Nominalization, Predication and Type Containment

the Journal of Logic, Language and Information 2, 171-215, 1993.

Fairouz Kamareddine [*]
Department of Computing Science
17 Lilybank Gardens
University of Glasgow
Glasgow G12 8QQ
email: `fairouz@dcs.glasgow.ac.uk`

and

Ewan Klein[†]
Centre for Cognitive Science
2 Buccleuch Place
University of Edinburgh
Edinburgh EH8 9LW
email: `klein@cogsci.ed.ac.uk`

5 February 1993

[*]Kamareddine is grateful to the Department of Mathematics and Computing Science, Technical University of Eindhoven, for their financial support and hospitality during the academic year 1991–92.

[†]Klein's work has been carried out as part of the research programmes of the DYANA project (BR 3175 and BR 6852), funded by CEC ESPRIT Basic Research Division, and of the Human Communication Research Centre, supported by the UK Economic and Social Research Council.

Abstract

In an attempt to accommodate natural language phenomena involving nominalization and self-application, various researchers in formal semantics have proposed abandoning the hierarchical type system which Montague inherited from Russell, in favour of more flexible type regimes. We briefly review the main extant proposals, and then develop a new approach, based semantically on Aczel's notion of Frege structure, which implements a version of *subsumption polymorphism*. Nominalization is achieved by virtue of the fact that the types of predicative and propositional complements are contained in the type of individuals. Russell's paradox is avoided by placing a type-constraint on lambda-abstraction, rather than by restricting comprehension.

Keywords: Typed λ-Calculus, Russell's Paradox, Property Theory, Polymorphism, Natural Language Semantics.

1 Introduction

1.1 Overview

Type disciplines have featured prominently in formal approaches to natural language since the work of Montague (e.g., [Montague 73]). Montague avoided the paradoxes of naive set theory by adopting a version of Russell's cumulative hierarchy of types. Despite the successes of Montague's type system for English, it has met with criticism in recent years for being excessively rigid. One line of research, initiated by Partee and Rooth [Rooth *et al.* 82, Partee *et al.* 83], has tried to achieve greater flexibility, especially in the treatment of quantifiers, by assigning each expression a *family* of types. Another line of work has moved in the direction of type-free theories of properties, in order to accommodate the difficulties raised by nominalization and self-application. In this paper, we will focus our attention on the second of these two endeavours.

Historically, type disciplines for languages have developed in close association with intended models for interpretation. The proposals we shall make can also be construed in this way, inasmuch as they were inspired in part by Aczel's [Aczel 80] notion of a *Frege structure*, which is intended to provide a consistent formulation of Frege's logical notion of set.

The paper falls into four sections. The first of these presents some background notions, and briefly surveys the natural language data which motivates our formal analysis. Section 2 presents the syntax, types and inference rules for a language $\mathcal{L}_{\underline{\prec}}$, while Section 3 deals with the models of $\mathcal{L}_{\underline{\prec}}$. The final section shows how a fragment of English can be treated within our formal framework.

1.2 Hierarchical Types

A system of types provides a classificatory scheme for the domain and range of functors. The type of an expression determines the domain in which that expression receives an interpretation. Thus, in (1) (where we use the notation $\alpha{:}\sigma$ to mean that expression α has type σ), the proper noun **Glasgow** might be assigned type e, the type of entities, while the predicate **fun** is assigned type $\langle e, p\rangle$, which we construe as the type of objects which combine with expresions of type e to yield expressions of type p.

(1) Glasgow:e is fun:$\langle e, p\rangle$.

If we make the plausible assumption that the copular verb **is** here denotes the identity function on predicates, then standard rules of type inference yield the result that (1) is an expression of type p, the type of propositions.

In recent years, the semantic problems of nominalization in linguistically motivated type theories have received increasing attention, particularly as a result of the work of Bealer, Chierchia and Turner [Bealer 82, Chierchia 84, Chierchia 85, Chierchia and Turner 88, Turner 87]. To illustrate, notice that we might want to assign different types to different kinds of syntactic subjects, as shown in the following two examples:

(2) a. [Running around the lake]:$\langle e, p\rangle$ is fun:$\langle\langle e, p\rangle, p\rangle$

b. [For us to run around the lake]:p is fun:$\langle p, p\rangle$

In (2a), we might expect the gerundive subject phrase to denote a property, hence to be assigned type $\langle e, p\rangle$. But if (2a) is to be of type p, **fun** will require a new type, namely $\langle\langle e, p\rangle, p\rangle$. Similarly, if the subject of (2b) denotes a proposition, then the type of the predicate has to be changed to $\langle p, p\rangle$. Yet there is no independent linguistic motivation for postulating distinct lexical entries for the different **fun**s of each type.

A related problem arises when we consider cases of self-application, illustrated in (3a) and the simpler (though more artificial) instance (3b).[1]

(3) a. [Being fun]:$\langle e, p\rangle$ is fun:$\langle\langle e, p\rangle, p\rangle$

b. Fun:$\langle e, p\rangle$ is fun:$\langle\langle e, p\rangle, p\rangle$

Suppose we postulate a first-order predicate **fun**:$\langle e, p\rangle$, and a second order predicate of predicates **fun**:$\langle\langle e, p\rangle, p\rangle$. This allows us to deal with (3); but what happens if we want to affirm that **fun**:$\langle\langle e, p\rangle, p\rangle$ is fun? We are at the bottom of an infinitely ascending ladder of types:

(4) Fun:$\langle\langle e, p\rangle, p\rangle$ is fun:$\langle\langle\langle e, p\rangle, p\rangle, p\rangle$

There seem to be broadly three classes of response to these problems of 'type inflation': type-lowering, type-freedom, and polymorphism. We briefly consider these in turn.

Type-Lowering

We have just observed the potential difficulties which arise if the subject **running** in (5) is assigned the type $\langle e, p\rangle$ of verb phrases:

(5) Running hurts.

For then we are apparently forced to assign a correspondingly higher type to **runs**. The approach proposed by Chierchia (e.g., in [Chierchia 84]) postulates a nominalisation operator $^{\cap}$ which maps propositional functions (and propositions) into entities.[2] That is, if **run**$'$ (the semantic translation of **run**—we

[1]Despite appearances, such locutions are not entirely restricted to the discourse of theoreticians; the following sentence was noted in the *Times Higher Education Supplement* of 28th September 1990, p.17:

In fact, the fun of research is more fun than fun.

[2]One of the earliest discussions of treating propositional arguments in a Montague framework, namely Thomason [Thomason 76], adopts a similar type-lowering operation.

use Montague's prime notation for semantic constants) denotes a propositional function f, then $^\cap\mathbf{run}'$ is an expression of type e which denotes an individual correlated to f. We might assume that the morphological operation which relates the gerundive form **running** to the finite form **runs** has as its semantic counterpart the introduction of this $^\cap$ operator. The resulting semantic analysis is illustrated in (6):

(6) $\mathbf{hurt}'{:}\langle e,p\rangle(^\cap\mathbf{run}'{:}e)$

Type-Freedom

From a technical point of view, it is not necessary to explicitly map propositional functions into their individual correlates. Instead, we can regard all properties as being a special sort of individual. Following Aczel [Aczel 80], Bealer [Bealer 82] and others, properties are those first-order objects which can be applied—using an explicit operation *app* of predication—to other objects so as to yield a proposition. This first-order approach is illustrated in (7):

(7) $\mathit{app}(\mathbf{hurt}'{:}e,\ \mathbf{run}'\ {:}e){:}p$

Although we have declared the types of the expressions in (7), they serve little purpose, since none of them are functional in nature.

Polymorphism

We say that a function is *polymorphic* if it yields appropriate outputs for inputs of a variety of types. There are at least two notions of polymorphism which can be invoked to deal with these problems. The first, called *parametric* polymorphism (cf. [Cardelli *et al.* 85]), obtains polymorphic types by admitting type variables. In Milner's [Milner 78] approach, as implemented for the programming language ML, types containing type variables are called *generic*. Suppose, for example, that v is a type variable, and that we assign to **fun** the generic type $\langle v,p\rangle$. What happens when we try to determine the type of an expression involving self-application like **fun(fun)**? Assuming that the second occurrence of **fun** has the most general type (i.e.,$\langle v,p\rangle$), the first occurrence will have to be assigned a more complex type, namely $\langle\langle v,p\rangle,p\rangle$, where the type variable v has itself been instantiated as $\langle v,p\rangle$. Although we are required to assign different types to functor and argument in such a case, it should be noted that the complexity of a functor's type is no greater than that required by the most general type of its argument; thus we avoid the 'infinitely ascending ladder of types' alluded to in our discussion of strictly hierarchical type systems. An approach similar in spirit to ML is adopted by Parsons [Parsons 79], where Montague's framework is modified to allow 'floating' types which again contain type variables. Although Parsons considers an interesting range of data, he does not explicitly discuss problems of nominalization.

A different route avoids type variables by using something which [Cardelli *et al.* 85] call *inclusion* polymorphism. Suppose, for example, that σ_1, σ_2, and τ are types such that $\sigma_2 \preceq \sigma_1$, i.e., σ_2 is subsumed by, or *contained* in (cf. [Mitchell 88]), σ_1, and let f be a functor of type $\langle \sigma_1, \tau \rangle$. Suppose further that α is a term, not of type σ_1, but of the more specific type σ_2. Then f is polymorphic in the sense that it can apply to α, and yields a value of type τ. From a semantic point of view, we model a type σ as a set D_σ of values, and containment as inclusion between such sets. Now if a function assigns values to members of a particular set D_{σ_1}, then it will also assign values to members of any subset D_{σ_2} of D_{σ_1}. How does this help us deal with nominalization? If we let the type $\langle e, p \rangle$ of predicates be contained within the type e of individuals, then, for example, **fun** of type $\langle e, p \rangle$ can apply to any expression of type $\sigma \preceq e$, including **fun** itself.

1.3 Individuals, properties and functions

Our treatment takes subsumption polymorphism as a starting point—that is, we will develop a notion of type containment, but avoid type variables. In fact, the formal framework that we develop is flexible enough to encompass a range of different approaches to nominalization, including type-free ones. However, within the space of options, we have made certain theoretical choices which allow us to model certain linguistic generalizations. In this section, therefore, we will consider some of the motivating data.

In order not to prejudge the issues to be decided, we use the term *propositional functor* to refer to any expression f of English which can combine with an argument a so that the result $f(a)$ is a declarative sentence, i.e., capable of being used to assert a proposition. Thus, a finite verb phrase such as **walks** is a propositional functor, as is a declarative sentence lacking a direct object, such as **John annoys ___**. We assume that propositional functors denote propositional *functions*, though just what these are supposed to be is left till later.

We will use the more neutral term *predicative* to cover both propositional functors and words or phrases which intuitively express properties but which cannot combine with other expressions to make sentences. Again, we leave till later what the denotation of predicatives is, if not propositional functions.

The first generalization which we wish to capture is:

Claim 1.1 *Predicative expressions can appear in the position of noun phrase (NP) arguments to propositional functors.*

For example, predicatives can occur in subject position of tensed sentences, i.e., a position which is typically occupied by *NP*s:

(8) a. *To run* will tire Mary.
    b. *Running* annoys Mary.

Thus, according to our terminology, (8a) contains two predicatives, **to run** and **will tire Mary**; the latter is, in addition, a propositional functor.

It can also be observed that the distribution of predicatives sometimes extends beyond that of *NP*s. Thus we have:

Claim 1.2 *Predicative expressions can appear as arguments to propositional functors where NPs are prohibited.*

In particular, certain lexical items are subcategorized to require predicative arguments, as opposed to ordinary noun phrases. The examples in (9) contrast with those in (10):

(9) a. John seems *to annoy Mary/happy*
  b. With John *annoying Mary/happy/in love*, we can stop worrying.
  c. Mary saw John *run/running/happy*

(10) a. *John seems *that boy*
  b. *With John *that boy*, we can stop worrying.
  c. *Mary saw John *that boy*

It might be claimed that this patterning of data is purely syntactic. Certainly, it is true that items which require predicatives are usually subcategorized to take only a subset thereof. Thus, **seems** takes infinitival complements but not bare or gerundive *VP*s, while **see** patterns the opposite way. Despite these idiosyncracies, however, there are a variety of generalizations that can only be expressed on the assumption that the class of predicatives can be somehow picked out (cf. [Bach 79], [Pollard *et al.* 87]). It would be desirable to give a semantic characterization of this class, rather than just invoking an arbitrary syntactic feature. As we will discuss later, our attempt to meet this criterion is only partially successful.

The next two claims have been particularly emphasized by Chierchia [Chierchia 85, Chierchia and Turner 88]. Recall Frege's view that a (propositional) function is 'unsaturated', or requires completion by an argument. On completion, the function yields a value, e.g., a proposition. Changing perspective slightly, we can say that only functions have the combinatorial potential to 'glue together' with arguments. The individual correlate of a function, by contrast, is 'inert': it cannot by itself combine with an argument to produce a value. Translated into the realm of grammar, we have:

Claim 1.3 *Tensed predicative expressions are propositional functors, but untensed predicatives are not.*

Thus, the examples in (11) do not express assertible propositions, whereas those in (12) do:

(11) a. *John *to run.*
  b. *John *(be) happy.*

(12) a. John *runs.*

b. John *is happy.*

This claim, though attractive, seems to require modification when embedded infinitives are considered. Thus, [Jacobson 90] has drawn attention to data like

(13) Everyone likes their tea to be hot.

The crucial question about such an example is whether the substring **their tea to be hot** is an infinitival sentence (as opposed to a sequence of two distinct complements of **like**). Evidence in favour of it being a single constituent is provided by standard tests:

(14) a. What everyone likes is their tea to be hot.

b. Everyone likes their tea to be hot and their beer to be cold.

Despite these examples, the fact that nonfinite verbs cannot combine directly with subjects in root clauses still requires explanation. In the present paper, therefore, we shall maintain Claim 1.3 as it stands, while accepting that further analysis of the issues is called for.

The fourth claim can be regarded as a further specification of Claim 1.1. Chierchia suggests that it is an empirical generalization which holds for many, if not all, natural languages:

Claim 1.4 *Tensed predicative expressions cannot occur as arguments of propositional functors.*

Thus, ungrammaticality results if we attempt to replace the untensed predicatives in our previous examples by tensed predicatives:

(15) a. **Runs* annoys Mary

b. *John seems *annoys Mary/is happy*

c. *John tries *annoys Mary/is happy*

Let us now consider how these observations might be rendered in a formal framework. The generally accepted interpretation of Claim 1.1 is that propositional functions have individual correlates. As a further terminological step, let us use the term *nominal predicatives* to refer to expressions which denote such individual correlates.[3] We make the standard assumption that a model determines a universe of individuals. However, this universe contains a greater diversity of objects than is usual in first order models; for example, it will contain all propositions as a subcollection. Following [Aczel 80], $\mathcal{F}_0$ is collection

[3]Although we will also follow standard practise in saying that such expressions have been 'nominalized', this is something of a misnomer inasmuch as we do not need to postulate any type or category changing operation.

of objects, and $\mathcal{F}_1$ is the collection of unary functions from $\mathcal{F}_0$ to $\mathcal{F}_0$. Within $\mathcal{F}_1$, we can identify the subcollection of propositional functions, i.e., functions from $\mathcal{F}_0$ to propositions, and this we call PF. It turns out that $\mathcal{F}_0$ is also 'big enough' to contain, for each function from objects to objects, an object that corresponds to that function. We can then implement the idea of individual correlates by letting the collection PF be explicitly mapped, *via* the $\boldsymbol{\lambda}$ operator, onto a subcollection SET of the domain of $\mathcal{F}_0$. That is, each object in SET is the individual correlate of a propositional function. (See Lemma 6 for a proof that $\boldsymbol{\lambda}$ is bijective.)

Claim 1.2 shows that some lexical items select as their arguments nominal predicatives. However, any solution to this is closely tied up with the problem of capturing the difference in combinatorial potential between tensed and untensed predicatives, as required by Claim 1.3. For on the one hand, we would like to say that a nominal predicative is the sort of thing which potentially combines with an argument; on the other hand, it can only do this under special circumstances, for example under the mediation of tense.

Let us be more concrete. If we assign the type $\langle e, p \rangle$ to nominal predicatives, then it is difficult to avoid the conclusion that such expressions should combine with arguments of type e to yield a result of type p, i.e, a proposition. If on the other hand we follow [Chierchia and Turner 88] in assigning them the type *nf* of nominalized functions, then it becomes hard to express the fact that there can be semantic constraints on the type of the argument which a predicative selects when it becomes 'denominalized'. Suppose, for example, that we wanted to define a subclass of untensed intransitive verbs which select for propositional subjects; it would be desirable to assign them the type $\langle p, p \rangle$ (which we could treat as a subtype of $\langle e, p \rangle$).

A third option, and the one we shall adopt, is to give nominal predicatives the type $\langle e, e \rangle$. This makes it clear that such expressions do select arguments, possibly of some proper subtype of e. At the same time, it does not claim that such expressions can combine with their arguments to make *propositions*. Instead, this type remains 'agnostic' about the precise nature of the resulting combination; we know that it is an object, but in the absence of further information, can neither affirm nor deny that the result is a proposition.[4]

As pointed out by Chierchia and Turner [Chierchia and Turner 88], the observation that propositional functions cannot act directly as arguments appears to be inadequately captured by first-order theories of properties such as that of Bealer [Bealer 82] in which propositions only result by virtue of explicitly applying a property to another object. For example, on such an approach, **John walks** would be expressed as (16):

(16) $\quad app(\mathbf{walk}'{:}e, \mathbf{john}'{:}e)$

[4]This is essentially Aczel's [Aczel 80] analysis of the Russell property—although the property is expressible in his system, the result of applying it to another object, including itself, is not provably a proposition.

The Fregean view (which Bealer [Bealer 82, Bealer 89] rejects) holds that propositional functions should not be thought of as objects, but indeed as functions. This is reflected in our framework, therefore, by the decision to view propositional functions as elements of PF, not $\mathcal{F}_0$. This has the virtue of providing a natural explanation for Claim 1.4. For although elements of PF do have individual correlates in $\mathcal{F}_0$, they are not themselves objects, and as such are not potential arguments for other propositional functions.

As we will see, 'nominal' types (including predicative nominals) are all constructed as subtypes of e. Since, according to what we have just said, propositional functors are not nominals, they cannot be assigned a nominal type. We therefore require a new kind of type for such functors, one which is *not* a subtype of e. Expressions whose denotations lie outside the domain $\mathcal{F}_0$ of objects will be assigned what we call *metatypes*. Whenever σ and τ are (meta-)types, $(\sigma \rightarrow \tau)$ will be a metatype. Note that we will not need to quantify over propositional functions, nor will we need λ-expressions whose domain of interpretation is the collection of propositional functions—we can use nominalized properties instead. Hence, variables in our language will never be assigned metatypes.

We shall assume that uninflected (or base form) verb phrases denote objects rather than propositional functions; for example, **walk** will be of type $\langle e, e\rangle$. When verb phrases receive tense, they are mapped by a predication operator $^\cup$ into propositional functions, with the metatype $(e \rightarrow p)$. Thus if nonfinite **walk** translates as $\mathbf{walk}'{:}\langle e, e\rangle$, then tensed **walks** translates as $^\cup\mathbf{walk}'{:}(e \rightarrow p)$. Putting the various pieces together, we replace (16) with (17), where the propositional functor is applied directly to its argument, rather than by the mediation of *app*:

(17) $[^\cup\mathbf{walk}'{:}(e \rightarrow p)(\mathbf{john}'{:}e)]{:}p$

By way of summary, we give the following tabular presentation of our articulation of the data. Note that our earlier notion of 'predicative' is now divided into two.

Syntactic Notion	**Semantic Notion**	**(Meta-)Type**	**Example**
propositional functor	function from $\mathcal{F}_0$ to propositions	$(e \rightarrow p)$	**walks**, **is fun**
nominal predicative	subcollection of $\mathcal{F}_0$	$\langle e, e\rangle$	**walk**, **be fun**

In this section, we have attempted to present and motivate the general structure of our approach, and it will be observed that we have followed [Chierchia and Turner 88] closely in favouring a Fregean analysis over a first order property theory. Nevertheless, our formal framework differs from that of [Chierchia and Turner 88] in many respects; this will become obvious in the following sections, where we give a more systematic presentation of the theory.

2 The Language $\mathcal{L}_{\preceq}$

2.1 Judgements and Type Containment

In the theory $\mathcal{L}_{\preceq}$ developed in this paper, we follow [Aczel 80] in starting from models of the type-free lambda calculus, on top of which an interpretation for logical connectives has been constructed; we then construct types within the set of objects. In place of the domain $\{0, 1\}$ of truth values, we have a domain PROP of propositions, included in which is the domain TRUTH of true propositions. These collections provide values for the types p and t respectively. As mentioned earlier, there is also a domain $\mathcal{F}_0$ of individuals, with associated type e. This domain turns out to be much richer than one might have expected. Indeed, it contains PROP (and hence TRUTH) as subcollections. In Section 3, we shall look in more detail at the intended models; for the time being, however, we present the type structure.

Following usual practice in type theory (e.g., [Cardelli *et al.* 85], [Mitchell 88]), we use a natural deduction format for rules of type inference. A simple example is the following:

$$\frac{\vdash \varphi{:}p}{\vdash \exists x{:}\sigma.\varphi{:}p}$$

The statement $\vdash \varphi{:}p$ is an assertion or *judgement* meaning that we can infer that φ is of type p. The rule as a whole is a logical implication; given the premiss, we can infer that $\exists x.\varphi$ is also of type p.

What we have presented is not quite sufficient, however; if φ contains occurrences of the variable x, the inference that it is of type p may in turn depend on the type of x; in other words, the judgement is made under the assumption, or in the context, $x{:}\sigma$. Using

$$\Gamma, x{:}\sigma$$

to represent a context Γ which contains the relevant assumption, we replace our earlier rule by the following:

$$\frac{\Gamma, x{:}\sigma \vdash \varphi{:}p}{\Gamma \vdash \exists x{:}\sigma.\varphi{:}p}$$

Let us now present these ideas in a more systematic format. A *type statement* is a pair, written $\alpha{:}\sigma$, consisting of an expression α and a type σ, read "α has type σ"; α is said to be the *subject* of the statement. A *signature* Σ is a finite set of distinct type statements the subjects of which are constants, while a *context* Γ is a finite set of distinct type statements, the subjects of which are variables or sentences. In the latter case, a statement of the form $\varphi{:}t$ indicates that φ is a sentence of the logic whose truth is being assumed in the course of a proof; that is, we are also using contexts in a sequent calculus style to encode the current set of assumptions required at each line of a proof.

As usual, we can regard signatures and contexts as functions from expressions to types. Thus, $dom(\Sigma)$ denotes the set of expressions to which the signature Σ assigns a type, and similarly for contexts. If $\mathcal{A}$ is a signature or a context, we write $\mathcal{A}, \alpha{:}\sigma$ in place of $\mathcal{A} \cup \{\alpha{:}\sigma\}$.

Although the system used here does not use the power of higher-order type theory (e.g., such as dependent types), we have nevertheless found it convenient to take as our framework the theory of expressions developed in the Edinburgh Logical Framework [Harper *et al.* 87]. As pointed out in the preceding section, we distinguish types, whose interpretations are constructed within the domain of objects, from metatypes, which have a disjoint interpretation as collections of functions and functionals. Types and metatypes are both *kinds*.

We need three further kinds, or classifications of types: *fixed point types* (fp-types) *well-behaved types* (wb-types) and *non-propositional types.* The latter have the characteristic that their leftmost type is e, hence we use le-type for short. All these types are all interpreted within the domain of objects. As we shall see later, there is a sense in which an fp-type is a complex type which does not have any proper subtypes.

We will use σ and τ for types, $m\sigma$ for metatypes, and η, η_1, η_2 to range over both types and metatypes. We use c for constants (a special instance of which is $\perp$), x, y for variables, α, β for arbitrary object language expressions and φ, ψ, χ for expressions which denote propositions. We use $\Gamma \vdash s$ to mean that s is derivable within context Γ, and $\Gamma \vdash_\Sigma s$ to mean that s is derivable from the signature Σ within context Γ. $\vdash s$ and $\vdash_\Sigma s$ stand respectively for $\emptyset \vdash s$ and $\emptyset \vdash_\Sigma s$, where $\emptyset$ is the empty context.

The syntax of the various sorts of expression can now be specified as follows:

Signatures	Σ	$::=$	$\emptyset \mid \Sigma, c{:}\eta$
Contexts	Γ	$::=$	$\emptyset \mid \Gamma, x{:}\sigma \mid \Gamma, \alpha{:}\ t$
Kinds	K	$::=$	*type* | *fp-type* | *le-type* | *wb-type* | *metatype*
Types	σ	$::=$	$e \mid t \mid p \mid \langle \sigma, \tau \rangle$
Metatypes	$m\sigma$	$::=$	$(\eta_1 \to \eta_2)$
Expressions	α	$::=$	$c \mid x \mid \lambda x{:}\sigma.\alpha \mid app(\alpha, \beta) \mid \alpha(\beta) \mid \neg\alpha \mid {}^{\cup}\alpha \mid [\alpha \wedge \beta]$
			$\mid [\alpha \vee \beta] \mid [\alpha \supset \beta] \mid [\alpha = \beta] \mid \forall x{:}\sigma.\alpha \mid \exists x{:}\sigma.\alpha$

We will omit square brackets around complex sentences except in those cases where the scope of a typing statement needs to be made explicit.

Type theory (cf. [Martin-Löf 79]) provides rules for making judgements of various forms. The ones which we are concerned with are the following:

Judgements

$\vdash \Sigma\ sig$	Σ *is a signature*
$\vdash_\Sigma \Gamma\ context$	Γ *is a context*
$\Gamma \vdash_\Sigma \eta\ K$	η *has kind* K
$\Gamma \vdash_\Sigma \sigma \preceq \tau$	*type* σ *is contained in type* τ
$\Gamma \vdash_\Sigma \sigma \approx \tau$	*type* σ *is equivalent to type* τ
$\Gamma \vdash_\Sigma \alpha{:}\sigma$	α *has type* σ

Examples of $\Gamma \vdash_\Sigma \eta\ K$ are: $\Gamma \vdash_\Sigma p$ type
$\Gamma \vdash_\Sigma (e \rightarrow p)$ metatype
$\Gamma \vdash_\Sigma \langle\langle e, e\rangle, e\rangle$ le-type

Note that the $\approx$ relation between types is the symmetric closure of $\preceq$, the containment relation.

We mentioned earlier that the inference rules by which judgements can be derived are formulated in natural deduction notation. We add glosses to a representative sample of the rules in order to help readers not familiar with this mode of presentation.

Valid Signature

$$(null\ \ sig\,) \qquad \frac{}{\vdash \emptyset\ \ sig}$$

The empty relation is a signature.

$$(:sig) \qquad \frac{\vdash \Sigma\ sig \qquad \vdash_\Sigma \eta\ K}{\vdash \Sigma, c{:}\eta\ sig} \quad \textit{if } c \notin\ \ dom\ (\Sigma)$$

If η is a kind, and Σ doesn't already assign a (meta-)type to the constant c, then we can augment Σ with the statement $c{:}\eta$.

Valid Context

$$(null\ \ context\,) \ \frac{\vdash \Sigma\ sig}{\vdash_\Sigma \emptyset\ \ context}$$

$$(:context) \qquad \frac{\vdash_\Sigma \Gamma\ context \qquad \Gamma \vdash_\Sigma \sigma\ type}{\vdash_\Sigma \Gamma, x{:}\sigma\ context} \quad \textit{if } x \notin\ \ dom\ (\Gamma)$$

$$(:truthcontext) \ \frac{\vdash_\Sigma \Gamma\ context}{\vdash_\Sigma \Gamma, \varphi{:}t\ context} \quad \textit{if } \varphi \notin\ \ dom\ (\Gamma)$$

We mentioned earlier that the type statements in a *context* have subjects which are variables or sentences. As far as the former is concerned, it can be seen that (: *context*) requires σ to be a type, not an arbitrary kind; thus, our contexts will not assign metatypes to any variables. As far as the latter kind of statement is concerned, we obseve that although we can derive the judgement $\varphi{:}p$, we do not require contexts which contain it; hence (:*truth context*) is sufficient.

As we pointed out above, the following semantic domains are ordered by inclusion:

$$\begin{array}{l} \text{TRUTH} \subseteq \text{PROP} \subseteq \mathcal{F}_0 \\ \text{SET} \subseteq \mathcal{F}_0 \end{array}$$

And indeed there are other inclusions in the domains. This structure is reflected by the *containment* relation $\preceq$ (in fact, a partial order) which is imposed on the types. When $\sigma \preceq \tau$, we say that σ is *contained in*, or is a *subtype of*, τ. $\sigma \preceq \tau$ means that any expression which is of type σ is also of type τ; moreover, any object in the model which belongs to the domain D_σ associated with σ also belongs to the domain D_τ associated with τ. The most salient containments in our system are the following:

$$\begin{array}{cccccc} t & \preceq & p & \preceq & e \\ & & \langle\sigma,\tau\rangle & \preceq & e \end{array}$$

Rules for inferring judgements about containment will be given shortly. Before that, however, we present the various kinds required.

Kinds, Types and Metatypes

$$(\textit{base types}) \quad \frac{\vdash\Sigma\ \textit{sig} \qquad \vdash_\Sigma \Gamma\ \textit{context}}{\Gamma\vdash_\Sigma e\ \textit{type}}$$

$$\frac{\vdash\Sigma\ \textit{sig} \qquad \vdash_\Sigma \Gamma\ \textit{context}}{\Gamma\vdash_\Sigma t\ \textit{type}}$$

$$\frac{\vdash\Sigma\ \textit{sig} \qquad \vdash_\Sigma \Gamma\ \textit{context}}{\Gamma\vdash_\Sigma p\ \textit{type}}$$

$$(\textit{complex types}) \quad \frac{\Gamma\vdash_\Sigma \sigma\ \textit{type} \qquad \Gamma\vdash_\Sigma \tau\ \textit{type}}{\Gamma\vdash_\Sigma \langle\sigma,\tau\rangle\ \textit{type}}$$

$$(\textit{le base}) \quad \frac{\vdash\Sigma\ \textit{sig} \qquad \vdash_\Sigma \Gamma\ \textit{context}}{\Gamma\vdash_\Sigma e\ \textit{le-type}}$$

$$(\textit{le complex}) \quad \frac{\Gamma\vdash_\Sigma \sigma\ \textit{le-type} \qquad \Gamma\vdash_\Sigma \tau\ \textit{type}}{\Gamma\vdash_\Sigma \langle\sigma,\tau\rangle\ \textit{le-type}}$$

$$(\textit{wb-types}) \quad \frac{\Gamma\vdash_\Sigma \tau\ \textit{le-type} \qquad \Gamma\vdash_\Sigma \sigma\ \textit{type} \qquad \Gamma\vdash_\Sigma \tau\preceq\sigma}{\Gamma\vdash_\Sigma \langle\sigma,\tau\rangle\ \textit{wb-type}}$$

$$(\textit{fp-types}) \qquad \frac{\Gamma\vdash_\Sigma \sigma \ \textit{type} \qquad \Gamma\vdash_\Sigma \tau \preceq p}{\Gamma\vdash_\Sigma \langle\langle\sigma,\tau\rangle,\tau\rangle \ \textit{fp-type}}$$

If σ is a type and τ is contained in p (that is, $\tau = t$ or p), then $\langle\langle\sigma,\tau\rangle,\tau\rangle$ is an fp-type.

$$(\textit{metatypes}) \qquad \frac{\Gamma\vdash_\Sigma \eta_1 \ K \qquad \Gamma\vdash_\Sigma \eta_2 \ K}{\Gamma\vdash_\Sigma (\eta_1 \rightarrow \eta_2) \ \textit{metatype}}$$

Here are some examples of le-types: e, $\langle e,e\rangle$, $\langle e,p\rangle$, $\langle e,\langle e,e\rangle\rangle$, $\langle e,\langle e,p\rangle\rangle$, $\langle\langle e,e\rangle,e\rangle$, $\langle\langle e,p\rangle,e\rangle$, and so on. We claimed earlier that the 'leftmost' type of such an le-type must be e. The following lemmas show that this property does indeed follow from the definitions we gave earlier.

Lemma 1 *If σ is an le-type, then the leftmost type of σ, call it* leftmost*(σ), is e.*

Proof *By induction on the judgement σ le-type.*

- *If σ is basic, then leftmost(σ) $= \sigma = e$.*
- *If $\langle\sigma,\tau\rangle$ is an le-type then by (le-complex), σ is an le-type. By hypothesis, the property holds of σ. So leftmost($\langle\sigma,\tau\rangle$) $=$ leftmost(σ) $= e$.* □

As we will see later, λ-abstraction will only be permitted when the type of the resulting abstract is a wb-type. A complex type $\langle\sigma,\tau\rangle$ is a wb-type just in case the range type τ is an le-type which is contained in the domain type σ. For example, $\langle e,e\rangle$, $\langle e,\langle e,e\rangle\rangle$, $\langle e,\langle e,p\rangle\rangle$ and $\langle\langle e,p\rangle,\langle e,p\rangle\rangle$ are wb-types. However, $\langle e,p\rangle$ is not a wb-type, because p is not an le-type, while $\langle p,e\rangle$ is not because e is not contained in p. A consequence of our condition on λ-abstraction is that we cannot form abstracts such as $\lambda x{:}e.app(a,x)$ where $app(a,x)$ is provably of type p. This might seem overly restrictive. However, it turns out that for most purposes, we need only to consider cases where $app(a,x)$, say, is provably of type e.

The containment relation is governed by the following conditions:[5]

Containment

$$(e\preceq) \qquad \frac{\Gamma\vdash_\Sigma \sigma \ \textit{type}}{\Gamma\vdash_\Sigma \sigma\preceq e}$$

Objects in the domain D_σ of any type σ are also in D_e.

$$(p\preceq) \qquad \frac{\vdash_\Sigma \Gamma \ \textit{context}}{\Gamma\vdash_\Sigma t\preceq p}$$

[5]For a similar proposal, see [Curien *et al.* 89].

Truths are propositions.

$$(Dom{\preceq})\qquad \frac{\Gamma\vdash_\Sigma \sigma_1{\preceq}\sigma_2}{\Gamma\vdash_\Sigma \langle\sigma_2,e\rangle{\preceq}\langle\sigma_1,e\rangle}$$

Every function (returning arguments in D_e) defined on a domain D_{σ_2} is also defined on subsets D_{σ_1} of D_{σ_2}.

$$(Ran{\preceq})\qquad \frac{\Gamma\vdash_\Sigma \sigma\ \textit{type}\qquad \Gamma\vdash_\Sigma \tau_1{\preceq}\tau_2}{\Gamma\vdash_\Sigma \langle\sigma,\tau_1\rangle{\preceq}\langle\sigma,\tau_2\rangle}$$

Every function with values in the range D_{τ_1} also yields values in supersets D_{τ_2} of D_{τ_1}.

$$(Id{\preceq})\qquad \frac{\Gamma\vdash_\Sigma \sigma\ \textit{type}}{\Gamma\vdash_\Sigma \sigma{\preceq}\sigma}$$

$$(Trans{\preceq})\qquad \frac{\Gamma\vdash_\Sigma \sigma{\preceq}\tau\qquad \Gamma\vdash_\Sigma \tau{\preceq}\rho}{\Gamma\vdash_\Sigma \sigma{\preceq}\rho}$$

$$(Anti{\preceq})\qquad \frac{\Gamma\vdash_\Sigma \sigma{\preceq}\tau\qquad \Gamma\vdash_\Sigma \tau{\preceq}\sigma}{\Gamma\vdash_\Sigma \sigma\approx\tau}$$

$$(Fix{\preceq})\qquad \frac{\Gamma\vdash_\Sigma \langle\langle\sigma,\tau\rangle,\tau\rangle\ \textit{fp-type}}{\Gamma\vdash_\Sigma \langle\langle\sigma,\tau\rangle,\tau\rangle{\approx}\langle\sigma,\tau\rangle}$$

We now prove some simple lemmas which help us to establish relationships between the different categories of types in our system.

Lemma 2 *The only judgement $e \preceq \tau$ is: $e \preceq e$.*
Proof *By induction on the definition of $\preceq$.*

- *The cases ($e \preceq$), ($p \preceq$) and ($Id \preceq$) are obvious.*
- *Of the recursive clauses, the only relevant one is ($Trans \preceq$), where σ is e and $e \preceq \rho$ is derived from $e \preceq \tau$ and $\tau \preceq \rho$. But by induction hypothesis, τ must be e. Now again from the induction hypothesis and $e \preceq \rho$ we derive that ρ must be e.* □

As a corollary of the Lemma 2, we obtain the result that $e \preceq p$ is not derivable.

Lemma 3 *If $\langle\sigma,\tau\rangle \preceq \rho$, then either $\rho = e$ or ρ is a complex type.*
Proof *By induction on the definition of $\preceq$.*

- *Case (e $\preceq$) is obvious.*
- *Cases (p $\preceq$), (Dom $\preceq$) and (Ran $\preceq$) do not apply.*
- *Case (Id $\preceq$) is obvious.*
- *Case (Trans $\preceq$), then*

$$\frac{\Gamma\vdash_\Sigma \langle\sigma,\tau\rangle\preceq\rho \qquad \Gamma\vdash_\Sigma \rho\preceq\rho'}{\Gamma\vdash_\Sigma \langle\sigma,\tau\rangle\preceq\rho'}$$

  By the induction hypothesis, either $\rho = e$ or ρ is complex. If $\rho = e$, we have that $e \preceq \rho'$, but then $\rho' = e$, as required. If, on the other hand, ρ is complex, then by the induction hypothesis, from $\rho \preceq \rho'$, we again conclude that $\rho' = e$ or ρ' is complex.

- *Cases (Anti $\preceq$) and (Fix $\preceq$) do not apply.* $\square$

Lemma 4 *If $\sigma \preceq p$, then σ is not le-type. (Or conversely, if σ is le-type, then $\sigma \not\preceq p$.)*

Proof *By induction on the judgement σ le-type.*

- *Case $\sigma = e$, then we use the corollary of Lemma 2.*
- *Cases $\sigma = \langle\sigma_1,\sigma_2\rangle$, where σ_2 is le-type, then from Lemma 3, $\langle\sigma_1,\sigma_2\rangle \not\preceq p$, because $\langle\sigma_1,\sigma_2\rangle \preceq p$ only if $p = e$ or p is complex.* $\square$

The axiom (*Fix* $\preceq$) gives us fixed points for type containment. That is, if $\tau \preceq p$, then $\langle\sigma,\tau\rangle \approx \langle\langle\sigma,\tau\rangle,\tau\rangle \approx \langle\langle\langle\sigma,\tau\rangle,\tau\rangle,\tau\rangle \ldots$. While types such as $\langle e,e\rangle$, $\langle\langle e,e\rangle,e\rangle$, $\langle\langle\langle e,e\rangle,e\rangle,e\rangle$, …are distinct, we need to be more restrictive about types such as $\langle e,p\rangle$, $\langle\langle e,p\rangle,p\rangle$, …if we are to avoid the paradoxes. According to (*Dom* $\preceq$), since $\langle e,p\rangle \preceq e$, we should have $\langle e,p\rangle \preceq \langle\langle e,p\rangle,p\rangle$. The intuition behind calling $\langle\langle e,p\rangle,p\rangle$ an fp-type is that this containment is not proper; that is, we cannot get anything extra by going from $\langle e,p\rangle$ to $\langle\langle e,p\rangle,p\rangle$. In other words, we can only map sets into propositions to the extent that we map those sets *qua* objects into propositions.

There is a complementarity between le-types and fp-types, in the following sense. Recall that for $\langle\langle\sigma,\tau\rangle,\tau\rangle$ to be an fp-type, we require $\tau \preceq p$. Now if $\Gamma\vdash_\Sigma \tau$ *le-type*, we can conclude that $\Gamma\not\vdash_\Sigma \langle\langle\sigma,\tau\rangle,\tau\rangle$ *fp-type*; for example, $\langle\langle\sigma,e\rangle,e\rangle$ is not an fp-type. The reason is that if τ is an le-type, then by Lemma 4, τ cannot be a subtype of p.

Conversely, from $\Gamma\vdash_\Sigma \langle\langle\sigma,\tau\rangle,\tau\rangle$ *fp-type*, we can conclude that $\Gamma\not\vdash_\Sigma \tau$ *le-type*. This also follows from Lemma 4, since $\tau\preceq p$, and is therefore not le-type. For example, $\langle\langle\sigma,p\rangle,p\rangle$ is an fp-type, but p is not an le-type.

Note, however, that while fp-types and wb-types are mutually exclusive, le-types and fp-types are not; for example, $\langle\langle e, p\rangle, p\rangle$ is both an le-type and an fp-type.

As already remarked, the containment relation plays a central role in our approach to polymorphism. In Section 3, we shall show that there are models of the typing system; that is, we will have functional domains from D_σ to D_τ which are included in D_e; moreover, when $D_\tau \subseteq$ PROP, we also have the result that objects in the function space domain 'D_σ to D_τ' are in D_σ.

2.2 Type Inference Rules

In the preceding subsection, we gave a definition of the syntax of expressions occurring in judgements. These definitions were deliberately general, and could encompass a variety of logical systems. In specifying a particular calculus, such as $\mathcal{L}_{\preceq}$, we need to make explicit how the types of expressions of $\mathcal{L}_{\preceq}$ are inferred. It is to this task that we now turn.

Not all functions can be mapped down into the collection of objects, and following Aczel [Aczel 80], we shall call these *functionals*. That is, adopting Frege's correlation thesis [Frege 77], we will see that all we need in the formal theory are objects, functions and functionals and that functions at a higher level than those three can be mapped down to the lower domains. Among the functionals we will count the interpretations of determiners and logical connectives—and indeed, these are expressions which do not admit of nominalization.

The signature Σ of $\mathcal{L}_{\preceq}$ contains a finite number of statements $c{:}\eta$ which assign types and metatypes to constants of the language. For now, we are only concerned with logical constants and functionals:

Signature of $\mathcal{L}_{\preceq}$

$$\begin{aligned}
\bot &: p \\
\neg &: (e \to e) \\
\wedge &: (e \to (e \to e)) \\
\vee &: (e \to (e \to e)) \\
\supset &: (e \to (e \to e)) \\
= &: (e \to (e \to e)) \\
{}^{\cup} &: (e \to (e \to e)) \\
\mathit{app} &: (e \to (e \to e)) \\
\lambda &: ((e \to e) \to e) \\
\forall &: ((e \to e) \to e) \\
\exists &: ((e \to e) \to e)
\end{aligned}$$

Two comments on the above are called for. First, it will be noticed that, for example, $\neg$ is interpreted as a functional which maps *any* object in $\mathcal{F}_0$ into another such object; we cannot tell, for a given expression α, whether $\neg\alpha$ is a proposition unless we have some way of proving that α itself is a proposition. This will be made explicit in the axioms for type inference given below. Second, we will use conventional notation for the syntax of the various constants, writing $\varphi \wedge \psi$ in place of $\wedge(\varphi)(\psi)$, $app(x, y)$ in place of $app(x)(y)$, and $\forall x.\varphi$ in place of $\forall(\lambda x.\varphi)$.

A context Γ for $\mathcal{L}_{\preceq}$ contains a finite number of statements of the form $x{:}\sigma$, for any type σ. Recall however that Γ never assigns metatypes to variables.

Before launching into the type inference rules, we first define substitution on expressions, where we take $\alpha[\beta/x]$ to be the result of substituting β for all free occurrences of x in α.

$x[\beta/x]$	$\equiv$	β	
$x[\beta/y]$	$\equiv$	x	if $x \not\equiv y$
$c[\beta/x]$	$\equiv$	c	
$(\lambda x.\alpha)[\beta/x]$	$\equiv$	$\lambda x.\alpha$	
$(\lambda x.\alpha)[\beta/y]$	$\equiv$	$\lambda x.\alpha[\beta/y]$	if $x \not\equiv y$ and x not free in β
$(\lambda x.\alpha)[\beta/y]$	$\equiv$	$\lambda z.\alpha[z/x][\beta/y]$	if $x \not\equiv y$ and x is free in β and z is not free in α or β
$\Theta_1(\alpha)[\beta/x]$	$\equiv$	$\Theta_1(\alpha[\beta/x])$	where Θ_1 is $\neg, {}^{\cup}, \wedge, \vee, \supset, =, app, \forall, \exists,$ or else represents functional application.

The other clauses for substitution in logically complex expressions carry on as usual.

The next definition serves the following functions:

1. It gives rules by which the type of an arbitrary expression of $\mathcal{L}_{\preceq}$ can be inferred.

2. It exploits the type t of truths to give introduction (I) and elimination (E) rules for the logical connectives in $\mathcal{L}_{\preceq}$.

Definition 1 (Type Inference for $\mathcal{L}_{\preceq}$)

$$(\textit{Base}) \qquad \frac{\vdash_{\Sigma} \Gamma \ \textit{context}}{\Gamma \vdash_{\Sigma} \alpha{:}\sigma} \quad \textit{where } \alpha{:}\sigma \in \Gamma$$

$$(\textit{Contain}) \qquad \frac{\Gamma \vdash_{\Sigma} \sigma \preceq \tau \qquad \Gamma \vdash_{\Sigma} \alpha{:}\sigma}{\Gamma \vdash_{\Sigma} \alpha{:}\tau}$$

$$(\lambda) \qquad \frac{\Gamma, x{:}\sigma \vdash_{\Sigma} \alpha{:}\tau \qquad \Gamma \vdash_{\Sigma} \langle \sigma, \tau \rangle \ \textit{wb-type}}{\Gamma \vdash_{\Sigma} (\lambda x{:}\sigma.\alpha){:}\langle \sigma, \tau \rangle}$$

$$(\textit{app}) \qquad \frac{\Gamma \vdash_{\Sigma} \alpha{:}\langle \sigma, \tau \rangle \qquad \Gamma \vdash_{\Sigma} \beta{:}\sigma}{\Gamma \vdash_{\Sigma} app(\alpha, \beta){:}\tau}$$

$$(\textit{Funct}) \qquad \frac{\Gamma \vdash_{\Sigma} f{:}(\sigma \rightarrow \tau) \qquad \Gamma \vdash_{\Sigma} \beta{:}\sigma}{\Gamma \vdash_{\Sigma} f(\beta){:}\tau}$$

$({}^{\cup}I)$ $$\frac{\Gamma \vdash_{\Sigma} \alpha{:}\langle e, e\rangle}{\Gamma \vdash_{\Sigma} {}^{\cup}\alpha{:}(e \to p)}$$

$(= prop)$ $$\frac{\Gamma \vdash_{\Sigma} \alpha{:}\eta_1 \qquad \Gamma \vdash_{\Sigma} \beta{:}\eta_2}{\Gamma \vdash_{\Sigma} [\alpha = \beta]{:}p}$$

$(= E)$ $$\frac{\Gamma \vdash_{\Sigma} [\alpha = \beta]{:}t \qquad \Gamma \vdash_{\Sigma} \alpha{:}\eta}{\Gamma \vdash_{\Sigma} \beta{:}\eta}$$

$(\neg prop)$ $$\frac{\Gamma \vdash_{\Sigma} \varphi{:}p}{\Gamma \vdash_{\Sigma} \neg\varphi{:}p}$$

$(\neg I)$ $$\frac{\Gamma \vdash_{\Sigma} \varphi{:}p \qquad \Gamma, \varphi{:}t \vdash_{\Sigma} \bot{:}t}{\Gamma \vdash_{\Sigma} \neg\varphi{:}t}$$

$(\neg E)$ $$\frac{\Gamma, \neg\varphi{:}t \vdash_{\Sigma} \bot{:}t \qquad \Gamma \vdash_{\Sigma} \varphi{:}p}{\Gamma \vdash_{\Sigma} \varphi{:}t}$$

$(\wedge prop)$ $$\frac{\Gamma \vdash_{\Sigma} \varphi{:}p \qquad \Gamma \vdash_{\Sigma} \psi{:}p}{\Gamma \vdash_{\Sigma} [\varphi \wedge \psi]{:}p}$$

$(\wedge I)$ $$\frac{\Gamma \vdash_{\Sigma} \varphi{:}t \qquad \Gamma \vdash_{\Sigma} \psi{:}t}{\Gamma \vdash_{\Sigma} [\varphi \wedge \psi]{:}t}$$

$(\wedge E)$ $$\frac{\Gamma \vdash_{\Sigma} [\varphi \wedge \psi]{:}t}{\Gamma \vdash_{\Sigma} \varphi{:}t} \qquad \frac{\Gamma \vdash_{\Sigma} [\varphi \wedge \psi]{:}t}{\Gamma \vdash_{\Sigma} \psi{:}t}$$

$(\vee prop)$ $$\frac{\Gamma \vdash_{\Sigma} \varphi{:}p \qquad \Gamma \vdash_{\Sigma} \psi{:}p}{\Gamma \vdash_{\Sigma} [\varphi \vee \psi]{:}p}$$

$(\vee I)$ $$\frac{\Gamma \vdash_{\Sigma} \varphi{:}t \qquad \Gamma \vdash_{\Sigma} \psi{:}p}{\Gamma \vdash_{\Sigma} [\varphi \vee \psi]{:}t} \qquad \frac{\Gamma \vdash_{\Sigma} \varphi{:}p \qquad \Gamma \vdash_{\Sigma} \psi{:}t}{\Gamma \vdash_{\Sigma} [\varphi \vee \psi]{:}t}$$

$(\vee E)$ $$\frac{\Gamma, \varphi{:}t \vdash_{\Sigma} \chi{:}t \qquad \Gamma, \psi{:}t \vdash_{\Sigma} \chi{:}t \qquad \Gamma \vdash_{\Sigma} [\varphi \vee \psi]{:}t}{\Gamma \vdash_{\Sigma} \chi{:}t}$$

$(\supset prop)$ $$\frac{\Gamma, \varphi{:}t \vdash_{\Sigma} \psi{:}p \qquad \Gamma \vdash_{\Sigma} \varphi{:}p}{\Gamma \vdash_{\Sigma} [\varphi \supset \psi]{:}p}$$

$(\supset I)$ $$\frac{\Gamma, \varphi{:}t \vdash_{\Sigma} \psi{:}t \qquad \Gamma \vdash_{\Sigma} \varphi{:}p}{\Gamma \vdash_{\Sigma} [\varphi \supset \psi]{:}t}$$

$(\supset E)$ $$\frac{\Gamma \vdash_{\Sigma} \varphi{:}t \qquad \Gamma \vdash_{\Sigma} [\varphi \supset \psi]{:}t}{\Gamma \vdash_{\Sigma} \psi{:}t}$$

$(\forall prop)$ $$\frac{\Gamma, x{:}\sigma \vdash_{\Sigma} \varphi{:}p}{\Gamma \vdash_{\Sigma} \forall x{:}\sigma.\varphi{:}p}$$

$$(\forall I) \qquad \frac{\Gamma, x{:}\sigma \vdash_\Sigma \varphi{:}t}{\Gamma \vdash_\Sigma \forall x{:}\sigma.\varphi{:}t} \quad \textit{where } x \textit{ is not free in } \varphi \textit{ or any assumptions in } \Gamma$$

$$(\forall E) \qquad \frac{\Gamma \vdash_\Sigma \forall x{:}\sigma.\varphi{:}t \qquad \Gamma \vdash_\Sigma \alpha{:}\sigma}{\Gamma \vdash_\Sigma \varphi[\alpha/x]{:}t}$$

$$(\exists prop) \qquad \frac{\Gamma, x{:}\sigma \vdash_\Sigma \varphi{:}p}{\Gamma \vdash_\Sigma \exists x{:}\sigma.\varphi{:}p}$$

$$(\exists I) \qquad \frac{\Gamma, x{:}\sigma \vdash_\Sigma \varphi[\alpha/x]{:}t}{\Gamma \vdash_\Sigma \exists x{:}\sigma.\varphi{:}t}$$

$$(\exists E) \qquad \frac{\Gamma \vdash_\Sigma \exists x{:}\sigma.\varphi{:}t \qquad \Gamma, \varphi[\alpha/x]{:}t \vdash_\Sigma \psi{:}t}{\Gamma \vdash_\Sigma \psi{:}t}$$

Although most of these clauses are standard, it should perhaps be pointed out that the definition ($\supset prop$) of implication is the one proposed by [Aczel 80] to enable him to interpret Martin-Löf's type theory in a Frege structure; it has the consequence that if the antecedent φ of a conditional is not true, then $\varphi \supset \psi$ is a proposition whatever object ψ is. For our purposes, it would also be possible to omit the extra condition on the antecedent.

2.3 Equality Axioms

We now give a set of equality axioms which are similar to those of the λ-calculus, except that we allow self-application and polymorphism. Note however that self-application is only possible for those expressions which have a complex type; indeed, this is what is required by clause (*app*) of the syntax above.

$$(\alpha) \quad \Gamma \vdash_\Sigma [(\lambda x{:}\sigma.\alpha) = (\lambda y{:}\sigma.\alpha[y/x])]{:}t, \text{where } y \text{ is not free in } \alpha.$$

$$(\beta) \quad \Gamma \vdash_\Sigma [app(\lambda x{:}\sigma.\alpha, a) = \alpha[a/x]]{:}t,$$

$$(\gamma) \quad \frac{\Gamma \vdash_\Sigma \alpha_1{:}\langle\sigma,\tau\rangle \qquad \Gamma \vdash_\Sigma \beta_1{:}\sigma \qquad \Gamma \vdash_\Sigma [\alpha_1 = \alpha_2]{:}t \qquad \Gamma \vdash_\Sigma [\beta_1 = \beta_2]{:}t}{\Gamma \vdash_\Sigma [app(\alpha_1, \beta_1) = app(\alpha_2, \beta_2)]{:}t}$$

$$(\delta) \quad \frac{\Gamma \vdash_\Sigma \alpha{:}\sigma}{\Gamma \vdash_\Sigma [\alpha = \alpha]{:}t}$$

$$(\epsilon) \quad \frac{\Gamma \vdash_\Sigma [\alpha_1 = \alpha_2]{:}t \qquad \Gamma \vdash_\Sigma [\alpha_1 = \alpha_3]{:}t}{\Gamma \vdash_\Sigma [\alpha_2 = \alpha_3]{:}t}$$

$$(\zeta^6) \quad \frac{\Gamma \vdash_\Sigma [app(\alpha_1, x) = app(\alpha_2, x)]{:}t}{\Gamma \vdash_\Sigma [\alpha_1 = \alpha_2]{:}t} \quad \text{where } x \text{ is not free in } \alpha_1, \alpha_2 \text{ or any assumptions in } \Gamma.$$

[6]This is the axiom of extensionality.

2.4 Russell's and Curry's Paradoxes

It might be thought that the theory presented above would fall foul of Russell's paradox, due to the fact that $\neg app(x,x)$ is a well-formed formula for x of any type $\langle\sigma,\tau\rangle$; hence by abstracting over $\neg app(x,x)$, we could obtain the equality

$$app(a,a) = \neg app(a,a)$$

where a is $\lambda x.\neg app(x,x)$.

For example, given the following proof,

$$\dfrac{\Gamma, x{:}\langle e,p\rangle \vdash_\Sigma x{:}\langle e,p\rangle \qquad \dfrac{\Gamma, x{:}\langle e,p\rangle \vdash_\Sigma x{:}\langle e,p\rangle}{\Gamma, x{:}\langle e,p\rangle \vdash_\Sigma x{:}e}(\mathit{Contain})}{\dfrac{\Gamma, x{:}\langle e,p\rangle \vdash_\Sigma app(x,x){:}p}{\Gamma, x{:}\langle e,p\rangle \vdash_\Sigma \neg app(x,x){:}p}(\neg prop)}(app)$$

we might conclude that we can set a equal to the abstraction

$$\lambda x{:}\langle e,p\rangle.\neg app(x,x){:}\langle\langle e,p\rangle,p\rangle$$

and infer that $app(a,a)$ is of type p, leading to a contradictory proposition from the above equality.

However, one of the steps necessary to derive this contradiction is incorrect. That is, even if x is of type $\langle e,p\rangle$, and even though $\neg app(x,x)$ is a proposition, $\lambda x{:}\langle e,p\rangle.\neg app(x,x)$ is not typable in $\mathcal{L}_{\overline{\prec}}$. More specifically, it is excluded by virtue of clause (λ) in the definition of type inference, since we cannot derive $\Gamma\vdash_\Sigma p$ *le-type*, and hence cannot derive that $\langle\langle e,p\rangle,p\rangle$ is a wb-type.

In fact we have a more general result: the paradox does not arise for x of any type $\langle\sigma,\tau\rangle$, where $\tau = t$ or p. This is a consequence of the following lemma.

Lemma 5 *If x is of type $\langle\sigma,\tau\rangle$, $\tau = t$ or p, then $\lambda x{:}\langle\sigma,\tau\rangle.\neg app(x,x)$ is not typable.*

Proof *According to the definition of type inference for $\mathcal{L}_{\overline{\prec}}$, it is enough to show that we cannot derive $\Gamma\vdash_\Sigma p$ le-type or $\Gamma\vdash_\Sigma t$ le-type. This is obvious.* □

Our manner of avoiding the paradox is somewhat new, we believe. It is similar to Russell's own approach in that type constraints are invoked to limit abstraction, but differs of course with respect to the non-hierarchical nature of the type system. Unlike Aczel [Aczel 80], we do not take the step of questioning the propositionhood of $app(a,a)$; and unlike Turner [Turner 87], we do not restrict the axiom of β-conversion.

Let us turn now to the question of Curry's paradox. Recall the Deduction Theorem (in fact, our rule $(\supset I)$):

$$(DT)\ \dfrac{\Gamma, \varphi{:}t \vdash_\Sigma \psi{:}t \qquad \Gamma\vdash_\Sigma \varphi{:}p}{\Gamma\vdash_\Sigma [\varphi\supset\psi]{:}t}$$

If we take a to be the formula

$$\lambda x{:}\sigma[app(x,x) \supset \bot],$$

then by β-conversion we derive

$$(ID)\;\; app(a,a) = [app(a,a) \supset \bot].$$

Now, it holds trivially that

$$app(a,a){:}t \vdash_\Sigma app(a,a){:}t.$$

Hence, by (ID) we derive

$$app(a,a){:}t \vdash_\Sigma [app(a,a) \supset \bot]{:}t,$$

and by $(\supset E)$ we get

$$app(a,a){:}t \vdash_\Sigma \bot{:}t.$$

In order to derive by (DT) that

$$\vdash_\Sigma [app(a,a) \supset \bot]{:}t$$

we must first be able to show

$$\emptyset \vdash_\Sigma app(a,a){:}p,$$

(where $\emptyset$ is the empty context). For we can derive the latter, we can use it in the following step:

$$\frac{app(a,a){:}t \vdash_\Sigma \bot{:}t \qquad \emptyset \vdash_\Sigma app(a,a){:}p}{\emptyset \vdash_\Sigma [app(a,a) \supset \bot]{:}t} \qquad (\supset prop)$$

and also by (ID) and $(= E)$

$$\emptyset \vdash_\Sigma app(a,a){:}t.$$

Given the last two steps, we can again apply $(\supset E)$ to get

$$\vdash_\Sigma \bot{:}t.$$

The proof only goes through, however, if $\emptyset \vdash_\Sigma app(a,a){:}p$ is derivable. For this, we would have to assign the type $\langle \sigma, p\rangle$ to a, i.e., to $\lambda x{:}\sigma[app(x,x) \supset \bot]$. How could we show that

(18) $\vdash_\Sigma \lambda x{:}\sigma[app(x,x) \supset \bot]{:}\langle \sigma, p\rangle$?

This can only be the last step of an inference involving the rules (λ) or $(Contain)$. We consider the two cases in turn.

Case 1: The premisses of the inference must include $x{:}\sigma \vdash_\Sigma [app(x,x) \supset \bot]{:}p$, which in turn is only derivable by ($\supset$ *prop*) from the assumption that $x{:}\sigma \vdash_\Sigma app(x,x){:}p$. However, we can only prove the latter if for some σ', $\sigma = \langle \sigma', p\rangle$, where $\langle \sigma', p\rangle \preceq \sigma'$. But in this case, we would have to show that $\Gamma \vdash_\Sigma p$ *le-type*, which is impossible.

Case 2: We have to find some type τ such that $\tau \preceq \langle \sigma, p\rangle$ and $x{:}\sigma \vdash_\Sigma \lambda x{:}\sigma[app(x,x) \supset \bot]{:}\tau$. The only applicable derivation rule is (*Ran* $\preceq$), setting τ to be $\langle \sigma, t\rangle$. However, the judgement $\vdash_\Sigma \lambda x{:}\sigma[app(x,x) \supset \bot]{:}\langle \sigma, t\rangle$ is not derivable, for the same reasons as those given in Case 1 above, since we cannot show that $\Gamma \vdash_\Sigma p$ *le-type*.

3 Models of $\mathcal{L}_\preceq$

3.1 Frege Structures

As pointed out earlier, our models are constructed using the notion of a Frege structure as defined by Aczel [Aczel 80]. We begin with a collection $\mathcal{F}_0$ of objects, and for each natural number $n \geq 1$, we define $\mathcal{F}_n$ as $\{\boldsymbol{f} : {\mathcal{F}_0}^n \mapsto \mathcal{F}_0\}$, where ${\mathcal{F}_0}^n = \overbrace{\mathcal{F}_0 \times \mathcal{F}_0 \times \cdots \mathcal{F}_0}^{n\ times}$. In particular, $\mathcal{F}_1$ is the set of all unary functions from $\mathcal{F}_0$ to $\mathcal{F}_0$. Within $\mathcal{F}_0$ we pick out PROP, the collection of propositions, and TRUTH, the collection of all true propositions. (Thus, Aczel makes a crucial departure from Frege in denying that all true propositions can be identified with the True).

So far, then our Frege structures contain objects, functions, propositions and truths. To these, we need to add functionals, logical connectives, and some closure conditions. We now show how they are supplied.

Two functionals are required in order to provide a model for the lambda calculus:[7]

$$\boldsymbol{\lambda} : \mathcal{F}_1 \mapsto \mathcal{F}_0$$
$$\mathbf{app} : \mathcal{F}_0 \times \mathcal{F}_0 \mapsto \mathcal{F}_0$$

These obey a *comprehension* principle such that whenever $\boldsymbol{f}$ is a function in $\mathcal{F}_1$, then

$$\mathbf{app}(\boldsymbol{\lambda x}.\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{a}) = \boldsymbol{f}(\boldsymbol{a}).$$

Let PF be the collection of unary propositional functions in a Frege structure, i.e., those functions $\boldsymbol{f}$ in $\mathcal{F}_1$ which map their arguments into PROP:

$$\text{PF} = \{\boldsymbol{f} \in \mathcal{F}_1 | \text{for all } \boldsymbol{x}\ in \mathcal{F}_0, \boldsymbol{f}(\boldsymbol{x}) \text{ is in PROP}\}$$

[7]We adopt the notational convention of using boldface terms to denote elements of the model, reserving italics for expressions of the object language. For example, **app** is a functional in the model which corresponds to the functor *app* in the language.

We can now identify a further subcollection of $\mathcal{F}_0$, namely SET, as the individual correlates of propositional functions under $\boldsymbol{\lambda}$:

Definition 2 (Sets) *An object is in* SET *iff it is* $\boldsymbol{\lambda f}$ *for some* $\boldsymbol{f}$ *in* PF.

The distinguishing characteristic of sets (i.e. elements of SET) is that they can be predicated of any object in $\mathcal{F}_0$ to yield a proposition:

Definition 3 (Predication) *If* $\boldsymbol{a}$ *is in* SET*, then* $\mathbf{app}(\boldsymbol{a}, \boldsymbol{b})$ *is in* PROP*, for any object* $\boldsymbol{b}$ *in* $\mathcal{F}_0$.

Comprehension can be restated as follows:

Definition 4 (Comprehension) *If* $\boldsymbol{f}$ *is in* PF*, then* $\boldsymbol{\lambda f}$ *is a set* $\boldsymbol{a}$ *such that for any object* $\boldsymbol{b}$*,* $\mathbf{app}(\boldsymbol{a},\ \boldsymbol{b})$ *is in* PROP*, and* $\mathbf{app}(\boldsymbol{a},\ \boldsymbol{b})$ *is in* TRUTH *iff* $\boldsymbol{f}(\boldsymbol{b})$ *is in* TRUTH.

The notions that we have introduced so far—objects, functions, PROP, TRUTH, SET, comprehension and predication—are based on a model of the λ-calculus. In order to ensure that they have the properties we want, our models should also contain a logic. We know that such a construction is not straightforward; for instance, logic cannot be built in a simple way on the top of Scott domains (cf. [Scott 76]). The construction provided by Aczel inductively increases the two basic collections of propositions and truths, and the fixed point theorem is then applied to provide the limit of these newly obtained collections, resulting in PROP and TRUTH. Hence PROP is closed under all the logical connectives $\wedge, \vee, \neg, \ldots$ (more strictly, the functionals corresponding to these connectives) and TRUTH is the collection of all true propositions. The organization of $\mathcal{F}_0$ and $\mathcal{F}_1$ in a Frege structure is illustrated in Figure 1.

We now need to ensure that we have full abstraction. That is, if $\xi[x_1, x_2, \ldots, x_n]$ is an expression formed out of objects, functions, functionals and variables (ranging over objects in a Frege structure), where $x_1, x_2, \ldots, x_n$ are free variables of ξ, then there is a function $\boldsymbol{f}$ in the Frege structure such that $\boldsymbol{f}(\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_n) = \xi[\boldsymbol{a}_1/x_1, \boldsymbol{a}_2/x_2, \ldots, \boldsymbol{a}_n/x_n]$, where the substition of objects for variables is simultaneous. This is called *explicit substitution.*

We assume our construction is based on the model E_∞ of the untyped λ-calculus [Scott 76]. We then take

$$\boldsymbol{B_p} = \{0, 1\} \subseteq \mathcal{F}_0 = E_\infty.$$

We next construct the logical constants so that PROP is the smallest set containing $\boldsymbol{B_p}$ which is closed under the relevant clauses for connectives presented in Definition 1. Here, we give just two examples.

($\wedge$ schema) If $\boldsymbol{\varphi}$ is in PROP and $\boldsymbol{\psi}$ is in PROP, then $\boldsymbol{\varphi} \wedge \boldsymbol{\psi}$ is in PROP, and $\boldsymbol{\varphi} \wedge \boldsymbol{\psi}$ is in TRUTH iff $\boldsymbol{\varphi}$ is in TRUTH and $\boldsymbol{\psi}$ is in TRUTH.

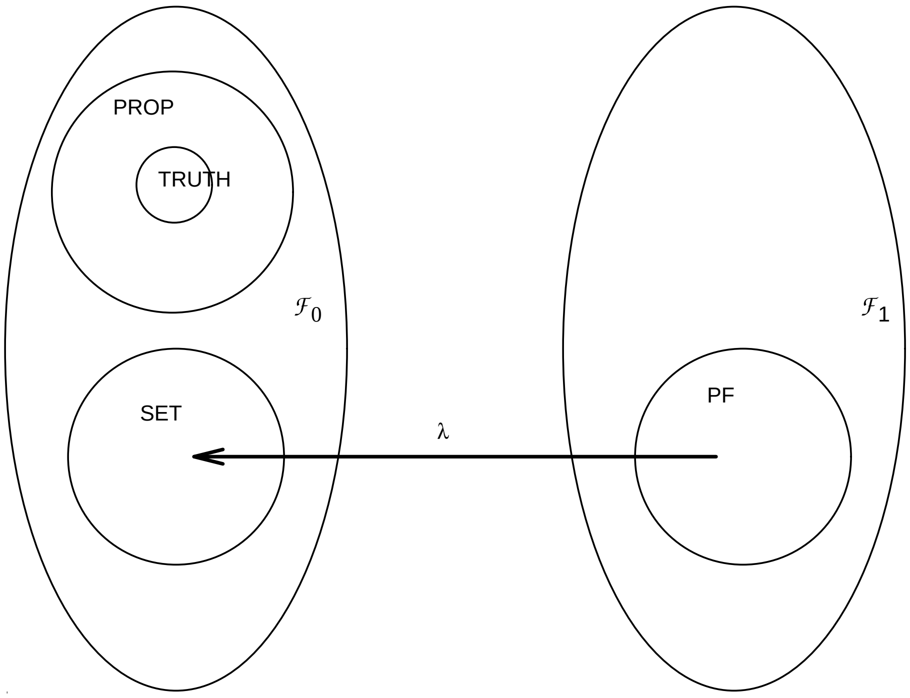

Figure 1: The functional λ

(∀ schema) For all n-ary functions $\boldsymbol{f}$, if $\boldsymbol{f(x_1, x_2, \ldots, x_n)}$ is in PROP for all $\langle \boldsymbol{x_1, x_2, \ldots, x_n} \rangle \in \mathcal{F}_0{}^n$, then $\forall \boldsymbol{f}$ is in PROP, and $\forall \boldsymbol{f}$ is in TRUTH iff $\boldsymbol{f(a_1, a_2, \ldots, a_n)}$ is in TRUTH for all $\langle \boldsymbol{a_1, a_2, \ldots, a_n} \rangle \in \mathcal{F}_0{}^n$.

Whenever φ is a wff open in x, we understand $\langle \varphi | \boldsymbol{x} \rangle$ to be the function $\boldsymbol{f}$ in $\mathcal{F}_1$ such that for any $\boldsymbol{a}$ in $\mathcal{F}_0$, $\boldsymbol{f}(\boldsymbol{a}) = \varphi[\boldsymbol{a}/x]$. Since we have full abstraction, we can assume that $\boldsymbol{f}$ exists. Now, we take $|\ |: \mathcal{F}_0 \mapsto \mathcal{F}_1$ to be the functional such that

$$|\boldsymbol{a}| \ = \langle \mathbf{app}(\boldsymbol{a}, \boldsymbol{x}) \mid \boldsymbol{x} \rangle.$$

In general, we define $|\ |_n : \mathcal{F}_0 \mapsto \mathcal{F}_n$ such that

$$|\ \boldsymbol{a}\ |_n = \langle \mathbf{app}_n(\boldsymbol{a}, \vec{\boldsymbol{x}}) \mid \vec{\boldsymbol{x}} \rangle$$

where $\mathbf{app}_1 = \mathbf{app}$ and $\mathbf{app}_{n+1}(\boldsymbol{a}, \boldsymbol{b}, \vec{\boldsymbol{b}}) = \mathbf{app}_n(\mathbf{app}(\boldsymbol{a}, \boldsymbol{b}), \vec{\boldsymbol{b}})$, and $\vec{\boldsymbol{x}}, \vec{\boldsymbol{b}}$ are sequences of n variables or elements of $\mathcal{F}_0$. Now,

$$\boldsymbol{\lambda}_n^m : \mathcal{F}_m \mapsto \mathcal{F}_n, m > n$$

is defined inductively, for $\vec{\boldsymbol{a}} = \boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ as

$$\begin{array}{rcl} (\boldsymbol{\lambda}_n^{n+1} \boldsymbol{f})(\vec{\boldsymbol{a}}) & = & \boldsymbol{\lambda} \langle \boldsymbol{f}(\vec{\boldsymbol{a}}, \boldsymbol{x}) \mid \boldsymbol{x} \rangle \\ (\boldsymbol{\lambda}_n^{n+m+1} \boldsymbol{f})(\vec{\boldsymbol{a}}) & = & \boldsymbol{\lambda}_n^{n+1} (\boldsymbol{\lambda}_{n+1}^{n+m+1} \boldsymbol{f}) \end{array}$$

In particular, $\boldsymbol{\lambda}_0^n$ nominalizes an n-ary function $\boldsymbol{f}$ returning $\boldsymbol{\lambda}_0^n \boldsymbol{f}$ in $\mathcal{F}_0$.
We take

$$\text{SET} \ = \{ \boldsymbol{\lambda}_0^n \boldsymbol{f} | \boldsymbol{f} \text{ is any propositional function} \}.$$

Definition 5 *A* Frege Structure *is a triple* $\mathcal{F} = \langle \mathcal{F}_0, \text{PROP}, \text{SET} \rangle$ *constructed as above.*

It might be unclear why we have only included $\mathcal{F}_0$, PROP, and SET in the structure and ignored functions in general (though not functionals). The reason for this is that the principle of extensionality holds in E_∞ and hence we have a bijection between $\mathcal{F}_0$ and $\mathcal{F}_n$ for $n \leq 1$. In fact, we can show that $\boldsymbol{\lambda}(|\boldsymbol{a}|) = \boldsymbol{a}$.

Lemma 6 *The functional* $\boldsymbol{\lambda} : \mathcal{F}_1 \mapsto \mathcal{F}_0$ *is bijective.*
Proof

1. $\boldsymbol{\lambda}$ *is* injective, *because*

$$\begin{array}{rll} \boldsymbol{\lambda f} \ = \boldsymbol{\lambda g} & \textit{implies} & \forall \boldsymbol{a}(\mathbf{app}(\boldsymbol{\lambda f}, \boldsymbol{a}) = \mathbf{app}(\boldsymbol{\lambda g}, \boldsymbol{a})) \\ & \textit{implies} & \forall \boldsymbol{a}(\boldsymbol{f}(\boldsymbol{a}) = \boldsymbol{g}(\boldsymbol{a})) \\ & \textit{implies} & \boldsymbol{f} \ = \boldsymbol{g}. \end{array}$$

2. $\boldsymbol{\lambda}$ *is* surjective, *because if* $\boldsymbol{a}$ *is in* $\mathcal{F}_0$ *then* $\langle \boldsymbol{a} \mid \boldsymbol{x} \rangle$ *is in* $\mathcal{F}_1$ *and for any* $\boldsymbol{b}$ *in* $\mathcal{F}_0$, $\boldsymbol{a} \neq \boldsymbol{b}$ *implies* $\langle \boldsymbol{a} \mid \boldsymbol{x} \rangle \neq \langle \boldsymbol{b} \mid \boldsymbol{x} \rangle$ $\square$

Hence we have sufficient structure within $\mathcal{F}_0$ to do everything we want without having to consider $\mathcal{F}_1$.

Up until now, we have not said anything about the interpretation of the predication operator $^{\cup}$. It will be recalled that, by virtue of ($^{\cup}$ I), whenever α is an expression of type $\langle e, e \rangle$, $^{\cup}\alpha$ is of type $(e \to p)$. If α denotes the nominalization $\boldsymbol{\lambda f}$ of a propositional function $\boldsymbol{f}$, then we want $^{\cup}\alpha$ to denote $\langle \mathbf{app}(\boldsymbol{\lambda f}, \boldsymbol{x}) \mid \boldsymbol{x} \rangle$. However, the functional corresponding to $^{\cup}$ must carry *any* object in $\mathcal{F}_0$ into an appropriate value in $\mathcal{F}_1$. What happens if α denotes an object $\boldsymbol{a}$ not in SET? In this case, $\langle \mathbf{app}(\boldsymbol{a}, \boldsymbol{x}) \mid \boldsymbol{x} \rangle$ will not be a propositional function; that is, for any argument $\boldsymbol{b}$, $\mathbf{app}(\boldsymbol{a}, \boldsymbol{b})$ will just denote an arbitrary object in $\mathcal{F}_0$. We take this to be an acceptable alternative to the approach used in [Partee 84] where $^{\cup}$ is interpreted as a partial function, defined only on objects in SET. Hence, we always let $^{\cup}\alpha$ denote $\langle \mathbf{app}([\![\alpha]\!], \boldsymbol{x}) \mid \boldsymbol{x} \rangle$, where $[\![\alpha]\!]$ is the interpretation of α.

We shall now show how to construct domains inside $\mathcal{F}_0$ such that the types described earlier can be mapped into them.

3.2 Domains

We distinguish between two kinds of domains, $\boldsymbol{Dom_1}$ and $\boldsymbol{Dom_2}$. We use $\boldsymbol{X_1}$, $\boldsymbol{Y_1}$ to range over $\boldsymbol{Dom_1}$, $\boldsymbol{X_2}$, $\boldsymbol{Y_2}$ to range over $\boldsymbol{Dom_2}$, and $\boldsymbol{X}$, $\boldsymbol{Y}$ to range over both domains. We also assume that $*$ is a distinguished element in $\mathcal{F}_0$ which will be used to give functions a undefined value.

$$\begin{array}{lcl} \boldsymbol{Dom_1} & ::= & \mathcal{F}_0 \mid \boldsymbol{X_1} \Rightarrow \boldsymbol{Y_1} \\ \boldsymbol{Dom_2} & ::= & \text{PROP} \mid \text{TRUTH} \mid \boldsymbol{X} \hookrightarrow \boldsymbol{X_2} \end{array}$$

$\boldsymbol{Dom_1}$ interprets those le-types which are not fp-types. It is appropriate for le-types because the leftmost type of an le-type will always be in $\mathcal{F}_0$. It does not interpret fp-types because $\forall \boldsymbol{X} \in \boldsymbol{Dom_1}$, $\boldsymbol{X} \not\subseteq$ PROP. However, fp-types will be interpreted by $\boldsymbol{Dom_2}$.

Definition 6 ($\Rightarrow$ Function Space)
$X \Rightarrow Y = \{x \in \mathcal{F}_0 :$ *for all* $x' \in X[\mathbf{app}(x, x') \in Y]\}$.

Definition 7 ($\hookrightarrow$ Function Space)
$X \hookrightarrow Y = \{x \in X :$ *for all* $x'[$ *if* $x' \in X$ *then* $\mathbf{app}(x, x') \in Y]$, *else* $app(x, x') = *\}$.

We shall write 'f is *true*' as an alternative for 'f is in TRUTH'. Similarly we use 'f is *false*' for 'f is in PROP–TRUTH'. We also assume the presence of two special elements of the Frege structure, 1 in TRUTH and 0 in PROP–TRUTH. Of course, 1 is not the only truth in the model.

Definition 8 (Internal Definability) *We say that a collection $\mathcal{C}$ is* internally definable *in a Frege structure if the following holds: there is some $\boldsymbol{f}$ in* PF *such that for all $\boldsymbol{x}$ in $\mathcal{F}_0$, $\boldsymbol{f(x)}$ is true iff $\boldsymbol{x}$ is in $\mathcal{C}$. In this case, $\boldsymbol{f}$ is the* characteristic function *of $\mathcal{C}$.*

The domain $\boldsymbol{Dom_1}$ can be understood as the collection of objects which provide interpretations for types not involving p and t. All the domains of $\boldsymbol{Dom_1}$ are internally definable. This can be proved by induction as follows:

Lemma 7

1. *$\mathcal{F}_0$ is internally definable, by taking the function $f : \mathcal{F}_0 \mapsto \text{PROP}$, where $f(x) = 1$, for all $x \in \mathcal{F}_0$*

2. *Assume $\boldsymbol{X_1}$, $\boldsymbol{Y_1}$ are internally definable by the propositional functions f and g respectively. Then we want to show that the collection $\boldsymbol{X_1} \Rightarrow \boldsymbol{Y_1}$ is also internally definable.*

$$f{:}\mathcal{F}_0 \mapsto \text{PROP} \textit{ where}$$
$$f(x) \textit{ is } \begin{cases} \textit{true} & \textit{for all } x \in \boldsymbol{X_1} \\ \textit{false} & \textit{otherwise} \end{cases}$$

$$g{:}\mathcal{F}_0 \mapsto \text{PROP} \textit{ where}$$
$$g(y) \textit{ is } \begin{cases} \textit{true} & \textit{for all } y \in \boldsymbol{Y_1} \\ \textit{false} & \textit{otherwise} \end{cases}$$

Let $h : \mathcal{F}_0 \mapsto \text{PROP}$, where

$$h(z) = \forall x[f(x) \supset g(\mathbf{app}(z, x))]$$

Now,

(a) h is a propositional function because f and g are, and

(b) we have to prove that $z \in \boldsymbol{X_1} \Rightarrow \boldsymbol{Y_1}$ iff $h(z)$ is true.

i. Assume $z \in \boldsymbol{X_1} \Rightarrow \boldsymbol{Y_1}$. Let $x \in \boldsymbol{X_1}$; then $f(x) \in$ PROP because $f \in$ PF and $\mathbf{app}(z, x) \in \boldsymbol{Y_1}$ because $z \in \boldsymbol{X_1} \Rightarrow \boldsymbol{Y_1}$. $g(\mathbf{app}(z, x))$ is true because g internally defines Y_1. Hence $f(x) \in$ PROP and $f(x) \in$ TRUTH implies $g(\mathbf{app}(z, x)) \in$ TRUTH. Hence $f(x) \supset g(\mathbf{app}(z, x))$ is true. But this holds for every x, hence $\forall x[f(x) \supset g(\mathbf{app}(z, x))]$ is true. Hence $h(z)$ is true.

ii. Assume $h(z)$ is true. $z \in \mathcal{F}_0$, of course. Let $x \in \boldsymbol{X_1}$, then $f(x) \supset g(\mathbf{app}(z, x))$ is true. But $f(x)$ is true because $x \in \boldsymbol{X_1}$. So $g(\mathbf{app}(z, x))$ is true, and $\mathbf{app}(z, x) \in \boldsymbol{Y_1}$, since g internally defines Y. Hence $z \in \boldsymbol{X_1} \Rightarrow \boldsymbol{Y_1}$.

Hence $z \in \boldsymbol{X_1}\Rightarrow\boldsymbol{Y_1} \iff h(z)$ *is true.* □

Now that we know $\boldsymbol{Dom_1}$ is the domain of internally definable collections, we can write $X\Rightarrow Y$ using f_X and f_Y, the characteristic functions of X and Y:

$$X\Rightarrow Y = \{x \in \mathcal{F}_0 : \text{ for all } x'[f_X(x') \supset f_Y(\mathbf{app}(x,x'))]\}$$

$\boldsymbol{Dom_2}$, on the other hand, involves domains which are not internally definable. For example, the two basic domains PROP and TRUTH cannot be internally defined. In fact, according to Tarski's theorem on the undefinability of truth, we cannot have a propositional function in the object language which internally defines truth; this implies that we cannot have a propositional function which internally defines propositions; see [Aczel 80] for discussion.

It might be asked whether the existence of judgements like $\Gamma\vdash_\Sigma \alpha{:}t$ means that we have in effect committed ourselves to the internal definability of truth. The first point to note is that typing statements are not propositions *in* $\mathcal{L}_{\preceq}$, but judgements *about* the language. Second, we have no way of telling for an arbitrary expression α whether the judgement $\Gamma\vdash_\Sigma \alpha : t$ holds. In particular, since contexts Γ are finite, they will not necessarily determine the type of an arbitrary variable.

Recall that **app** is the functional in the Frege structure which corresponds to *app* in the language of $\mathcal{L}_{\preceq}$. We saw that in a standard Frege structure, $\mathcal{F}_1 = \{\boldsymbol{f} : \mathcal{F}_0 \mapsto \mathcal{F}_0\}$ is the collection of all functions from $\mathcal{F}_0$ to $\mathcal{F}_0$, and contains a subcollection PF of unary propositional functions. We also saw earlier that $\boldsymbol{\lambda}$ is a bijective map from $\mathcal{F}_1$ to $\mathcal{F}_0$. What we now have to check is that, as a special case of Definition 6, there is an appropriate domain $\mathcal{F}_0 \hookrightarrow$PROP inside $\mathcal{F}_0$ which will contain the nominals of propositional functions. In fact $(\mathcal{F}_0 \hookrightarrow \text{PROP}) = \text{SET}$ (easy to prove).

Our next lemma illustrates the fact that the domains constructed above do indeed model the types in our language.

Lemma 8 *If* $\boldsymbol{X_1}$, $\boldsymbol{Y_1}$ *are any domains in* $\boldsymbol{Dom_1}$, *then* $(\boldsymbol{X_1}\Rightarrow\boldsymbol{Y_1}) \subseteq \mathcal{F}_0$.
The proof is trivial. □

In other words, every function in $\boldsymbol{Dom_1}$ is an object. This enables us to interpret self-application and nominalization.

Lemma 9 *If* $\boldsymbol{X}$ *is any domain and* $\boldsymbol{Y_2}$ *is in* $\boldsymbol{Dom_2}$ *then* $(\boldsymbol{X}\hookrightarrow\boldsymbol{Y_2}) \subseteq \boldsymbol{X}$.
The proof is trivial. □

Lemma 10 *If* $\boldsymbol{X_1}, \boldsymbol{Y_1}, \boldsymbol{Y_1}'$ *are in* $\boldsymbol{Dom_1}$, *then* $\boldsymbol{Y_1} \subseteq \boldsymbol{Y_1}'$ *implies* $(\boldsymbol{X_1}\Rightarrow\boldsymbol{Y_1}) \subseteq (\boldsymbol{X_1}\Rightarrow\boldsymbol{Y_1}')$.

Proof *If* $x \in \boldsymbol{X_1}\Rightarrow\boldsymbol{Y_1}$, *then* $\forall x' \in \boldsymbol{X_1}, \mathbf{app}(x,x') \in \boldsymbol{Y_1}$, *by Definition 6. Since* $\boldsymbol{Y_1} \subseteq \boldsymbol{Y_1}'$, *it follows that for all* $x' \in \boldsymbol{X_1}, \mathbf{app}(x,x') \in \boldsymbol{Y_1}'$ *and so* $x \in \boldsymbol{X_1}\Rightarrow\boldsymbol{Y_1}'$. □

Lemma 11 *If* $\boldsymbol{X}, \boldsymbol{Y_2}, \boldsymbol{Y_2}'$ *are domains such that* $\boldsymbol{Y_2}, \boldsymbol{Y_2}'$ *are in* $\boldsymbol{Dom_2}$*, then* $\boldsymbol{Y_2} \subseteq \boldsymbol{Y_2}'$ *implies* $(\boldsymbol{X} \hookrightarrow \boldsymbol{Y_2}) \subseteq (\boldsymbol{X} \hookrightarrow \boldsymbol{Y_2}')$.

Proof *Same as above.* □

Lemma 12 *If* $\boldsymbol{X_1}, \boldsymbol{X_1}'$ *and* $\boldsymbol{Y_1}$ *are in* $\boldsymbol{Dom_1}$*, then* $\boldsymbol{X_1} \subseteq \boldsymbol{X_1}'$ *implies* $(\boldsymbol{X_1}' \Rightarrow \boldsymbol{Y_1}) \subseteq (\boldsymbol{X_1} \Rightarrow \boldsymbol{Y_1})$.

Proof *If* $x \in (\boldsymbol{X_1}' \Rightarrow \boldsymbol{Y_1})$*, then by Definition 6, for all* $x' \in \boldsymbol{X_1}'$, $\mathbf{app}(x, x') \in \boldsymbol{Y_1}$*. Since* $\boldsymbol{X_1} \subseteq \boldsymbol{X_1}'$*, then for all* $x' \in \boldsymbol{X_1}, \mathbf{app}(x, x') \in \boldsymbol{Y_1}$*. Therefore* $x \in \boldsymbol{X_1} \Rightarrow \boldsymbol{Y_1}$. □

A *model* for $\mathcal{L}_{\preceq}$ is a 6-tuple $\mathcal{M} = \langle \mathcal{F}, \Rightarrow, \hookrightarrow, \mathcal{I}, D, g \rangle$, where

1. $\mathcal{F}$ is a Frege Structure ,
2. $\Rightarrow$ and $\hookrightarrow$ are defined as above,
3. $\mathcal{I}$ is an interpretation function which takes any constant of kind η to an object in D_η, and takes $\perp$ to the element 0 in PROP,
4. D is a function which maps types into domains of $\mathcal{M}$ as follows:
   (a) $D_e = \mathcal{F}_0$
   (b) $D_p =$ PROP
   (c) $D_t =$ TRUTH
   (d) $D_{\langle \sigma, \tau \rangle} = \begin{cases} D_\sigma \hookrightarrow D_\tau, \text{ if } D_\tau \in \boldsymbol{Dom_2} \\ D_\sigma \Rightarrow D_\tau \text{ if } D_\sigma \in \boldsymbol{Dom_1} \text{ and } D_\tau \in \boldsymbol{Dom_1} \end{cases}$
   (e) $D_{(\sigma \to \tau)} = \{\boldsymbol{f} : \boldsymbol{f}$ is an $\mathcal{F}$-functional such that for all $x \in D_\sigma, \boldsymbol{f}(x) \in D_\tau\}$,
5. g is an assignment function which takes any variable of type σ to an object in D_σ.

Note that $\boldsymbol{Dom_1} \cap \boldsymbol{Dom_2}$ is empty and that $\boldsymbol{Dom_1}$ will interpret le-types which are not fp-types, among others. $\boldsymbol{Dom_2}$ will interpret the fp-types, among others.

Since we do not allow variables to range over $\mathcal{F}$-functionals, the interpretation function $\mathcal{I}$ is sufficient to determine the denotation of functors.

We now define a valuation function $[\![\ ]\!]$ which given an expression α and an assignment g yields a value in $\boldsymbol{Dom_1} \cup \boldsymbol{Dom_2}$.

1. $[\![c]\!]_{\mathcal{M},g} = \mathcal{I}(c)$
2. $[\![x]\!]_{\mathcal{M},g} = g(x)$
3. $[\![app(\alpha, \beta)]\!]_{\mathcal{M},g} = \mathbf{app}([\![\alpha]\!]_{\mathcal{M},g}, [\![\beta]\!]_{\mathcal{M},g})$
4. $[\![\alpha(\beta)]\!]_{\mathcal{M},g} = [\![\alpha]\!]_{\mathcal{M},g}([\![\beta]\!]_{\mathcal{M},g})$

5. $[\![{}^{\cup}(\alpha)]\!]_{\mathcal{M},g} = \mid [\![\alpha]\!]_{\mathcal{M},g} \mid$

6. $[\![\lambda x{:}\sigma.\varphi]\!]_{\mathcal{M},g} = \boldsymbol{\lambda} f$, where $f \in \mathcal{F}_1$ and $f(\boldsymbol{a}) = [\![\varphi]\!]_{\mathcal{M},g[\boldsymbol{a}/x]}$ for all $\boldsymbol{a} \in D_\sigma$

7. $[\![\neg\varphi]\!]_{\mathcal{M},g} = \neg[\![\varphi]\!]_{\mathcal{M},g}$

8. $[\![\varphi \wedge \psi]\!]_{\mathcal{M},g} = [\![\varphi]\!]_{\mathcal{M},g} \wedge [\![\psi]\!]_{\mathcal{M},g}$

9. $[\![\varphi \vee \psi]\!]_{\mathcal{M},g} = [\![\varphi]\!]_{\mathcal{M},g} \vee [\![\psi]\!]_{\mathcal{M},g}$

10. $[\![\varphi \supset \psi]\!]_{\mathcal{M},g} = [\![\varphi]\!]_{\mathcal{M},g} \supset [\![\psi]\!]_{\mathcal{M},g}$

11. $[\![\forall x{:}\sigma.\varphi]\!]_{\mathcal{M},g} = \forall \boldsymbol{f}$, where $\boldsymbol{f} \in \mathcal{F}_1$ and $f(\boldsymbol{a}) = [\![\varphi]\!]_{\mathcal{M},g[\boldsymbol{a}/x]}$ for all $\boldsymbol{a} \in D_\sigma$

12. $[\![\exists x{:}\sigma.\varphi]\!]_{\mathcal{M},g} = \exists \boldsymbol{f}$, where $\boldsymbol{f} \in \mathcal{F}_1$ and $\boldsymbol{f}(\boldsymbol{a}) = [\![\varphi]\!]_{\mathcal{M},g[\boldsymbol{a}/x]}$ for all $\boldsymbol{a} \in D_\sigma$

It will be observed that these valuation clauses depend on the existence of the appropriate functionals (e.g., *app*, $\neg, \wedge, \vee, \forall, \exists$) in the Frege structure. It would be straightforward to convert the clauses for propositions into truth-theoretic definitions, along the following lines:

8′ $[\![\varphi \supset \psi]\!]_{\mathcal{M},g} \in$ TRUTH $\iff$ $[\![\psi]\!]_{\mathcal{M},g} \in$ TRUTH whenever $[\![\varphi]\!]_{\mathcal{M},g} \in$ TRUTH

9′ $[\![\forall x{:}\sigma.\varphi]\!]_{\mathcal{M},g} \in$ TRUTH $\iff$ $[\![\varphi]\!]_{\mathcal{M},g[\boldsymbol{a}/x]} \in$ TRUTH for all $\boldsymbol{a} \in D_\sigma$

Lemma 13 $D_{\langle\sigma,\tau\rangle} = (D_\sigma \Rightarrow D_\tau) \subseteq D_e$ *if* $D_\sigma, D_\tau \in \boldsymbol{Dom_1}$
Proof *Obvious by Lemma 8.* □

Lemma 14 $D_{\langle\sigma,\tau\rangle} \subseteq D_\sigma$ *if* $D_\tau \in \boldsymbol{Dom_2}$
Proof *If* $D_\tau \in \boldsymbol{Dom_2}$, *then* $D_{\langle\sigma,\tau\rangle} = (D_\sigma \hookrightarrow D_\tau) \subseteq D_\sigma$ *by Lemma 9 .* □

Lemma 15 *For any type* σ, D_σ *is either in* $\boldsymbol{Dom_1}$ *or in* $\boldsymbol{Dom_2}$.
Proof *By induction on the construction of types.*

- *If* $\sigma = p$, t, *or* e, *then obvious.*
- *If* $\sigma = \langle\sigma_1, \sigma_2\rangle$, *where the property holds for* σ_1 *and* σ_2, *then also obvious.* □

Lemma 16 $D_\sigma \subseteq D_e$ *for any type* σ.[8]
Proof *By induction on* σ.

[8]Of course, the domain for $(\tau_1 \to \tau_2)$ is not contained in D_e; but this follows from the fact that $(\tau_1 \to \tau_2)$ is not a type but a metatype.

- *σ is base type (i.e., e, t or p). Obvious.*
- *Assume $\sigma = \langle\sigma_1, \sigma_2\rangle$, where $D_{\sigma_1} \subseteq D_e$ and $D_{\sigma_2} \subseteq D_e$.*

  Case 1 *$D_\sigma = D_{\sigma_1} \hookrightarrow D_{\sigma_2}$. Then $D_\sigma \subseteq D_{\sigma_1}$ by Lemma 14. Since $D_{\sigma_1} \subseteq D_e$, by induction hypothesis, we have $D_\sigma \subseteq D_e$.*

  Case 2 *$D_\sigma = D_{\sigma_1} \Rightarrow D_{\sigma_2}$. Then $D_\sigma \subseteq D_e$ by Lemma 13.* □

Lemma 17 *If $\sigma \preceq \tau$, then $D_\sigma \subseteq D_\tau$.*
Proof

($e \preceq$) *Case $\sigma \preceq e$.*
$D_\sigma \subseteq D_e$ always holds, by Lemma 16.

($p \preceq$) *Case $t \preceq p$.*
$D_t \subseteq D_p$, since TRUTH $\subseteq$ PROP.

(*Ran* $\preceq$) *Case $\tau_1 \preceq \tau_2$.*

1. *If $D_{\tau_1}, D_{\tau_2} \in$ **Dom₂** then use Lemma 11.*

2. *If $D_{\tau_1}, D_{\tau_2} \in$ **Dom₁** then use Lemma 10.*

3. *It cannot be the case that $D_{\tau_1} \in$ **Dom₁** and $D_{\tau_2} \in$ **Dom₂**.*

4. *If $D_{\tau_1} \in$ **Dom₂** and $D_{\tau_2} \in$ **Dom₁** then $D_{\langle\sigma, \tau_1\rangle} = (D_\sigma \hookrightarrow D_{\tau_1})$ and $D_{\langle\sigma, \tau_2\rangle} = (D_\sigma \Rightarrow D_{\tau_2})$. It is easy to check $(D_\sigma \hookrightarrow D_{\tau_1}) \subset (D_\sigma \Rightarrow D_{\tau_2})$.*

(*Id* $\preceq$, *Trans* $\preceq$, *Anti* $\preceq$) *Obvious.*

(*Prop* $\preceq$) *$\tau \preceq p$ implies $D_{\langle\sigma,\tau\rangle} \subseteq D_\sigma$. This holds since $D_{\langle\sigma,\tau\rangle} = (D_\sigma \hookrightarrow D_\tau) \subseteq D_\sigma$, by Lemma 9.*

(*Fix* $\preceq$) *$\tau \preceq p$ implies $D_{\langle\langle\sigma,\tau\rangle,\tau\rangle} = D_{\langle\sigma,\tau\rangle}$. We need to show that $(D_\sigma \hookrightarrow D_\tau) \hookrightarrow D_\tau = D_\sigma \hookrightarrow D_\tau$. From the proof of (Prop $\preceq$) above, it follows that $D_{\langle\langle\sigma,\tau\rangle,\tau\rangle} \subseteq D_{\langle\sigma,\tau\rangle}$. The reverse inclusion is established as follows. Let $\boldsymbol{x} \in (D_\sigma \hookrightarrow D_\tau)$. This implies that for all $\boldsymbol{x}' \in (D_\sigma \hookrightarrow D_\tau) \subseteq D_\sigma$, $\mathbf{app}(\boldsymbol{x}, \boldsymbol{x}') \in D_\tau$. Hence $\boldsymbol{x} \in (D_\sigma \hookrightarrow D_\tau) \hookrightarrow D_\tau$.*

(*Dom* $\preceq$) *$\sigma_1 \preceq \sigma_2$ implies $D_{\langle\sigma_2, e\rangle} \subseteq D_{\langle\sigma_1, e\rangle}$. By the induction hypothesis, $\sigma_1 \preceq \sigma_2$ implies $D_{\sigma_1} \subseteq D_{\sigma_2}$. D_e is in **Dom₁** and as we restrict types so that a domain type is never strictly less than the range type, then D_{σ_1}*

and D_{σ_2} must be in $\boldsymbol{Dom_1}$. Hence by definition, $D_{\langle\sigma_2,e\rangle} = D_{\sigma_2}\Rightarrow D_e$. Let $\boldsymbol{x} \in D_{\langle\sigma_2,e\rangle}$. Hence $\boldsymbol{x} \in (D_{\sigma_2}\Rightarrow D_e)$. So $\boldsymbol{x} \in \mathcal{F}_0$ and for all $\boldsymbol{x}' \in D_{\sigma_2}, \mathbf{app}(\boldsymbol{x},\boldsymbol{x}') \in D_e$ and $D_{\langle\sigma_1,e\rangle} = D_{\sigma_1}\Rightarrow D_e$. Since $D_{\sigma_1} \subseteq D_{\sigma_2}$, it follows that $\boldsymbol{x} \in \mathcal{F}_0$ and for all $\boldsymbol{x}' \in D_{\sigma_1}, \mathbf{app}(\boldsymbol{x},\boldsymbol{x}') \in D_e$. Hence $\boldsymbol{x} \in D_{\sigma_1}\Rightarrow D_e$. □

Theorem 1 *If $\Gamma\vdash\alpha{:}\sigma$, where $D_\sigma \in \boldsymbol{Dom_1}$ then $[\![\alpha]\!]_{\mathcal{M},g} \in D_\sigma$.*

Proof

- *If α is a constant c or variable x, this is obvious from the definition of $\mathcal{I}$ and g.*
- *Let us assume the property holds for expressions α, β, and show that it holds for $app(\alpha,\beta)$. $\Gamma\vdash app(\alpha,\beta){:}\tau$ iff $\Gamma\vdash\alpha{:}\langle\sigma,\tau\rangle$ and $\Gamma\vdash\beta{:}\sigma$. So $[\![\alpha]\!]_{\mathcal{M},g} \in D_{\langle\sigma,\tau\rangle}$, $[\![\beta]\!]_{\mathcal{M},g} \in D_\sigma$, and $[\![app(\alpha,\beta)]\!]_{\mathcal{M},g} = \mathbf{app}([\![\alpha]\!]_{\mathcal{M},g}, [\![\beta]\!]_{\mathcal{M},g})$. The latter belongs to D_τ, as $D_{\langle\sigma,\tau\rangle} = D_\sigma \Rightarrow D_\tau$.*
- *Let us prove $[\![\lambda x{:}\sigma.\alpha]\!]_{\mathcal{M},g} \in D_{\langle\sigma,\tau\rangle}$, where $\Gamma\vdash_\Sigma \tau$ le-type, $\Gamma \not\vdash_\Sigma \sigma \prec \tau$, and by induction hypothesis $[\![\alpha]\!]_{\mathcal{M},g[\boldsymbol{a}/x]} \in D_\tau$ for all $x{:}\sigma$.*

  Since $D_{\langle\sigma,\tau\rangle}$ is in $\boldsymbol{Dom_1}$ then $D_{\langle\sigma,\tau\rangle} = D_\sigma \Rightarrow D_\tau$. Hence $[\![\lambda x{:}\sigma.\alpha]\!]_{\mathcal{M},g} \in \mathcal{F}_0$, and for all $\boldsymbol{a} \in D_\sigma$, $\mathbf{app}([\![\lambda x{:}\sigma.\alpha]\!]_{\mathcal{M},g}, \boldsymbol{a}) = \mathbf{app}(\boldsymbol{\lambda f}, \boldsymbol{a}) = \boldsymbol{f}(\boldsymbol{a})$, where $\boldsymbol{f}(\boldsymbol{a}) = [\![\alpha]\!]_{\mathcal{M},g[\boldsymbol{a}/x]} \in D_\tau$. Hence $[\![\lambda x{:}\sigma.\alpha]\!]_{\mathcal{M},g} \in D_\sigma\Rightarrow D_\tau$. □

4 A Fragment of English

The English fragment that we consider is intentionally simple,[9] and will focus attention on issues of polymorphism and self-application. One possible way of setting up the grammar would be to follow Montague in using the standard fractional notation of categorial grammar, together with a homomorphism which maps the categories into semantic types. However, for our purposes, it would be preferable to build the syntactic categories directly on top of the types. Consequently, the categories of the grammar will consist of *decorated* types and metatypes of $\mathcal{L}_{\preceq}$; that is, types and metatypes annotated with phrase structure labels. The latter will provide us with the power to draw somewhat finer distinctions of the kind required for English syntax. For example, (untensed) intransitive verbs, adjectives and common nouns will all belong to the type $\langle e,e\rangle$; however, this type will be annotated as $\langle e,e\rangle^V$, $\langle e,e\rangle^A$, or $\langle e,e\rangle^N$, respectively.

[9]In particular, we do not treat quantified noun phrases. It would be straightforward to implement [Chierchia and Turner 88]'s treatment of type-shifting for quantifier arguments. It is unclear to us, however, what the appropriate analysis of scope would be in the current setting. Promising approaches include [Pereira 91] and [Emms 91].

The list of admissible labels is the following: S (sentences), V (verbs), N (nouns), CN (common nouns), A (adjectives), P (prepositions), Adv (adverbials).

In some cases, we extend these labels with feature specifications along the lines of [Gazdar *et al* 85]. For example, we use 'P[to]' as the label for prepositional phrases whose head is the word **to**, and 'VFORM' to specify the inflectional status of a verb or verb phrase. Thus V[VFORM BSE], V[VFORM FIN], V[VFORM PSP] indicate verbal categories in, respectively, base form (e.g., **be**), finite form (e.g., **is**), and present participial form (e.g., **being**). We use X as an underspecified category label; this will be useful when we want to give a maximally general decoration to a type.

Whenever η is a kind, and C is a category label, then η^C is a decorated kind. The rules given previously for constructing a complex kind can be generalized in the obvious way to decorated kind. We use the symbols 's, t, r' as metavariables ranging over decorated kinds. It is obvious that we can simply strip the labels off a decorated kind s to recover our original kind. We use '$^\circ s$' to denote the stripped-down version of s, where $^\circ\langle s,t\rangle^C = \langle ^\circ s, ^\circ t\rangle$, and $^\circ(s \to r)^C = (^\circ s \to ^\circ t)$.

An English grammar object will be a triple

$$(w, s, \alpha)$$

where w is a phonological (in practice, orthographic) form, s is a decorated kind, α is an expression of $\mathcal{L}_{\preceq}$, and moreover $\Gamma \vdash_\Sigma \alpha{:}^\circ s$, with Σ as specified before.

As a typographical convenience, we shall also employ the following vertical format for these triples:

$$\begin{array}{l} w \\ s \\ \alpha \end{array}$$

For example, the representations of the words **John** and **kiss** can be stated as:

(19)
$$\begin{array}{ll} \textbf{John} & \textbf{kiss} \\ e^N & \langle e^N, \langle e,e\rangle^V\rangle^{\,V} \\ \textbf{john}' & \textbf{kiss}' \end{array}$$

Thus, **kiss** has the type of a verbal expression which will combine with something of type e^N to make something of type $\langle e,e\rangle^V$. The decorated type therefore combines standard categorial information, which would usually be notated VP/NP (i.e., a functor which combines with an NP to make VP), together with the semantic type that such a category would be mapped into.

The rules of type inference are like those for $\mathcal{L}_{\preceq}$ with some provisos which we will come to shortly.

It will be noticed that the type assigned to **kiss**, namely,

(20) $\langle e^N, \langle e,e\rangle^V\rangle^{\,V}$,

appears redundant in the sense that not only is the type as whole specified to be V, but the result type, $\langle e, e\rangle^V$, is also so specified. Yet inasmuch as **kiss** is the head of verb phrase, it should be predictable that the result type has the same category decoration as the whole complex type. In response to this observation, we adopt the convention that if the result type lacks a decoration, then it can be inferred from the decoration of the enclosing decorated type; in other words, a type like (21) is shorthand for (20).

(21) $\langle e^N, \langle e, e\rangle\rangle^V$

We can make this more explicit by means of a modified inference rule of the following sort (where σ is restricted to *un*decorated types):

(22) $$\frac{\alpha{:}\langle s, \sigma\rangle^C \qquad \beta{:}s}{\mathit{app}(\alpha, \beta){:}\sigma^C}$$

Our grammar for English is non-directional, in the sense that we do not encode whether a functor seeks its argument to the left or to the right. Modifying the notation to allow this would be trivial, but would add an extra degree of complexity which would detract from the main thrust of the exposition. For convenience, we shall simply write the premisses of a type inference rule in the correct left-to-right order, and stipulate that the string in the conclusion is the right of concatenating the strings of the premisses. This is shown in the following schema for type inference in the fragment, (where '$\frown$' indicates concatenation):

Definition 9 (Concatenation Schema for English)

$$(\mathit{Concat}) \quad \frac{(w_1, s_1, \alpha_1) \qquad (w_2, s_2, \alpha_2)}{(w_1{}^\frown w_2, s_3, \alpha_3)}$$

is valid only if the corresponding inference

$$\frac{\Gamma \vdash_\Sigma \alpha_1{:}{}^\circ s_1 \qquad \Gamma \vdash_\Sigma \alpha_2{:}{}^\circ s_2}{\Gamma \vdash_\Sigma \alpha_3{:}{}^\circ s_3}$$

is derivable for the undecorated types $^\circ s_1$, $^\circ s_2$, *and* $^\circ s_3$.

4.1 Verb-Object Combination

Whether a verb is tensed affects its ability to combine with a subject, but not its ability to combine with object arguments and complements:

(23) a. to kiss Mary/kissed Mary

b. *John to kiss Mary/John kissed Mary

Now, in the semantic framework we have developed there are two distinct ways in which a syntactic functor can combine semantically with an argument: either via the *app* relation, or by normal functional application. Moreover, *app* is invoked for functors which we earlier called 'nominal predicatives', i.e., expressions which denote objects in the Frege structure domain $\mathcal{F}_0$; by contrast, expressions which denote propositional functions live outside $\mathcal{F}_0$ and therefore cannot act as nominal arguments of *app*. In this section, we examine how *app* comes into play when we combine verbs with their objects.

Let us start by looking at transitive and intransitive verbs. The base, or nontensed, form of an intransitive verb like **run** is translated as a constant **run′** of type $\langle e, e\rangle^V$; as we observed in Section 1, such constants denote (a special sort of) nominal objects, not propositional functions. Similarly, the base form of a transitive verb such as **kiss** is translated as a constant **kiss′** of type $\langle e^N, \langle e, e\rangle\rangle^V$, which also denotes a sort of nominal object.

The schema in Definition 9 licenses derivations like the following:

(24)
$$\frac{\begin{array}{l}\textbf{kiss}\\ \langle e^N, \langle e^N, e\rangle\rangle^{V[\text{VFORM BSE}]}\\ \textbf{kiss}'\end{array} \qquad \begin{array}{l}\textbf{Mary}\\ e^N\\ \textbf{mary}'\end{array}}{\begin{array}{l}\textbf{kiss Mary}\\ \langle e^N, e\rangle^{V[\text{VFORM BSE}]}\\ \mathit{app}(\textbf{kiss}', \textbf{mary}')\end{array}}\,(\mathit{Concat})$$

The same approach extends to verbs which combine with more than one complement. Before considering such a case, let us introduce some new notation to indicate the iterated application of a functor α to a series of arguments:

Definition 10 (Multiple Application)

$$[\alpha, x_1, \ldots, x_n] =_{\mathit{df}} \mathit{app}(\ldots(\mathit{app}(\alpha, x_1), \ldots), x_n)$$

Assuming **give′** to be of type $\langle e^{P[\text{to}]}, \langle e^N, \langle e, e\rangle\rangle\rangle^V$, we have the following semantic translation for **give the cat to Mary**:

(25) $[\textbf{give}', \textbf{mary}', \textbf{(the cat)}']{:}\langle e, e\rangle = \mathit{app}(\mathit{app}(\textbf{give}', \textbf{mary}'), \textbf{(the cat)}')$

The last step in the derivation of (25) is:[10]

(26)
$$\frac{\begin{array}{l}\textbf{give to Mary}\\ \langle e^N, \langle e, e\rangle\rangle^V\\ [\textbf{give}', \textbf{mary}']\end{array} \qquad \begin{array}{l}\textbf{the cat}\\ e^N\\ \textbf{the cat}'\end{array}}{\begin{array}{l}\textbf{give the cat to Mary}\\ \langle e, e\rangle^V\\ [\textbf{give}', \textbf{mary}', \textbf{(the cat)}']\end{array}}$$

[10]As presented, this derivation would require a wrapping operation to produce the conclusion, rather than just string concatenation; cf. [Dowty 82] for some arguments in favour of this approach. Alternatively, we could have combined **give the cat** with **to Mary** after making appropriate modifications to the type of **give**.

4.2 Verb-Subject Combination

In our discussion of predication in Section 1, we argued that untensed verb phrases should be assigned the type $\langle e, e\rangle$. Let us consider how this type enters into our English fragment, taking the string ***John walk** as an example.[11] As the following derivation shows, we can infer a type for the string, namely e:

$$(27) \qquad \frac{\begin{array}{ll}\textbf{John} & \textbf{walk} \\ e^{N} & \langle e, e\rangle^{V[\text{VFORM BSE}]} \\ \textbf{john}' & \textbf{walk}'\end{array}}{\begin{array}{l}\textbf{John walk} \\ e^{V[\text{VFORM BSE}]} \\ \mathit{app}(\textbf{walk}', \textbf{john}')\end{array}}\,(\mathit{Concat})$$

But **John walk** comes out as anomalous *qua* sentence. That is, since it does not have the type p, it does not have the semantic value which we would expect a sentence to bear. And although we know that **John walk** does denote *some* object in the semantic domain, our type rules give us no means of inferring the more specific conclusion that it expresses a proposition. Similar reasoning will label as deviant strings like ***John to run**.

What we must do now is make explicit the way in which tense is introduced. From a semantic point of view, it is easiest map untensed verb phrases into tensed ones.[12] That is, we require a rule which will convert a phrase like **give the cat to Mary** into **gives the cat to Mary**. We accomplish this by means of an inference schema like the following (where C is a metavariable over categories):

Definition 11 (Tense Introduction)

$$(\mathit{Tense}) \qquad \frac{(w, \langle e^{C}, e\rangle^{V[\text{VFORM BSE}]}, \alpha)}{(\text{INFL}(w), (e^{C} \rightarrow p)^{V[\text{VFORM FIN}]}, {}^{\cup}\alpha)}$$

The type change in this rule is closely coupled with the introduction of the predication operator in the semantics. That is, given an expression w which combines with an e(in effect, any nominalizable expression) to yield an e, we can infer that $\text{INFL}(w)$ will combine with that same argument to yield a proposition. And whereas w denoted some object in $\mathcal{F}_0$, $\text{INFL}(w)$ denotes a function from $\mathcal{F}_0$ to PROP.

INFL is intended to be a morphological operation which assigns appropriate inflections to the verbal head(s) of its argument. In a more detailed treatment, the operation would need to be parameterized for person, number, and case. Moreover, in addition to denominalizing the interpretation of α *via* $^{\cup}$, the semantic correlates of, say, past tense would need to be accommodated. To

[11] We are ignoring the analysis under which **walk** is present tense, but not third person.

[12] This has been the standard approach in most Montagovian approaches, and is also the one adopted by [Chierchia and Turner 88].

simplify exposition, however, we shall confine our attention to one instance of INFL, namely third person singular present.

This inference rule is illustrated in (28) and (29) below.

(28)
$$\frac{\begin{array}{l}\textbf{be fun}\\ \langle e^X, e\rangle^{V[\text{VFORM BSE}]}\\ [\mathbf{be}', \mathbf{fun}']\end{array}}{\begin{array}{l}\textbf{is fun}\\ (e^X \to p)^{V[\text{VFORM FIN}]}\\ {}^{\cup}[\mathbf{be}', \mathbf{fun}']\end{array}}\,(\mathit{Tense})$$

Notice that **be** places no restrictions on the syntactic decoration of its subject argument, requiring only that it be of type e. By contrast, **walk** is subcategorized to take a nominal subject:

(29)
$$\frac{\begin{array}{l}\textbf{walk}\\ \langle e^N, e\rangle^{V[\text{VFORM BSE}]}\\ \mathbf{walk}'\end{array}}{\begin{array}{l}\textbf{walks}\\ (e^N \to p)^{V[\text{VFORM FIN}]}\\ {}^{\cup}\mathbf{walk}'\end{array}}\,(\mathit{Tense})$$

As a further illustration, we show how a tensed intransitive verb combines with a subject noun phrase:

(30)
$$\frac{\begin{array}{ll}\textbf{John} & \textbf{walks}\\ e^N & (e^N \to p^S)^V\\ \mathbf{john}' & {}^{\cup}\mathbf{walk}'\end{array}}{\begin{array}{l}\textbf{John walks}\\ p^S\\ {}^{\cup}\mathbf{walk}'(\mathbf{john}')\end{array}}\,(\mathit{Funct})$$

Table 1 summarizes the assignment of categories to expressions of English in our fragment. A major distinction is drawn between those expressions which receive ordinary types, and are therefore open to nominalization, and those which receive metatypes, and can never be nominalized. The notion of 'predicative', which we appealed to at the beginning of this paper, cuts across this distinction. That is, it was intended to cover expressions with type $\langle e, e\rangle$, which can be nominalized, and expressions with metatype $(\sigma \to \tau)$, which cannot.

It will be observed that there is a broad correspondence between our type '$\langle e, e\rangle$' and the [Chierchia and Turner 88] sort '*nf*', standing for nominalized functions, and to this extent the two fragments are quite similar.

Note in passing that we have chosen to analyse **fun** as a mass noun rather than an adjective, on the grounds that collocations involving noun modifiers, as (31a), seem significantly better than those involving adjectival modifiers, as (31b):

Informal Name	*Type*	*Basic Expressions*
	Nominalizable Expressions	
NP	e^N	**John, Mary**
CN$_{count}$	$\langle e,e\rangle^{CN}$	**dog, man, woman, park**
CN$_{mass}$	$\langle e,e\rangle^{N}$	**water, gold, fun**
ADJ	$\langle e,e\rangle^{A}$	**happy, drunk, old**
PP	$\langle\langle e,e\rangle^X,\langle e,e\rangle^X\rangle^P$	$\emptyset$
TV	$\langle e^N,\langle e^N,e\rangle\rangle^V$	**kiss, seek**
	$\langle e^X,\langle e^N,e\rangle\rangle^V$	**believe, know**
	$\langle\langle e,e\rangle^{V[\text{VFORM INF}]},\langle e^N,e\rangle\rangle^V$	**seem, try, want**
	$\langle\langle e,e\rangle^X,\langle e^X,e\rangle\rangle^V$	**be**
TTV	$\langle e^{P[\text{to}]},\langle e^N,\langle e^N,e\rangle\rangle\rangle^V$	**give, send**
	$\langle\langle e,e\rangle^{V[\text{VFORM INF}]},\langle e^N,\langle e^N,e\rangle\rangle\rangle^V$	**force, believe**
IV	$\langle e^N,e\rangle^V$	**run, walk, talk**
S	p^S	$\emptyset$
S′	$p^{S[\text{COMP}]}$	$\emptyset$
IV$_{inf}$	$\langle e,e\rangle^{V[\text{VFORM INF}]}$	$\emptyset$
	Non-nominalizable Expressions	
Det	$(\langle e,e\rangle^{CN}\rightarrow e^N)^{Det}$	**the, a, every**
VP	$(e^X\rightarrow p^S)$	$\emptyset$
AdSent	$(p^S\rightarrow p^S)^{Adv}$	**necessarily, possibly**
AdVerb	$(\langle e,e\rangle^V\rightarrow\langle e,e\rangle^V)^{Adv}$	**almost, slowly**
P	$(e^N\rightarrow\langle\langle e,e\rangle^X,\langle e,e\rangle^X\rangle)^P$	**in, with, to**
AdNom	$(\langle e,e\rangle^N\rightarrow\langle e,e\rangle^N)^A$	**former, alleged**
COMP	$(\langle e,e\rangle^V\rightarrow\langle e,e\rangle^{V[\text{VFORM INF}]})$	**to**
	$(p^S\rightarrow p^{S[\text{COMP THAT}]})$	**that**

Table 1: Categories and expressions in the fragment

(31) a. It wasn't much/a lot of fun.

b. ?It was extremely/very fun.

Nothing crucial hangs on this decision. Nevertheless, it follows on our account that all mass nouns can occur as nominal arguments. They can also occur as predicative complements by virtue of the polymorphic type assigned to **be**.

4.3 Nominalization and Polymorphism

As we indicated at the beginning of this paper, we do not employ a rule of nominalization as such. Rather, some expressions—the ones categorised as 'nominalizable' in Table 1—have kinds which are contained in the type e of individuals.

Then type containment is invoked to derive the more general type.

Let us take the following strings to illustrate the mechanisms:

(32)
 a. Mary is fun.
 b. Fun is fun.
 c. Running is fun.
 d. For us to run is fun.

We start off by considering how the untensed phrase **be fun** is derived:[13]

(33)
$$\dfrac{\begin{array}{l}\textbf{be}\\ \langle\langle e,e\rangle^X,\langle e^X,e\rangle\rangle^V\\ \textbf{be}'\end{array} \qquad \begin{array}{l}\textbf{fun}\\ \langle e,e\rangle^X\\ \textbf{fun}'\end{array}}{\begin{array}{l}\textbf{be fun}\\ \langle e^X,e\rangle^V\\ \mathit{app}(\textbf{be}',\textbf{fun}')\end{array}}\,(\mathit{Concat})$$

This in turn will constitute a premise for the inference (30) which derives the string **is fun**. The latter can be predicated of any string whose category is a possible argument for the type $(e^X \rightarrow p^S)^V$, that is, any string for which the category e^X can be inferred. Recall the axiom we presented earlier for deriving type containments:

$$(\mathit{Contain})\quad \frac{\Gamma\vdash_\Sigma \sigma\preceq\tau \qquad \Gamma\vdash_\Sigma \alpha{:}\sigma}{\Gamma\vdash_\Sigma \alpha{:}\tau}$$

As we showed in preceding sections, this gives us an account of inclusion polymorphism for the typed language $\mathcal{L}_{\preceq}$. In order to deal with polymorphism in the English fragment, we need to extend containment to our decorated types. To do this, we supplement $\preceq$ over types with a new partial order $\preceq^*$ over category labels. In the following definition s, t are decorated types, while C, D are category decorations:

Definition 12 (Containment of decorated types)

1. *$s^C \preceq t^D$ iff $^\circ s \preceq {^\circ t}$ and $C\preceq^* D$.*
2. *$\preceq^*$ is reflexive, transitive and antisymmetric.*
3. *$Cat \preceq^* X$, where Cat ::= { N, P, V, S }, and where V is not specified as* VFORM FIN.

[13]As we will shortly explain, the type $\langle e,e\rangle^X$ can be inferred for **fun** by a modified containment axiom.

This now gives rise to a modified Containment rule:

$$(\mathit{Contain*})\ \frac{s^C \preceq t^D \qquad (w, s^C, \alpha)}{(w, t^D, \alpha)}$$

The following example shows how the rule is invoked for the subject of (32b).

(34) $$\frac{\begin{array}{l}\textbf{fun}\\ \langle e,e\rangle^N\\ \textbf{fun}'\end{array}}{\begin{array}{l}\textbf{fun}\\ e^X\\ \textbf{fun}'\end{array}}(\mathit{Contain*})$$

The inference works in a completely parallel fashion for the other two cases:

(35) **running**
$\langle e,e\rangle^{V[\text{VFORM PSP}]}$
$\textbf{run}'$

(36) **for mary to run**
p^S
$\textbf{for}'(\mathit{app}(\textbf{run}', \textbf{mary}'))$

After the (*Contain**) inference, **fun** can combine with **is fun** as shown in (37):

(37) $$\frac{\begin{array}{l}\textbf{fun}\\ e^X\\ \textbf{fun}'\end{array} \qquad \begin{array}{l}\textbf{is fun}\\ (e^X \to p^S)^V\\ {}^{\cup}[\textbf{be}', \textbf{fun}']\end{array}}{\begin{array}{l}\textbf{fun is fun}\\ p^S\\ {}^{\cup}[\textbf{be}', \textbf{fun}'](\textbf{fun}')\end{array}}(\mathit{Funct})$$

We also need to show that certain expressions *cannot* act as nominal arguments. Consider for example

(38) *Runs is fun.

In order to derive a type for this, we would have to deduce that the type of **runs**, namely $(e^N \to p)^{V[\text{VFORM FIN}]}$ is contained in e^X. But $(e^N \to p)^{V[\text{VFORM FIN}]}$ is a metatype, not a proper type, and therefore not a subtype of e. Hence the (*Contain**) step which would be required for (38) in fact fails. The same reasoning shows that types cannot be derived for strings like (39) where an expression with a metatype is filling an argument role:[14]

[14]Examples like **Slowly would be fun** seem relatively acceptable. One conclusion could be that manner adverbs should have (at least) the type $\langle e, p\rangle$, perhaps as predicates of events. Alternatively, such examples might involve ellipsis of a modified verb phrase; cf. **Jogging fast would be a pain, but slowly would be fun**.

(39) *The/possibly/almost/to is fun.

In Section 1.3, we noted that certain verbs, such as **seem**, required a predicative rather than a nominal complement. This is witnessed by the following contrast:

(40) a. John seems to annoy Mary.

b. *John seems the boy.

In Table 1, we categorized **seem** as $\langle\langle e,e\rangle^{V[\text{VFORM INF}]},\langle e,e\rangle\rangle^{V}$; that is, it requires an expression of type $\langle e,e\rangle^{V[\text{VFORM INF}]}$ as argument.[15] This, of course, is the type which would be assigned to **to annoy Mary**. But definite noun phrases such as **the boy** are of type e^{N}. Could we use the (*Contain**) rule to infer that **the boy** is also of type $\langle e,e\rangle$? No, because the only type τ such that $e \preceq \tau$ is e itself. Hence (40b) will not be well-typed.

There is an interesting difference between our approach and that of [Chierchia and Turner 88] when nominalizations of verbs are considered. For Chierchia and Turner, only expressions of type *nf* are nominals. Since their nominalization operator is exclusively defined for expressions of type $\langle e,e\rangle$[16], and they do not have any kind of type containment for functional types, they do not allow transitive verbs like **love** and ditransitives like **give** to be nominalised. Yet examples such as (41a) (from [Partee 86]) and (41b) show that untensed transitive verbs enter into the same nominal patterns as intransitives:

(41) a. To love is to exalt.

b. To give is better than to receive.

By contrast, we have $\langle e^{N},\langle e^{N},e\rangle\rangle^{V[\text{VFORM INF}]} \preceq e^{X}$, and can thus accommodate such data straightforwardly.

5 Acknowledgements

We are grateful to David Beaver, Inge Bethke, Max Cresswell, Lex Holt, Uwe Mönnich and an anonymous referee for their comments on previous versions of this paper.

[15]In fact, this needs to be generalized, since it excludes **John seems happy**. However, this would have to form part of a more comprehensive analysis of syntactic containments in English.

[16]This type corresponds to our metatype $(e \rightarrow e)$.

References

[Aczel 80] Aczel, P. (1980) 'Frege Structures and the Notions of Proposition, Truth and Set.' In Barwise, J., Keisler, H. J. and Kunen, K. (eds.) *The Kleene Symposium*, pp31-59. Amsterdam: North Holland.

[Bach 79] Bach, E. (1979) 'Control in Montague Grammar.' *Linguistic Inquiry* 10, 515–531.

[Bealer 82] Bealer, G. (1982) *Quality and Concept.* Oxford: Clarendon Press.

[Bealer 89] Bealer, G. (1989) 'On the Identification of Properties and Propositional Functions.' *Linguistics and Philosophy* 12, 1–14.

[Cardelli *et al.* 85] Cardelli, L. and P. Wegner (1985) 'On understanding types, data abstraction and polymorphism.' *Computing Surveys* 17, 471–522.

[Chierchia 84] Chierchia, G. (1984) *Topics in the Syntax and Semantics of Infinitives and Gerunds.* Unpublished PhD Thesis, University of Massachusetts.

[Chierchia 85] Chierchia, G. (1985) 'Formal Semantics and the Grammar of Predication.' *Linguistic Inquiry* 16, pp.417–443.

[Chierchia and Turner 88] Chierchia, G. and R. Turner (1988) 'Semantics and Property Theory.' *Linguistics and Philosophy* 11, pp.261–302.

[Curien *et al.* 89] Curien, P.-L. and G. Ghelli (1989) 'Coherence of Subsumption.' Unpublished ms, Liens (CNRS), Paris.

[Dowty 82] Dowty, D. (1982) 'Grammatical Relations and Montague Grammar, pp. 79–130 in *The Nature of Syntactic Representation*, P. Jacobson and G. K. Pullum, eds., Dordrecht: Reidel.

[Emms 91] Emms, M. (1991) 'Polymorphic Quantifiers', pp. 65–112 in *Studies in Categorial Grammar*, G. Barry and G. Morrill, eds., Edinburgh: Centre for Cognitive Science, U of Edinburgh.

[Frege 77] Frege, G. (1977) *Translations from the Philosophical Writings of Gottlob Frege.* Geach, P. and Black, M. (eds.), 3rd Edition, pp56-78. Oxford: Basil Blackwell.

[Gazdar *et al* 85] Gazdar, G., Klein, E., Pullum G.K., and I.A. Sag (1985) *Generalized Phrase Structure Grammar.* Oxford: Basil Blackwell.

[Harper *et al.* 87] Harper, R., Honsell, F., and G. Plotkin (1987) 'A Framework for Defining Logics.' *Second Annual Symposium on Logic in Computer Science*, IEEE, pp.194–204.

[Jacobson 90] Jacobson, P. (1990) 'Raising as Function Composition.' *Linguistics and Philosophy* 13, pp.423–475.

[Martin-Löf 79] Martin-Löf, P. (1978) 'Constructive Mathematics and Computer Programming.' In *Logic, Methodology and Philosophy of Science, VI, 1979*, pp.153–175, North-Holland.

[Milner 78] Milner, R. (1978) 'A Theory of Type Polymorphism in Programming.' *Journal of Computer and System Sciences* 17, pp.348–375.

[Mitchell 88] Mitchell, J. C. (1988) 'Polymorphic Type Inference and Containment.' *Information and Computation* 76, pp.211–249.

[Montague 73] Montague, R. (1973) 'The proper treatment of quantification in ordinary English.' In Hintikka, J., Moravcsik, J. M. E. and Suppes, P. (eds.) *Approaches to Natural Language.* Dordrecht: D. Reidel. Reprinted in R. H. Thomason (ed.) (1974), *Formal Philosophy: Selected Papers of Richard Montague*, pp247-270. Yale University Press: New Haven, Conn.

[Parsons 79] Parsons, T. (1979) 'The theory of types and ordinary language.' In S. Davies and M. Mithun (eds.) *Linguistics, Philosophy and Montague Grammar*, University of Texas Press, Austin.

[Partee *et al.* 83] Partee, B. H. and M. Rooth (1983) 'Generalized conjunction and type ambiguity.' In R. Bäuerle, C. Schwarze, and A. von Stechow (eds.) *Meaning, Use, and Interpretation of Language*, De Gruyter.

[Partee 84] Partee, B. H. (1984) 'Compositionality.' In F. Landman and F. Veltman (eds.) *Varieties of Formal Semantics: Proceedings of The Fourth Amsterdam Colloquium, Sept 1982* Foris Press, Dordrecht.

[Partee 86] Partee, B. H. (1986) 'Ambiguous pseudo-clefts with ambiguous *be*.' In S. Berman, J. Choe and J. McDonough (eds.) *Proceedings of the Sixteenth Annual Meeting of the North Eastern Linguistic Society*, University of Massachusetts, Amherst.

[Pereira 91] Pereira, F. (1991) 'Deductive Interpretation', pp. 117–133 in *Natural Language and Speech*, E. Klein and F. Veltman, eds., Berlin: Springer-Verlag.

[Pollard *et al.* 87] Pollard, C. and I. A. Sag (1987) *Information-Based Syntax and Semantics, Vol. 1.* CSLI Lecture Notes, No. 13.

[Rooth *et al.* 82] Rooth, M. and B. H. Partee Mats Rooth 'Conjunction, type ambiguity, and wide scope 'or'.' In M. Barlow, D. Flickinger and M. Westcoat (eds.) *Proceedings of the Second West Coast Conference on Formal Linguistics*, pp353-362.

[Scott 76] Scott, D. (1976) ‘Data Types as Lattices.’ *SIAM* Journal of Computing, 5, 522–587.

[Thomason 76] Thomason, R. H. (1976) ‘On the Semantic Interpretation of the Thomason 1972 Fragment’. Distributed by Indiana University Linguistics Club, Bloomington, Indiana.

[Turner 87] Turner, R. (1987) ‘A Theory of Properties.’ *Journal of Symbolic Logic* 52, 63–86.