Using Type Theory as a Language for Negotiation Objects in Online Exchanges

Rod Moten

July 26, 2002

In this paper, we present LpX, a type theory for agents to specify negotiation objects in an online exchange. An online exchange is a multi–agent system in which agents buy and sell. We have created LpX to study the use of type theory in resolving semantic heterogeneity. By using type theory we can formulate the entire process of resolving semantic heterogeneity in a single framework. Resolving semantic heterogeneity involves merging independently created ontologies into a single ontology and performing translation between schemas. Using an expressive type theory, we can use a collection of types as an explication of an ontology as well as represent schemas. As a result, we can use proof rules of the type system to relate independently created ontologies and use implicit coercion to perform schema translation between related types.

1 Introduction

An aspect of agent–mediated electronic commerce involves implementing electronic marketplaces as a multi–agent systems. An *online exchange* is a type of these marketplaces. In an online exchange, agents negotiate contracts for requiring some goods or services. Agents negotiate with each other by transmitting negotiation objects to each other. The negotiation objects encode the issues of a contract that agents use to make their decision to accept or reject a proposal for a contract. Each agent uses it own *local language* for formulating negotiation objects. A local language is a subset of terms of the language the online exchange requires agents to encode negotiation objects. Semantic heterogeneity occurs when two agents use different terms to have the same meaning.

Effective solutions for resolving semantic heterogeneity is important for the growth of agent–mediated electronic commerce. Some vendors will not sell their products at agent–mediated electronic marketplaces if price is the only attribute agents use to base their decision to purchase a product [1, 4, 5]. Price wars—an unfavorable position for vendors whose competitive advantage is quality—will less likely occur when attributes other than price can be used in negotiations [8, 6]. However, semantic heterogeneity is a major obstacle for allowing online exchanges to permit agents to negotiate on attributes of a good or service other than price [2].

In this paper, we demonstrate how to use type theory for completely formalizing a solution for resolving semantic heterogeneity. In particular, we present a type theory called LpX. LpX is intended to be a meta–language for formulating negotiation objects in an online exchange that supports one–to–one or one–to–many negotiations. A local language is a type. Semantic heterogeneity is resolved using coercion between types related by LpX's subtyping relation. Using this approach assumes that the types representing local languages are defined in such a manner so that conceptual relationships can be deduced from the types.

1.1 Outline

This paper is organized as follows. In Section 2 we present LpX, the type theory we use for resolving semantic heterogeneity. In Section 3, we describe the roles polymorphism, subtyping, and coercion play in resolving semantic heterogeneity. In Section 4, we demonstrate by example how semantic heterogeneity is resolved during negotiations in an online exchange that uses LpX. In Section 5, we conclude the paper and briefly describe related work. We assume the reader is familiar with category theory on an elementary level and with type theory.

2 LpX

In this section, we give an overview of LpX. LpX is a strongly–typed language for agents to represent offers. The terms of LpX are constructed from a countable set of atom symbols $\mathcal{A}$, a countable set of type constant symbols $\mathcal{C}$, and a countable set of label symbols Ł. We assume that $\mathcal{A}$ contains the symbol $\perp$. Also, the terms of LpX are constructed from a countable disjoint sets of *label variables* and type variables. $\mathcal{A}$, $\mathcal{C}$, Ł, the label variables, and type variables are pairwise disjoint.

The terms of LpX are partitioned into two disjoint sets, types and objects. The syntax of types is represented as the following grammar.

Kinds	$K ::=$	$\textsc{type} \mid \{T_1, \ldots, T_n\}$
Label types	$L ::=$	$\textsc{label} \mid \{l_1, \ldots, l_n\}$
Types	$t ::$	$\pi x : L \,.\, t \mid \Lambda X : K \,.\, t \mid T$
Object types	$T ::=$	$c \mid X \mid r_1 : T_1 \times \cdots \times r_n : T_n \mid T_1 \times \cdots \times T_n \mid (T)$
Label expressions	$r ::=$	$x \mid l$

In the above grammar, x ranges over label variables, X ranges over type variables, c ranges over $\mathcal{C}$, and l ranges over Ł. For all occurrences of n, $n \geq 1$. We call $\{l_1, \ldots, l_n\}$ an *enumerated label type* and $\{T_1, \ldots, T_n\}$ an *enumerated kind.* Intuitively, LABEL represents the label type of all labels and TYPE is the kind of all ground object types. The object type $r_1 : T_1 \times \cdots \times r_n : T_n$ denotes a record type. Notice that the labels in a record type can be variables, a feature not present in any other type theory we are aware of. We do not require the labels

$$\frac{\Phi \vdash_\Sigma c}{\vdash_{(\Phi;\Psi)} a^c : c} \qquad \frac{\vdash_\Sigma e_1 : T_1, \ldots, \vdash_\Sigma e_n : T_n}{\vdash_\Sigma (e_1, \ldots, e_n) : T_1 \times \cdots \times T_n}$$

$$\frac{\vdash_\Sigma e_1 : T_1 \ \cdots \ \vdash_\Sigma e_n : T_n}{\vdash_\Sigma (l_1 = e_1, \ldots, l_n = e_n) : l_1 : T_1 \times \cdots \times l_n : T_n}$$

Figure 1: Rules for Typing Objects

in a record type to be distinct so that we can represent offers in *combinatorial auctions* [3, 7]. In combinatorial auctions, agents may negotiate on the sell or purchase of bundles of the same good or service. Therefore, LpX needs the capability to allow offers to have multiple attributes that are the same.

The scope of the binders π and Λ is the same as λ, $\forall$, and $\exists$. However, π and Λ can bind variables in enumerated kinds. For example, y and Y are free in $\Lambda X : y : Y . X$, but they are bound in $\pi y : \textsc{label} . \Lambda Y : \textsc{type} . \Lambda X : y : Y . X$. A type is closed if all variable occurrences are bound; otherwise its open. A ground type is an object type with no variables. Because a Λ binder does not bind anything in a label type, we assume for the rest of the paper all types are written as

$$\pi x_1 : L_1 . \cdots . \pi x_n : L_n . \Lambda X_1 : K_1 . \cdots . \Lambda X_m : K_m . T$$

where T is an object type and $m, n \geq 0$.

The syntax of objects is represented as the following grammar rule.

$$e ::= \quad a^c \mid (e_1, \ldots, e_n) \mid (l_1 = e_1, \ldots l_n = e_n)$$

In the above rule, a ranges over $\mathcal{A}$, c ranges over $\mathcal{C}$, and each l_i ranges over Ł. For all occurrences of n, $n \geq 1$. We call the objects of the form a^c *atoms*.

In addition to types and objects LpX contains *signatures*. Due to space limitations, we omit defining a signature formally, but define them informally as follows. A signature is a pair (Φ, Ψ) where Φ is a non-empty set of type constant symbols and Ψ is a non-reflexive transitive relation between type constants in Φ. We denote $(c, c') \in \Psi$ as $c \rightarrow c'$ and say c is *reachable* from c'.

We type objects with respect to a signature. The rules for typing records and tuples are similar to the typing rules pervasive in type theory. In LpX, however, the order of fields in record types is significant. We give the type rules in Figure 1. It is easy to see from our rules that each object has a unique type.

An interpretation of a signature is a category $(\mathcal{S}, \mathcal{F})$, where $\mathcal{S}$ is the objects and $\mathcal{F}$ is the morphisms, and a valuation $\delta : \mathcal{C} \rightarrow \mathcal{S}$. $\mathcal{S}$ consists of subsets of $\mathcal{A}$. $\mathcal{F}$ is a set of *coercion embeddings* on subsets in $\mathcal{A}$. A coercion embedding is a function $f : A \rightarrow B$ such that f is an injection and given some equivalence relation $\approx$ on A and equivalence $\sim$ on B, if $a \approx a'$ then $f(a) \sim f(a')$ $\forall a, a' \in A$. An interpretation $\mathcal{I} = ((\mathcal{S}, \mathcal{F}), \delta)$ satisfies a signature $\Sigma = (\Phi, \Psi)$ if for each $c \in \Phi$, $\delta(c) \in \mathcal{S}$. Also if $c \rightarrow c'$, there is a morphism $f : \delta(c) \rightarrow \delta(c') \in \mathcal{F}$ and there is a morphism $g : \delta(c') \rightarrow \delta(c) \in \mathcal{F}$.

3 Resolving Data Heterogeneity Using LpX

We use polymorphism and subtyping to relate local languages. Once we have determined two local languages are related, we use coercion to convert objects of one local language to objects of another local language.

The terms of a local language consist of tuples and records. Therefore, semantic heterogeneity occurs when an object e of a local language β does not equal an object e' of a local language γ, but e and e' are semantically the same. Since e and e' are semantically the same, then e and e' encode the same attributes of the same product on which the agents negotiate. However, they may be unequal because of one or more of the following reasons.

1. e or e' may have some additional fields that do not correspond to any attribute in the product.

2. The fields can be ordered differently.

3. Either e or e' may be a record and the other is a tuple.

4. e and e' can use different names for the same attribute.

5. The type used for the value of the same attribute may be different.

We can resolve some of the difference between e and e' caused by 1, 2 and 3 by using subtyping, a topic of Section 3.1. We can resolve some of the differences between e and e' caused by 4 and 5 by using polymorphic types.

By using polymorphic types, an agent developer can specify a *synonym set*—a range of equivalent names—for an attribute and a range of types that can be used as the value of an attribute. For example, suppose an agent developer wanted to specify that the names wt and weight refer to the weight attribute. He can express this relationship as the polymorphic type

$$\pi w : \{\mathsf{weight}, \mathsf{wt}\} \,.\, w : \mathsf{g} \times \mathsf{price} : \mathsf{usd}.$$

An agent developer can specify that the values of the weight attribute can be either grams or ounces as the type

$$\Lambda W : \{\mathsf{g}, \mathsf{ozs}\} \,.\, \mathsf{weight} : W \times \mathsf{price} : \mathsf{usd}.$$

An agent developer can specify that any name could be used for the weight attribute as long as the type of the value is g or ozs as the type

$$\pi : \textsc{label} \,.\, \Lambda W : \{\mathsf{g}, \mathsf{ozs}\} \,.\, w : W \times \mathsf{price} : \mathsf{usd}.$$

An agent developer can specify that any type could be used for the value of the identifier attribute as the type

$$\Lambda D : \textsc{type} \,.\, (\mathsf{id{:}D}) \times (\mathsf{weight{:}g} \times \mathsf{price{:}usd}).$$

The need for such a type may arise if one agent requires a unique identifier to accompany each offer, but an agent negotiating with it ignores the identifier.

$$\frac{}{\vdash_\Sigma T \leq T} \qquad\qquad \frac{c \rightarrow c' \in \Psi}{\vdash_\Sigma c \leq c'} \text{ where } \Sigma = (\Phi, \Psi)$$

$$\frac{\begin{array}{c} n \geq m \quad l_1 = l_i \quad \vdash_\Sigma T_1 \leq T_i' \\ \vdash_\Sigma l_2 : T_2 \times \cdots \times l_n : T_n \;\leq\; l_1' : T_1' \times \cdots \times l_{i-1}' : T_{i-1}' \times l_{i+1}' : T_{i+1}' \times \cdots \times l_n' : T_n' \end{array}}{\vdash_\Sigma l_1 : T_1 \times \cdots \times l_n : T_n \leq l_1 : T_1 \times \cdots \times l_m : T_m} \quad (\textsc{rec-rec})$$

$$\frac{n \geq m \;\; \vdash_\Sigma T_1 \leq T_1' \;\; \vdash_\Sigma l_2 : T_2 \times \cdots \times l_n : T_n \leq T_2' \times \cdots \times T_m'}{\vdash_\Sigma l_1 : T_1 \times \cdots \times l_n : T_n \leq T_1' \times \cdots \times T_m'} \quad (\textsc{rec-prod})$$

$$\frac{\vdash_\Sigma T_1 \leq T_1' \;\; \vdash_\Sigma T_2 \times \cdots \times T_n \leq T_2' \times \cdots \times T_n'}{\vdash_\Sigma T_1 \times \cdots \times T_n \leq T_1' \times \cdots \times T_n'} \quad (\textsc{prod-prod})$$

Figure 2: Subtyping Rules

3.1 Subtyping and Coercion

We resolve semantic heterogeneity between subtypes and supertypes via coercion. The subtype relation is defined on object types. We indicate T is a subtype of T with respect to a signature Σ as the judgment $\vdash_\Sigma T \leq T'$. When Σ is known we write $\vdash T \leq T'$. We use the rules in Figure 2 to prove these judgments. Subtyping between constant types is specified by a signature. The REC-REC and PROD-PROD rules in Figure 2 support standard inheritance subtyping. Notice that the REC-REC rule indicates that the order of fields is insignificant. In the rec-prod rule, on the other hand, the order of fields is significant. Noticed that subtyping is only defined on ground types.

Coercion can be performed to and from a supertype and a subtype. We give the algorithm that an exchange uses to perform coercion as pseudo ML functions in Figure 3. The functions are parameterized by a signature Σ and an interpretation $((\mathcal{S}, \mathcal{F}), \delta)$ that satisfies Σ. The functions use the morphisms of $\mathcal{F}$ to convert atoms in one type to atoms in another type. Both algorithms take as input an object e and two types T and T' where $\vdash_\Sigma e : T$. The function tosup in Figure 3(a) assumes $\vdash_\Sigma T \leq T'$ and the function tosub in Figure 3(b) assumes $\vdash_\Sigma T' \leq T$. The result is an object e' such that $\vdash_\Sigma e' : T'$. (See Theorem 3.2 below.) The function ν in the last clauses of tosup and tosub is used to permute the fields of the supertype in the same way the fields were permuted in a proof of $\vdash_\Sigma T \leq T'$ or a proof of $\vdash_\Sigma T' \leq T$. Notice in Figure 3(b) how tosub uses a bottom element as the value for fields that are in the subtype, but are not in the supertype. Each ground object type T has a unique bottom element, denoted $\perp^T$. A bottom element is an object in which all atoms in the object are of the form $\perp^c$.

Theorem 3.2 indicates that tosub and tosup actually perform coercion between two types. To show that tosub and tosup are nontrivial, requires that we show that tosub and tosup are coercion embeddings.

Our coercion algorithm is actually an encoding of a coercion embedding. To

$\text{tosup}(a^c, c, c') = \text{let } b = [\![c \to c']\!](a) \text{ in } b^{c'}$
$\text{tosup}((e_1, \ldots, e_n), T_1 \times \cdots \times T_n, T'_1 \times \cdots \times T'_n) = (\text{tosup}(e_1, T_1, T'_1), \ldots, \text{tosup}(e_n, T_n, T'_n))$
$\text{tosup}((l_1 : e_1, \ldots, l_n : e_n), l_1 : T_1 \times \cdots \times l_n : T_n, l'_1 : T'_1 \times \cdots \times l'_m : T'_m) =$
  let $\nu : \{1, \ldots, m\} \to \{1, \ldots, n\}$ be an injection such that $\vdash_\Sigma l_{\nu(i)} : T_{\nu(i)} \leq l'_i : T'_i$
  in $(l'_1 = \text{tosup}(e_{\nu(1)}, T_{\nu(1)}, T'_1), \ldots, l'_m = \text{tosup}(e_{\nu(m)}, T_{\nu(m)}, T'_m))$

(a) Converting to Supertype

$\text{tosub}(a^c, c, c') = \text{let } b = [\![c \to c']\!]^{-1}(a) \text{ in } b^{c'}$
$\text{tosub}((e_1, \ldots, e_n), (T_1 \times \cdots \times T_n), (T'_1 \times \cdots \times T'_m)) = (\text{tosub}(e_1, T_1, T'_1), \ldots, \text{tosub}(e_n, T_n, T'_n))$
$\text{tosub}((l_1 = e_1, \ldots, l_n = e_n, l_1 : T_1 \times \cdots \times l_n : T_n, l'_1 : T'_1 \times \cdots \times l'_m : T'_m) =$
  let $\nu : \{1, \ldots, n\} \to \{1, \ldots, m\}$ be an injection such that $\vdash_\Sigma l'_i : T'_i \leq l_{\nu(i)} : T_{\nu(i)}$
  let $\mathcal{E}$ be the set of all objects
  let $G : \{1, \ldots, m\} \to \mathcal{E}$ such that $G(j) = \begin{cases} \text{tosub}(e_i, T_i, T'_j) & \text{if } j = \nu(i) \text{ for some } i \\ \bot^{T_j} & \text{otherwise} \end{cases}$
  in $(l'_1 = G(1), \ldots, l'_m = G(m))$

(b) Converting to Subtype

Figure 3: Coercion Algorithm for LpX

prove this, we have to define a model of product and record types and show that our coercion algorithm behaves the same as a coercion embedding on the entities that model product and record types.

We model product types and record types as set Cartesian products and *record sets*, respectively. A record set is the set Cartesian product of a singleton set containing a tuple of labels and a set Cartesian product. To create a model for product types and record types we extend $\mathcal{S}$ of an interpretation $((\mathcal{S}, \mathcal{F}), \varepsilon)$. We denote the extension of $\mathcal{S}$ as $\mathcal{S}^+$. We define $\mathcal{S}^+$ inductively as follows. $S \subset \mathcal{S}^+$. If $S_1, \ldots, S_n \subset \mathcal{S}^+$ then $S_1 \times \cdots \times S_n \subset S^+$ and $\{(l_1, \ldots, l_n)\} \times (S_1 \times \cdots \times S_n) \subset S^+$. If $((\mathcal{S}, \mathcal{F}), \varepsilon)$ is an interpretation of Σ, then $\mathcal{S}^+[\![\cdot]\!]$ is a map from object types to sets in $\mathcal{S}^+$ and a map from objects to elements of sets in $\mathcal{S}^+$ defined as follows.

$$\begin{array}{rcl} \mathcal{S}^+[\![a^c]\!] & = & a \text{ if } \Phi \vdash c \text{ for } \Sigma = (\Phi; \Psi) \\ \mathcal{S}^+[\![(e_1, \ldots, e_n)]\!] & = & (\mathcal{S}^+[\![e_1]\!], \ldots, \mathcal{S}^+[\![e_n]\!]) \\ \mathcal{S}^+[\![(l_1 = e_1, \ldots, l_n = e_n)]\!] & = & ((l_1, \ldots, l_n), (\mathcal{S}^+[\![e_1]\!], \ldots, \mathcal{S}^+[\![e_n]\!])) \\ \mathcal{S}^+[\![c]\!] & = & \langle c \rangle \\ \mathcal{S}^+[\![T_1 \times \cdots \times T_n]\!] & = & S^+[\![T_1]\!] \times \cdots \times S^+[\![T_n]\!] \\ \mathcal{S}^+[\![l_1 : T_1 \times \cdots \times l_n : T_n]\!] & = & \{(l_1, \ldots, l_n)\} \times (\mathcal{S}^+[\![T_1]\!] \times \cdots \times \mathcal{S}^+[\![T_n]\!] \end{array} \qquad (1)$$

Notice that (1) requires that we have access to the binary relation on $\mathcal{A}$ and $\mathcal{C}$ used to create the elements of $\mathcal{S}$. When $\mathcal{S}^+$ is known we omit it from $\mathcal{S}^+[\![\cdot]\!]$.

Theorem 3.1 Let Σ be a well–formed signature and $((\mathcal{S}, \mathcal{F}), \varepsilon)$ an interpretation that satisfies Σ. Let T and T' be ground object types where $\vdash_\Sigma T : \text{TYPE}$ and $\vdash_\Sigma T' : \text{TYPE}$. Let e be an object where $\vdash_\Sigma e : T$. Suppose $\vdash_\Sigma T \leq T'$ and $\mathsf{tosup}(e, T, T') = e'$ or $\vdash_\Sigma T' \leq T$ and $\mathsf{tosub}(e, T, T') = e'$. Then there exists a concept embedding $f : [\![T]\!] \rightarrow [\![T']\!]$ such that $f([\![e]\!]) = [\![e']\!]$.

Proof: The proof is by induction on the structure of e, T, and T'.
$\square$

In the next section, we give an overview of how an online exchange makes use of LpX and properties about it to resolve semantic heterogeneity.

Theorem 3.2 Let Σ be a signature and T and T' be ground object types. Let e be an object such that $\vdash_\Sigma e : T$. If $\vdash_\Sigma T \leq T'$ and $\mathsf{tosup}(e, T, T') = e'$ then $\vdash_\Sigma e' : T'$. If $\vdash_\Sigma T' \leq T$ and $\mathsf{tosub}(e, T, T') = e'$ then $\vdash_\Sigma e' : T'$.

3.2 Subtype Unification

Subtyping deduces relationships between object types, but how do we deduce relationships between polymorphic types? We use *subtype unification* to deduce relationships between polymorphic types. Subtype unification is the process of determining the satisfiability of *subtype constraints*. A subtype constraint $t \preceq t'$ with respect to a signature Σ is satisfied by a *valuation* ρ, denoted $\rho \models_\Sigma t \preceq t'$, if $\vdash_\Sigma \rho(t) \leq \rho(t')$. A valuation is an assignment of label variables to label expressions and type variables to object types. The result of applying a valuation ρ to a term t, denoted $\rho(t)$, is always an object type. More specifically, if

$$t \quad = \quad \pi x_1 : L_1 \,.\, \cdots \,.\, \pi x_n : L_n \,.\, \Lambda X_1 : K_1 \,.\, \cdots \,.\, \Lambda X_m : K_m \,.\, T$$

where $n, m \geq 0$ and T is an object type, then $\rho(t) = \rho(T)$.

Subtype unification is a procedure for finding a valuation ρ that satisfies a subtype constraint $t \preceq t'$. Either t or t' has to be a ground object type. We give the algorithm for subtype unification as a collection of rules in Figure 4. Subtype unification takes as input a subtype constraint and a list of declarations. When subtype unification begins, the list of declarations is empty. If subtype unification terminates without aborting, then the result is a valuation that satisfies the subtype constraint. In the rules in Figure 4, $\rho[X \mapsto T]$ is the valuation that is the same as ρ everywhere except on X. On X, the valuation assigns X to T. $\rho_1 \circ \rho_2$ is composition of valuations. Therefore, $\rho_1 \circ \rho_2(t) = \rho_1(\rho_2(t))$. The valuation ρ_ι assigns each variable to itself. Therefore, $\rho_\iota(t) = t$.

Merely determining that a subtype constraint $t \preceq t'$ is satisfiable is not sufficient for relating two types to resolve semantic heterogeneity. We need the valuation to be *consistent* with the declaration of variables specified within the type. For example, the valuation ρ_0 that assigns all label variables to the label l and all type variables to the type real satisfies the subtype constraint $\Lambda X : \{\mathsf{string}\} \,.\, X \preceq \mathsf{real}$ with respect to all signatures (Φ, Ψ) in which $\mathsf{real} \in \Phi$. However, real is not a member of the family of types represented

$$\frac{\vdash_\Sigma c \leq c'}{(\Gamma; c \preceq c') \Longrightarrow_\Sigma \rho_\iota} \qquad \frac{(\Gamma; X \preceq T) \Longrightarrow_\Sigma \rho}{(\Gamma, Y : K; X \preceq T) \Longrightarrow_\Sigma \rho}\; Y \neq X$$

$$\frac{(\Gamma'; T_i \preceq T) \Longrightarrow_\Sigma \rho}{(\Gamma', X : \{T_1, \ldots, T_n\}; X \preceq T) \Longrightarrow_\Sigma [X \mapsto \rho(T_i)]} \qquad \frac{(\Gamma'; T \preceq T_i) \Longrightarrow_\Sigma \rho}{(\Gamma', X : \{T_1, \ldots, T_n\}; T \preceq X) \Longrightarrow_\Sigma [X \mapsto \rho(T_i)]}$$

$$\frac{}{(\Gamma', X : \textsc{type}; X \preceq T) \Longrightarrow_\Sigma \rho_\iota[X \mapsto T]} \qquad \frac{}{(\Gamma', X : \textsc{type}; T \preceq X) \Longrightarrow_\Sigma \rho_\iota[X \mapsto T]}$$

$$\frac{(\Gamma; T_1 \preceq T_1') \Longrightarrow_\Sigma \rho_1 \quad (\Gamma; \rho_1(T_2, \ldots, T_n) \preceq \rho_1(T_2', \ldots, T_n')) \Longrightarrow_\Sigma \rho_2}{(\Gamma; T_1 \times \cdots \times T_n \preceq T_1' \times \cdots \times T_n') \Longrightarrow_\Sigma \rho_2 \circ \rho_1}$$

$$\frac{(\Gamma; T_1 \preceq T_1') \Longrightarrow_\Sigma \rho_1 \quad (\Gamma; \rho_1(T_2, \ldots, T_n) \preceq \rho_1(T_2', \ldots, T_n')) \Longrightarrow_\Sigma \rho_2}{(\Gamma; r_1 : T_1 \times \cdots \times r_n : T_n \preceq T_1' \times \cdots \times T_n') \Longrightarrow_\Sigma \rho_2 \circ \rho_1}$$

$$\frac{(\Gamma; r_i \equiv r') \Longrightarrow \rho_1 \quad (\Gamma; T_i \preceq T') \Longrightarrow_\Sigma \rho_2}{(\Gamma; r_1 : T_1 \times \cdots \times r_{i-1} : T_{i-1} \times r_{i+1} : T_{i+1} \times \cdots \times r_n : T_n \preceq r' : T') \Longrightarrow_\Sigma \rho_2 \circ \rho_1}$$

$$\frac{\begin{array}{c}(\Gamma; r_i \equiv r_1') \Longrightarrow \rho_1 \quad (\Gamma; T_i \preceq T_1') \Longrightarrow_\Sigma \rho_2 \\ (\Gamma; r_1 : T_1 \times \cdots \times r_{i-1} : T_{i-1} \times r_{i+1} : T_{i+1} \times \cdots \times r_n : T_n \preceq r_2' : T_2', \ldots, r_m' : T_m') \Longrightarrow_\Sigma \rho_3\end{array}}{(\Gamma; r_1 : T_1, \ldots, r_n : T_n \preceq r_1' : T_1' \times \cdots \times r_m' : T_m') \Longrightarrow_\Sigma \rho_3 \circ \rho_2 \circ \rho_1}$$

Figure 4: Rules for Solving Subtype Constraints

by $\Lambda X : \{\mathsf{string}\} . X$. Only string is in the family of types represented by $\Lambda X : \{\mathsf{string}\} . X$. Subtype unification produces a valuation such that the assignment of variables to labels and ground object types is consistent with the declarations of the variables in the type. In particular, suppose for some signature Σ, $(; t \preceq T) \Longrightarrow_\Sigma \rho$ or $(; T \preceq t) \Longrightarrow_\Sigma \rho_\iota$. If ρ assigns a type variable X to T, then X is bound in t by $\Lambda X : \textsc{type}$ or $\Lambda X : \{T_1, \ldots, T_n\}$ where $T = \rho(T_i)$ for some i. If ρ assigns a label variable x to l, then x is bound by $\pi x : \textsc{label}$ or $\pi x : \{l_1, \ldots, l_n\}$ where $l = l_i$ for some i.

The properties of subtype unification are summarized in the following theorem.

Theorem 3.3 Let t be a closed type and T be a ground object type. Let $\rho_{l,c}$ be the valuation that assigns all label variables to l and all type variables to c where $c \in \Phi$ for some signature $\Sigma = (\Phi, \Psi)$. If $(; t \preceq T) \Longrightarrow_\Sigma \rho$ then $\vdash_\Sigma \rho(t) \leq T$. If $(; T \preceq t) \Longrightarrow_\Sigma \rho$ then $\vdash_\Sigma T \leq (\rho_{l,c} \circ \rho)(t)$. Furthermore, ρ and $\rho_{l,c} \circ \rho$ are consistent with the declaration of variables in t.

4 Example

To use our approach, an online exchange implements LpX including the functions tosup and tosub. The implementation requires that an online exchange specify a language for atom symbols. This language could be the set of all non-empty finite binary strings or XML terms. For our example, the atoms symbols consist of the rationals, the finite strings of printable ASCII characters enclosed in single quotes, and $\perp$. Also, the online exchange has to specify a signature that it will use. An online exchange could technically support more than one signature. For each signature the online exchange has to provide an interpretation that satisfies the signature. We omit describing the signature and interpretation fully due to space limitation. We assume that all type constant symbols that occur throughout our example are in the signature unless otherwise stated. We will mention the relationship between constant symbols and their interpretation, if they are not obvious from the context.

When an agent enters the online exchange, it tells the online exchange its *input type*. The input type is a closed type and specifies the family of types of negotiation objects that the agent can consider as offers. The input type may specify relationships with the agent's local language and other local languages if it is polymorphic.

Negotiations between two agents are performed by the online exchange taking an offer produced by an agent and giving it to another agent to consider. For this example, we assume negotiations are one–to–one. One–to–many and many–to–many negotiations can be implemented by extending the one–to–one negotiation process so that an agent sends an a single offer to multiple agents to consider or taking multiple offers from several agents to evaluate collectively. The online exchange can use the input types to match agents or some other mechanism. Matching agents is beyond the scope of this paper, so we omit discussing it in this example.

Suppose the online agent matches an agent for a same–day courier and an agent of a law firm to negotiate the courier's delivery services. The courier uses as its local language records belonging to the type

$$\mathsf{wt}\!:\!\mathsf{lbs} \times \mathsf{time}\!:\!\mathsf{hrs} \times \mathsf{cost}\!:\!\mathsf{usd} \times \mathsf{from}\!:\!\mathsf{num} \times \mathsf{to}\!:\!\mathsf{num}. \tag{1}$$

The local language is used by a group of same–day couriers that operate in NYC. Suppose the group of couriers interpret the fields of (1) as follows.

field	semantics
wt	weight of freight in pounds
time	delivery time in hours
to	district of sender
from	district of receiver
cost	amount to charge customer to ship freight in US dollars

Suppose an organization of U.S. law firms develops software to allow members of its organization to buy courier services on the online exchange. The local language it uses is intended to represent negotiation objects for purchasing services from same–day couriers as well as multiple–day couriers. As a result, the local language of this organization consists of records of the type

$$\mathsf{weight}\!:\!\mathsf{lbs} \times \mathsf{from}\!:\!A \times \mathsf{to}\!:\!A \times \mathsf{time}\!:\!\mathsf{days} \times \mathsf{cost}\!:\!\mathsf{usd}. \tag{2}$$

where

$$A \;=\; \mathsf{string} \times \mathsf{string} \times \mathsf{string} \times \mathsf{zipcode}$$

Suppose the law firm uses the following interpretation of the fields of (2) as the semantics of its local language.

field	semantics
weight	weight of freight in pounds
from	street, city, state, and zip code of sender
to	street, city, state, and zip code of recipient
time	delivery time in days
cost	amount courier charges to ship freight in US dollars

If a courier's agent uses the type in (1) as its input type and a law firm's agent uses the type in (2) as its input type, then semantic heterogeneity can occur. For example, the offer to deliver a four pound package for \$100 within eight hours from 777 51st St.,NY, NY, 10023 to 126-34 178th St., Queens, NY 11411 is represented as

$$(\mathsf{wt} = 4, \mathsf{time} = 4, \mathsf{cost} = 100, \mathsf{from} = 23, \mathsf{to} = 191)$$

in the courier's local language and as

$$\begin{aligned}(\quad & \mathsf{weight} = 4, \mathsf{time} = 0.167, \mathsf{from} = (\text{'77 51st St.'}, \text{'NY'}, \text{'NY'}, 10023),\\ & \mathsf{to} = (\text{'126-34 178th St.'}, \text{'Jamaica'}, \text{'NY'}, 11413), \mathsf{cost} = 100)\end{aligned}$$

in the law firm's local language. Semantic heterogeneity can be resolved if the courier's agent used

$$\begin{aligned}&\pi w\!:\!\{\mathsf{wt}, \mathsf{weight}\}\,.\,\Lambda P\!:\!\{A, \mathsf{num}\}\,.\\ &\qquad w\!:\!\mathsf{lbs} \times \mathsf{time}\!:\!\mathsf{hrs} \times \mathsf{cost}\!:\!\mathsf{usd} \times \mathsf{orig}\!:\!P \times \mathsf{dest}\!:\!P\end{aligned} \tag{3}$$

as its input type. We describe how semantic heterogeneity is resolved during negotiation if the courier's agent input type is (3) and the law firm's agent input type is (2).

Negotiation begins by the online exchange selecting one of the agents to generate a negotiation object representing an initial offer. Suppose, for example, that the law firm's agent generates the initial offer

$$e'_0 = (\mathsf{weight} = 2, \mathsf{to} = a, \mathsf{from} = b, \mathsf{time} = 2, \mathsf{cost} = 20)$$

where $a = (\text{'13 Grand St.'}, \text{'Brooklyn'}, \text{'NY'}, 11211)$ and $b = (\text{'19 Rector St'}, \text{'NY'}, \text{'NY'}, 10006)$. Then the exchange deduces that its type is (2). Next the online exchange will seek to convert e'_0 into an object that belongs to a type produced from applying a valuation to t that is consistent with t. The online exchange begins this process by using subtype unification to create a consistent valuation. In particular, the online exchange attempts to subtype unify $t \preceq T'$ or $T' \preceq t$, where T' is (2) and t is (3). Using the subtype unification procedure specified by the rules in Figure 4 and $\mathsf{hours}{\rightarrow}\mathsf{days}$, the online exchange will deduce that $t \preceq T'$ is satisfied by $\rho_0 = \rho_\iota[w \mapsto \mathsf{weight}, P \mapsto A]$. Afterward, the exchange applies ρ_0 to t to produce the type

$$\mathsf{weight}\!:\!\mathsf{lbs} \times \mathsf{time}\!:\!\mathsf{hrs} \times \mathsf{cost}\!:\!\mathsf{usd} \times \mathsf{from}\!:\!A \times \mathsf{to}\!:\!A.$$

which we denote as T_0. By Theorem 3.3 $\vdash T_0 \leq T'$, the exchange can use tosub to convert e'_0 to an object with type T_0. Suppose $\text{tosub}(e'_0, T_0, T) = e_0$. After obtaining e_0, the exchange gives it to the courier's agent to consider. After considering e_0, suppose the courier's agent generates e_1. The type of e_1 may or may not be T_0. For example, e_1 may have more fields than e_0. The additional field may indicate that the recipient also intends to have the courier transport a package for it on the same day. The courier may give a discount when this scenario occurs. For this example, we will assume that type of the offer generated by the courier's agent has the same type as the offer that it received as input. In any case, The exchange performs the same process of inferring the type of the offer, performing subtype unification, using the valuation produced by subtype unification to construct a type, and converting an object to an object of the type produced from applying the valuation. This process continues until one of the agents accepts an offer, rejects an offer, or a time limit for performing negotiations expires. An agent accepts an offer by returning the same offer given to it to consider. An agent rejects an offer by returning a bottom element.

To illustrate how a user can change a local language without involvement of the administrators consider the following. Suppose the group of NYC same–day couriers want to make agents that can be used to sell same–day courier services in major cities in the US and Canada. As a result, they extend their local language to consist of records of the type

$$\mathsf{wt}\!:\!\mathsf{g} \times \mathsf{time}\!:\!\mathsf{hrs} \times \mathsf{cost}\!:\!\mathsf{cad} \times \mathsf{from}\!:\!\mathsf{num} \times \mathsf{to}\!:\!\mathsf{num}. \tag{4}$$

All the fields have the same meaning except wt is the weight of the freight in grams and cost is the price to charge the customer in Canadian dollars. To allow this agent to participate in negotiations with an agent that uses (2) as its input type, we need to change the input type of the courier's agent to

$$\begin{aligned}&\pi w\!:\!\{\mathsf{wt},\mathsf{weight}\}\,.\,\Lambda P\!:\!\{A,\mathsf{num}\}\,.\,\Lambda C\!:\!\{\mathsf{usd},\mathsf{cad}\}\,.\,\Lambda W\!:\!\{\mathsf{lbs},\mathsf{g}\}\,.\\ &\qquad w\!:\!W \times \mathsf{time}\!:\!\mathsf{hrs} \times \mathsf{cost}\!:\!\mathsf{usd} \times \mathsf{orig}\!:\!P \times \mathsf{dest}\!:\!P.\end{aligned}$$

Also the agent has to be changed to handle objects that can have the weight field in grams and the cost in dollars and the address in some city other than NYC.

5 Conclusion

In this paper we have presented LpX, a type theory that could be used as a language for negotiation objects in an online exchange LpX is to a first step in trying to demonstrate the use of type theory as a framework understanding and analyzing solutions for resolving semantic heterogeneity. We have chosen type theory as a framework because type theory allows one to integrate the formalize a conceptualization with a procedure for performing schema matching and translation. As a result, type theory can be used to formulate the entire process of resolving semantic heterogeneity.

References

[1] Y. Bakos. The emerging role of electronic marketplaces on the Internet. *Communications of the ACM*, 41:35–42, Aug 1998.

[2] M. Bichler. *Future of E-markets: Multi-dimensional Market Mechanisms.* Cambridge University Press, Cambridge, 2001.

[3] S. de Vries and R. Vohra. Combinatorial auctions: A survey. Unpublished manuscript, 2000.

[4] M. M. Handling. Working on the B2B and B2C chain gang: An executive roundable (Part 1). *Mordern Materials Handling*, 55(7):70–75, Feb. 2000.

[5] M. M. Handling. Working on the B2B and B2C chain gang: An executive roundable (Part 2). *Modern Materials Handling*, 55(8):47–52, June 2000.

[6] J. O. Kephart, J. E. Hanson, and J. Sairamesh. Price and niche wars in a free market economy of software agents. *Artificial Life*, 4(1):1–23, 1998.

[7] N. Nisan. Bidding and allocation in combinatorial auctions. In *ACM Conference on Electronic Commerce*, pages 1–12, 1999.

[8] J. Teich, H. Wallenius, and J. Wallenius. Multiple issue auction and market algorithms for the world wide web. *Decision Support Systems*, 26:49–66, 1999.