Lexical selection, coercion, and record types

William Babonnaud and Philippe de Groote

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy,

Abstract. This paper introduces a simply-typed λ-calculus dedicated to the treatment of the lexical phenomena of restrictive selection and type coercion. This calculus features records and record types, subtyping through explicit coercion, and bounded polymorphism. We show that coercion inference is decidable and discuss the canonicity of the inferred solutions.

1 Introduction

The use of records and record types in computational linguistics (especially in semantics) has been widely advocated and acknowledged. A possible reason for this is that records resemble feature structures, and allow numerous linguistic concepts to be formalized in a type-theoretic way (see [6] for instance).

In this paper, we propose to use records and record types to accommodate, in a compositional way, the lexical phenomena of restrictive selection and type coercion. The literature on lexical semantics, in general, and on lexical selection and type coercion, in particular, is very large. Our purpose here is not to propose a new lexical semantics theory, or a new organisation of the lexicon. We rather want to show how existing proposals can be expressed using records and record types. In particular, the solution we propose relies on some of the analyses made in [15].

In order to illustrate the kind of problem we want to solve, consider the following sentence, which will serve as a running example:

John bought a book which is voluminous but interesting (1)

Now, suppose that the semantics of the word "*voluminous*" is expressed by means of a predicate **voluminous** that only applies to physical objects (p). Similarly, suppose that the semantics of "*interesting*" and of "*bought*" are respectively expressed using the predicate **interesting** and the binary relation **buy**. The former only applies to informational contents (i), while the later takes a human being (h) as first argument, and a commodity (c) as second argument. The questions we want to answer are then the following ones:

- how can one allow a book to be at the same time a commodity, a physical object, and an informational content?
- according to these three cases, how does one select the corresponding meaning of the word "*book*"?

What is needed here is an object that would feature both aspects of a book (its physical aspect and its informational one). Such an object, in fact, is known in the literature under the name of *dot-object* [14]. But then, the question becomes: what could be an appropriate type-theoretic model of dot-objects, and how could one implement it?

The nature of dot-objects suggests that dot-types should feature introduction and elimination rules of the following form:

$$\frac{e_1 : A \qquad e_2 : B}{\mathit{Intro}(e_1, e_2) : A \bullet B} \qquad\qquad \frac{e : A \bullet B}{\mathit{Elim}_1(e) : A} \qquad\qquad \frac{e : A \bullet B}{\mathit{Elim}_2(e) : B}$$

These rules are reminiscent of two well-known type-theoretic constructs: intersection types and cartesian products (and also record types, which are cartesian products under disguise). In [1], Asher argues against the use of intersection types or cartesian products as a model of dot-types. He then proposes a categorical formalisation of the dot-types that essentially amounts to a pullback (see [2] for a discussion). Pullbacks are in fact a kind of product. In set-theoretic terms, they are known as *fibrated products*. In some sense, it is precisely those pullbacks that we want to implement using records and record types.

2 Record types and inheritance

The object language we use to express semantic recipes is the simply typed λ-calculus provided with a standard notion of record, that is, a finite association of labels and values. Accordingly, a record type is a finite set of labels together with their types. For instance, for the purpose of our running example, we define the type book as follows

$$\mathsf{book} = \{object : \mathsf{a}\,;\, content : \mathsf{i}\} \tag{2}$$

where a and i stands for the type of *artifacts* and the type of *informational contents*, respectively.

More precisely, the notion of type we use is given in the following definition.

Definition 1. *Let $\mathcal{A}$ be a set of atomic types (containing the Montagovian types e and t), and let $\mathcal{L}$ be a set of labels. The set of* types *is inductively defined as follows:*

1. *every $\mathsf{a} \in \mathcal{A}$ is a type;*
2. *if α and β are types, so is $(\alpha \to \beta)$;*
3. *if $\alpha_1, \ldots, \alpha_n$ are types, and $l_1, \ldots l_n \in \mathcal{L}$, then $\{l_1 : \alpha_1\,;\, \ldots\,;\, l_n : \alpha_n\}$ is a type.*

The notion of term is defined accordingly.

Definition 2. *Let $\mathcal{X}$ be a set of λ-variables, and let $\mathcal{L}$ be a set of labels. The set of* λ-terms *is inductively defined as follows:*

1. *every $x \in \mathcal{X}$ is a λ-term;*
2. *if t is a λ-term, and $x \in \mathcal{X}$, then $(\lambda x.\, t)$ is a λ-term;*
3. *if t and u are λ-terms, so is $(t\, u)$;*
4. *if $t_1, \ldots, t_n$ are λ-terms, and $l_1, \ldots l_n \in \mathcal{L}$, then $\{l_1 = t_1\,;\, \ldots\,;\, l_n = t_n\}$ is a λ-term;*
5. *if t is a λ-term, and $l \in \mathcal{L}$, then $t_\bullet l$ is a λ-term.*

In a record (resp., a record type), the order in which the labels and their associated terms (resp., associated types) are listed is not relevant. We sometimes use a vector-like notation to denote records or record types. For instance, we write:

$$\left\{\overline{l_i : \alpha_i}\right\}_n \ \text{ for } \ \{l_1 : \alpha_1\,;\, \ldots\,;\, l_n : \alpha_n\}$$

The set $\mathcal{A} \setminus \{\mathsf{t}\}$ is provided with a partial order relation $\mathcal{O}$, such that for every $\mathsf{a} \in \mathcal{A}$, $\mathsf{a} \leq \mathsf{e}$. This order induces a subtyping relation that obeys the following rules.

$$\lambda x.\, x : \alpha \leq \alpha \qquad\qquad \lambda x.\, x : \mathsf{p} \leq \mathsf{q},\ \text{for } (p, q) \in \mathcal{O}$$

$$\frac{c_1 : \alpha \leq \beta \qquad c_2 : \beta \leq \gamma}{\lambda x.\, c_2\,(c_1\, x) : \alpha \leq \gamma} \qquad\qquad \frac{c_1 : \alpha_1 \leq \beta_1 \qquad c_2 : \alpha_2 \leq \beta_2}{\lambda f x.\, c_2\,(f\,(c_1\, x)) : \beta_1 \to \alpha_2 \leq \alpha_1 \to \beta_2}$$

$$\lambda x.\, x : \{l_1 : \alpha_1\,;\, \ldots\,;\, l_n : \alpha_n\} \leq \{l_1 : \alpha_1\,;\, \ldots\,;\, l_{n-1} : \alpha_{n-1}\}$$

$$\frac{c_1 : \alpha_1 \leq \beta_1 \qquad \cdots \qquad c_n : \alpha_n \leq \beta_n}{\lambda r.\, \left\{\overline{l_i = c_i\,(r_\bullet l_i)}\right\}_n : \left\{\overline{l_i : \alpha_i}\right\}_n \leq \left\{\overline{l_i : \beta_i}\right\}_n}$$

The above rules, which are quite standard [3, 10], allows one to derive judgements of the form $f : \alpha \leq \beta$, where α and β are types, and f is a function that coerces a value of type α into a value of type β. If one does not consider any additional rule, computing the coercion function is not quite interesting because it always amount to identity. We therefore add, to the above set of rules, the following inequation:

$$\lambda r.\, r_\bullet l : \{\rho\,;\, l : \alpha\} \leq \alpha \tag{3}$$

In order to complete the definition of our object language we should specify the typing rules. For reasons that will become clear later, we delay this question to the next section. In the meantime, let us illustrate how the coercion calculus works by considering an example. Remember that we have defined the type book as follows:

$$\mathsf{book} = \{\mathit{object} : \mathsf{a}\,;\, \mathit{content} : \mathsf{i}\}$$

where a stands for the type of *artifacts*, and i for the type of *informational contents*. Assume that the predicate **voluminous**, which only applies to physical objects, is of type $(\mathsf{p} \rightarrow \mathsf{t})$, and that a is a subtype of p. Then, using the following coercion rule:

$$\frac{\vdash t : \alpha \qquad c : \alpha \leq \beta}{\vdash c\, t : \beta}$$

we may coerce **voluminous** to be of type $(\mathsf{book} \rightarrow \mathsf{t})$:

$$\dfrac{\vdash \textbf{voluminous} : \mathsf{p} \rightarrow \mathsf{t} \qquad \dfrac{\dfrac{\lambda r.\, r_\bullet \mathit{object} : \mathsf{book} \leq \mathsf{a} \qquad \lambda x.\, x : \mathsf{a} \leq \mathsf{p}}{\lambda r.\, r_\bullet \mathit{object} : \mathsf{book} \leq \mathsf{p}} \qquad \lambda x.\, x : \mathsf{t} \leq \mathsf{t}}{\lambda f r.\, f\,(r_\bullet \mathit{object}) : \mathsf{p} \rightarrow \mathsf{t} \leq \mathsf{book} \rightarrow \mathsf{t}}}{\vdash \lambda r.\, \textbf{voluminous}\,(r_\bullet \mathit{object}) : \mathsf{book} \rightarrow \mathsf{t}}$$

where

$$\lambda r.\, r_\bullet \mathit{object} : \mathsf{book} \leq \mathsf{a}$$

is an instance of inequation (3).

The above derivation allows one to coerce the semantic recipe assigned to "*voluminous*" to be of type $\mathsf{book} \rightarrow t$:

$$\textsc{voluminous} := \lambda r.\, \textbf{voluminous}\,(r_\bullet \mathit{object}) : \mathsf{book} \rightarrow t \tag{4}$$

Similarly, one can obtain:

$$\textsc{interesting} := \lambda r.\, \textbf{interesting}\,(r_\bullet \mathit{content}) : \mathsf{book} \rightarrow t \tag{5}$$

3 Bounded polymorphism

The next step is to coordinate "*voluminous*" and "*interesting*" using the conjunction "*but*". We interpret the latter as a logical conjunction:

$$\textsc{but} := \lambda p q x.\, (p\, x) \wedge (q\, x)$$

The Montagovian type assigned to the above λ-term is $(\mathsf{e} \rightarrow \mathsf{t}) \rightarrow (\mathsf{e} \rightarrow \mathsf{t}) \rightarrow \mathsf{e} \rightarrow \mathsf{t}$. This type, unfortunately, does not subsume $(\mathsf{book} \rightarrow \mathsf{t}) \rightarrow (\mathsf{book} \rightarrow \mathsf{t}) \rightarrow \mathsf{book} \rightarrow \mathsf{t}$

because e occurs both positively and negatively in it. What is needed here is a kind of polymorphism, as suggested in [15]. We do not wish, however, to enhance the type system with system F full polymorphism [8, 9] because we would loose decidability. We could use prenex polymorphism à la ML, but it would allow to instantiate BUT with any two terms of type $(\alpha \to \mathsf{t})$, whatever α is. This is not either what we want. What we need, in fact, is instantiating BUT with terms of type $(\alpha \to \mathsf{t})$ where α is a subtype of e. Consequently, we extend our type system with bounded polymorphism [4, 5], or more precisely, prenex bounded polymorphism (in order to keep decidability). To this end, we consider a set of type variables $\mathcal{V}$, and we extend the definition of a type as follows.

Definition 3. *Let $\mathcal{A}$ be a set of atomic types (containing the Montagovian types e and t), let $\mathcal{V}$ be a set of type variables, and let $\mathcal{L}$ be a set of labels. The set of* types, *and the set of* type schemes *are inductively defined as follows:*

1. *every $\mathsf{a} \in \mathcal{A}$ is a type;*
2. *every $a \in \mathcal{V}$ is a type;*
3. *if α and β are types, so is $(\alpha \to \beta)$;*
4. *if $\alpha_1, \ldots, \alpha_n$ are types, and $l_1, \ldots l_n \in \mathcal{L}$, then $\{l_1 : \alpha_1 \,;\, \ldots \,;\, l_n : \alpha_n\}$ is a type.*
5. *every type is a type scheme;*
6. *if $a \in \mathcal{V}$, α is a type, and β is a type scheme, then $\forall a \leq \alpha.\, \beta$ is a type scheme.*

Because we are dealing with a notion of bounded polymorphism, the typing rules will depend of subtyping assumptions of the form "$a \leq \alpha$", where a is a type variable and α a type. In addition, because we are using explicit coercions, these subtyping assumptions will be associated to coercion-variables by means of declarations of the following shape: "$c : a \leq \alpha$", where c is such a coercion-variable. This notion of coercion variable also play a part in the new definition of a λ-term.

Definition 4. *Let $\mathcal{X}$ be a set of λ-variables, $\mathcal{C}$ be a set of coercion-variables, and $\mathcal{L}$ be a set of labels. The set of* λ-terms *and of* schematic λ-terms *are inductively defined as follows:*

1. *every $x \in \mathcal{X}$ is a λ-term;*
2. *if t is a λ-term, and $x \in \mathcal{X}$, then $(\lambda x.\, t)$ is a λ-term;*
3. *if t and u are λ-terms, so is $(t\, u)$;*
4. *if $t_1, \ldots, t_n$ are λ-terms, and $l_1, \ldots l_n \in \mathcal{L}$, then $\{l_1 = t_1 \,;\, \ldots \,;\, l_n = t_n\}$ is a λ-term;*
5. *if t is a λ-term, and $l \in \mathcal{L}$, then $t \bullet l$ is a λ-term.*
6. *if t is λ-term, and $c \in \mathcal{C}$, then $(c\, t)$ is a λ-term;*
7. *every λ-term is a schematic λ-term;*
8. *if t is a schematic λ-term, and $c \in \mathcal{C}$, then $(\Lambda c.\, t)$ is a schematic λ-term.*

We are now in a position of giving the typing system of our calculus. It is specified by means of the five following kinds of judgements:

Γ **Cenv**	(Γ is a well-formed coercion environment)
$\Gamma \vdash \Delta$ **env**	(Δ is a well-formed typing environment)
$\Gamma \vdash \alpha$ **type**	(α is a well-formed type)
$\Gamma \vdash s : \alpha \leq \beta$	(s is a coercion from type α into type β)
$\Gamma; \Delta \vdash t : \alpha$	(t is a term of type α)

The inference rules that allow these different judgement to be derived are listed here below.

Coercion-environment formation rules

$$\frac{}{\varnothing\ \mathbf{Cenv}}\ (\textsc{Cempty}) \qquad \frac{\Gamma\ \mathbf{Cenv} \quad \Gamma \vdash \alpha\ \mathbf{type}}{(\Gamma, c : \mathsf{a} \leq \alpha)\ \mathbf{Cenv}}\ (\textsc{Ccons})$$

Typing-environment formation rules

$$\frac{}{\Gamma \vdash \varnothing\ \mathbf{env}}\ (\textsc{empty}) \qquad \frac{\Gamma \vdash \Delta\ \mathbf{env} \quad \Gamma \vdash \alpha\ \mathbf{type}}{\Gamma \vdash (\Delta, x : \alpha)\ \mathbf{env}}\ (\textsc{cons})$$

Type formation rules

$$\frac{\mathsf{a} \in \mathcal{A}}{\Gamma \vdash \mathsf{a}\ \mathbf{type}}\ (\textsc{Tconst})$$

$$\frac{}{\Gamma, c : a \leq \alpha \vdash a\ \mathbf{type}}\ (\textsc{Tvar}) \qquad \frac{\Gamma \vdash a\ \mathbf{type} \quad b \neq a}{\Gamma, c : b \leq \alpha \vdash a\ \mathbf{type}}\ (\textsc{Tweak})$$

$$\frac{\Gamma \vdash \alpha\ \mathbf{type} \quad \Gamma \vdash \beta\ \mathbf{type}}{\Gamma \vdash (\alpha \to \beta)\ \mathbf{type}}\ (\textsc{Tfun}) \qquad \frac{\Gamma \vdash \alpha_1\ \mathbf{type} \quad \cdots \quad \Gamma \vdash \alpha_n\ \mathbf{type}}{\Gamma \vdash \left\{\overline{l_i : \alpha_i}\right\}_n\ \mathbf{type}}\ (\textsc{Trec})$$

Coercion rules

$$\frac{(p, q) \in \mathcal{O}}{\Gamma \vdash \lambda x.\, x : \mathsf{p} \leq \mathsf{q}}\ (\textsc{Cconst})$$

$$\frac{}{\Gamma, c : a \leq \alpha \vdash c : a \leq \alpha}\ (\textsc{Cvar}) \qquad \frac{\Gamma \vdash c_2 : a \leq \alpha \quad b \neq a}{\Gamma, c_1 : b \leq \alpha \vdash c_2 : a \leq \alpha}\ (\textsc{Cweak})$$

$$\frac{}{\Gamma \vdash \lambda x.\, x : \alpha \leq \alpha}\ (\textsc{refl}) \qquad \frac{\Gamma \vdash s_1 : \alpha \leq \beta \quad \Gamma \vdash s_2 : \beta \leq \gamma}{\Gamma \vdash \lambda x.\, s_2\,(s_1\, x) : \alpha \leq \gamma}\ (\textsc{trans})$$

$$\frac{\Gamma \vdash s_1 : \alpha_1 \leq \beta_1 \quad \Gamma \vdash s_2 : \alpha_2 \leq \beta_2}{\Gamma \vdash \lambda f x.\, s_2\,(f\,(s_1\, x)) : (\beta_1 \to \alpha_2) \leq (\alpha_1 \to \beta_2)}\ (\textsc{Cfun})$$

$$\frac{}{\Gamma \vdash \lambda x.\, x : \left\{\overline{l_i : \alpha_i}\right\}_n \leq \left\{\overline{l_i : \alpha_i}\right\}_{n-1}}\ (\textsc{Crec1})$$

$$\frac{\Gamma \vdash s_1 : \alpha_1 \leq \beta_1 \quad \cdots \quad \Gamma \vdash s_n : \alpha_n \leq \beta_n}{\Gamma \vdash \lambda r.\, \left\{\overline{l_i = s_i\,(r\bullet l_i)}\right\}_n : \left\{\overline{l_i : \alpha_i}\right\}_n \leq \left\{\overline{l_i : \beta_i}\right\}_n}\ (\textsc{Crec2})$$

$$\frac{}{\Gamma \vdash \lambda r.\, r\bullet l_k : \left\{\overline{l_i : \alpha_i}\right\}_n \leq \alpha_k}\ (\textsc{Csel})$$

Typing rules

$$\frac{}{\Gamma\,;\, \Delta, x : \alpha \vdash x : \alpha}\ (\textsc{var}) \qquad \frac{\Gamma\,;\, \Delta \vdash x : \beta \quad y \neq x}{\Gamma\,;\, \Delta, y : \alpha \vdash x : \beta}\ (\textsc{weak})$$

$$\frac{\Gamma\,;\,\Delta, x:\alpha \vdash t:\beta}{\Gamma\,;\,\Delta \vdash \lambda x.\,t:\alpha\to\beta}\ (\textsc{abs}) \qquad \frac{\Gamma\,;\,\Delta \vdash t:\alpha\to\beta \quad \Gamma\,;\,\Delta \vdash u:\alpha}{\Gamma\,;\,\Delta \vdash t\,u:\beta}\ (\textsc{app})$$

$$\frac{\Gamma\,;\,\Delta \vdash t_1:\alpha_1 \quad \cdots \quad \Gamma\,;\,\Delta \vdash t_n:\alpha_n}{\Gamma\,;\,\Delta \vdash \left\{\overline{l_i = t_i}\right\}_n : \left\{\overline{l_i : \alpha_i}\right\}_n}\ (\textsc{rec}) \quad \frac{\Gamma\,;\,\Delta \vdash t : \left\{\overline{l_i : \alpha_i}\right\}_n}{\Gamma\,;\,\Delta \vdash t\bullet l_i : \alpha_i}\ (\textsc{sel})$$

$$\frac{\Gamma, c : a \le \alpha\,;\, \vdash t:\beta}{\Gamma\,;\, \vdash \Lambda c.\,t : \forall a \le \alpha.\,\beta}\ (\textsc{gen}) \qquad \frac{\Gamma\,;\, \vdash \Lambda c.\,t : \forall a \le \alpha.\,\beta \quad \Gamma \vdash s : \gamma \le \alpha}{\Gamma\,;\, \vdash t[c := s] : \beta[a := \gamma]}\ (\textsc{inst})$$

$$\frac{\Gamma;\Delta \vdash t:\alpha \quad \Gamma \vdash s:\alpha \le \beta}{\Gamma\,;\,\Delta \vdash s\,t:\beta}\ (\textsc{coer})$$

The above rules are necessary because of the presence of type variables and coercion variables. In particular, the type formation rules (see Rule (TVAR)) ensure that all the type variables occurring in a type are declared in the coercion environment, and that they may eventually be bound by using Rule (GEN). Similarly, the coercion rules (see Rule (CVAR)) ensure that all the coercion variables occurring in a term are declared, and that they may eventually be bound by using Rule (GEN). In order to ensure these properties, the above rules obey the following conventions:

– In the typing-environment formation rules, in the type formation rules, in the coercion rules, and in the typing rules, all the coercion environments that are used are implicitly considered to be well-formed.
– In the typing rules, all the typing environments that are used are implicitly considered to be well-formed.
– In the coercion rules and in the typing rules, all the types that are used are implicitly considered to be well-formed.

Let us now illustrate the way this type system works by presenting a few derivations. Consider the following coercion environment, and typing environments:

$$\begin{aligned}
\Gamma &\equiv c : a \le \mathsf{e}\\
\Delta_1 &\equiv p : a \to \mathsf{t}\\
\Delta_2 &\equiv p : a \to \mathsf{t}, q : a \to \mathsf{t}\\
\Delta_3 &\equiv p : a \to \mathsf{t}, q : a \to \mathsf{t}, x : a
\end{aligned}$$

Let us start by establishing that Γ is a well-formed coercion environment:

$$\frac{\dfrac{}{\varnothing\ \mathbf{Cenv}}\ (\textsc{Cempty}) \quad \dfrac{\mathsf{e} \in \mathcal{A}}{\vdash \mathsf{e}\ \mathbf{type}}\ (\textsc{Tconst})}{c : a \le \mathsf{e}\ \mathbf{Cenv}}\ (\textsc{Ccons})$$

Similarly, we can show that Δ_1, Δ_2, and Δ_3 are well-formed typing environments:

$$\frac{\dfrac{}{\Gamma \vdash \varnothing\ \mathbf{env}}\ (\textsc{empty}) \quad \dfrac{\dfrac{}{\Gamma \vdash a\ \mathbf{type}}\ (\textsc{Tvar}) \quad \dfrac{\mathsf{t} \in \mathcal{A}}{\Gamma \vdash \mathsf{t}\ \mathbf{type}}\ (\textsc{Tconst})}{\Gamma \vdash (a \to \mathsf{t})\ \mathbf{type}}\ (\textsc{Tfun})}{\Gamma \vdash \Delta_1\ \mathbf{env}}\ (\textsc{cons})$$

$$\frac{\Gamma \vdash \Delta_1\ \mathbf{env} \quad \dfrac{\dfrac{}{\Gamma \vdash a\ \mathbf{type}}\ (\textsc{Tvar}) \quad \dfrac{\mathsf{t} \in \mathcal{A}}{\Gamma \vdash \mathsf{t}\ \mathbf{type}}\ (\textsc{Tconst})}{\Gamma \vdash (a \to \mathsf{t})\ \mathbf{type}}\ (\textsc{Tfun})}{\Gamma \vdash \Delta_2\ \mathbf{env}}\ (\textsc{cons})$$

$$\dfrac{\Gamma \vdash \Delta_2\ \mathbf{env} \quad \dfrac{}{\Gamma \vdash a\ \mathbf{type}}\,(\textsc{Tvar})}{\Gamma \vdash \Delta_3\ \mathbf{env}}\,(\textsc{cons})$$

Finally, let us show how the term $\lambda pqx.\,(p\,x) \wedge (q\,x)$ may be assigned a polymorphic type:

$$\left.\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma\,;\,\Delta_1 \vdash p : a \to \mathsf{t}}\,(\textsc{var}) \quad q \neq p}{\Gamma\,;\,\Delta_2 \vdash p : a \to \mathsf{t}}\,(\textsc{weak}) \quad x \neq p}{\Gamma\,;\,\Delta_3 \vdash p : a \to \mathsf{t}}\,(\textsc{weak}) \quad \dfrac{}{\Gamma\,;\,\Delta_3 \vdash x : a}\,(\textsc{var})}{\Gamma\,;\,\Delta_3 \vdash p\,x : \mathsf{t}}\,(\textsc{app})\right\}(1)$$

$$\left.\dfrac{\dfrac{\dfrac{}{\Gamma\,;\,\Delta_2 \vdash q : a \to \mathsf{t}}\,(\textsc{var}) \quad x \neq q}{\Gamma\,;\,\Delta_3 \vdash q : a \to \mathsf{t}}\,(\textsc{weak}) \quad \dfrac{}{\Gamma\,;\,\Delta_3 \vdash x : a}\,(\textsc{var})}{\Gamma\,;\,\Delta_3 \vdash q\,x : \mathsf{t}}\,(\textsc{app})\right\}(2)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma\,;\,\Delta_3 \vdash \mathbf{and} : \mathsf{t} \to \mathsf{t} \to \mathsf{t} \quad \left.\begin{matrix}\vdots\\ \Gamma\,;\,\Delta_3 \vdash p\,x : \mathsf{t}\end{matrix}\right\}(1)}{\Gamma\,;\,\Delta_3 \vdash \mathbf{and}\,(p\,x) : \mathsf{t} \to \mathsf{t}}\,(\textsc{app}) \quad \left.\begin{matrix}\vdots\\ \Gamma\,;\,\Delta_3 \vdash q\,x : \mathsf{t}\end{matrix}\right\}(2)}{\Gamma\,;\,\Delta_3 \vdash \mathbf{and}\,(p\,x)\,(q\,x) : \mathsf{t}}\,(\textsc{app})}{\Gamma\,;\,\Delta_2 \vdash \lambda x.\,\mathbf{and}\,(p\,x)\,(q\,x) : a \to \mathsf{t}}\,(\textsc{abs})}{\Gamma\,;\,\Delta_1 \vdash \lambda qx.\,\mathbf{and}\,(p\,x)\,(q\,x) : (a \to \mathsf{t}) \to a \to \mathsf{t}}\,(\textsc{abs})}{\Gamma\,; \vdash \lambda pqx.\,\mathbf{and}\,(p\,x)\,(q\,x) : (a \to \mathsf{t}) \to (a \to \mathsf{t}) \to a \to \mathsf{t}}\,(\textsc{abs})}{\vdash \Lambda c.\,\lambda pqx.\,\mathbf{and}\,(p\,x)\,(q\,x) : \forall a \leq \mathsf{e}.\,(a \to \mathsf{t}) \to (a \to \mathsf{t}) \to a \to \mathsf{t}}\,(\textsc{gen})$$

This derivation allows us to assign to BUT the expected type scheme:

$$\textsc{but} := \Lambda c.\,\lambda pqx.\,(p\,x) \wedge (q\,x) : \forall a \leq e.\,(a \to \mathsf{t}) \to (a \to \mathsf{t}) \to a \to \mathsf{t}$$

Then, book being a subtype of e, one may instantiate the above type scheme as follows:

$$\textsc{but} := \lambda pqx.\,(p\,x) \wedge (q\,x) : (\mathsf{book} \to \mathsf{t}) \to (\mathsf{book} \to \mathsf{t}) \to \mathsf{book} \to \mathsf{t} \tag{6}$$

One may now apply (5) to (4) and (3), which yields the following equation (after β-reduction):

$$\begin{aligned}&\textsc{but voluminous interesting} =\\ &\qquad \lambda r.\,(\mathbf{voluminous}\,(r_\bullet\,\mathit{object})) \wedge (\mathbf{interesting}\,(r_\bullet\,\mathit{content})) : \mathsf{book} \to \mathsf{t}\end{aligned}$$

Bounded polymorphism is also used to provide a semantics to the indefinite determiner "*a*", or to the relative pronoun "*which*". We give in appendix a complete grammar that allows example (1) to be treated.

4 Coercion inference

An attentive reader might argue that coercions are just a particular case of λ-terms and that they are not needed. For instance, the derivation given at the end of

Section 2 can be replaced by a derivation that uses only a trivial coercion:

$$\dfrac{r:\mathsf{book}\vdash \mathbf{voluminous}:\mathsf{p}\to\mathsf{t} \qquad \dfrac{\dfrac{\dfrac{}{r:\mathsf{book}\vdash r:\mathsf{book}}\,(\textsc{var})}{r:\mathsf{book}\vdash r\bullet \mathit{object}:\mathsf{a}}\,(\textsc{sel}) \quad \dfrac{(\mathsf{a},\mathsf{p})\in\mathcal{O}}{\vdash \lambda x.\,x:\mathsf{a}\le\mathsf{p}}\,(\textsc{Cconst})}{r:\mathsf{book}\vdash r\bullet \mathit{object}:\mathsf{p}}\,(\textsc{coer})}{\dfrac{r:\mathsf{book}\vdash \mathbf{voluminous}\,(r\bullet \mathit{object}):\mathsf{e}}{\vdash \lambda r.\,\mathbf{voluminous}\,(r\bullet \mathit{object}):\mathsf{book}\to\mathsf{e}}\,(\textsc{abs})}\,(\textsc{app})$$

It is indeed a fact that every coercion $s : \alpha \leq \beta$ may be shown to be a λ-term of type $(\alpha \to \beta)$. But this is not a defect of the system. Indeed, explicite coercions have not been introduced in order to increase the power of the calculus. They have been introduced them with the purpose of inferring them automatically. This can be done by deriving judgements of the form:

$$\varDelta \vdash t : \alpha \ \mid\ C$$

where $\varDelta$ is a typing environment, t is a λ-term, α is a type, and C is a set of collected constraints to be solved. The inference rules are the following ones.

$$\frac{}{\vdash t : \alpha \ \mid\ \varnothing}\ (\textsc{start})$$

$$\frac{\vdash \Lambda c.\,t : \forall a \leq \alpha.\,\beta \ \mid\ C}{\vdash t : \beta \ \mid\ C \cup \{c : a \leq \alpha\}}\ (\textsc{i-inst})$$

$$\frac{}{\varDelta_1, x : \alpha_1, \varDelta_2 \vdash c\,x : \alpha_2 \ \mid\ \{c : \alpha_1 \leq \alpha_2\}}\ (\textsc{i-var})$$

$$\frac{\varDelta, x : \alpha \vdash t : \beta \ \mid\ C}{\varDelta \vdash \lambda x.\,t : \alpha \to \beta \ \mid\ C}\ (\textsc{i-abs})$$

$$\frac{\varDelta_1 \vdash t : \alpha_1 \to \beta \ \mid\ C_1 \qquad \varDelta_2 \vdash u : \alpha_2 \ \mid\ C_2}{\varDelta_1, \varDelta_2 \vdash t\,(c\,u) : \beta \ \mid\ C_1 \cup C_2 \cup \{c : \alpha_2 \leq \alpha_1\}}\ (\textsc{i-app})$$

$$\frac{\varDelta \vdash t_1 : \alpha_1 \ \mid\ C_1 \quad \cdots \quad \varDelta \vdash t_n : \alpha_n \ \mid\ C_n}{\varDelta \vdash \left\{\overline{l_i = t_i}\right\}_n : \left\{\overline{l_i : \alpha_i}\right\}_n \ \mid\ \bigcup_{i=1}^{n} C_i}\ (\textsc{i-rec})$$

$$\frac{\varDelta \vdash t : \left\{\overline{l_i : \alpha_i}\right\}_n \ \mid\ C}{\varDelta \vdash t\bullet l_i :\ \alpha_i \ \mid\ C}\ (\textsc{i-sel})$$

The above typing system relies on a few assumptions that need to be made explicit. First, the typing environments that are used are finite sets of declarations such that every declared variable is declared only once. Consequently, in a rule such as (i-app), $\varDelta_1$ and $\varDelta_2$ must be compatible in the sense such that every variable which is declared in both environments must be declared with the same type in both environments. In addition, the coercion variables that are introduced by Rules (i-inst), (i-var), (i-app) must be fresh. Similarly, the type variables that are introduced by Rule (i-inst), and possibly by Rule (i-var), must also be fresh. This means that when taking the union of sets of constraints (in Rules (i-app) or (i-rec)) the set of coercion variables and type variables occurring in these sets of constraints must be disjoint. All these assumptions are not constraining because they may be easily satisfied by variable renaming without any loss of generality.

Let us now illustrate how the above rules can be used to compute the semantic representation of the phrase

$$\textit{a book which is voluminous but interesting} \tag{7}$$

The abstract syntax of this phrase (see the grammar given in appendix) is expressed by the following term:

$$\textsc{some}\,(\textsc{which}\,(\textsc{is}\,(\textsc{but voluminous interesting}))\,\textsc{book}) \tag{8}$$

Let us start with the subterm ($\textsc{but voluminous interesting}$) that we abbreviate as ($\textsc{b}\,\textsc{v}\,\textsc{i}$) for the sake of conciseness.

$$\dfrac{\dfrac{\dfrac{\overline{\vdash \textsc{b} : \forall a \leq \mathsf{e}.\,(a \to \mathsf{t}) \to (a \to \mathsf{t}) \to a \to \mathsf{t} \;\mid\; \varnothing}}{\vdash \textsc{b} : (a_1 \to \mathsf{t}) \to (a_1 \to \mathsf{t}) \to a_1 \to \mathsf{t} \;\mid\; C_1} \quad \overline{\vdash \textsc{v} : \mathsf{p} \to \mathsf{t} \;\mid\; \varnothing}}{\vdash \textsc{b}\,(c_2\,\textsc{v}) : (a_1 \to \mathsf{t}) \to a_1 \to \mathsf{t} \;\mid\; C_2} \quad \overline{\vdash \textsc{i} : \mathsf{i} \to \mathsf{t} \;\mid\; \varnothing}}{\vdash \textsc{b}\;(c_2\,\textsc{v}\,)\,(c_3\,\textsc{i}) : a_1 \to \mathsf{t} \;\mid\; C_3}$$

where

$$\begin{aligned}
C_1 &= \{c_1 : a_1 \leq \mathsf{e}\}\\
C_2 &= C_1 \cup \{c_2 : (\mathsf{p} \to \mathsf{t}) \leq (a_1 \to \mathsf{t})\}\\
C_3 &= C_2 \cup \{c_3 : (\mathsf{i} \to \mathsf{t}) \leq (a_1 \to \mathsf{t})\}
\end{aligned}$$

By continuing to apply this typing process to term (8), we end up with the following typing judgement:

$$\vdash \textsc{s}\,(c_{10}\,(\textsc{w}\,(c_7\,(\textsc{is}\,(c_5\,(\textsc{b}\,(c_2\,\textsc{v})\,(c_3\,\textsc{i})))))\,(c_8\,\textsc{bk}))) : (a_4 \to \mathsf{t}) \to \mathsf{t} \;\mid\; C \tag{9}$$

where

$$\begin{aligned}
C = \{\; & c_1 : a_1 \leq \mathsf{e},\\
& c_2 : (\mathsf{p} \to \mathsf{t}) \leq (a_1 \to \mathsf{t}),\\
& c_3 : (\mathsf{i} \to \mathsf{t}) \leq (a_1 \to \mathsf{t}),\\
& c_4 : a_2 \leq \mathsf{e},\\
& c_5 : (a_1 \to \mathsf{t}) \leq (a_2 \to \mathsf{t}),\\
& c_6 : a_3 \leq \mathsf{e},\\
& c_7 : (((a_2 \to \mathsf{t}) \to \mathsf{t}) \to \mathsf{t}) \leq (((a_3 \to \mathsf{t}) \to \mathsf{t}) \to \mathsf{t}),\\
& c_8 : (\mathsf{book} \to \mathsf{t}) \leq (a_3 \to \mathsf{t}),\\
& c_9 : a_4 \leq \mathsf{e},\\
& c_{10} : (a_3 \to \mathsf{t}) \leq (a_4 \to \mathsf{t})\}
\end{aligned}$$

In order to infer the coerced type of term (8), we must solve the above constraints. A solution to such a set of constraints C consists of two substitutions ρ and σ such that:

1. ρ is a substitution that assigns a closed coercion to each coercion variable occurring in C;
2. σ is a substitution that assigns a ground type to each type variable occurring in C;
3. for every constraint $(c : \alpha \leq \beta) \in C$ the coercion judgement:

   $$\vdash \rho(c) : \sigma(\alpha) \leq \sigma(\beta)$$

   is derivable.

In the present case, one easily checks that the following pair of substitutions provides a solution to C:

$$
\begin{array}{rl@{\qquad\qquad}l}
c_1 \leftarrow & \lambda x.\, x & a_1 \leftarrow \mathsf{book} \\
c_2 \leftarrow & \lambda f r.\, f\,(r_\bullet\, \mathit{object}) & a_2 \leftarrow \mathsf{book} \\
c_3 \leftarrow & \lambda f r.\, f\,(r_\bullet\, \mathit{content}) & a_3 \leftarrow \mathsf{book} \\
c_4 \leftarrow & \lambda x.\, x & a_4 \leftarrow \mathsf{book} \\
c_5 \leftarrow & \lambda f x.\, f\, x & \\
c_6 \leftarrow & \lambda x.\, x & \\
c_7 \leftarrow & \lambda f g.\, f\,(\lambda h.\, g\,(\lambda x.\, h\, x)) & \\
c_8 \leftarrow & \lambda f x.\, f\, x & \\
c_9 \leftarrow & \lambda x.\, x & \\
c_{10} \leftarrow & \lambda f x.\, f\, x &
\end{array}
$$

Applying these substitutions to the term obtained in (9) yields the following term:

$$
\begin{array}{l}
\textsc{some} \\
\quad (\textsc{which} \\
\quad\quad (\textsc{is}\,(\textsc{but}\,(\lambda r.\,\textsc{voluminous}\,(r_\bullet\,\mathit{object}))\,(\lambda r.\,\textsc{interesting}\,(r_\bullet\,\mathit{content})))) \\
\quad\quad \textsc{book})
\end{array}
\tag{10}
$$

which is of type $(\mathsf{book} \to \mathsf{t}) \to \mathsf{t}$.

Finally, by replacing in (10) each defined constant by its definition (see the grammar given in appendix) and then β-reducing the resulting term, we obtained the following semantic interpretation for phrase (7):

$$\lambda q.\, \exists x.\, (\mathbf{book}\, x) \wedge (\mathbf{voluminous}\,(x_\bullet\,\mathit{object})) \wedge (\mathbf{interesting}\,(x_\bullet\,\mathit{content})) \wedge (q\, x)$$

5 Decidability and canonicity

In spirit, our proposal is similar to the approach presented in [15]. The formal framework we use, however, is rather different. In fact, it is weaker because it relies only on a fragment of second-order λ-calculus. The reason for this choice is that we want the operations of selection and coercion to be performed automatically. To achieve this aim, two key properties are needed:

- the decidability of constraint solving;
- the canonicity of the computed coercions.

The first property ensures that the coercion functions may be computed automatically. The second one guarantees that selectional restriction by type coercion does not result in erroneous interpretations, or in fallacious semantic ambiguities.

Decidability does not present any real difficulty. A constraint system can be solved by reducing it to a closed system of atomic constraints [7]. This can be done by applying type decomposition rules. Typically, in the case of a functional type, the decomposition rules are as follows:

$$c : (\alpha_1 \to \alpha_2) \leq (\beta_1 \to \beta_2) \;\longrightarrow\; \left\{ \begin{array}{l} c_1 : \beta_1 \leq \alpha_1 \\ c_2 : \alpha_2 \leq \beta_2 \\ \hline c \leftarrow \lambda f x.\, c_2\,(f\,(c_1\, x)) \end{array} \right.$$

Remark that the above rule does not only decompose the constraint. It also contributes to the construction of the solution by generating a substitution. Then, decidability of constraint solving relies on the decidability of the subtyping relation (which is not difficult to establish in our case).

Canonicity, on the other hand, is more problematic. The simplest form of canonicity would be ensured by a unicity property: given two types α and β such that $\alpha \leq \beta$, there is a unique coercion function such that $f : \alpha \leq \beta$. This property however, does not hold because of inequation (2). There are several ways out of this failure. For instance, one could restrict inequation (2) to the case of atomic types (different from e) , and impose that any two atomic types occurring in a record type do not have any supertype but e. Remark that such a solution constrains the structure of the lexicon. To our mind, this is not surprising because we think that canonicity should not be a property of the underlying mathematical framework, but must derive from the organisation of the lexicon. In particular, the solutions that are proposed in [15] are adaptable to our framework. Remark also that the unicity of the coercion is not always a desirable property. For instance, a sentence such as "*Mary began a novel*" presumably means that Mary began reading a novel, but in case Mary is a writer, it might means that Mary began writing a novel.

A final note should be added on the adaptability of our proposal to creative uses of language as illustrated in the case of metonymy or metaphor. A well-known example of such a situation is the possible use of "*the sandwich*" to mean "*the customer who ordered a sandwich*" in the context of a restaurant. It is doubtful that one may want to encode the possibility of such a use directly into the lexicon, since only a few sandwiches may have an obvious relation to some customer. But our system of inference is able, to some extent, to account for such phenomena in an explicit way: in the typing judgement of a term involving polymorphism, the type mismatch between food and customer is likely to arise in the form of constraints reducible to some $c_1 : a \leq \mathsf{food}$ and $c_2 : a \leq \mathsf{customer}$. Then, even if no type satisfying those constraints is to be found in our grammar, the typing-environment formation rules allows for the introduction of a new record type capturing both aspects, which appears in the term as a new variable encoding a relation between the food and the customer. The very nature of this relation will not be explained by the system, but it leaves open the way for inference by subsequent analyses using world knowledge and context knowledge. Moreover, if we apply the restrictions on record types described above, the new record type introduced can be uniquely defined as the greatest record type which satisfies the constraints.

However a few cases, especially when no polymorphic term is involved, may not be covered by such an analysis, and will require some additional treatment. Yet the examples described in this paper show that the use of records and record types can be adapted to account for restrictive selection and type coercion similarly to existing proposal, while enabling dynamic inference of types and coercion where needed.

References

1. N. Asher. *Lexical Meaning in Context: A Web of Words.* Cambridge University Press, 2011.
2. William Babonnaud. A topos-based approach to building language ontologies. In Raffaella Bernardi, Greg Kobele, and Sylvain Pogodalla, editors, *Formal Grammar - 24th International Conference, FG 2019, Riga, Latvia, August 11, 2019, Proceedings*, volume 11668 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2019.
3. L. Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76:138–164, 1988.
4. L. Cardelli, S. Martini, J.C. Mitchell, and A. Scedrov. An extension of system f with subtyping. *Information and Computation*, 109(1):4 – 56, 1994.
5. L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–523, December 1985.

6. R. Cooper. Records and Record Types in Semantic Theory. *Journal of Logic and Computation*, 15(2):99–112, 2005.
7. J. Eifrig, S. Smith, and V. Trifonov. Type inference for recursively constrained types and its application to oop. *Electronic Notes in Theoretical Computer Science*, 1:132 – 153, 1995. MFPS XI, Mathematical Foundations of Programming Semantics, Eleventh Annual Conference.
8. J-Y. Girard. Une extension de l'interprétation de gödel à l'analyse et son application à l'élimination des coupures dans l'analyse et la théorie des types. In *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 63–92. North-Holland, 1971.
9. J.-Y. Girard. The system F of variable types, fifteen years later. *Theoretical Computer Science*, 45:159–192, 1986.
10. Z. Luo, S. Soloviev, and T. Xue. Coercive subtyping: Theory and implementation. *Information and Computation*, 223:18–42, 2013.
11. R. Montague. The proper treatment of quantification in ordinary english. In J. Hintikka, J. Moravcsik, and P. Suppes, editors, *Approaches to natural language: proceedings of the 1970 Stanford workshop on Grammar and Semantics*, Dordrecht, 1973. Reidel. Reprinted: [12, pages 247–270].
12. R. Montague. *Formal Philosophy: selected papers of Richard Montague, edited and with an introduction by Richmond Thomason.* Yale University Press, 1974.
13. J. Pustejovsky. Type coercion and lexical selection. In J. Pustejovsky, editor, *Semantics and the Lexicon*, volume 49 of *Studies in Linguistics and Philosophy.* Springer, Dordrecht, 1993.
14. J. Pustejovsky. *The Generative Lexicon.* The MIT Press, 1995.
15. C. Retoré. The montagovian generative lexicon ΛTy_n: a type theoretical framework for natural language semantics. In Ralph Matthes and Aleksy Schubert, editors, *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 202–229. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014.

Appendix

This appendix provides a toy grammar made of an abstract syntax and its semantic interpretation.

Abstract syntax

$$\begin{aligned}
\textsc{John} &: NP \\
\textsc{book} &: N \\
\textsc{voluminous} &: Adj \\
\textsc{interesting} &: Adj \\
\textsc{is} &: Adj \to NP \to S \\
\textsc{bought} &: NP \to NP \to S \\
\textsc{some} &: N \to NP \\
\textsc{but} &: Adj \to Adj \to Adj \\
\textsc{which} &: (NP \to S) \to N \to N
\end{aligned}$$

Semantic interpretation

1 Atomic types

t	(*truth value*)
e	(*entities*)
h	(*human beings*)
i	(*informational contents*)
c	(*commodities*)
p	(*physical object*)
a	(*artifacts*)

$$\begin{tikzcd}
 & & & \mathsf{e} \arrow[dlll, dash] \arrow[dl, dash] \arrow[dr, dash] \arrow[drrr, dash] & & & \\
\mathsf{h} & & \mathsf{i} & & \mathsf{c} \arrow[dr, dash] & & \mathsf{p} \arrow[dl, dash] \\
 & & & & & \mathsf{a} &
\end{tikzcd}$$

2 Non logical constants

$$\begin{aligned}
\textbf{book} &: \mathsf{book} \to \mathsf{t}, \text{ where } \mathsf{book} = \{\mathit{object} : \mathsf{a}\,;\mathit{content} : \mathsf{i}\}\\
\textbf{buy} &: \mathsf{h} \to \mathsf{c} \to \mathsf{t}\\
\textbf{interesting} &: \mathsf{i} \to \mathsf{t}\\
\mathbf{j} &: \mathsf{h}\\
\textbf{voluminous} &: \mathsf{p} \to \mathsf{t}
\end{aligned}$$

3 Semantic interpretation

$$\begin{aligned}
\textsc{John} &:= \lambda k.\, k\,\mathbf{j} : (\mathsf{h} \to \mathsf{t}) \to \mathsf{t}\\
\textsc{book} &:= \lambda x.\, \textbf{book}\, x : \mathsf{book} \to \mathsf{t}\\
\textsc{voluminous} &:= \lambda x.\, \textbf{voluminous}\, x : \mathsf{p} \to \mathsf{t}\\
\textsc{interesting} &:= \lambda x.\, \textbf{interesting}\, x : \mathsf{i} \to \mathsf{t}\\
\textsc{is} &:= \Lambda c.\, \lambda es.\, s\, e : \forall a \leq \mathsf{e}.\, (a \to \mathsf{t}) \to ((a \to \mathsf{t}) \to \mathsf{t}) \to \mathsf{t}\\
\textsc{bought} &:= \lambda os.\, s\, (\lambda x.\, o\, (\lambda y.\, \mathsf{P}\, (\textbf{buy}\, x\, y))) :\\
&\qquad ((\mathsf{c} \to \mathsf{t}) \to \mathsf{t}) \to ((\mathsf{h} \to \mathsf{t}) \to \mathsf{t}) \to \mathsf{t}\\
\textsc{some} &:= \Lambda c.\, \lambda pq.\, \exists x.\, (p\, x) \wedge (q\, x) : \forall a \leq \mathsf{e}.\, (a \to \mathsf{t}) \to (a \to \mathsf{t}) \to \mathsf{t}\\
\textsc{but} &:= \Lambda c.\, \lambda pqx.\, (p\, x) \wedge (q\, x) : \forall a \leq \mathsf{e}.\, (a \to \mathsf{t}) \to (a \to \mathsf{t}) \to a \to \mathsf{t}\\
\textsc{which} &:= \Lambda c.\, \lambda rnx.\, (n\, x) \wedge (r\, (\lambda k.\, k\, x)) :\\
&\qquad \forall a \leq \mathsf{e}.\, (((a \to \mathsf{t}) \to \mathsf{t}) \to \mathsf{t}) \to (a \to \mathsf{t}) \to a \to \mathsf{t}
\end{aligned}$$