Contextual Analysis of Word Meanings in Type-Theoretical Semantics

Zhaohui Luo*

Dept of Computer Science, Royal Holloway, Univ of London
Egham, Surrey TW20 0EX, U.K.
zhaohui@cs.rhul.ac.uk

Abstract. Word meanings are context sensitive and may change in different situations. In this paper, we consider how contexts and the associated contextual meanings of words may be represented in type-theoretical semantics, the formal semantics based on modern type theories. It is shown, in particular, that the framework of coercive subtyping provides various useful tools in the representation.

1 Introduction

Word meanings are context sensitive. In any lexical semantics, it is important to spell out how different word meanings may be disambiguated. In this paper, we make proposals on how some word meanings may be represented in type-theoretical semantics [34,21], the formal semantics based on modern type theories, and how disambiguation can be done automatedly based on the representation method. In particular, it is shown that coercive subtyping [20] provides important tools for such a representation in type theory.

We start by dealing with homonymy, when the meanings of a homonym (e.g., 'run') can be disambiguated by their typings in the type-theoretical semantics. In such cases, *overloading* provides a suitable representation mechanism for sense enumeration. We show how coercive subtyping supports the overloading of word meanings and the automated sense selection.

As word meanings may change from context to context, some uses are only meaningful in certain contexts, not in others. Since subtyping relations are crucial in representing such informal contexts, we should extend the formal notion of context (in type theory and other logical systems) to incorporate the assumption of subtyping relations. We formally introduce coercion contexts and show how they may be used in contextual analysis.

The meanings of some homonyms may not be distinguished by their typings. For example, in the type-theoretical semantics, common nouns (e.g., 'bank') are interpreted as types; therefore, the semantic typings of CNs are the same (different types do not have different 'typings') and cannot be used in disambiguation.

* This work is partially supported by the research grant F/07-537/AJ of the Leverhulme Trust in U.K.

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 159–174, 2011.

For such disambiguation, we need to introduce *local coercions* (i.e., coercion contexts for terms) so that semantic interpretations can be given as intended.

There are some proposals that words should be given complex and structured meanings so that contributions to meaning generation can be made effectively (cf., Pustejovsky's work on Generative Lexicon Theory [31] and the opposing views of lexical atomism [12,32]). Although the author does not take a philosophical stand on this, it is worth stating that structured lexical entries have the potential to contribute to a computational treatment of the semantics in, for instance, building an inference engine based on a formal semantics. We shall study how structured lexical meanings of CNs may be represented by Σ-types and show that such representations are consistent with the successful treatment of copredication in type-theoretical semantics [21].

For all of the proposals in this paper, experiments have been done in the proof assistant Coq [9], which implements a modern type theory. This may serve to verify the proposals, but more importantly, it can be seen as the first step towards computer-assisted linguistic reasoning with proof engines that implement the type-theoretical semantics.

In §2, we first give an introduction to type-theoretical semantics. The representation of sense enumeration and selection by coercive subtyping is described in §3. Coercion contexts and local coercions, and their uses in contextual analysis, are studied in §4. In §5, we consider how some of the structured lexical entries may be represented as Σ-types in the context of studying copredication. Finally, we briefly describe our Coq experiments in §6, followed by the conclusion where some future work is discussed.

2 Type-Theoretical Semantics

By a *type-theoretical semantics*, we mean a formal semantics of natural languages based on modern type theories such as Martin-Löf's predicative type theory [25,29] and the impredicative type theory UTT [18]. Such a semantics is in the tradition of the Montague grammar [27] but the powerful type structures in a modern type theory provide new useful mechanisms for formal semantics of various linguistic features, some of which have been found difficult to describe in the Montagovian setting.

In this section, we give a brief overview of the basics of type-theoretical semantics, partly to lay down the background and partly to set up notations and terminologies.

2.1 A Brief Overview

The Montague grammar is based on Church's simple type theory [6], which is a *single-sorted* logic. In Montague grammar, there is a universal type e of entities: a common noun is interpreted as a function of type $e \to t$ and a verb or an adjective as a function of type $(e \to t) \to (e \to t)$, where t is the type of truth values.

In contrast, a modern type theory can be considered as a *many-sorted* logical system, where there are many sorts called *types* that may be used to stand for the domains to be represented. These types include:

- the *propositional types* (or logical propositions – see §2.2),
- the *inductive types* such as the type of natural numbers and Σ-types of dependent pairs (see the latter in §2.3), and
- other more advanced type constructions such as *type universes* (see §2.5).

Because of this many-sortedness, it is natural to interpret the noun phrases as types. Here are several basic interpretation principles one may adopt in a type-theoretical semantics [34]:

- Common nouns are interpreted as types. For instance, the CNs 'man' and 'human' can be interpreted as types $[\![man]\!]$ and $[\![human]\!]$, respectively.
- An adjective is interpreted as a predicate over the type that interprets the domain of the adjective. For instance, 'handsome' may be interpreted as a predicate $[\![handsome]\!] : [\![man]\!] \to Prop$, where $Prop$ is the type of logical propositions (see §2.2).
- Modified CNs are interpreted as Σ-types (see §2.3 for more details).

Furthermore, the framework of coercive subtyping provides us additional richer means for type-theoretical semantics [20,21] (see §2.4).

2.2 Embedded Logic

A modern type theory has an *embedded logic* (or *internal logic*) based on the propositions-as-types principle [10,14]. For example, in Martin-Löf's predicative type theory, the logical proposition $A\&B$ corresponds to the product type $A \times B$ (a special case of Σ-type – see below) and a pair of a proof of A and a proof of B corresponds to an object of the product type. Similarly, this correspondence extends to other logical operators: the logical implication ($\supset$) corresponds to the function types ($\to$), the universal quantifier ($\forall$) to the dependent Π-types, etc.

For Martin-Löf's type theory, the embedded logic is first-order and, for impredicative type theories such as ECC/UTT [18], the embedded logics are second-order or higher-order, where there is a type $Prop$ of logical propositions. Formally, $Prop$ is a totality and one can quantify over it to form other propositions (and this process is regarded as 'circular' by predicativists [11] or 'impredicative', in the technical jargon).[1]

In this paper, we shall use $Prop$ in linguistic interpretations. In a type-theoretical semantics, an assertive sentence is interpreted as a proposition of type $Prop$ and a verb or an adjective as a predicate of type $A \to Prop$, where A is the domain whose objects the verb or adjective can be meaningfully applied to. For instance, consider the following sentence:

[1] $Prop$ is very much like the type t in the simple type theory. The main difference is that, in modern type theories, we have explicit proof terms of logical propositions.

(1) John is handsome.

With $[\![John]\!] : [\![man]\!]$ and $[\![handsome]\!] : [\![man]\!] \to Prop$, the above sentence (1) is interpreted as proposition $[\![handsome]\!]([\![John]\!])$ of type $Prop$.

2.3 Dependent Types

Modern type theories contain *dependent types*. For instance, When A is a type and P is a predicate over A, $\Pi x{:}A.P(x)$ is the dependent function type that, in the embedded logic, stands for the universally quantified proposition $\forall x{:}A.P(x)$. Π-types degenerates to the function type $A \to B$ in the non-dependent case.

Another example of dependent type is the so-called Σ-types. If A is a type and B is an A-indexed family of types, then $\Sigma(A, B)$, or sometimes written as $\Sigma x{:}A.B(x)$, is a type, consisting of pairs (a, b) such that a is of type A and b is of type $B(a)$. When $B(x)$ is a constant type (i.e., always the same type no matter what x is), the Σ-type degenerates into product type $A \times B$ of non-dependent pairs. Σ-types (and product types) are associated projection operations π_1 and π_2 so that $\pi_1(a, b) = a$ and $\pi_2(a, b) = b$, for every (a, b) of type $\Sigma(A, B)$ or $A \times B$.

In a type-theoretical semantics, modified common nouns are interpreted as Σ-types. For instance, $\Sigma([\![man]\!], [\![handsome]\!])$ is the type of handsome men (or more precisely, of those men together with proofs that they are handsome).

Notations for Σ-types. A nested Σ-type can be seen as a type of tuples/modules. The following notations will be adopted (in §5). We shall use

$$\left\{ \begin{array}{l} x_1 : A_1 \\ \ldots \\ x_n : A_n \end{array} \right\}$$

to stand for the Σ-type $\Sigma x_1 : A_1 \Sigma x_2 : A_2$ ... $\Sigma x_{n-1} : A_{n-1}$. A_n. For example,

$$\left\{ \begin{array}{ll} name & : [\![man]\!] \\ hproof & : [\![handsome]\!](name) \end{array} \right\} \quad \text{stands for} \quad \Sigma([\![man]\!], [\![handsome]\!]).$$

Coq implementation of Σ-types. In the proof assistant Coq [9], Σ-types are implemented by means of the 'record' mechanism, which provide many automatic tools, including the following:

- If the record mechanism is used, the projection operators of the Σ-types are automatedly generated as globally defined terms, named by the 'labels' such as *name* and *hproof* in the above example.
- A projection operator may be indicated as a coercion. For instance, if we want to declare the first projection of the above Σ-type as a coercion, we may simply write `name :> Man` when defining the Σ-type. (cf., the definition of the dot-type `PhyInfo` in Appendix B.2.)

These automatic tools have greatly helped our experiments in Coq on the type-theoretical semantics, as reported in §6 and Appendix B.

2.4 Coercive Subtyping

Coercive subtyping [19,20] is an adequate theory of subtyping for modern type theories. In computer science, coercive subtyping has been implemented in many proof assistants such as Coq [9,35], Lego [23,4], Matita [26] and Plastic [5], and used effectively in interactive theorem proving. It has also been applied to type-theoretical semantics [21].

The basic idea of coercive subtyping is to consider subtyping as an abbreviation mechanism: A is a (proper) subtype of B ($A < B$) if there is a unique implicit coercion c from type A to type B and, if so, an object a of type A can be used in any context $\mathfrak{C}_B[\_]$ that expects an object of type B: $\mathfrak{C}_B[a]$ is legal (well-typed) and equal to $\mathfrak{C}_B[c(a)]$.

For instance, one may introduce $[\![man]\!] < [\![human]\!]$. Then, if we assume that $[\![John]\!] : [\![man]\!]$ and $[\![shout]\!] : [\![human]\!] \to Prop$, the interpretation (3) of (2) is well-typed:

(2) John shouts.

(3) $[\![shout]\!]([\![John]\!])$

according to the rule of coercive subtyping, because $[\![man]\!] < [\![human]\!]$.

2.5 Universes

Other more advanced features in a modern type theory are useful in developing the theory of type-theoretical semantics. For example, one may collect (the names of) some types into a type called a *universe* [25]. Introducing universes can be considered as a reflection principle: such a universe reflects those types whose names are its objects.

In type-theoretical semantics, universes can be introduced to help semantic interpretations. For instance, one may consider the universe CN : $Type$ of all common noun interpretations and, for each type A that interprets a common noun, there is a name $\overline{A}$ in CN. For example,

$$\overline{[\![man]\!]} : \text{CN} \quad \text{and} \quad T_{\text{CN}}(\overline{[\![man]\!]}) = [\![man]\!] .$$

In practice, we do not distinguish a type in CN and its name by omitting the overlines and the operator T_{CN} by simply writing, for instance, $[\![man]\!]$.

Type universes can be used in semantic interpretations. For instance, the universe CN can be used to give semantic interpretations to adverbs.[2] An adverb modifies a verb (an adjective) to result in a verb (adjective) phrase.[3] Since, in a type-theoretical semantics, verbs and adjectives are interpreted as predicates over a variety of domains (rather than over a single domain as in the Montagovian

[2] As far as the author is aware, there are no proposals in the literature on how adverbs should be interpreted in a type-theoretical semantics based on modern type theories.

[3] There are other adverbs. For example, an adverb may modify sentences to result in new sentences and, as in Montague grammar [27], such adverbs are interpreted as functions from $Prop$ to $Prop$.

setting), adverbs such as 'loudly' in 'John talked loudly' and 'simply' in 'That idea is simply ridiculous' would be interpreted as of type

$$\Pi A : \text{CN}.\ (A \to Prop) \to (A \to Prop).$$

For instance, for $[\![talk]\!] : [\![human]\!] \to Prop$, the following phrase (4) can be interpreted as (5), which is of type $[\![human]\!] \to Prop$:

(4) talk loudly

(5) $[\![loudly]\!]([\![human]\!], [\![talk]\!])$

Such interpretations of adverbs are experimented in Coq as, for example, shown in an example at the end of Appendix B.1.

3 Sense Selection via Overloading

In this paper, we shall consider how word meanings can be formalised in a type-theoretical semantics based on modern type theories. It is important to emphasise that the formal presentation of word meanings should naturally give rise to *automated disambiguation* in contexts so that, when considered in an interpretation of sentences or even larger linguistic entities where the concerned word occurs, the correct meaning will be automatedly selected.

In this section, we first start with the simple cases of homonymy and show that, when the meanings of a homonym can be differentiated by means of typing, its sense enumeration can be represented with coercive subtyping.

3.1 Sense Enumeration

A word may be homonymous with several unrelated meanings.[4] For instance, the word 'run' can be used in the following two sentences with different meanings.

(6) John runs quickly.

(7) John runs a bank.

In a type-theoretical semantics, we may have the following two different meanings of 'run', corresponding to the above uses:

(8) $[\![run]\!]_1 : [\![human]\!] \to Prop$

(9) $[\![run]\!]_2 : [\![human]\!] \to [\![institution]\!] \to Prop$

To represent the sense selection model, we need a mechanism that allows *automated selection* of the correct meaning when a sentence is interpreted. For instance, $[\![run]\!]_2$ in (9) should be selected automatedly when (7) is interpreted.

[4] Sense enumeration lexicons have been discussed, e.g., by Pustejovsky [31].

3.2 Simple Sense Selection via Overloading Based on Coercive Subtyping

In type theory, simple cases in the sense selection model can be represented by means of *overloading* (or *ad hoc polymorphism*) [36]. Intuitively, in the above example, the word 'run' is *overloaded* in the sense that it is associated with more than one meaning. Overloading can be supported by coercive subtyping [20,4], as explained below.

Assume that w be an arbitrary homonym with different meanings $[\![w]\!]_i : A_i$ $(i = 1, ..., n)$, where $A_j \neq A_k$ if $j \neq k$. Let $\mathbf{1}_w : \mathit{Type}$ be the inductive unit type with only one object $w : \mathbf{1}_w$ (see Appendix A for the formal details of the unit type). Then, the meaning of w is represented as the coercions $c_i : \mathbf{1}_w \to A_i$ $(i = 1, ..., n)$, defined as:

$$c_i(w) = [\![w]\!]_i : A_i.$$

For instance, the above word 'run' has two meanings $[\![\mathit{run}]\!]_1$ and $[\![\mathit{run}]\!]_2$ (in (8) and (9) in the previous subsection). The sense selection model for these two meanings of 'run' is given by the following two coercions:[5]

$$c_1(\mathit{run}) = [\![\mathit{run}]\!]_1 \quad \text{and} \quad c_2(\mathit{run}) = [\![\mathit{run}]\!]_2 .$$

This has the effect that, for example, in any context $\mathcal{C}_1[\mathit{run}]$ that requires an object of type $[\![\mathit{human}]\!] \to \mathit{Prop}$, we have

$$\mathcal{C}_1[\mathit{run}] = \mathcal{C}_1[c_1(\mathit{run})] = \mathcal{C}_1[[\![\mathit{run}]\!]_1],$$

and, in any context $\mathcal{C}_2[\mathit{run}]$ that requires an object of type $[\![\mathit{human}]\!]$ $\to [\![\mathit{institution}]\!] \to \mathit{Prop}$, we have

$$\mathcal{C}_2[\mathit{run}] = \mathcal{C}_2[c_2(\mathit{run})] = \mathcal{C}_2[[\![\mathit{run}]\!]_2].$$

Therefore, through automated insertions of coercions, the sentences (6) and (7) will both be interpreted correctly as expected.

Remark 1. For some homonyms, their different meanings may have the same type and therefore cannot be differentiated by typing. For instance, in a type-theoretical semantics, common nouns are interpreted as types and, therefore, the disambiguation of a homonymous common noun (e.g., 'bank') may depend on further linguistic information (e.g., in the case of 'bank', it may refer to some financial matters). One can resort to local coercions, as to be discussed in §4.2, to give formal interpretations in such situations. □

We have experimented in the proof assistant Coq [9] on sense selection based on the unit type and coercive subtyping. See Appendix A for unit types and Appendix B.1 for an example of homonymy.

[5] If other meanings of 'run' are considered, further coercions are defined accordingly.

4 Representation of Contexts: Coercion Contexts and Local Coercions

Word meanings are context sensitive. For instance, as discussed in the above section, a homonym has different meanings when used in different sentences, or in different *sentential contexts*. In other circumstances, the contexts are not sentential; what they describe can be either a special situation or a specific background. Many usages are only meaningful in such special situations or *local contexts* in which, for instance, the meanings of some words change.

Example 1 (reference transfer). *Consider the following utterance (cf., [30]):*

(10) The ham sandwich shouts.

Assuming that the act of shouting requires that the argument be human, it is obvious that sentence (10) is not well-formed, unless it is uttered by somebody in some special extralinguistic context (e.g., by a waiter in a café to refer to a person who has ordered a ham sandwich). □

4.1 Coercion Contexts in Type Theory

In a type-theoretical semantics, such local contexts can be described by means of the formal notion of context in type theory. Traditionally, a context in type theory is of the form

$$x_1 : A_1,\ ...,\ x_n : A_n$$

where A_i is either a data type, in which case x_i is assumed to be an object of that type, or a logical proposition, in which case the proposition A_i is assumed to be true and x_i a proof of A_i. For example, one may have the following context:

$$m : [\![man]\!],\ hproof : [\![handsome]\!](m)$$

which assumes, in layman's terms, that 'm is a man' and 'm is handsome' (with 'hproof' being a proof).

The formal notion of context can be extended by coercion declarations or subtyping assumptions, as proposed in [21]. A *coercion context* is a context whose entries may be of the form $A <_c B$ as well as the usual form $x : A$. For instance, the following context may be used to describe the special circumstances in a café:

(11) ..., $[\![ham\ sandwich]\!] < [\![human]\!]$, ...

where the subtyping assumption says that a ham sandwich can be coerced into a person (i.e., the person who has ordered a ham sandwich). In a context such as (11), the above sentence (10) can be interpreted satisfactorily as intended.

Formally, we have:

$$\frac{\Gamma \vdash A : Type \quad \Gamma \vdash B : Type \quad \Gamma \vdash c : (A)B}{\Gamma,\ A <_c B\ \ valid} \qquad \frac{\Gamma,\ A <_c B,\ \Gamma'\ \ valid}{\Gamma,\ A <_c B,\ \Gamma' \vdash A <_c B : Type}$$

where $(A)B$ is the functional kind from A to B in the logical framework (see Chapter 9 of [18] for formal details.) In other words, coercions can now be introduced in contexts and they are only valid 'locally' in the context where they are introduced. For example, sentence (10) can be reasonably interpreted in a context which contains the subtyping as shown in (11) and, otherwise, it cannot.

Remark 2. (coherent context) Please note that validity of a context is not enough anymore for it to be legal. One needs to make sure that the context is *coherent*, in the sense that the declared coercions in the context do not lead to more than one coercion between two types. Since it requires some formal backgrounds to be treated more concisely, its details are omitted here. □

Example 2. *Consider the following example (adapted from [24]):*

(12) Every linguist drinks a glass.

Let's assume that 'drink' be interpreted as:

$$[\![drink]\!] : [\![animated]\!] \to [\![liquid]\!] \to Prop.$$

Now, since not every container contains drinks, there should be a special context in which the above sentence (12) can be interpreted. The coercion context should contain the following subtyping relations:

$$[\![glass]\!] < [\![beverage]\!], \quad [\![beverage]\!] < [\![liquid]\!], \quad [\![linguist]\!] < [\![animated]\!]$$

Then, in a coercion context with the above, (12) can be interpreted as:

(13) $\forall l : [\![linguist]\!] \,.\, \exists g : [\![glass]\!] \,.\, [\![drink]\!](l, g).$

In the coercion context, (13) is well-typed. □

4.2 Local Coercions in Terms

Consider the following phrases that use the homonym 'bank':

(14) the bank of the river
(15) the richest bank in the city

From the previous subsection, we know that (14) and (15) can be interpreted in the following contexts (16) and (17), respectively:

(16) ... $[\![bank]\!] < [\![riverside]\!]$, ...
(17) ... $[\![bank]\!] < [\![financial\ institution]\!]$, ...

Now, what if we want to use the word 'bank' twice with these two different meanings in the same text (e.g., in a text where (14) is followed not far by

(15))? Although one is not forbidden to introduce a context that contains both subtyping assumptions:

(18) ... $[\![bank]\!] < [\![riverside]\!]$, ... $[\![bank]\!] < [\![financial\ institution]\!]$, ...

it may not be known which subtyping relation should be used in which sentence (unless some extra information is available). Automatic selection fails. Sometimes, the situation is even worse: it may be impossible to assume all the subtyping relations because it would lead to incoherence.

Such a problem can be solved by introducing *local coercions* – coercions that are only effective locally for some terms (expressions in type theory). Coercions may be introduced into terms by the following rule:

$$\frac{\Gamma,\ A <_c B \vdash J}{\Gamma \vdash \mathbf{coercion}\ A <_c B\ \mathbf{in}\ J}$$

where J is any of the following four forms of judgement:

$$k : K, \quad k = k' : K, \quad K\ kind, \quad \text{and} \quad K = K'.$$

For instance, with $J \equiv k : K$, we have

$$\frac{\Gamma,\ A <_c B \vdash k : K}{\Gamma \vdash \mathbf{coercion}\ A <_c B\ \mathbf{in}\ k : K}$$

Intuitively, the coercions declared locally are only effective in the expressions in the scope of the keyword **in**. For instance, for

$$\mathbf{coercion}\ [\![bank]\!] < [\![financial\ institution]\!]\ \mathbf{in}\ e,$$

the subtyping relation between $[\![bank]\!]$ and $[\![financial\ institution]\!]$ is only effective in the expression e, not outside e.

The key word **coercion** distributes through the components of J. For example, the following two judgements are identified:

$$\mathbf{coercion}\ A <_c B\ \mathbf{in}\ (k : K)$$
$$(\mathbf{coercion}\ A <_c B\ \mathbf{in}\ k)\ :\ (\mathbf{coercion}\ A <_c B\ \mathbf{in}\ K)$$

The introduction of local coercions broadens the scope of interpretation in applying the above techniques. For example, assuming that (14) and (15) have interpretations $[\![(14)]\!]$ under $[\![bank]\!] < [\![riverside]\!]$ and $[\![(15)]\!]$ under $[\![bank]\!] < [\![financial\ institution]\!]$, respectively, then the following two terms give their semantics and can be used together with no problem:

(19) $\mathbf{coercion}\ Bank < Riverside\ \mathbf{in}\ [\![(14)]\!]$

(20) $\mathbf{coercion}\ Bank < FinancialInst\ \mathbf{in}\ [\![(15)]\!]$

Since the coercions only take effects in the relevant expressions, the intended semantics gets represented correctly.

5 Structured Lexical Entries: Copredication and Beyond

Lexical entries can either be atomic, as advocated by Fodor and Lepore's lexical atomism [12,32], or complex and structured as proposed and studied by Pustejovsky and others [31]. According to lexical atomism, the basic words such as 'book' may only be properly interpreted in an atomic way, while Pustejovsky and others believe that they should be represented by means of rich and structured entities which, when combined with other linguistic entities in a sentence, make important contributions to meaning generations.

In this paper, we shall not argue for or against whether the lexicon should be generative and structured, but only to investigate, if words should be represented as complex and structured entities, how it can be done in a type-theoretical semantics. We want to add that, if one can represent word meanings successfully as structured entities that contribute to meaning generation, it will make substantial contribution to natural language processing in practice.

In this section, we shall first review how copredication (cf., [3]) can be captured in the type-theoretical semantics [21] and then show how complex and structured lexical entries, represented by Σ-types,[6] can be dealt with satisfactorily in this respect.

5.1 Copredication and Dot-Types

The dot-type and its use in lexical semantics were first proposed by Pustejovsky [31] and further studied by many others including, for example, Asher in the study of copredication [2,3]. In [21], the author has proposed a type-theoretic formal treatment of dot-types, with the help of coercive subtyping, and shown that the type-theoretical semantics with coercive subtyping gives a satisfactory treatment of copredication, among others.

Example 3 (copredication). *Let* PHY *and* INFO *be the types of physical objects and informational objects, respectively. One may consider the dot-type* PHY$\bullet$ INFO *as the type of the objects with both physical and informational aspects. A dot-type is a subtype of its constituent types:* PHY $\bullet$ INFO $<$ PHY *and* PHY $\bullet$ INFO $<$ INFO*. A book may be considered as having both physical and informational aspects, reflected as:*

$$(*) \qquad [\![book]\!] < \text{PHY} \bullet \text{INFO}.$$

Now, consider the following sentence:

(21) John picked up and mastered the book.

In a type-theoretical semantics, we may assume

$$[\![pickup]\!] : [\![human]\!] \to \text{PHY} \to Prop$$
$$[\![master]\!] : [\![human]\!] \to \text{INFO} \to Prop$$

[6] Cooper [8,7] has proposed to use dependent record kinds in Martin-Löf's type theory to represent lexical entries. The idea of using Σ-types to represent structured lexical entries was proposed, but not studied in any depth, in [22].

Because of the above subtyping relationship $(*)$ *(and contravariance of subtyping for the function types), we have*

$$\begin{array}{rcl} [\![pickup]\!] & : & [\![human]\!] \to \textsc{Phy} \to Prop \\ & < & [\![human]\!] \to \textsc{Phy} \bullet \textsc{Info} \to Prop \\ & < & [\![human]\!] \to [\![book]\!] \to Prop \end{array}$$

$$\begin{array}{rcl} [\![master]\!] & : & [\![human]\!] \to \textsc{Info} \to Prop \\ & < & [\![human]\!] \to \textsc{Phy} \bullet \textsc{Info} \to Prop \\ & < & [\![human]\!] \to [\![book]\!] \to Prop \end{array}$$

Therefore, $[\![pickup]\!]$ *and* $[\![master]\!]$ *can both be used in a context where terms of type* $[\![human]\!] \to [\![book]\!] \to Prop$ *are required and the interpretation of the sentence (21) can proceed as intended.* □

5.2 Generative Lexical Entries as Σ-types

When lexical entries are represented as complex and structured entities, copredication should be able to be treated in a similar way. For the above example, the key is to make sure that, if we interpret the word 'book' as a structured entity, the subtyping relation $(*)$ still holds.

Our proposal is that the basic common nouns such as 'book' be represented as Σ-types in type theory (see §2.3 for a brief introduction to Σ-types and the relevant notations). For example, the lexical entry for 'book' (p.15 of [33]) may be interpreted as the following Σ-type:

$$[\![book]\!] = \left\{ \begin{array}{ll} Arg & : \textsc{Phy} \bullet \textsc{Info} \\ Qualia : & \left\{ \begin{array}{ll} Formal & : Hold(p_1(Arg), p_2(Arg)) \\ Telic & : R(Arg) \\ Agent & : \exists h{:}Human.W(h, Arg) \end{array} \right\} \end{array} \right\}$$

where p_1 and p_2 are the associated projection operators for the dot-type $\textsc{Phy} \bullet \textsc{Info}$ (see [21] for formal details) and $R(x)$ and $W(h, x)$ informally stand for 'x to be read' and 'h wrote x', respectively.

If 'book' is given the above structured interpretation, is it true that the subtyping relation $(*)$ still holds? (i.e., do we still have $[\![book]\!] \leq \textsc{Phy} \bullet \textsc{Info}$?) The answer is yes, because we can take the first projection π_1 for Σ-types as a coercion. Therefore, the formal calculations in the previous section goes through.

6 Implementations in Coq

Type theories have been implemented in several proof assistants such as Agda [1] and Coq [9]. We have used Coq to experiment with the proposal as reported in this paper. Coq supports the use of coercions. Although it is not completely satisfactory, it can be used well to implement the examples. (We do not get into the details here.) Some of the Coq codes are given in Appendix B:

- In Appendix B.1, we give the Coq codes that implement the examples of homonyms as described in §3. Please note that, since Coq has certain restrictions on what coercions can be defined (esp. concerning function types), we have to use type-casting in order to make some coercion insertions work.
- In Appendix B.2, we give the Coq codes that interpret 'book' as a Σ-type (implemented in Coq as the 'record' mechanism), as in §5.2, and implement examples of copredication etc; in particular, it shows how the following sentences can be interpreted as intended:

  (22) John burned a boring book.

  (23) John picked up and mastered book B.

7 Conclusion

Lexical semantics must consider how to give appropriate meaning explanations to the words which have different meanings in different contexts. We have studied how this may be done in various situations in the type-theoretical semantics based on modern type theories. The proposals can all be implemented on the computer based on the automated selection mechanisms with the help of coercive subtyping, among others. It would be interesting to see how such a system can be built based on the type-theoretical semantics.

The study of meta-theory of coercion contexts and local coercions is beyond the scope of the current paper. Although we do not foresee key difficulties, a challenge may be concerned with the notion of 'coherent context' as mentioned in Remark 2 in §4.1. The meta-theoretic properties such as strong normalisation do not just concern valid contexts, but coherent contexts and this new phenomena may need new tools to be dealt with satisfactorily.

In this paper, we have used Σ-types to represent structured lexical entries of CNs. However, other words such as verbs should not be represented as types. In general, the structured semantics of a word may be represented as a pair

$$(A, \phi), \text{ where } A : \mathit{Type} \text{ and } \phi : A \to \mathit{Prop}.$$

Such constructions have been studied in the contexts of mathematical theories [16] and program specifications [17]. Similar constructions seem to be needed in the current context and further studies are called for in this respect.

Among related work, we should mention that by Francez et al [13], where the proof-theoretic semantics (cf., [15]) of a fragment of natural language has been studied. Although it is quite different from type-theoretic semantics, there is one thing in common: proof-theoretic ideas play a central role in such semantics, in contrast to the more dominant model-theoretic approaches, including the Montague semantics.

Finally, it may also be helpful to mention that, in natural language processing, there is a substantial amount of work on the word sense disambiguation (WSD) problem (see, for example, [28] for a recent survey) and to emphasise that this is quite different (e.g., in aims) from the work reported here.

References

1. The Agda proof assistant (2008), http://appserv.cs.chalmers.se/users/ulfn/wiki/agda.php?
2. Asher, N.: A type driven theory of predication with complex types. Fundamenta Infor. 84(2) (2008)
3. Asher, N.: Lexical Meaning in Context: A Web of Words, draft, CUP (2010)
4. Bailey, A.: The Machine-checked Literate Formalisation of Algebra in Type Theory. Ph.D. thesis, University of Manchester (1999)
5. Callaghan, P., Luo, Z.: An implementation of LF with coercive subtyping and universes. Journal of Automated Reasoning 27(1), 3–27 (2001)
6. Church, A.: A formulation of the simple theory of types. J. Symbolic Logic 5(1) (1940)
7. Cooper, R.: Records and record types in semantic theory. J. Logic and Compututation 15(2) (2005)
8. Cooper, R.: Copredication, dynamic generalized quantification and lexical innovation by coercion. Proceedings of GL 2007, the Fourth International Workshop on Generative Approaches to the Lexicon (2007)
9. The Coq Development Team: The Coq Proof Assistant Reference Manual (Version 8.1), INRIA (2007)
10. Curry, H., Feys, R.: Combinatory Logic, vol. 1. North-Holland, Amsterdam (1958)
11. Feferman, S.: Predicativity. In: Shapiro, S. (ed.) The Oxford Handbook of Philosophy of Mathematics and Logic. Oxford Univ. Press, Oxford (2005)
12. Fodor, J.A., Lepore, E.: The emptiness of the lexicon: Reflections on james pustejovskys the generative lexicon. Linguistic Inquiry 29(2), 269–288 (1998)
13. Francez, N., Dyckhoff, R.: Proof-theoretic semantics for a natural language fragment. In: Ebert, C., Jäger, G., Michaelis, J. (eds.) MOL 10. LNCS, vol. 6149, pp. 56–71. Springer, Heidelberg (2010)
14. Howard, W.A.: The formulae-as-types notion of construction. In: Hindley, J., Seldin, J. (eds.) To H. B. Curry: Essays on Combinatory Logic. Academic Press, London (1980)
15. Kahle, R., Schroder-Heister, P. (eds.) Synthese, vol. 148(3) (2006)
16. Luo, Z.: A higher-order calculus and theory abstraction. Information and Computation 90(1) (1991)
17. Luo, Z.: Program specification and data refinement in type theory. Mathematical Structures in Computer Science 3(3) (1993)
18. Luo, Z.: Computation and Reasoning: A Type Theory for Computer Science. Oxford Univ. Press, Oxford (1994)
19. Luo, Z.: Coercive subtyping in type theory. In: van Dalen, D., Bezem, M. (eds.) CSL 1996. LNCS, vol. 1258. Springer, Heidelberg (1997)
20. Luo, Z.: Coercive subtyping. J. of Logic and Computation 9(1), 105–130 (1999)
21. Luo, Z.: Type-theoretical semantics with coercive subtyping. In: Semantics and Linguistic Theory 20 (SALT 20), Vancouver (2010)
22. Luo, Z., Callaghan, P.: Coercive subtyping and lexical semantics (extended abstract). In: Logical Aspects of Computational Linguistics (LACL 1998) (1998)
23. Luo, Z., Pollack, R.: LEGO Proof Development System: User's Manual. LFCS Report ECS-LFCS-92-211, Dept. of Computer Science, Univ. of Edinburgh (1992)
24. Marlet, R.: When the generative lexicon meets computational semantics. In: 4th Inter. Workshop on Generative Approaches to the Lexicon (2007)
25. Martin-Löf, P.: Intuitionistic Type Theory. Bibliopolis (1984)

26. The Matita proof assistant (2008), http://matita.cs.unibo.it/
27. Montague, R.: Formal Philosophy. Yale University Press, New Haven (1974)
28. Navigli, R.: Word sense disambiguation: a survey. ACM Computing Surveys 41(2) (2009)
29. Nordström, B., Petersson, K., Smith, J.: Programming in Martin-Löf's Type Theory: An Introduction. Oxford University Press, Oxford (1990)
30. Nunberg, G.: Transfers of meaning. J. of Semantics 12(2) (1995)
31. Pustejovsky, J.: The Generative Lexicon. MIT, Cambridge (1995)
32. Pustejovsky, J.: Generativity and explanation in semantics: A reply to fodor and lepore. Linguistic Inquiry 29(2), 289–311 (1998)
33. Pustejovsky, J.: The semantics of lexical underspecification. Folia Linguistica (1998)
34. Ranta, A.: Type-Theoretical Grammar. Oxford University Press, Oxford (1994)
35. Saïbi, A.: Typing algorithm in type theory with inheritance. In: POPL 1997 (1997)
36. Strachey, C.: Fundamental concepts in programming languages. Higher-Order and Symbolic Computation 13(1-2) (2000), (Paper based on 1967 lectures)

A The Unit Type

A unit type is an inductive type that has only one object. It is one of the inductive types in Martin-Löf's type theory or UTT, whose formal details can be found in, for example, Chapter 9 of [18]. In this paper, we consider a class of unit types: a unit type $\mathbf{1}_w$ for each word w. These unit types can be introduced by means of the following rules

$$\frac{}{\mathbf{1}_w : Type} \quad \frac{}{w : \mathbf{1}_w} \quad \frac{C : (\mathbf{1}_w)Type \quad c : C(w) \quad z : \mathbf{1}_w}{\mathcal{E}_w(C, c, z) : C(z)}$$

together with the computation rule $\mathcal{E}_w(C, c, w) = c : C(w)$ stating that, when applied to the canonical object w, its elimination operator $\mathcal{E}_w$ computes to c. For Coq implementation, see Appendix B.1 for that of $\mathbf{1}_{run}$.

B Implementations in Coq

B.1 Homonymy in Coq

```
(* Lexical Semantics using Coq's records: simple homonymy *)

(* Categories of Sentences & CNs *)
Definition S := Prop.
Definition CN := Set.
Parameters Bank Institution Human Man : CN.
Parameter John : Man.
Axiom mh : Man->Human.  Coercion mh : Man >-> Human.
Axiom bi : Bank->Institution.  Coercion bi : Bank >-> Institution.

(* unit type for "run" *)
Inductive Onerun : Set := run.
```

```
Definition RSem1 := Human->S.
Definition RSem2 := Human->Institution->S.
Parameter run1 : RSem1.
Parameter run2 : RSem2.
Definition r1 (r:Onerun) : RSem1 := run1.  Coercion r1 : Onerun >-> RSem1.
Definition r2 (r:Onerun) : RSem2 := run2.  Coercion r2 : Onerun >-> RSem2.

(* John runs quickly *)
Parameter quickly : forall (A:CN), (A->S)->(A->S).
Definition john_runs_quickly := quickly Human (run:RSem1) John.
(* John runs a bank *)
Definition john_runs_a_bank := exists b:Bank, (run:RSem2) John b.
```

B.2 Copredications in Coq

```
(* Note: Coq's "record-types" is Sigma-types plus auto-defns of projections & coercions *)

(* Categories of Sentences and CNs *)
Definition S := Prop.
Definition CN := Set.

Parameter Human Man : CN.
Axiom mh : Man->Human.
Coercion mh : Man >-> Human.

(* Phy dot Info *)
Parameter Phy Info : CN.
Record PhyInfo : CN := mkPhyInfo { phy :> Phy; info :> Info }.

(* Book as Sigma-type with PhyInfo & BookQualia *)
Parameter Hold : Phy->Info->Prop.
Parameter R : PhyInfo->Prop.
Parameter W : Human->PhyInfo->Prop.
Record BookQualia (A:PhyInfo) : Set :=
  mkBookQualia { Formal : Hold A A;
                 Telic  : R A;
                 Agent  : exists h:Human, W h A }.
Record Book : Set := mkBook { Arg :> PhyInfo; Qualia : BookQualia Arg }.

(* "John burned a boring book" *)
Parameter John : Man.
Parameter boring : Info->S.
Record BBook : CN := mkBBook { b :> Book; _ : boring b }.
Parameter burn : Human->Phy->S.
Definition John_burned_a_boring_book := exists b:BBook, (burn John b) : S.

(* copredication: "John picked up and mastered book B" *)
Parameter B : Book.
Parameter pickup : Human->Phy->S.
Parameter master : Human->Info->S.
Definition John_picked_up_and_mastered_book_B := and (pickup John B) (master John B).
```