (30) into a paradox are: (30) is uttered by Jones, whose other statements about Iraq are all true. Blair's statements about Iraq, on the other hand, are evenly divided among the True and the False, with the exception of one extra claim, the contention that everything Jones says about Iraq is true. It is easy to see that paradox results.

This shows that circular sentences need not wear self-referentiality upon their sleeves, that highly normal utterances can turn out to be circular, and that the normal mechanisms that allow reference to other statements, the kind of mechanisms linguistic theory has to deal with, are sufficient to also obtain circular reference.

What about the other ingredient for the paradox, the possibility to pronounce a statement true (or false)? We are conditioned to think about languages that can express their own syntax here, and about a truth predicate that can hold or fail to hold of syntactic objects expressed with the help of coding (e.g. Gödel numbering). But in a setting such as the present one, where propositions are available as first-class citizens, it seems much more natural to let truth and falsity attach directly to the latter (see also Barwise and Etchemendy, 1987). In fact, as Moschovakis (1994) makes clear, as soon as self-reference is present in such a setting, the paradoxes already pop up with the help of negation only. The following are informal examples of the Liar (31a) and the Truth-teller (31b) considered by Moschovakis (1994).

(31) a. ¬(31a)
b. (31b)

Let us build truth, falsity, and reference to propositions into the $\mathcal{T}_{\mathrm{LF}}$ language by adding the non-logical constants true and false of type pp and the constants this, that, this_0, that_0, this_1, $\mathsf{that}_1, \ldots$ of type p to the list of sans serif constants in Table II and by closing off as before.

(32) a. A: If a woman is walking no man is talking. B: That's true.
b. $\mathsf{(if((a\ woman)walk))((no\ man)talk)}$

[18] I will take a *sentence* to be circular if the *proposition* it expresses (on the intended reading) is circular.

$$\begin{tikzcd}
\leftarrow d(\mathsf{true\ that}, \tau) \arrow[d] \\
\leftarrow d(\mathsf{that}, \tau) \arrow[d] \\
\leftarrow \mathit{ant}(\mathsf{that}, \pi),\ d(\pi, \tau) \arrow[d, "\pi := (\mathsf{if}((\mathsf{a\ woman})\mathsf{walk}))((\mathsf{no\ man})\mathsf{talk})"] \\
\leftarrow d((\mathsf{if}((\mathsf{a\ woman})\mathsf{walk}))((\mathsf{no\ man})\mathsf{talk}), \tau) \\
\vdots \\
\text{(continued as in Figure 1)}
\end{tikzcd}$$

Figure 4. A refutation tree for $d(\mathsf{true\ that}, \tau)$. Added to the database is the information that $\mathit{ant}(\mathsf{that}, (\mathsf{if}((\mathsf{a\ woman})\mathsf{walk}))((\mathsf{no\ man})\mathsf{talk}))$.

c. $(\mathsf{true\ that})$

d. $\mathit{ant}(\mathsf{that}, (\mathsf{if}((\mathsf{a\ woman})\mathsf{walk}))((\mathsf{no\ man})\mathsf{talk}))$

The little dialogue in (32a)ncan be translated with the help of (32b) and (32c), which are closed $\mathcal{T}_{\mathrm{LF}}$ terms of type p under the new definition. But the translation is incomplete if it does not additionally represent the fact that the demonstrative in B's utterance is anaphorically linked to A's observation. This is expressed in (32d), where the $p(pt)$ constant *ant* expresses antecedenthood. It is assumed that $\mathit{ant} \in \mathcal{A}$, so that the constant is treated on a par with the d^n.

We need meaning postulates for the new constants. Those for the truth predicates in (33), essentially treat false as negation and true as the redundant connective that Frege already thought it was.

(33) a. $d(\pi, \tau) \rightarrow d(\mathsf{true}\ \pi, \tau)$

b. $d(\pi, \tau) \rightarrow d(\mathsf{false}\ \pi, \lambda i.\neg\tau i)$

The postulates for the constants translating demonstrative pronouns are given in (34). The basic idea is that the set of worlds determined by such a constant is simply the set of worlds determined by its antecedent.

(34) a. $\mathit{ant}(\mathsf{this}, \pi) \wedge d(\pi, \tau) \rightarrow d(\mathsf{this}, \tau)$

b. $\mathit{ant}(\mathsf{that}, \pi) \wedge d(\pi, \tau) \rightarrow d(\mathsf{that}, \tau)$

Now assume that whenever a hearer figures out that an anaphoric relation holds between a demonstrative pronoun that (or this) and some sentence translated by the p term Φ, he adds $\mathit{ant}(\mathsf{that}, \Phi)$ to his database of definite clauses. For example, a person overhearing (32a) who interprets the anaphoric relation between *that* and A's utterance in the way obviously intended by B may add (32d) to his logic program.

$$\begin{tikzcd}
\leftarrow d(\mathsf{false\ this}, \tau) \arrow[d, "\tau := \lambda i.\neg\tau_1 i"] & \leftarrow d(\mathsf{true\ this}, \tau) \arrow[d] \\
\leftarrow d(\mathsf{this}, \tau_1) \arrow[d] & \leftarrow d(\mathsf{this}, \tau) \arrow[d] \\
\leftarrow \mathit{ant}(\mathsf{this}, \pi),\ d(\pi, \tau_1) \arrow[d, "\pi := \mathsf{false\ this}"] & \leftarrow \mathit{ant}(\mathsf{this}, \pi),\ d(\pi, \tau) \arrow[d, "\pi := \mathsf{true\ this}"] \\
\leftarrow d(\mathsf{false\ this}, \tau_1) \arrow[d, "\tau_1 := \lambda i.\neg\tau_2 i"] & \leftarrow d(\mathsf{true\ this}, \tau) \arrow[d] \\
\leftarrow d(\mathsf{this}, \tau_2) & \leftarrow d(\mathsf{this}, \tau) \\
\vdots & \vdots
\end{tikzcd}$$

a.– Liar b.– Truth-teller

Figure 5. Search trees for the Liar (a) and the Truth-teller (b). In case of the Liar $\mathit{ant}(\mathsf{this}, \mathsf{false\ this})$ was added to the program; for the Truth-teller this was $\mathit{ant}(\mathsf{this}, \mathsf{true\ this})$. In both cases search results in an infinite loop.

This seems to be a natural way to deal with anaphoric elements. While other words have more or less fixed meanings and thus can obtain meaning from postulates permanently present in the database, the meaning of an anaphoric element depends on its antecedent and therefore must lead to an addition to the logic program after that antecedent has been established.

Now consider the query $\leftarrow d(\mathsf{true\ that}, \tau)$ in a context where (32d) was added to the database. Figure 4 gives a refutation tree,[19] that in a few steps leads to the query considered in Figure 1. Executing this query (or remembering its result) will establish that (32b) and (32c) determine the same set of worlds.

We have now set up a rudimentary mechanism dealing with reference to propositions and truth that will work in perfectly unspectacular ways in normal cases. But it will also lead to circular behaviour. Consider the Liar, formalized in (35).

(35) a. $\mathsf{false\ this}$
b. $\mathit{ant}(\mathsf{this}, \mathsf{false\ this})$

When $\leftarrow d(\mathsf{false\ this}, \tau)$ is evaluated against a database which contains (35b), a search as in Figure 5a results, with a variant of the original query reached in a few steps. A very similar behaviour results if the Truth-teller is evaluated.

[19] Note that here we have an example where adding *control* is necessary. It would not be a good idea to select the atom $d(\pi, \tau)$ from the query $\leftarrow \mathit{ant}(\mathsf{that}, \pi),\ d(\pi, \tau)$ in Figure 4 as this would lead to unnecessary indeterminism.

(36) a. $\mathsf{true\ this}$

b. $ant(\mathsf{this}, \mathsf{true\ this})$

The latter's formalization is given in (36) and a search tree is given in Figure 5b. A simple example of a *Liar cycle* is found in (37), with (37a) referring to (37b) and vice versa. If $\leftarrow d(\mathsf{true\ that}_0, \tau)$ is evaluated with (37c) and (37d) added to the database, circular behaviour will occur.

(37) a. $\mathsf{true\ that}_0$

b. $\mathsf{false\ that}_1$

c. $ant(\mathsf{that}_0, \mathsf{false\ that}_1)$

d. $ant(\mathsf{that}_1, \mathsf{true\ that}_0)$

A difference between the Liar and the Truth-teller is that it seems possible to arbitrarily assume that the Truth-teller is true (or false), while no such assumption is possible in case of the Liar. The present formalization agrees with this intuition, as adding an extra $d(\mathsf{true\ this}, \top)$ (or $d(\mathsf{true\ this}, \bot)$) to the database considered in (36) is perfectly possible (and leads to a one-step refutation of $\leftarrow d(\mathsf{true\ this}, \tau)$), but adding $d(\mathsf{false\ this}, \top)$ to the database for (35) leads to conflict with (14).

Thus the senses-as-queries view, when combined with mechanisms for referring to propositions and talking about truth, leads to normal results in normal cases and to circular results in cases of simple Liar-like sentences. Queries may diverge and propositions therefore may fail to determine a set of worlds. What about more elaborate forms of the Liar paradox such as the *Strengthened Liar* in (38)?

(38) This sentence is false or does not denote

Unlike the simple Liar, the strengthened Liar uses a word that is not part of the common vernacular but is technical, and it may well be that here we reach a limit of what is expressible in ordinary language. Can a language contain a word *denote* that is applicable to any statement of that language? We are used to be able to express about anything we like and this creates an illusion that the answer must be 'yes', but the senses-as-queries view strongly suggests a negative answer. After all, we know that, in general, there can be no program H that, given the description of any program Π, decides whether Π halts or not, i.e. the *Halting Problem* is undecidable. But since, in the senses-as-queries view, the question whether a sentence denotes is closely related to the question whether a certain query halts, we should not be surprised if this result carries over to natural language. If senses are queries then the limits of computation are also limits of language.

Suppose we had a *pp* term $\mathsf{denotes}$ with the property that, in any model M, $d(\mathsf{denotes}\ \Phi, \lambda i.\top)$ holds in M if there is a φ such that $d(\Phi, \varphi)$ holds in M and $d(\mathsf{denotes}\ \Phi, \lambda i.\bot)$ is true in M if there is no such φ. The usual diagonalization proof that shows the undecidability of the Halting Problem can be turned into a reductio ad absurdum of this assumption provided that meaning postulate (29e) is adopted. Consider (39).

(39) a. $(\mathsf{if}\,(\mathsf{denotes\ this}_1))(\mathsf{false\ this}_2)$

b. $ant(\mathsf{this}_2, \mathsf{false\ this}_2)$

c. $ant(\mathsf{this}_1, (\mathsf{if}\,(\mathsf{denotes\ this}_1))(\mathsf{false\ this}_2))$

It is clear that in any model satisfying (39b) and the previous meaning postulates (39a) will denote iff this_1 does not denote. Adding (39c) therefore immediately leads to inconsistency. We conclude that a term $\mathsf{denotes}$ with the required properties cannot be part of our system. The reductio follows that of the existence of a machine H deciding the halting problem by constructing a machine H' from it that halts iff its input does not halt ((39a) with (39b) added to the database) and then feeding H' to itself (adding (39c)).

8. Conclusion

We have shown how a relatively standard fragment of logical grammar can deal with the foundational problems of intentionality and circularity if propositions are accepted as primitive objects, in Thomason's way, and if the determination relation which associates propositions with sets of possible worlds is axiomatized by means of a logic program. Such a set-up pushes part of the usual Tarski-style interpretation procedure from the metalevel of the logic to the object level and allows us to interpret propositions as queries. Often the result of a query will be some set of possible worlds, but in some cases the query will not lead to a result because it diverges. Distinct queries can lead to the same result and identity criteria on queries can be very strict, thus leading to an intentional (hyperintensional) semantics.

A third possibility for queries, besides diverging or duly returning an answer, we have not gone into because it did not fall within the boundaries drawn up for this paper. This is the possibility of *aborting with error* because necessary preconditions for computation have not been satisfied. This possibility corresponds to the notion of *presupposition* in natural language but its treatment within (a liberalized version of) the present framework will have to await future work.

Acknowledgements

This paper is based on talks held at the *Sinn und Bedeutung* workshop in Osnabrück 2001 and the *Sinn und Bedeutung* workshop in Konstanz 2002. I would like to thank my audiences there for asking the right questions. Two anonymous referees gave valuable comments.

References

Andrews, P.: 1971, 'Resolution in Type Theory'. *Journal of Symbolic Logic* **36**(3), 414–432.

Apt, K.: 1990, 'Introduction to Logic Programming'. In: J. van Leeuwen (ed.): *Handbook of Theoretical Computer Science*, Vol. B. Amsterdam: Elsevier, pp. 495–574.

Barwise, J.: 1997, 'Information and Impossibilities'. *Notre Dame Journal of Formal Logic* **38**(4), 488–515.

Barwise, J. and J. Etchemendy: 1987, *The Liar: An Essay on Truth and Circularity.* New York, N.Y.: Oxford University Press.

van Benthem, J.: 1988, 'The Semantics of Variety in Categorial Grammar'. In: W. Buszkowski, W. Marciszewski, and J. v. Benthem (eds.): *Categorial Grammar.* Amsterdam: John Benjamins, pp. 37–55.

Benzmüller, C., C. E. Brown, and M. Kohlhase: 2004, 'Higher Order Semantics and Extensionality'. *Journal of Symbolic Logic* **69**, (to appear).

Blackburn, P., J. Bos, and K. Striegnitz: 2001, *Learn Prolog Now!* www.consem.org.

Carnap, R.: 1947, *Meaning and Necessity.* Chicago: Chicago UP.

Chierchia, G. and R. Turner: 1988, 'Semantics and Property Theory'. *Linguistics and Philosophy* **11**, 261–302.

Church, A.: 1940, 'A Formulation of the Simple Theory of Types'. *Journal of Symbolic Logic* **5**, 56–68.

Cooper, R.: 1983, *Quantification and Syntactic Theory.* Dordrecht: Reidel.

Cresswell, M.: 1972, 'Intensional Logics and Logical Truth'. *Journal of Philosophical Logic* **1**, 2–15.

Cresswell, M.: 1985, *Structured Meanings.* Cambridge, MA: MIT Press.

Dowek, G.: 2001, 'Higher-Order Unification and Matching'. In: A. Robinson and A. Voronkov (eds.): *Handbook of Automated Reasoning.* Amsterdam: Elsevier, pp. 1009–1062.

Dummett, M.: 1978, *Truth and Other Enigmas.* London: Duckworth.

Fitting, M.: 2002, *Types, Tableaus, and Gödels God.* Dordrecht: Kluwer Academic Publishers.

Fox, C. and S. Lappin: 2001, 'A Framework for the Hyperintensional Semantics of Natural Language with Two Implementations'. In: P. De Groote, G. Morrill, and C. Retoré (eds.): *Logical Aspects of Computational Linguistics.* Berlin, pp. 175–192, Springer-Verlag.

Fox, C., S. Lappin, and P. C.: 2002, 'A Higher-order Fine-grained Logic for Intensional Semantics'. In: G. Alberti, K. Balough, and P. Dekker (eds.): *Proceedings of the Seventh Symposium for Logic and Language.* Pecs, Hungary, pp. 37–46.

Goldfarb, W.: 1981, 'The Undecidability of the Second-order Unification Problem'. *Theoretical Computer Science* **13**, 225–230.

Hintikka, J.: 1975, 'Impossible Possible Worlds Vindicated'. *Journal of Philosophical Logic* **4**, 475–484.

Huet, G.: 1973, 'The Undecidability of Unification in Third-Order Logic'. *Information and Control* **22**, 257–267.

Kripke, S.: 1975, 'Outline of a Theory of Truth'. *Journal of Philosophy* **72**, 690–716.

van Lambalgen, M. and F. Hamm: 2003, 'Moschovakis' Notion of Meaning as Applied to Linguistics'. In: M. Baaz and J. Krajicek (eds.): *Logic Colloquium '01*, ASL Lecture Notes in Logic.

Lappin, S. and C. Pollard: 2000, 'Strategies for Hyperintensional Semantics'. ms.

Larson, R. and G. Segal: 1995, *Knowledge of Meaning.* Cambridge, MA: MIT Press.

Lewis, D.: 1972, 'General Semantics'. In: D. Davidson and G. Harman (eds.): *Semantics of Natural Language.* Dordrecht: Reidel, pp. 169–218.

Miller, D.: 1991, 'A Logic Programming Language with Lambda-abstraction, Function Variables, and Simple Unification'. *Journal of Logic and Computation* **1**, 497–536.

Montague, R.: 1970, 'Universal Grammar'. In: *Formal Philosophy.* New Haven: Yale University Press, pp. 222–246.

Montague, R.: 1973, 'The Proper Treatment of Quantification in Ordinary English'. In: *Formal Philosophy.* New Haven: Yale University Press, pp. 247–270.

Moschovakis, Y.: 1994, 'Sense and Denotation as Algorithm and Value'. In: *Logic Colloquium '90 (Helsinki 1990)*, Vol. 2 of *Lecture Notes in Logic.* Berlin: Springer, pp. 210–249.

Moschovakis, Y.: 2003, 'A Logical Calculus of Meaning and Synonymy'. Corrected and edited notes for a course in NASSLLI 2003.

Muskens, R.: 1991, 'Hyperfine-Grained Meanings in Classical Logic'. *Logique et Analyse* **133/134**, 159–176.

Muskens, R.: 2005, 'Higher Order Modal Logic'. In: P. Blackburn, J. van Benthem, and F. Wolter (eds.): *Handbook of Modal Logic*, Studies in Logic and Practical Reasoning. Dordrecht: Elsevier. (to appear).

Prawitz, D.: 1968, 'Hauptsatz for Higher Order Logic'. *Journal of Symbolic Logic* **33**(3), 452–457.

Rantala, V.: 1982, 'Quantified Modal Logic: Non-normal Worlds and Propositional Attitudes'. *Studia Logica* **41**, 41–65.

Takahashi, M.: 1967, 'A Proof of Cut-elimination Theorem in Simple Type Theory'. *Journal of the Mathematical Society of Japan* **19**(4), 399–410.

Thomason, R.: 1980, 'A Model Theory for Propositional Attitudes'. *Linguistics and Philosophy* **4**, 47–70.

Tichý, P.: 1988, *The Foundations of Frege's Logic.* Berlin: De Gruyter.

Turner, R.: 1987, 'A Theory of Properties'. *Journal of Symbolic Logic* **52**(2), 455–472.

Zalta, E.: 1997, 'A Classically-Based Theory of Impossible Worlds'. *Notre Dame Journal of Formal Logic* **38**(4), 640–660.