

Type Polymorphism in English

Elias Ponvert

The University of Texas at Austin, 1 Univ. Sta. Austin TX 78712
efp@uts.cc.utexas.edu

Abstract. The quantifiers and connectives of English present difficulties to standard type theory in linguistics, in that they can apply to terms of different syntactic and logical types while preserving their core denotations. In this sense they are polymorphic. This paper presents a type-logical perspective on Montague-style types, based on the Curry-Howard correspondence; the emergent system is powerful enough not only to handle functional abstraction and application, but also to handle type-raising elegantly. However, it is inadequate to articulate type-expressions for logical elements, failing in particular to capture their polymorphism. A solution to this problem is presented by the polymorphic lambda calculus, a higher-order lambda calculus that explicitly quantifies and abstracts over types. The paper concludes with a discussion of the consequences this solution presents to the semantics of the type theory.

1 Background and Outline

It should go without saying that we expect natural languages such as English to have at least the expressive power as programming languages. One area of research in programming language semantics is *type theory*, which has also been studied explicitly in linguistics since at least Bar-Hillel’s work in the early 1950s.

Type theory in contemporary linguistics is inspired by the system of syntactic and semantic types introduced by Montague [5]’s PTQ system. In PTQ, Montague specified algebras of each syntactic and semantic types with a (many-one) homomorphism between them. Each type system contained an internal language: the semantic types’ internal language was the lambda calculus with Church-typing wrapped around Montague’s intensional logic (IL), the syntactic types’ internal language was the English fragment itself, or a disambiguated form thereof.

Research on quantification since Montague has tended to dispense with the intensional type of world-time pairs \mathbf{s} in lieu of the extensional types of entities \mathbf{e} and truth-values \mathbf{t} , and functional types $A \rightarrow B$ ($\langle A, B \rangle$) for types A, B —i.e. the set $\{\mathbf{e}, \mathbf{t}\}$ closed under the type constructor \rightarrow .

One can treat a type system like this as a logic wherein the functional types $A \rightarrow B$ are treated as implicational statements: e.g. the combination of expressions of type \mathbf{e} (“*John*”) and type $\mathbf{e} \rightarrow \mathbf{t}$ (“*sleeps*”) combine to form an expression of type \mathbf{t} (“*John sleeps*”) akin to the logical inference

$$\mathbf{e}, \mathbf{e} \rightarrow \mathbf{t} \vdash \mathbf{t}$$

This last point introduces a distinction between the *information-content logic* and the *information-packaging logic*. The ICL, which characterizes the truth-conditions and object-level meanings of terms and phrases, is less discussed here. The IPL, however, the subject of this paper, has only indirect influence on the truth-conditions of NL clauses in that it characterizes the *form* of the content logic expressions.

So, seen as a logic, the type system built from Montague’s PTQ is strong enough to prove Montague and Partee-style type-raising. But, it is not strong enough to capture the polymorphic behavior of the logical elements, the quantifiers and connectives. I argue that a polymorphic extension to the term language, which corresponds to a second-order extension to the type system, is necessary to capture the polymorphic nature of natural language terms. This will have strong consequences in the picture above, namely that, the standard naive interpretation of the type system will have to be rejected. I conclude with a discussion of alternative type interpretations, and what consequences they will have for the term language.

2 Quantifiers in English

Consider

Ex 1. *Every person in this room knows Tai Chi.*

Ex 2. *Every time I eat Thai food, I order fried tofu.*

Ex 3. *John is trying to lose weight every way he can.*

In each example, a generalization is made over a certain domain: in Ex 1 over persons, in Ex 2 over times or time-indices, and in Ex 3 over manners or causes. These distinctions induce difference senses for each indicative. However, the distinctions between these expressions extends beyond sense; each quantified noun phrase (QNP) induces difference grammatical dependencies as well. Dropping the QNP from each clause illustrates this point:

Ex 4. ** Knows Tai Chi.*

Ex 5. *? I order fried tofu.*

Ex 6. *John is trying to lose weight.*

The QNP in Ex 4 provides the subject to the indicative, hence its absence renders the clause totally ungrammatical. Such QNPs provide *generalized arguments* to the clause. Likewise for QNPs in a clause’s object position such as

Ex 7. *John knows every person in the room.*

Ex 5 is loosely grammatical but missing a temporal argument to be sensible; it is reminiscent of the temporal deixis in

Ex 8. *(Darn!) I left the stove on.*

from Partee [6]. In such cases the QNPs provide a *generalized time* to the clause.

More so than the others, Ex 6 is grammatically and meaningfully OK. What is provided by the QNP in Ex 3 is a *generalized manner*. While structurally an NP, generalized manners have the distribution of manner adverbials such as “*by cutting down on soy lattes*” and “*quickly*” (hence, both causative and non-causative).

It is easy to see that many of the generalized quantifiers studied by Barwise and Cooper [1], which focused exclusively on generalized arguments, have analogous generalized times and generalized manners. The following are all meaningful

Ex 9. *Three out of four times I eat Thai food, I order fried tofu.*

Ex 10. *Most times I eat Thai food, I order fried tofu.*

although certain generalized times, especially when they assert the existence of a particular event-time, cannot not be paired with the English generic present tense:

Ex 11. * *One time I eat Thai food, I order fried tofu.*

Ex 12. *One time I ate Thai food, I ordered friend tofu.*

As mentioned above in Section 1, I do not so much address the truth-theoretic semantics of these expressions, which, especially in the case of generalized arguments, have been studied extensively in the field of generalized quantifiers. Rather, I am interested here in how these expressions can be provided with a meaningful type within the type-system, the information-packaging logic. First, however, let us sketch linguistic type theory before attempting to apply it to “every” in Section 4.

3 A Type-theoretic Sketch

Take the extensional fragment of the type theory of Montague [5]’s PTQ, as was adopted in Partee and Rooth [7] with basic types **e** and **t**. The set **Type** then is

Def 1 (Basic Type Syntax). $\text{Type} ::= \mathbf{e} \mid \mathbf{t} \mid \text{Type} \rightarrow \text{Type}$

Functional types $A \rightarrow B$ are often denoted $\langle A, B \rangle$ in the post-Montague literature. The naive interpretation (given by the interpretation function $\|\cdot\|$) for **Type** is

Def 2 (Basic Type Semantics).

$$\begin{aligned} \|\mathbf{e}\| &=_{\text{def}} \text{The set of model-entities} \\ \|\mathbf{t}\| &=_{\text{def}} \text{The set of truth-values} \\ \|A \rightarrow B\| &=_{\text{def}} \text{The (set) function space } \|B\|^{\|A\|} \end{aligned}$$

The *term language* of *internal language* of this system is the *Church-typed lambda calculus*, corresponding to the denotations of English terms. The term language is built from primitive terms $\text{Const} = \{ \text{j, LOVES, TOFU, ...} \}$ and term variables $\text{Var} = \{ f, g, \dots, x, y, z \}$ as such:

Def 3 (Basic Term Syntax). $\text{Term} ::= \text{Const} \mid \text{Var} \mid \lambda \text{Var}^{\text{Type}} \text{Term} \mid \text{Term}(\text{Term})$

Assume β and η reductions hold, modulo bound variables, between lambda terms:

Def 4 (Beta and Eta).

$$\begin{aligned} (\lambda x^A f)(a) &\Longrightarrow_{\beta} f[a/x] \\ \lambda x^A f(x) &\Longrightarrow_{\eta} f \text{ when } x \text{ not free in } f \end{aligned}$$

Typing judgments are set up as follows: a type environment is a partial function \mathcal{T} that takes elements of Const and Var into Type .

Def 5 (Type Environment).

$$\mathcal{T}, a : A =_{\text{def}} x \mapsto \begin{cases} A & \text{if } x = a \\ \mathcal{T}(x) & \text{otherwise} \end{cases}$$

So, if $\mathcal{T}(a)$ is undefined, then $\mathcal{T}, a : A = \mathcal{T} \cup a \mapsto A$. A *typing judgment*, then, is a relation over type environments and term-type pairs $a : A^1$. For a type environment \mathcal{T} and $x \text{ Var}$ and f in either Var or Const ,

Def 6 (Type Judgment Rules).

$$\begin{aligned} \frac{\mathcal{T}(x) = A}{\mathcal{T} \vdash a : A} \text{Ax} \quad \frac{\mathcal{T}, x : A \vdash f : B}{\mathcal{T} \vdash \lambda x^A f : A \rightarrow B} \text{Abs} \\ \frac{\mathcal{T} \vdash f : A \rightarrow B \quad \mathcal{T} \vdash x : A}{\mathcal{T} \vdash f(x) : B} \text{App} \end{aligned}$$

Invoking the Curry-Howard correspondence, it is easy to see that Abs and App correspond to the logical rules of \rightarrow -introduction and -elimination respectively. In fact, this system corresponds to the positive implicative intuitionistic calculus, which has a neat properties. For example, type-raising, as studied from Montague onwards, takes a term j of type \mathbf{e} and produces a term $\lambda X^{\mathbf{e} \rightarrow \mathbf{t}} X(j)$ of type $(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}$. That is, from an object j e.g. for John, you get the (characteristic function for) all properties of John. In symbols

$$j : \mathbf{e} \vdash \lambda X^{\mathbf{e} \rightarrow \mathbf{t}} X(j) : (\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}.$$

Type raising falls out of this system immediately, with the following:

Thm 1 (Type-raising). $j : \mathbf{e} \vdash \lambda X^{\mathbf{e} \rightarrow \mathbf{t}} X(j) : (\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}$.

Proof. Via

$$\frac{\begin{array}{c} j : \mathbf{e}, X : \mathbf{e} \rightarrow \mathbf{t} \vdash j : \mathbf{e} \quad (\text{Ax}) \\ j : \mathbf{e}, X : \mathbf{e} \rightarrow \mathbf{t} \vdash X : \mathbf{e} \rightarrow \mathbf{t} \quad (\text{Ax}) \\ \hline j : \mathbf{e}, X : \mathbf{e} \rightarrow \mathbf{t} \vdash X(j) : \mathbf{t} \quad (\text{App}) \end{array}}{j : \mathbf{e} \vdash \lambda X^{\mathbf{e} \rightarrow \mathbf{t}} X(j) : (\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t} \quad (\text{Abs})}$$

¹ in abuse of notation with type environments; this is OK because type environments will only appear on the left of the typing judgment turnstile \vdash , and type judgment pairs on the left.

4 The Inadequacy of Simple Types

In spite of the power and elegance of the simple type system sketched above, English logical elements exhibit polymorphic behaviors, such as that in Section 2, that the system cannot accommodate.

The generalized argument denoted by the QNP “*every person in this room*” in Ex 1 carries the type $(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}$, however the generalized (event) time denoted by the QNP “*every time I eat fried tofu*” in Ex 2 does not. The corresponding underspecified clause, Ex 5 isn’t missing an entity argument, that is, Ex 5 is not itself of type $\mathbf{e} \rightarrow \mathbf{t}$ as Ex 4 is. Rather it is missing a temporal argument—a term denoting the event-time that is normally carried by tense. Without extending the above system to the fully intensional type system of Montague’s PTQ (with \mathbf{s}), include the additional type \mathbf{i} for objects denoting times. Thus, the underspecified clause Ex 5 carries the type $\mathbf{i} \rightarrow \mathbf{t}$ and the generalized time “*every time I eat fried tofu*” the type $(\mathbf{i} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}$.

Neither does the generalized manner denoted by the QNP “*every way he can*” in Ex 3 carry the entity-oriented type $(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}$ exactly. Its corresponding clause and arguments are (approximately) fully specified clauses of type \mathbf{t} , hence the QNP is (approximately) of type $\mathbf{t} \rightarrow \mathbf{t} \rightarrow \mathbf{t}$. An alternative analysis is that the manner adverbial in fact applies to VPs, returning a VP, i.e. of type $\mathbf{t} \rightarrow (\mathbf{e} \rightarrow \mathbf{t}) \rightarrow (\mathbf{e} \rightarrow \mathbf{t})$. I leave manner adverbials unanalyzed for now. Nevertheless, the basic point remains that they do not carry the same type as generalized arguments.

So, even leaving out generalized manners, the picture that is beginning to emerge for the lexical item “*every*” is this:

Ex 13 (Varieties of “*every*”).

| | Environment Type | Denotation |
|---|---|---|
| a | argument $(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow (\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}$ | $\lambda X^{\mathbf{e} \rightarrow \mathbf{t}} \lambda Y^{\mathbf{e} \rightarrow \mathbf{t}} \ X\ \subseteq \ Y\ $ |
| b | tense/time $(\mathbf{i} \rightarrow \mathbf{t}) \rightarrow (\mathbf{i} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}$ | $\lambda X^{\mathbf{i} \rightarrow \mathbf{t}} \lambda Y^{\mathbf{i} \rightarrow \mathbf{t}} \ X\ \subseteq \ Y\ $ |

$\|\cdot\|$ is interpreted as a function from terms to their extensions². As there is an isomorphism between the power-set $\wp(A)$ and the set $\{0,1\}^A$, and as we normally take \mathbf{t} to be isomorphic to $\{0,1\}$, it is common to stipulate that objects of type $A \rightarrow \mathbf{t}$ denote subsets of $\|A\|$. It is important to point out here that this stipulation underlies a convention of the term language, and is not tied to the interpretation of the type system.

Ex 13a and Ex 13b then are totally comprehensible. Let $\|\text{every}\|_{\mathbf{e}}$ be the denotation $\lambda X^{\mathbf{e} \rightarrow \mathbf{t}} \lambda Y^{\mathbf{e} \rightarrow \mathbf{t}} \|X\| \subseteq \|Y\|$ of Ex 13a and $\|\text{every}\|_{\mathbf{i}}$ the denotation $\lambda X^{\mathbf{i} \rightarrow \mathbf{t}} \lambda Y^{\mathbf{i} \rightarrow \mathbf{t}} \|X\| \subseteq \|Y\|$ of Ex 13b. $\|\text{every}\|_{\mathbf{e}}$ takes as arguments two entity-set-denoting terms (i.e. terms of type $\mathbf{e} \rightarrow \mathbf{t}$) A, B and asserts that the extension of A is a subset of the extension of B : this is the standard story for the denotation

² This is abuse of notation with Def 2, although the symbol’s basic sense, that of an interpretation function, is preserved. Here, it applies over terms, in Def 2 it applies over types.

of “every” in the study of generalized quantifiers. Likewise with $\|\text{every}\|_i$, where the sets related are sets of instances or events.

The thesis of this paper is that a stronger type theory is needed to articulate quantificational types in a way that “every” can be effectively assigned a type. This thesis has two immediate critiques to respond to.

Stronger types are unnecessary because the variants of “every” can be articulated by defining variant terms, what the glosses $\|\text{every}\|_e$ and $\|\text{every}\|_i$ seem to suggest.

Stronger types are unnecessary because “every” can quantify over the universal domain \top without resorting to quantificational types.

The former critique invokes a strategy that is what Pustejovsky [8] describes as *sense enumeration*, here applied to logical elements. And, this strategy fails for many of the reasons Pustejovsky raises in criticism of it. Firstly, sense enumeration does not account for the polyadicity demonstrated in Ex 13 wherein “every” has structurally the same denotation only with different type-labels. And secondly, “every” can operate over a relatively wide array of types. “and” can as well. A sense enumeration lexicon, then, must contain a very large number of “every”s and “and”s to accommodate a relatively small corpus. Only a unified analysis of logical terms can address these concerns such that, basically, $\|\text{every}\|_e$ and $\|\text{every}\|_i$ not be separate terms.

The second critique is more subtle. It suggests that a single type judgment can accommodate “every” and “and”, namely

Ex 14 (Analysis with \top).

$$\begin{aligned} \|\text{every}\| &:= \lambda X^{\top \rightarrow \mathbf{t}} \lambda Y^{\top \rightarrow \mathbf{t}} \|X\| \subseteq \|Y\| \\ \|\text{and}\| &:= \lambda X^{\top} \lambda Y^{\top} \|X\| \sqcap \|Y\| \end{aligned}$$

These analyses are inadequate because they do not express a relationship between the two types in each expression. In particular, they fail to express type-identity, i.e.

Ex 15 (Type identity constraints).

$$\begin{aligned} \text{every}: (A \rightarrow \mathbf{t}) \rightarrow (A \rightarrow \mathbf{t}) \rightarrow \mathbf{t} \\ \text{and}: A \rightarrow A \rightarrow A \end{aligned}$$

This could lead to passing in terms of different types into the expressions’ argument positions, such as **e** and **i** in for example

Ex 16. * *Every person in this room order fried tofu.*

Ex 17. * *Every person and ordered fried tofu.*

Expressing the identity constraint between types in these expressions entails quantifying, either explicitly or implicitly, over the type expressions. And this is precisely the strategy introduced in the following section.

5 A Calculus for Polymorphism

System F or the polymorphic lambda calculus was discovered independently by Girard [2] and Reynolds [9]. It adds to the system outlined in Section 3 a new universally quantified type schema and a new term schema abstracting over types.

5.1 Syntax

We now distinguish formally between term variables $\text{TermVar} = \{x, y, z, \dots\}$ and type variables $\text{TypeVar} = \{X, Y, Z\}$. (Previous to this our type variables have been meta-variables.) The type system and term language are given by

Def 7 (System F Type Syntax).

$$\text{Type} ::= \mathbf{e} \mid \mathbf{t} \mid \mathbf{i} \mid \text{Type} \rightarrow \text{Type} \mid \forall \text{TypeVar} . \text{Type}$$

Def 8 (System F Term Syntax).

$$\begin{aligned} \text{Term} ::= & \\ & \text{TermVar} \mid \text{Const} \mid \lambda \text{TermVar}^{\text{Type}} \text{Term} \mid \text{Term}(\text{Term}) \mid \\ & \Lambda \text{TypeVar} \text{Term} \mid \text{Term}(\text{Type}) \end{aligned}$$

β and η rules (Def 4) are still valid but now permit (valid) substitutions of types. Finally, we extend the typing judgments to accommodate reasoning about quantifiable types:

Def 9 (System F Type Judgments).

$$\frac{\mathcal{T}(x) = A}{\mathcal{T} \vdash x : A} \text{Ax} \quad \frac{\mathcal{T}, x : A \vdash f : B}{\mathcal{T} \vdash \lambda x^A . f : A \rightarrow B} \text{Abs}$$

$$\frac{\mathcal{T} \vdash f : A \rightarrow B \quad \mathcal{T} \vdash x : A}{\mathcal{T} \vdash f(x) : B} \text{App}$$

$$\frac{\mathcal{T} \vdash p : \forall X . A}{\mathcal{T} \vdash p(B) : A[B/X]} \text{Inst} \quad \frac{\mathcal{T} \vdash p : A}{\mathcal{T} \vdash \Lambda X . p : \forall X . A} \text{Gen}$$

System F is often called the second-order lambda calculus, because under Curry-Howard it corresponds to the second order (positive implicative) intuitionistic calculus, wherein formulae can quantify over propositions.

This calculus can now provide a concise denotation for “every”.

Ex 18 (every).

Let $\text{ev} =_{\text{def}} \Lambda X . \lambda p^{X \rightarrow \mathbf{t}} . \lambda q^{X \rightarrow \mathbf{t}} . \|p\| \subseteq \|q\|$ then $\|\text{every}\|_{\mathbf{e}}$ and $\|\text{every}\|_{\mathbf{i}}$ from Section 4 fall right out, as $\|\text{every}\|_{\mathbf{e}} =_{\text{def}} \text{ev}(\mathbf{e})$ and $\|\text{every}\|_{\mathbf{i}} =_{\text{def}} \text{ev}(\mathbf{i})$.

5.2 Semantics

The naive set-theoretic semantics sketched in Def 2 cannot apply to System F. As mentioned, the polymorphic lambda calculus corresponds to the second order propositional calculus, wherein formulae quantify over other formulae. As types are mapped to sets under the naive semantics, then the domain of quantification for the higher order types would be the class of all sets. For example,

Ex 19.

$$\vdash \lambda X. X \notin X : X \rightarrow \mathbf{t}$$

would be the characteristic function on the Russell set.

It is worth pointing out that this argument is only valid for classical set-theory, and that constructive formulations of set theory can serve as models of the polymorphic lambda calculus. In Ex 19, I'm glossing over the admissibility of combining type theoretic notation with set-theoretic notation, which would be illegitimate constructively.

As of Huet [4], at least, characterizing models of the polymorphic lambda calculus is an active area of research in theoretical computer science. According to Reynolds [10] (same volume), there are several classes of non-trivial models for the polymorphic lambda calculus. There are two known classes of such models. In one, types are mapped (approximately) to partial equivalence relations on a model of the untyped lambda calculus. In the second, the meaning of a type is a Scott domain. (The symmetry here should not be surprising as Scott domains constitute models for the untyped lambda calculus.) Additionally, Girard [3] presents a model for this calculus based on qualitative domains with stable functions between them, that informally are like finitary Scott domains.

5.3 Type-theoretic semantics and truth-conditions

While a consistent semantic account for the validity of typing judgments and the truth-conditions would be nice, is it necessary? There is a sense in which types and terms constitute the structure of logical forms, not the truth-theoretic content per se. And, indeed, until this point, the truth conditions of the one generalized quantifier I've discussed, "every", have been all but absent, as well as most of the major significant results of GQs, such as monotonicity and persistence (although, there is a wealth of research yet to be done, I believe, on the truth-conditions of generalized quantifiers over temporal expressions).

Nevertheless, generalized quantifier theory is typically presented with classical set-theory as an underpinning and it is an open question whether the problem of coherently typing GQs like "every" precludes such an underpinning.

The lynch-pin between the type theory and the set-theoretic truth conditions is that expressions of type $A \rightarrow \mathbf{t}$ are typically interpreted as *sets*, that is, *subsets of A*. This is not a consequence of the type-theory but a convention of the term language. That is to say, this is a specification in the interpretation of the term language into a model, rather than the interpretation of the type system. Thus,

we can with certain equanimity drop the declaration that the type expression $A \rightarrow B$ denotes a set (of functions from $\|A\|$ to $\|B\|$) without dropping the predicates/characteristic functions/subsets convention familiar from GQ theory.

6 Directions of further research

Discussing linguistic type theory at the second-order or polymorphic level opens the door to a wide array of further research. Following the proposals in this paper, the two immediate needs are to provide a story for the nature of terms of arrow types, such as $\lambda x^{e \rightarrow t} P(x)$ and those of higher-order arrow-types such as $\lambda X \lambda x^{e \rightarrow t} X(x)$.

An example of what that might look like is as follows. [7] sketch a simple and appealing interpretation for the generalized conjunction for “and” (\sqcap), roughly:

Ex 20 (and (\sqcap)).

$$X \sqcap Y \mapsto \begin{cases} X \wedge Y & \text{if } X, Y : \mathbf{t} \\ \lambda Z^{e \rightarrow t} Z(X) \wedge Z(Y) & \text{if } X, Y : \mathbf{e} \\ \lambda x^A X(x) \sqcap Y(x) & \text{if } X, Y : A \rightarrow B \end{cases}$$

This is in the term language, though it admits the recursive definition akin to the type-system. Proving that this interpretation is valid relative to a type (in particular, the type $\forall X. X \rightarrow X \rightarrow X$) interpreted by, e.g., finite approximations of functions could be shown by demonstrating that the above function has a (finitely approximate-able) fixed-point.

Secondly, there is much descriptive work to be done on how generalized quantifiers behave when relating properties of type $\mathbf{i} \rightarrow \mathbf{t}$, to say nothing of the manner adverbials. While the general denotation of such quantifiers will remain very much the same, as sketched above in (13), certain new properties will emerge under different types: grammatical distribution, as above in Section 4 but also model-theoretic properties. For example, contrast the following clauses:

Ex 21. *Every time I eat Thai food, I order fried tofu.*

Ex 22. *Every time I ate Thai food, I ordered fried tofu.*

Ex 23. *One time I ate Thai food, I ordered fried tofu.*

Ex 24. ** One time I eat Thai food, I order fried tofu.*

Ex 25. *One time I'll eat Thai food and order fried tofu.*

The apparent constraint against the simple existential GQ in the present tense is interesting, as is the conjoined construction necessary to get the existential to go through in a future-tensed clause. There are plenty of similar examples with the other GQs similar. My hunch as to Ex 24 is that the existential GQ requires the existence of such an object in the model and that, metaphysically, most events and time-indices occur in the past or the future. It is intriguing, though, that it still doesn't seem to work in the historical present.

References

1. Jon Barwise and Robin Cooper. Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4, 1981.
2. Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupes dans l'arithmétique d'ordre supérieure*. Thèse de doctorat d'état, Université de Paris VII, 1972.
3. Jean-Yves Girard. The System F of variable types, fifteen years later. In Huet [4], pages 87–126.
4. Gérard Huet, editor. *Logical Foundations of Functional Programming* The UT Year of Programming series, Reading, Mass., Addison-Wesley, 1990.
5. Richard Montague. The proper treatment of quantification in ordinary English. In J. Hintikka, J. Moravcsik and P. Suppes, editors, *Approaches to Natural Language*, Dordrecht, D. Reidel, 1973.
6. Barbara H. Partee. Some structural analogies between tenses and pronouns in English. *The Journal of Philosophy*, 70:601–609, 1973.
7. Barbara H. Partee and Matts Rooth. Generalized conjunction and type ambiguity. In Rainer Bäuerle, Christoph Schwarze and Arnim von Stechow, editors, *Meaning, Use and the Interpretation of Language*, pages 361–93. Walter de Gruyter, 1983.
8. James Pustejovsky. *The Generative Lexicon*. Cambridge, Mass., MIT, 1995.
9. John C. Reynolds. Toward a theory of type structure. In J. E. Fenstad, editor, *Proceedings, Colloque sur la programmation*, Volume 19 of *Lecture Notes in Computer Science*, pages 408–425, Berlin, Springer-Verlag, 1974.
10. John C. Reynolds. Introduction to Part II. In Huet [4], pages 77–86.