Lexical Ambiguity as Type Disjunction

Nicholas Asher
Philosophy Department
The University of Texas at Austin
1, University Station C3500
Austin, TX 78712
USA
nasher@mail.utexas.edu

Pascal Denis
Linguistics Department
The University of Texas at Austin
1, University Station B5100
Austin, TX 78712
USA
denis@mail.utexas.edu

Abstract

Phenomena such as logical polysemy and logical metonymy have recently received a precise treatment by using a rich composition logic that assumes complex types and rich introduction and exploitation rules for manipulating them (Asher and Pustejovsky, 2005). An interesting question is whether this approach can be extended and applied to other lexical semantics phenomena, in particular to *contrastive ambiguity* (aka *homonymy*). This paper enriches Asher and Pustejovsky's Type Composition Logic with another type of complex types, namely *disjunctive types*, and accompanying exploitation and introduction rules to model homonymy. This results both in a more elegant and economical treatment of homonymy and in a more general account of lexical ambiguity.

1 Introduction

It has long been noted that the problem of polysemy is not a uniform one. In particular, a distinction is often drawn between *homonymy* (or contrastive polysemy) and *logical polysemy* (or complementary polysemy). Informally, the former designates words that have two (or more) unrelated senses; the fact that these different words share the same form is purely accidental. When logically polysemous expressions are concerned, however, the two (or more) word senses are closely related: they express different "facets" of the same meaning. Common examples of homonymy include words like *bank* or *pitcher*. Thus, a word like *bank* for instance denotes either a financial institution or a strip of land along a river. Examples of logical polysemy are *book* or *lunch*. Thus, a word like *book* either denotes a physical object or *its* informational content.

How do we tell homonymy and logical polysemy apart? Among the different tests have been proposed, few are really conclusive. A pretty reliable test is so-called *copredication* (Pustejovsky, 1995). Copredication occurs when the same term has simultaneous predications selecting for different semantic types. Homonyms are usually semantically infelicitous in such contexts, whereas polysemous words tend to be acceptable. This is illustrated in the following examples:

(1) a. # The bank specializes in IPO and is being quickly eroded by the river.
    b. # The pitcher was being filled up with Hoegarden and hit a home run.

(2) a. The book was 800 pages long but turned out to be very interesting.
    b. Lunch was delicious but took for ever.

The contrast observed in these examples obviously calls for different treatments of (logical) polysemy and homonymy. Following (Pustejovsky, 1995) and (Asher and Pustejovsky, 2005), one can account for examples (2) by assuming that polysemous nouns like *book* receive a *dot-type*, i.e. a complex type with two (or more) constituent types. These constituent types correspond to different aspects of the object and thus license predications over either of the two dot element types.

While a lot of work has been devoted to the treatment of polysemy (e.g., in the context of the Generative Lexicon), the representation of homonymy has received comparatively little attention. The received view is the sense-enumeration view (Pustejovsky, 1995) whereby one posits different word senses. These different senses are often taken to correspond to different lexical entries, and at least to different semantic representations. So, a word like *bank* for instance would contribute different bits of logical form, depending on its selected sense (e.g., $\lambda x bank\_1(x)$, …, $\lambda x bank\_n(x)$). The question then comes to the problem of selecting an appropriate sense for a given context of predication (which we understand generally as the application of some predicate to some argument). A type free system would have to test the consistency of the lexical meanings involved in the predication, and if such meanings are associated with a large body of background information the consistency test could be in principle undecidable. A lexicon with a semi-lattice of simple types on the other hand, would avoid onerous consistency tests by precompiling these into the type system itself, but there is still the matter of checking different lexical entries for truly ambiguous expressions. Our view pushes the type driven approach one step further. We will move the ambiguity to the level of the semantic types (so to speak), and consider that homonymous words introduce another type of complex types: *disjunctive types*.

2 Many potential complex types

In general, the approach provided to lexical information by Asher and Pustejovsky supposes that composition, the process of putting together representations of lexical meanings to arrive at a logical form, is very strongly type-driven and that a great deal of crucial lexical information is stored in the type system for the lexicon. In particular, it is type mismatch that drives processes of logical metonymy, which is represented via a calculus of type shifting rules, each one corresponding to something like a proof rule. This approach takes its inspiration from the famous Curry-Howard isomorphism (Howard, 1980) between types and formulas, and proofs and operations on lambda terms. The simple application of a predicate to an argument is here a sort of type *exploitation* rule that corresponds to *modus ponens*: we take the predicate whose type is the functional type $\alpha \multimap \beta$[1] and apply it to its argument of type α to get a result of type β.

Of course, understanding application in this way is altogether familiar from the work of Church and Montague. But a Curry-Howard inspired perspective leads us to ask some interesting questions. We expect that in a natural deduction style proof system for intuitionistic logic (which is what the Curry Howard isomorphism is based on), each logical operator should have both introduction and elimination or exploitation rules. The lexical use of types and proofs hasn't led to a system so far where each complex type forming operator which corresponds in the Curry Howard way of thinking to a connective or quantifier has both an introduction and an elimination rule; for instance, though the rule of application corresponds as we have just seen to a conditional elimination rule or MP, what would constitute an introduction rule for $\multimap$? One thing that comes to mind here is the lexical process, according to which names of individuals become predicates or verbs as in:

[1]We follow (Asher and Pustejovsky, 2005) in using $\multimap$ as our type forming operator to keep it distinct from the object language conditional.

(3) a. He's no Einstein.
b. Can we pegasize that name away?
c. Every Lex Luthor has his comeuppance.

The type shifting operation of (Partee and Rooth, 1983) that lifts expressions of type e to type $(e \multimap t) \multimap t$ can also be modeled as a (two step) $\multimap$ introduction rule. From our perspective such type shifting should be expected and needed —e.g., for generalized conjunctions like *Pat and three bears*.

As soon as lexical theory becomes more complex—e.g., it admits the sort of rich representations that are at the core of GL, then the matter of constructing precise proof rules corresponding to operations on types becomes pressing. The use in GL of complex operations has largely been underconstrained and under-specified formally. Exploiting the correspondence to construct precise proof rules constitutes one way to get a precise idea concerning the nature of the lexical operations that GL postulates. It will be clear, for instance, that the introduction and elimination rules for types corresponding to dot objects ($\bullet$) and qualia ($\otimes$) no longer correspond to anything in standard intuitionistic logic. Nevertheless, once these rules are formulated we can ask questions about the logical nature of the system, and we can determine complexity results for the logic. We'll review a couple of these rules below.

The question that motivates this paper is, how does homonymy fit into the type calculus? Can we extend the lexical information as types scheme to take account of other phenomena in lexical semantics?

3 The type composition logic (TCL)

Before we answer these questions let us have a closer look at the composition logic proposed in (Asher and Pustejovsky, 2005). We start with the type definitions:

(4) a. PRIMITIVE TYPES: e the general type of entities and $\underline{t}$ the type of truth values. Below σ, τ range over all simple types, the subtypes of e as well as e and $\underline{t}$.
b. FUNCTIONAL TYPES: If σ and τ are types, then so is $(\sigma \multimap \tau)$
c. DOT TYPES: If σ and τ are types, then so is $(\sigma \bullet \tau)$
d. PRODUCT TYPES: If σ and $\tau_1, \ldots \tau_n$ are types, then so is $(\sigma \otimes_{R_1,\ldots R_n} (\tau_1 \ldots \tau_n))$ (where R_i is a relation over (σ, τ_i)).

Such a typed composition logic has several motivations. First of all, a typed composition logic seems needed just to account for basic facts of predication. For instance, a semantic type mismatch is a standard way to explain semantic anomalies like:

(5) # The beer carried the table.

(6) # The quick house

Of course, some type mismatches can become good when we avail ourselves of metaphorical interpretations. Some authors (Asher and Lascarides, 2001) have argued that it's precisely the type mismatch in predication that triggers the seach for a metaphorical interpretation. Second, complex types appear to be at work in a wide array of phenomena. Thus, GL's account of copredication relies on the existence of •- or "dot" types. As for dependent or ⊗-types, these play an important role in (Pustejovsky, 2001) definition of artifacts as well as in (Asher and Denis, 2004) analysis of the genitive construction.

We now turn to the basic operations on these types.

(7) **Application:**

$$\frac{\lambda x\phi[t],\quad c(x\colon\alpha, t\colon\alpha)}{\phi[t/x], c}$$

The *Application* rule is the classical function application augmented with *type contexts*; it corresponds in terms of the type calculus itself to a rule of modus ponens for $\multimap$.

This first rule is accompanied by another rule that allows contexts to be combined and updated:

(8) **Merging Contexts:**

$$\frac{\lambda x\phi, c[t, c']}{\lambda x\phi[t], (c^\wedge c')}$$

The last rule, *Type Accommodation*, complements the *Application* rule by allowing type unification rather than simple type matching:

(9) **Type Accommodation:**

$$\frac{\lambda x\phi[t], c(x\colon\alpha, t\colon\beta),\ \ \alpha\sqcap\beta\neq\bot}{\lambda x\phi[t], c*(x, t\colon\alpha\sqcap\beta)}$$

One thing not provided by Asher and Pustejovsky is a notion of greatest lower bound that generalizes to the complex types. To remedy this defect, let us define:

- $\alpha\sqcap^*\beta = \alpha\sqcap\beta$, where α,β are simple types.
- $(\alpha\multimap\beta)\sqcap^*(\gamma\multimap\delta) = (\alpha\sqcap^*\gamma)\multimap(\beta\sqcap^*\delta)$
- $(\alpha\bullet\beta)\sqcap^*(\gamma\bullet\delta) = (\alpha\sqcap^*\gamma)\bullet(\beta\sqcap^*\delta)$
- $(\alpha\otimes(\beta_1,\ldots\beta_n)\sqcap^*(\gamma\otimes(\delta_1\ldots\delta_n) = (\alpha\sqcap^*\gamma)\otimes((\beta_1\sqcap^*\delta_1,\ldots\beta_n\sqcap^*\delta_n)$

From now on, we take $\sqcap$ in our rules to reflect this generalized greatest lower bound.

Finally, here are the rules for •-Introduction and •-Exploitation. We provide one version of •-Exploitation which we express with a pair of substitutions. They look like this $\chi\{\frac{\phi}{\psi}\}$, where ϕ uniformly replaces every occurrence of ψ in χ. One other bit of notation has to do with the square brackets; they represent an application that hasn't yet taken place. That is with $P[x]$ we haven't yet applied the property that P stands for to x; similarly the lambda expression with its typing context, $[\psi, c']$ hasn't yet been integrated with the lambda expression with its context on its left. We enclose the complex expression that is to apply to $[\psi, c']$ in curly brackets to help for readability. Below $\phi(P[x])$ represents the fact that the property variable P is to apply to x in the expression ϕ, and $\psi\colon\left[\begin{smallmatrix}\alpha'\\ \beta'\end{smallmatrix}\right]\multimap\gamma$ represents the fact that ψ is typed either as $\alpha\multimap\gamma$ or as $\beta\multimap\gamma$. In this rule and the following rules concerning •-types, we will assume that $\alpha\sqcap\alpha'\neq\bot, \beta\sqcap\beta'\neq\bot$.

Here is the first exploitation rule:

(10) **•-Exploitation (•E):**

$$\frac{\{\lambda P\phi(P(x)),\ c(P\colon(\alpha\bullet\beta)\multimap\gamma)\}[\psi,\ c'(\psi\colon[\begin{smallmatrix}\alpha'\\ \beta'\end{smallmatrix}]\multimap\gamma)], \mathrm{head}(\psi)}{\{\lambda P\phi[\frac{\exists v(\Delta(\phi,x)[\frac{v}{x}]\wedge \text{O-Elab}(x,v))}{\Delta(\phi,x)}],\ c*(x\colon[\begin{smallmatrix}\alpha\sqcap\alpha'\\ \beta\sqcap\beta'\end{smallmatrix}], v\colon\alpha\bullet\beta)\}[\psi, c']}$$

This rule does two things; it adds material to the logical form of the lambda term to which it applies and it also revises the type contexts to reflect a shift in the typing of some of the variables in the altered lambda term. If we look just at what happens to the type for x, •-Exploitation corresponds to something like a conjunction elimination rule for •-types, but it is more complicated than that since it forces us in reintroduce a variable of •-type. It is in fact an *ampliative* rule.

Here is the basic •-Introduction rule. Once again, we assume $\alpha\sqcap\alpha'\neq\bot$ and $\beta\sqcap\beta'\neq\bot$.

(11) **•-Introduction (•I):**

$$\frac{\{\lambda P\phi(P[x]), c(P\colon[\begin{smallmatrix}\alpha'\\ \beta'\end{smallmatrix}]\multimap\gamma)\}[\psi,\ c'(\psi\colon(\alpha\bullet\beta)\multimap\gamma)], \mathrm{head}(\psi)}{\{\lambda P\phi[\frac{\exists v(\Delta(\phi,x)[\frac{v}{x}]\wedge \text{O-Elab}(v,x))}{\Delta(\phi,x)}], c*(v\colon[\begin{smallmatrix}\alpha\sqcap\alpha'\\ \beta\sqcap\beta'\end{smallmatrix}], x:\alpha\bullet\beta)\}\ [\psi,\ c']}$$

Note that in Asher and Pustejovsky's system, the different Exploitation and Introduction rules are constrained by an additional principle, the so-called *Head Principle*:

(12) **Head Typing Principle:**

> Given a compositional environment X with constituents A and B, and type assignments $A{:}\,\alpha$ and $B{:}\,\beta$ in the type contexts for A and B respectively that clash, if A is the syntactic head in the environment, then the typing of A must be preserved in any composition rule for A and B to produce a type for X.

This principle make sure that heads impose their type over that of a modifier and are also the only ones to be able to introduce complex types. As we will see, it seems that we actually want to have the Head Typing Principle apply to type introduction rules (e.g., +-Introduction) but not the elimination rules. One possible motivation behind this is that in composition, we are aiming to reduce ambiguity not increase it.

4 Homonomy as Type Disjunction

We propose to model homonymy as type disjunction. This involves as a first step augmenting our repertoire of types with a new brand of complex type, namely *disjunctive types* or +-types:

(13) DISJUNCTIVE TYPES: If σ and τ are types, then so is $(\sigma + \tau)$

We suppose that some orthographic entries in the lexicon like *bank* will have multiple senses. Even though a GL style approach seeks to minimize such multiple sense entries in the lexicon and we are wholly in favor of that, it seems silly to think that these can be eliminated altogether. However, we can represent homonymous entries in a compact form using the new sort of type constructor +.

The main objective from our perspective of constructing a logical form for a clause (or a discourse) is to reduce and manage ambiguities introduced by various lexical and compositional processes in as efficient a manner as possible. We can exploit the correspondence between proofs and types to reduce homonymic ambiguity in the lexicon by means of an exploitation rule for +-types that corresponds to one form of disjunction exploitation, that is:

(14)
$$\frac{\phi \vee \psi, \quad \neg\psi}{\phi}$$

Below is the +-Exploitation rule:

(15) **+-Exploitation:**

$$\frac{\{\lambda P\phi(P(x)), c(P : (\alpha + \beta) \multimap \gamma)\}[\psi, c'(\psi : [\begin{smallmatrix}\alpha' \\ \beta'\end{smallmatrix}] \multimap \gamma)]; \alpha \sqcap \beta = \bot}{\{\lambda P\phi(P(x)), c * (P : [\begin{smallmatrix}\alpha \sqcap \alpha' \\ \beta \sqcap \beta'\end{smallmatrix}] \multimap \gamma)\}[\psi, c']}$$

The reason why we can't exactly mimic the proof rule familiar from the propositional calculus is that in our type system we don't have anything directly corresponding to negation, other than the idea that two types are incompatible in the sense of $\alpha \sqcap \beta = \bot$.

Note that this rule is simpler than •-Exploitation. +-Exploitation is basically *destructive*: the complex disjunctive type can be exploited once, which corresponds to the intuition that ambiguity gets filtered out as a result of semantic composition.

Let us move to a concrete application of the above rule. Consider the following simple sentence:

(16) The bank is eroding.

We start by assuming the following semantic representations for the NP *the bank* and the VP *is eroding*, respectively:[2]

(17) a. $[\![\text{the bank}]\!] = \lambda P\exists! x(bank(x) \wedge P(x))\langle x{:}\,\textsc{fin\_inst} + \textsc{shore}, P{:}\,(\textsc{fin\_inst} + \textsc{shore}) \multimap t\rangle$

b. $[\![\text{is eroding}]\!] = \lambda y(erode(y))\langle y{:}\,\textsc{shore}\rangle$

Now we simply apply the different TCL rules to derive the correct logical form for (16):

a. $[\![\text{the bank is eroding}]\!] = \lambda P\exists! x(bank(x) \wedge P(x))\langle x{:}\,\textsc{fin\_inst} + \textsc{shore}, P{:}\,(\textsc{fin\_inst} + \textsc{shore}) \multimap t\rangle[\lambda y(erode(y))\langle y{:}\,\textsc{shore}\rangle]$

b. By *+-Exploitation*: $\lambda P\exists! x(bank(x) \wedge P(x))\langle x{:}\,\textsc{shore}, P{:}\,\textsc{shore} \multimap t\rangle[\lambda y(erode(y))\langle y{:}\,\textsc{shore}\rangle]$

c. By *Merging*: $\lambda P\exists! x(bank(x) \wedge P(x))[\lambda y(erode(y))]\langle x{:}\,\textsc{shore}, P{:}\,\textsc{shore} \multimap t, y{:}\,\textsc{shore}\rangle$

d. By *Application*: $\exists! x(bank(x) \wedge erode(x))\langle x{:}\,\textsc{shore}\rangle$

Note that the ambiguity of *bank* doesn't "survive" the semantic composition. This is turn explains why copredication is impossible in these cases. For instance, one cannot continue sentence (16) with the following sentence, since the VP here selects for FIN_INST type which is no longer available for *bank*:

(18) It was offering new interest rates.

As we mentioned above, the *Head Principle* doesn't seem to apply during the exploitation of a disjunctive type. Evidence for this is that copredication fails in in an example like

(19) The eroding bank was offering new interest rates.

In (19), the modifier *eroding* like the corresponding verb in (16), filters out the FIN_INST reading, which is selected by the the VP *was offering new interest rates*.

[2]Time and aspect information are left out for simplicity.

4.1 A slightly more complex example

Let us move to a slightly more complex example and to another language. In Spanish, the adjective *listo* is ambiguous between a individual-level meaning where it means *clever* and a stage-level meaning where it means *ready*. As is often the case in Spanish, this difference is correlates with a difference in terms of auxiliary selection: the individual-level meaning requires the use of copular *ser* while the stage-level meaning requires the use of *estar*. This is shown in the following examples:

(20) a. Joan és listo. (Joan is clever/#ready)
b. Joan está listo. (Joan is #clever/ready)

We propose to account for this contrast as follows. First, we posit the following lexical entry for *listo*, wherein the types ST and IND stand for *stage* and *individual* (cf. (Carlson, 1977)), respectively:

(21) $[\![\text{listo}]\!] = \lambda x\, listo(x)\langle x\colon \textsc{st} + \textsc{ind}\rangle$

We could also assume a couple of meaning postulates to transform the disambiguated *listo* into a more transparent logical form:

(i) $\lambda x\, listo(x)\langle x\colon \textsc{st}\rangle \mapsto \lambda x\, ready(x)$
(ii) $\lambda x\, listo(x)\langle x\colon \textsc{ind}\rangle \mapsto \lambda x\, intelligent(x)$

The semantic representations for *ser* and *estar* are as follows:

(22) a. $[\![\text{ser}]\!] = \lambda P \lambda x P(x)\langle x\colon \textsc{ind}, P\colon \textsc{ind} \multimap t\rangle$
b. $[\![\text{estar}]\!] = \lambda P \lambda x P(x)\langle x\colon \textsc{st}, P\colon \textsc{st} \multimap t\rangle$

That is, the verb *ser* is a function from stage properties to stage properties, whereas the verb *estar* is a function from individual properties to individual properties.

As it stands, the +-Exploitation rule doesn't handle these examples. That is, we need a second, symmetrical version of that rule:

(23) **+-Exploitation (bis):**

$$\frac{\{\lambda P\phi(P(x)), c(P : [\,{}^{\alpha'}_{\beta'}\,] \multimap \gamma))\}[\psi, c'(\psi : (\alpha + \beta) \multimap \gamma]; \alpha \sqcap \beta = \bot}{\{\lambda P\phi(P(x)), c\}[\psi, c' * (\psi : [\,{}^{\alpha \sqcap \alpha'}_{\beta \sqcap \beta'}\,] \multimap \gamma)]}$$

A comparison with English *be* is instructive. Given that copular *be* in English can take both stage-level and individual-level predicates, we assume that *be* has the following logical representation:

(24) $[\![\text{be}]\!] = \lambda P \lambda x P(x)\langle x\colon \textsc{st} + \textsc{ind}, P\colon (\textsc{st} + \textsc{ind}) \multimap t\rangle$

Note once again that we locate the ambiguity at the level of types by assuming a complex disjunctive type, hence providing a more economical management of homonymy. In English, it will be the complement of *be* that disambiguates the verb's function.

An interesting issue that crops up in connection with these types concerns the conjunction of stage-level and individual-level predicates. Thus, assuming as usual that conjunction like *and* require type identity of the conjoined constituents, then one predicts that conjoining a stage-level predicate and an individual-level predicate results is unfelicitous. This prediction seems to be, to a large extent, born out:

(25) # Jim is sick and six feet tall.[3]

Also, note that if we take stages and individuals to be of incompatible types, then we also predict that a NP like *a sick banker* cannot be interpreted in the standard generalized conjunction way (as in (Heim and Kratzer, 1998) for instance).

We are committed to stage-level and individual level (and kinds) types in the type hierarchy. But how do these relate to the extant type lattice? Should we assume a multi-dimensional lattice? In a way the GL framework invites a multi-dimensional lattice with the postulation of complex types. But we leave the details for future research.

4.2 Disambiguation through complementation

In the nominal and adjectival cases we looked so far, disambiguation is done through predication. Now, moving to the verbal predicates, there are also examples where ambiguity reduction arises through complementation (i.e., where it is the direct object that disambiguate the predicate).[4] This is shown in the following example:

(26) a. George shot his gun.
b. George shot a rabbit.

The direct object is the *instrument* of the shooting in (26a), but it is the *target* in (26b). It is worth noting that "co-complemenation" is impossible:

(27) # George shot his gun and a rabbit.

This suggests the use of a disjunctive type, rather than a dot type; more precisely, the type for *shoot* is:

This suggests the use of a disjunctive type, rather than a dot type; more precisely, the type for *shoot* is:

[3]The fact that proper nouns like *Jim* can be predicated of either stage-level or individual-level predicate suggests that its type is also complex (i.e., either ST + IND or ST • IND).

[4]We thank one of our anymous reviewers for this observation.

(28) *shoot*: $((\textsc{phys\_obj} + \textsc{gun}) \multimap \textsc{human}) \multimap t$

Neither of the two versions of +-Exploitation already introduced will apply directly. Rather, we will have to invoke a type-shifted version of this rule, which we dubb +-Exploitation^{TS} (on the model of •-Exploitation^{TS} in (Asher and Pustejovsky, 2005)):

(29) **+-Exploitation^{TS} :**

$$\frac{\{\lambda\mathcal{P}\phi, c(\mathcal{P} : ((\alpha+\beta) \multimap \gamma) \multimap \delta)\}[\lambda P\psi(P(x)), c'(P : [\begin{smallmatrix}\alpha' \\ \beta'\end{smallmatrix}] \multimap \gamma)]; \alpha \sqcap \beta = \perp}{\{\lambda\mathcal{P}, c(\mathcal{P} : ([\begin{smallmatrix}\alpha \sqcap \alpha' \\ \beta \sqcap \beta'\end{smallmatrix}] \multimap \gamma) \multimap \delta)\}[\psi,\ c']}$$

Now that we have this rule, the derivation for (26b) becomes straightforward:

a. First, we type-raise the type for *shoot* to: $\lambda\mathcal{P}\lambda x\mathcal{P}[\lambda y(shoot(x,y))]\langle x\colon \textsc{human}, y\colon \textsc{animate} + \textsc{gun}, \mathcal{P}\colon ((\textsc{animate} + \textsc{gun}) \multimap t) \multimap t\rangle$

b. $[\![$shoot a rabbit$]\!]$ = $\lambda\mathcal{P}\lambda x\mathcal{P}[\lambda y(shoot(x,y))]\langle x\colon \textsc{human}, y\colon \textsc{phys\_obj} + \textsc{gun}, \mathcal{P}\colon ((\textsc{phys\_obj} + \textsc{gun}) \multimap t) \multimap t\rangle(\lambda P\exists z(rabbit(z) \wedge P(z))\langle z\colon \textsc{animal}, P\colon \textsc{animal} \multimap t\rangle)$

c. By $+ - Exploitation^{TS}$: $\lambda\mathcal{P}\lambda x\mathcal{P}[\lambda y(shoot(x,y))]\langle x\colon \textsc{human}, y\colon \textsc{animal}, \mathcal{P}\colon (\textsc{animal} \multimap t) \multimap t\rangle(\lambda P\exists z(rabbit(z) \wedge P(z))$ $\langle z\colon \textsc{animal}, P\colon \textsc{animal} \multimap t\rangle)$

d. By *Merging* and *Application*: $\lambda x[\exists z(rabbit(z) \wedge shoot(x,z))]\langle x\colon \textsc{human}, z\colon \textsc{animal}\rangle$

4.3 Ambiguous derived nominals: + or •

The type of disambiguation through complementation that we discuss in the previous section is not limited to the verbal domain *stricto sensu*. There is indeed a class of derived nominals (e.g., *invention*, *repreduction*, *examination*, *destruction*) discussed in (Grimshaw, 1990) that show a similar, though slightly more complex, behavior. Typically, these nominals are said to be ambiguous between a *process* and *result* reading:

(30) a. The invention of the phonograph took place in the late 19th century. (PROCESS)

b. The invention was presented at the Paris Universal Exposition. (RESULT)

According to (Grimshaw, 1990), this ambiguity reflects different argument-taking properties of the nominal: the nominals that denote a process would be the nominals that have retained the argument structure of their verb, whereas the nominals that denote a result the nominals that haven't. This proposal seems to make some correct prediction; for instance, it predicts that *the invention of the phonograph* can only have a process reading:

(31) # The invention of the phonograph was presented at the Paris universal exposition.

But this proposal also faces serious problems. On the one hand, derived nominals can have a process reading without realizing their object (overtly). On the other hand, certain nominals have a result reading *even* when they realize their object. Both points are illustrated in (32a) and (32b), respectively:

(32) a. The examination lasted three hours.

b. A reproduction of the *Sunflowers* was sold for one million dollars.

Before moving to the analysis, it is also worth noting that some of these nominals allow for copredication:

(33) a. The reproduction (of the painting) took place in that workshop and is eight feet tall.

b. The translation that took place in this room is a classic.

But this is by no means a general phenomenon, and many examples are zeugmatic at best:

(34) *? The examination lasted three hours and is now ready for you to grade.

The way we propose to deal with these nominals is to assume that these nominals: (i) always preserve their (verbal) internal structure, and (ii) have a unique complex type. This type can be of either two forms, depending on whether the nominals allow for copredication:

(35) a. $(\textsc{created-obj} \multimap (\{\textsc{ag}, \textsc{ev}\} \multimap t)) + (\textsc{ev} \multimap (\{\textsc{ag}, \textsc{created-obj}\} \multimap t))$

b. $(\textsc{created-obj} \multimap (\{\textsc{ag}, \textsc{ev}\} \multimap t)) \bullet (\textsc{ev} \multimap (\{\textsc{ag}, \textsc{created-obj}\} \multimap t))$

The above types AG and EV stand for agent and eventuality, respectively; and they remain unordered in the lexical entry, thus making our ambiguity a disjunction between polymorphic types. The notation $\{\vec{\lambda x}\}$ is borrowed from (Asher, 1993) and allows for "unordered" function application. The CREATED-OBJ corresponds to the entity created by the verb; the term whose logical form it decorates is the direct object of the verb. This characteristic only holds for certain nominals like *invention*, in contrast to other nominals like *reproduction* or *proof* (with which the created object and the entity referred to by the direct object are distinct). To spell this out fully would require us to look at another sort of dependent ⊗-type, but we defer a detailed analysis to a future time.

Let's turn instead to the analysis of one of the examples, namely (30a). The logical form for a nominal like *invention* is as follows:

(36) $[\![\text{invention}]\!] : \{\lambda y \lambda e \lambda x\}\ \mathit{invention}(e, x, y)$

The derivation for sentence (30a) runs as follows:

a. $[\![\text{of the phonograph}]\!]$: $\lambda P \exists ! z(\mathit{phonograph}(z) \wedge P(z))\langle z\colon \textsc{obj}, P\colon \textsc{obj} \multimap t\rangle$

b. We type-raise *invention* distributively across + to combine with its object, giving it the complex type:[5]
$(((\textsc{created-obj} \multimap t) \multimap t) \multimap (\{\textsc{ag}, \textsc{ev}\} \multimap t)) + (((\textsc{ev} \multimap t) \multimap t) \multimap (\{\textsc{ag}, \textsc{created-obj}\} \multimap t))$

c. We assume $\textsc{ev} \sqcap \textsc{obj} = \perp$, so the generalized greatest lower bound of the argument and functor here is $\perp$, when *invention* has type $((((\textsc{ev} \multimap t) \multimap t) \multimap (\{\textsc{ag}, \textsc{created-obj}\} \multimap t))$. But since $\textsc{created-obj} \sqcap \textsc{obj} = \textsc{created-obj}$, the generalized greatest lower bound of the complement's meaning and *invention* is not $\perp$. So, by *+-Exploitation*, the type of $[\![\text{invention}]\!]$ becomes:
$((\textsc{created-obj} \multimap t) \multimap t) \multimap (\{\textsc{ag}, \textsc{ev}\} \multimap t)$.

d. By type *Accommodation*, the semantics for $[\![\text{of the phonograph}]\!]$ is adjusted to:
$\lambda P \exists ! z(\mathit{phonograph}(z) \wedge P(z))\langle P\colon \textsc{created-obj} \multimap t\rangle$

e. Finally, by *Application* and *Merging* contexts: $\{\lambda e, \lambda x\}\exists ! z(\mathit{phonograph}(z) \wedge \mathit{invention}(e, x, z))$ which has type $\{\textsc{ev}, \textsc{ag}\} \multimap t$.

f. We assume the AGent is either specified by a DP in the genitive or by an adjoining *by*-PP. In the absence of these constructions, the agent argument is closed off via an operation of existential closure (Asher, 1993). Combining the logical form in (e) with the determiner's meaning together then yields the right, event interpretation for the nominal.

Now what happens with a nominalization like *reproduction*? We postulate that such a nominalization has in fact a similar argument structure that it shares with its root verb, but the type of one of the arguments is already complex: it consists of an AG(ent), a CREATED-OBJ • EV, and an OBJect (which is the "source" of the reproduction). But here the λ binders are at least partially ordered:

(37) $[\![\text{reproduction}]\!] = \{\lambda y \lambda z\}\lambda u\ \mathit{reproduction}(u, z, y)\langle y : \textsc{obj}, z : \textsc{ag}, u : \textsc{created-obj} \bullet \textsc{ev}\rangle$

[5]We conjecture that type raising is always distributive with respect to +; we leave open the question as to whether type raising across other complex types can be distributive as well. We conjecture not, pointing to a fundamental logical difference between polysemy and homonymy.

This entry would predict that we can copredicate on the argument of the complex type which would be the head argument of the nominal that will combine with the determiner meaning. Notice that in this example we do not have a homonomy but rather a logical polysemy.

To conclude, note that our treatment of derived nominals sharply contrasts with Pustejovsky's account of these nominals (e.g., in (Pustejovsky, 1995)): (i) he always assumes dot objects, and (ii) he assumes EV • STATE. We think that these nominals come in two varieties: they may have arguments of complex type or they may themselves come with a complex type of the +-variety, even though all their arguments may have simple type.

4.4 +-Introduction

One might hypothesize something like the following for the rule of +-Introduction:

(38) **+-Introduction:**

$$\frac{\{\lambda P\phi(P(x)), c(P : [\,{}^{\alpha'}_{\beta'}\,] \multimap \gamma)\}[\psi, c'(\psi : (\alpha + \beta) \multimap \gamma)];\ \alpha \sqcap \beta = \perp}{\lambda P\phi(P(x))[\psi], c * (P : (\alpha + \beta) \multimap \gamma)}$$

Prima facie, there doesn't seem to be much room for such a rule. And this is what one might expect, for ambiguity tends to decrease (not to increase) as a result of semantic composition. There is however a case where this operation appears to be required, namely definitional cases like the following:

(39) A bank is either a financial institution or the shore of a river.

Here, the conjunction *a financial institution or the shore of a river* seems to force the introduction of the type FIN_INST + SHORE.

In its current form, +-Introduction would terribly overgenerate, producing spurious ambiguities. So we would have to put a guard on the +-Introduction rule, limiting it to discourse contexts of certain types. This kind of endeavor opens up a whole new territory for exploration: particular type rules mandated by a particular discourse context. This is something we believe is crucial to understanding the nature of the interaction between lexical information and discourse, but we forego this topic for another time.

Before we leave the subject of definitions, however, we note that something related may also be at work with negation:

(40) A rock is not an animal.

It seems that the effect of negation in this case is to select the *complement* of the type present in the predication; that is, it's a little like negation gets into the types, so to speak.

4.5 Comparison with ambiguity-as-set

There is another way one could account for lexical ambiguity, namely by assuming that ambiguous lexical items are associated with a *set* of types, and that this set gets filtered through composition. This view can be simply implemented by reviewing the *Type Accommodation* rule as follows:

(41) **Type Accommodation (generalized to sets):**

$$\frac{\lambda x\phi[t], c(x\colon \{\alpha_1, \ldots, \alpha_n\}, t\colon \{\beta_1, \ldots, \beta_m\}); \{\alpha_i \sqcap \beta_j\colon \alpha_i \sqcap \beta_j \neq \bot\} \neq \emptyset}{\lambda x\phi[t], c * (x, t\colon \{\alpha_i \sqcap \beta_j\colon \alpha_i \sqcap \beta_j \neq \bot\})}$$

How does this view compare to type disjunction? Well, there's a very close connection of course because we can define any finite set of elements $\{a_1, \ldots, a_n\}$ via a disjunction:

(42) $\exists x \forall y (y \in x \leftrightarrow (y = a_1 \vee y = a_2 \vee \ldots \vee y = a_n))$

We can model any finite sense enumeration of a homonymy via our +-Exploitation rule and our use of +-types. This seems preferable to us to introducing sets of types into the lexicon, which would complicate the type calculus considerably (right now it corresponds to a simple propositional calculus system with rules that are very constrained and so provide easy computations) and by introducing sets into the picture, even just finite sets, we would need much more resources.

5 Conclusion

What remains of the sense-enumeration view? It looks as though we might still need different lexical entries for cases of ambiguity where there is a change in syntactic category and in valence (e.g. *book* as a verb or a noun). But there may be a way to finesse this issue as well using complex types. For instance *book* could have the following complex +-type:

(43) $((\textsc{phys-obj} \bullet \textsc{info}) \multimap t) + ((\textsc{phys-obj} + \textsc{ev}) \multimap (\textsc{ev} \multimap (\textsc{agent} \multimap t)))$

If we hypothesize that combining two functional types with different numbers of arguments are incompatible, then we could use *+-Exploitation* in the same way as we have done here to build up logical forms for sentences with ambiguous words like *book*. Thus, we conjecture that our +-types may be quite useful to encoding all sorts of ambiguities other than logical polysemies.

6 Acknowledgements

We would like to thank Sabrina Parent and two anonymous reviewers for their comments and suggestions.

References

N. Asher and P. Denis. 2004. Dynamic typing for lexical semantics. a case study: the genitive construction. In A.C. Varzi and L. Vieu, editors, *Formal Ontology in Information Systems. Proceedingds of the Third International Conference (FOIS 2004)*, pages 165–176. IOS Press.

N. Asher and A. Lascarides. 2001. The semantics and pragmatics of metaphor. In P. Bouillon and F. Busa, editors, *The Language of Word Meaning*, pages 262–289. Cambridge University Press.

N. Asher and J. Pustejovsky. 2005. Word meaning and commonsense metaphysics. University of Texas at Austin and Brandeis University, available from `nasher@bertie.la.utexas.edu`.

N. Asher. 1993. *Reference to Abstract Objects in Discourse*. Kluwer Academic Publishers.

Gregory N. Carlson. 1977. *Reference to kinds in English*. Ph.D. thesis, Umass, Amherst.

Jane Grimshaw. 1990. *Argument Structure*. MIT Press, Cambridge, Mass.

Irene Heim and Angelika Kratzer. 1998. *Semantics in Generative Grammar*. Blackwell Publishers, Oxford.

W.A. Howard. 1980. The formulas-as-types notion of construction. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, New York.

Barbara Partee and Mats Rooth. 1983. Generalized conjunction and type ambiguity. In C. Schwarze R. Bauerle and A. von Stechow, editors, *Meaning, Use and Interpretation of Language*, pages 361–383. Walter de Gruyter, Berlin.

James Pustejovsky. 1995. *The Generative Lexicon*. MIT Press, Cambridge, MA.

James Pustejovsky. 2001. Type construction and the logic of concepts. In P. Bouillon and F. Busa, editors, *The Syntax of Word Meaning*. Cambridge University Press.