February 1991

Inheritance as Implicit Coercion

Val Tannen
University of Pennsylvania, val@cis.upenn.edu

Thierry Coquand
INRIA

Carl A. Gunter
University of Pennsylvania

Andre Scedrov
University of Pennsylvania

Recommended Citation

Val Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov, "Inheritance as Implicit Coercion", . February 1991.

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-89-01.

Inheritance as Implicit Coercion

Abstract

We present a method for providing semantic interpretations for languages with a type system featuring *inheritance* polymorphism. Our approach is illustrated on an extension of the language Fun of Cardelli and Wegner, which we interpret via a translation into an extended polymorphic lambda calculus. Our goal is to interpret inheritances in Fun via *coercion functions* which are definable in the target of the translation. Existing techniques in the theory of semantic domains can be then used to interpret the extended polymorphic lambda calculus, thus providing many models for the original language. This technique makes it possible to model a rich type discipline which includes parametric polymorphism and recursive types as well as inheritance.

A central difficulty in providing interpretations for explicit type disciplines featuring inheritance in the sense discussed in this paper arises from the fact that programs can type-check in more than one way. Since interpretations follow the type-checking derivations, *coherence* theorems are required: that is, one must prove that the meaning of a program does not depend on the way it was type-checked. The proof of such theorems for our proposed interpretation are the basic technical results of this paper. Interestingly, proving coherence in the presence of recursive types, variants, and abstract types forced us to reexamine fundamental equational properties that arise in proof theory (in the form of commutative reductions) and domain theory (in the form of strict vs. non-strict functions).

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-89-01.

Inheritance As Implicit Coercion

MS-CIS-89-01
LOGIC & COMPUTATION 1

Val Breazu-Tannen
Thierry Coquand
Carl A. Gunter
Andre Scedrov

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

Second Revision
February 1991

Acknowledgements:
This research was supported in part by a grant from the State of Pennsylvania, by DARPA grant N00014-85-K-0018, NSF grants MCS-8219196-CEr, IRI84-10413-A02, and U.S. Army grants DAA29-84-K-0061, DAA29-84-9-0027.

INHERITANCE AS IMPLICIT COERCION [1]

Val Breazu-Tannen *Thierry Coquand* *Carl A. Gunter* *Andre Scedrov*[2]

Abstract. We present a method for providing semantic interpretations for languages with a type system featuring *inheritance* polymorphism. Our approach is illustrated on an extension of the language Fun of Cardelli and Wegner, which we interpret via a translation into an extended polymorphic lambda calculus. Our goal is to interpret inheritances in Fun via *coercion functions* which are definable in the target of the translation. Existing techniques in the theory of semantic domains can be then used to interpret the extended polymorphic lambda calculus, thus providing many models for the original language. This technique makes it possible to model a rich type discipline which includes parametric polymorphism and recursive types as well as inheritance.

A central difficulty in providing interpretations for explicit type disciplines featuring inheritance in the sense discussed in this paper arises from the fact that programs can type-check in more than one way. Since interpretations follow the type-checking derivations, *coherence* theorems are required: that is, one must prove that the meaning of a program does not depend on the way it was type-checked. The proof of such theorems for our proposed interpretation are the basic technical results of this paper. Interestingly, proving coherence in the presence of recursive types, variants, and abstract types forced us to reexamine fundamental equational properties that arise in proof theory (in the form of commutative reductions) and domain theory (in the form of strict *vs.* non-strict functions).

1 Introduction

In this paper we will discuss an approach to the semantics of a particular form of inheritance which has been promoted by John Reynolds and Luca Cardelli. This inheritance system is based on the idea that one may axiomatize a relation $\leq$ between type expressions in such a way that whenever the *inheritance judgement* $s \leq t$ is provable for type expressions s and t, then an expression of type s can be "considered as" an expression of type t. This property is expressed by the *inheritance* rule (sometimes also called the *subsumption* rule), which states that if an expression e is of type s and $s \leq t$, then e also has type t. The consequences from a semantic point of view of the inclusion of this form of typing rule are significant. It is our goal in this paper to look carefully at what we consider to be a robust and intuitive approach to systems which have this form of inheritance and examine in some detail the semantic implications of the inclusion of inheritance judgements and the inheritance rule in a type discipline.

Several attempts have been made recently to express some of the distinctive features of object-oriented programming, principally *inheritance*, in the framework of a rich type discipline which can accommodate

[1] To appear in **Information and Computation**'s special issue dedicated to LICS'89

[2] Author's addresses. Breazu-Tannen and Gunter: Department of Computer and Information Sciences, University of Pennsylvania, Philadelphia PA 19104, USA. Coquand: INRIA, Domaine de Voluceau, 78150 Rocquencourt, France. Scedrov: Department of Mathematics, University of Pennsylvania, Philadelphia PA 19104, USA.

strong static type-checking. This endeavor searches for a language that offers some of the flexibility of object-oriented programming [GR83] while maintaining the reliability, and sometimes increased efficiency of programs which type-check at compile-time (see [BBG88] for a related comparison).

A type system of Reynolds introduced in [Rey80] captured some basic intuitions about inheritance relations between familiar type expressions built from records, variants (sums) and higher types. A language which exploited this form of type discipline was developed by Cardelli in [Car84, Car88a] where the first attempt was made to describe a rigorous form of mathematical semantics for such a system. His approach uses ideals and it is shown that the type discipline is consistent with the semantics in the sense that type-checking is shown to "prevent type errors". Subsequent work has aimed at combining inheritance with richer type disciplines, in particular featuring *parametric polymorphism.* One direction of research [Wan87, JM88, OB88, Sta88], has investigated expressing inheritance and type inference mechanisms, similarly to the way in which parametric polymorphism is expressed in ML-like languages. Another direction of research investigates expressing inheritance through explicit subtyping mechanisms which are part of the type-checking systems, such as in Cardelli and Wegner's language Fun [CW85] and further work [Car88b, Car89a, CM89]. Cardelli and Wegner sketch a model for Fun based on ideals. An extensional model for Fun was subsequently described by Bruce and Longo [BL88]. Their model interprets inheritances as identity relations between partial equivalence relations (PER's). Another model of Fun, using the interval interpretation of Cartwright [Car85] has been given by Martini [Mar88]. In Martini's semantics, inheritance is interpreted as a form of inclusion between intervals. This model also includes a general recursion operator for functions (but not types).

In this paper we present a novel approach to the problem of developing a simple mathematical semantics for languages which feature inheritance in the sense of Reynolds and Cardelli. The form of semantics that we propose will take a significant departure from the characteristic shared by the semantics mentioned above. We will not attempt to model inheritance as a binary relation on a family of types. In particular, our interpretation will not use anything like an inclusion relation between types. Instead, we interpret the inheritance relation between type *expressions* as indicating a certain coercion which remains implicit in instances in which the inheritance is used in type-checking. We show how these coercions can be made explicit using *definable* terms of a calculus without inheritance, and thus depart from the "relational" interpretation of the inheritance concept. Using this idea, we are able to show how many of the models of polymorphism and recursive types which have no relevant concept of type inclusion, can nevertheless be seen as models for a calculus with inheritance.

We illustrate our approach on the language Fun of Cardelli and Wegner extended with recursive types but, the kind of results we obtain are non-trivial for any calculus that combines inheritance, parametric polymorphism, and recursive types. The method we propose proceeds first with a translation of Fun into an extended polymorphic lambda calculus with recursive types. As we mentioned above, this translation interprets inheritances in Fun as *coercion functions* already *definable* in the extended polymorphic lambda calculus. Then, we can use existing techniques for modeling polymorphism and recursion (such as those described in [ABL86, Gir86, CGW87, CGW89]) to interpret the extended polymorphic lambda calculus, thus providing models for the original language with inheritance. This method achieves simultaneous modeling of parametric polymorphism, recursive types, and inheritance. In the process, the paradigm "inheritance as definable coercion" proves itself remarkably robust, which makes us confident that it will apply to a large class of rich type disciplines with inheritance.

The paper is divided into seven sections. Following this introduction, the second section provides some general examples and motivation to prepare the reader for the technical details in the subsequent sections. The third section discusses how our semantics applies to a calculus **SOURCE** which has inheritance, exponentials, records, generics and recursive types. We show how this is translated into a calculus **TARGET** without inheritance and state our results about the coherence of the translation. We hope that the results in this simpler setting will help the reader get an idea of what our program is before we proceed to a more interesting calculus in the remainder of the paper. The fourth section is devoted to developing a translation for an expanded calculus which adds variants. Fundamental equational properties of variants lead us to develop a target language which has a *type of coercions*. The fifth section, which contains the difficult technical results of the paper, shows that our translation is coherent. In the sixth section we discuss mathematical models for the full calculus. Since most of the work has already been done, we are able to produce many models using standard domain-theoretic techniques. The concluding section makes some remarks about what we feel has been achieved and what new challenges still need to be confronted.

2 Inheritance as implicit coercion.

A simple analogy will help explain our translation-based technique. Consider how the ordinary *untyped* λ-calculus is interpreted semantically in such sources as [Sco80, Mey82, Koy82, Bar84]. One begins by postulating the existence of a semantic domain D and a pair of arrows $\Phi : D \to (D \to D)$ and $\Psi : (D \to D) \to D$ such that $\Phi \circ \Psi$ is the identity on $D \to D$. Certain conditions are required of $D \to D$ to insure that "enough" functions are present. To interpret an untyped λ-term, one defines a translation $M \mapsto M^*$ on terms which takes an untyped term M and creates a typed term M^*. This operation is defined by induction:

- for a variable, $x^* \equiv x : D$,
- for an application, $M(N)^* \equiv \Phi(M^*)(N^*)$ and,
- for an abstraction, $(\lambda x.\ M)^* \equiv \Psi(\lambda x : D.\ M^*)$

(where we use $\equiv$ for syntactic equality of expressions). For example, the familiar term

$$\lambda f.\ (\lambda x.\ f(xx))(\lambda x.\ f(xx))$$

translates to

$$\Psi(\lambda f : D.\ \Phi(\Psi(\lambda x : D.\ \Phi(f)(\Phi(x)(x))))(\Psi(\lambda x : D.\ \Phi(f)(\Phi(x)(x))))).$$

The fact that the latter term is unreadable is perhaps an indication of why we use the former term *in which the semantic coercions are implicit.* Nevertheless, this translation provides us with the desired semantics for the untyped term since we have converted that term into a term in a calculus which we know how to interpret. Of course, this assumes that we really do know how to provide a semantics for the typed calculus supplemented with triples such as D, Φ, Ψ. Moreover, there are some equations we must check to show that the translation is sound. But, at the end of the day, we have a simple, intuitive explanation of the interpretation of untyped λ-terms based on our understanding of a certain simply typed λ-theory. In this paper we show how a similar technique may be used to provide an intuitive interpretation for inheritance, even in the presence of parametric polymorphism and type recursion. As mentioned earlier, our interpretation is carried out by translating the full calculus into a calculus without inheritance (the *target* calculus) whose semantics we already understand. However, our idea differs significantly from the interpretation of the untyped λ-calculus as described above in at least one important respect: typically, the coercions (such as Φ and Ψ above) which we introduce will be *definable* in the target calculus. Hence our target calculus needs to be an extension of the ordinary polymorphic λ-calculus with records, variants, abstract types, and recursive types. But it need not have any inheritance.

From this lead, we may now propose a way to explain the semantics of an expression in a language with inheritance. Our semantics interprets typing judgements, *i.e.* assertions $\Gamma \vdash e\colon s$ that expression e has type s in context Γ. Ordinarily such a judgement is assigned a semantics inductively in the proof of the judgement using the typing rules. However, the system we are considering may also include instances of the *inheritance rule* which says that if e has type s and s is a subtype of t, then e has type t. How are we to relate the interpretation of the type expressions s and t so that the meaning of e can be viewed as living in both places? Our proposal: the proof that s is a subtype of t generates a *coercion* P from s into t. The inheritance (subsumption) rule is interpreted by the application of the coercion P to the interpretation of e as an element of s. It will be seen below that this technique can be made to work very smoothly since the language we are interpreting may have a familiar *inheritance-free* fragment in which coercions such as P can be defined. In effect, we can therefore "project" the language onto an inheritance-free fragment of itself.

For further illustration, let us now look at an example which combines parametric polymorphism and inheritance. In the polymorphic λ-calculus, it is possible to form expressions in which there are abstractions over *type variables*. For example, the term $e \equiv \Lambda a.\ \lambda x\colon a.\ x$ is an operator which takes a type s as an argument and returns the identity function $\lambda x\colon s.\ x$ on that type as a value. The type of e is indicated by the expression $\forall a.\ a \to a$. Semantically, one may think of the meaning of this expression as an indexed product where a ranges over all types. Although this explanation is a bit too simple as it stands, it does help with the basic intuition. If one wishes to make an abstraction over the *subtypes* of a given type, one may use the concept of a *bounded quantification* [CW85]. Consider, for example, the term

$$e' \equiv \Lambda a \leq \{l\colon s\}.\ \lambda x\colon a.\ (x.l)$$

where $\{l\colon s\}$ is a *record* expression which has one field, labelled l, with type s. The expression e' denotes an operator which takes a subtype t of $\{l\colon s\}$ (we write $t \leq \{l\colon s\}$) and returns as value a function from t to s. (The reader should not confuse a, a type variable, with t, a type expression.) Intuitively, a subtype of $\{l\colon s\}$ is a record which has an l field whose type is a subtype of s. The type of e' is indicated by the expression $u' \equiv \forall a \leq \{l\colon s\}.\ a \to s$. How should we think of this type semantically? Taking an analogy with the intuitive semantics of polymorphic quantification, we want to think of the meaning of

u' as some kind of indexed product. But indexed over what? In this paper we argue that one may get an intuitive semantics of bounded quantification by thinking of a type expression such as u' as a family of types *indexed over coercions (i.e. certain functions) from a type a into the type s.*

To support this intuition we must explain the meaning of the application $e'(t)$ of the expression e' to a type expression t which is a subtype of $\{l\colon s\}$. The key fact is this: given type expressions v and w and a proof that v is a subtype of w, there is a canonical coercion from v into w. Hence, the application $e'(t)$ has, as its meaning, the element of $t \to s$ obtained by applying the meaning of e'—which is an element of an indexed product—to the canonical coercion from t to $\{l\colon s\}$. This leads us to consider u' as the type

$$\forall a.\ (a \multimap \{l\colon s\}) \to a \to s$$

where $a \multimap \{l\colon s\}$ is a "type of coercions". In category-theoretic jargon: the meaning of a bounded quantification with bound v will be an adjoint to a fibration over the *slice category* over v. This follows the analogy with models of polymorphism which are based on adjoints to fibrations over the category of all domains (as in [CGW89] for example).

Although we believe that the translation just illustrated is intuitive, we need to show that it is *coherent.* In other words, we must show that the semantic function is well defined. The need for coherence comes from the fact that a typing judgement may have many different derivations. In general, it is customary to present the semantics of typed lambda calculi as a map defined inductively on type-checking derivations. Such a method would therefore assign a meaning to each derivation tree. We do believe though, that the *language* consists of the derivable typing judgements, rather than of the derivation trees. For many calculi, such as the simply typed or the polymorphic lambda calculus, there is at most one derivation for any typing judgement. Therefore, in such calculi, giving meaning to derivations is the same as giving meaning to derivable judgements. But for other calculi, such as Martin-Löf's Intuitionistic Type Theory (ITT) [Mar84] (see [Sal88]), and the Calculus of Constructions [CH88] (see [Str88]), and—of immediate concern to us—Cardelli and Wegner's Fun, this is not so, and one must prove that derivations yielding the same judgement are given the same meaning. This idea has also appeared in the context of category theory and our use of the term "coherence" is partially inspired by its use there, where it means the uniqueness of certain canonical morphisms (see *e.g.* [KL71] and [LP85]). Although we have not attempted a rigorous connection in this paper, the possibility of unifying coherence results for a variety of different calculi offers an interesting direction of investigation. In the case of Fun, we show the coherence of our semantic approach by proving that *translations of any two derivations of the same typing judgement are equated in the target calculus.*

Hence, the coherence of a given translation is a property of the equational theory of the target calculus. When the target calculus is the polymorphic lambda calculus extended with records and recursive types, the standard axiomatization of its equational theory is sufficient for the coherence theorem. But when we add variants, the standard axiomatization of these features, while sufficient for coherence, clashes with the standard axiomatization of recursive types, yielding an inconsistent theory (see [Law69, HP89a] for variants, that is, coproducts). The solution lies in two observations: (1) the (too) strong axioms are only needed for "coercion terms", and (2) in the various models we examined these coercion terms have special interpretations (such as *strict*, or *linear* maps), so special in fact, that they satisfy the corresponding restrictions of the strong axioms! Correspondingly, one has to restrict the domains over which "coercion variables" can range, which leads naturally to the type of coercions mentioned above.

3 Translation for a fragment of the calculus

For pedagogical reasons, we begin by considering a language whose type structure features function spaces (exponentials), record types, bounded generic types (an inheritance-generalized form of universal polymorphism), recursive types, and, of course, inheritance. In the next section we will enrich this calculus by the addition of variants. As we have mentioned before, this leads to some (interesting) complications which we avoid by restricting ourselves to the simpler calculus of this section. Since the calculus in the next section is stronger, we omit details for the proofs of results in this section. They resemble the proofs for the calculus with variants, but the calculations are simpler. Rather than generate four different names for the calculi which we shall consider in this section and the next we simply refer to the calculus with inheritance as **SOURCE** and the inheritance-free calculus into which it is translated as **TARGET**. The fragment of the calculus which we consider in this section is fully described in the appendices to the paper.

We provide semantics to **SOURCE** via a *translation* into a language for which several well-understood semantics already exist. This "target" language, which we shall call **TARGET**, is an extension with record and recursive types of the Girard-Reynolds polymorphic lambda calculus (see [CGW87] for the semantics of **TARGET**). Therefore, **SOURCE** extends with inheritance and bounded generics **TARGET**, which is at its turn an extension of what Girard calls **System F** in [Gir86]. Our translation takes derivations of inheritance and typing judgements in **SOURCE** into derivations of typing judgements in **TARGET**. We translate the inheritance judgements of **SOURCE** into definable terms of **TARGET** which can be thought of as *canonical explicit coercions*. Bounded generics translate into usual generics, but of "higher" type, which take an additional argument which can be thought of as an *arbitrary coercion*.

In arguing that this translation yields a semantics for **SOURCE**, we encounter, as mentioned in the introduction, an important complication: as we shall see, in **SOURCE** as well as in Fun, there may be *several* distinct derivations of the *same* typing judgement (or inheritance judgement, for that matter). We consider, however, the *language* to consists of the derivable typing judgements, rather than of the derivation trees. This distinction can be ignored in System **F** or **TARGET**, where there is at most one derivation for any typing judgements, so giving meaning to derivations is the same as giving meaning to derivable judgements. But for **SOURCE** and Fun, this is not so, and one must show that derivations yielding the same judgement are given the same meaning. This meaning is then defined to be the meaning of the judgement. This crucial problem was overlooked by publications on the semantics of inheritance prior to [BCGS89].

We solve the problem as follows. It turns out that our translation takes syntactically distinct derivations of the same **SOURCE** judgement into syntactically distinct derivations in **TARGET**. But we give an *equational axiomatization* as an integral part of **TARGET**, and we show that our translation takes derivations of the same **SOURCE** judgement into derivations of *provably equal* judgements in **TARGET**. By this *coherence* result, *any* model of **TARGET**, being also a model of its equational theory, will provide a well-defined semantics for the derivable judgements of **SOURCE.**

The source calculus. For notation, we will follow the spirit of Fun [CW85] making precise only the differences. The type expressions include type variables a and a distinguished constant Top. If s and t are type expressions, then $s \rightarrow t$ is the type of functions from s to t. If $s_1, \ldots, s_n$ are type expressions, and $l_1, \ldots, l_n$ is a collection of distinct *labels,* then $\{l_1 : s_1, \ldots, l_n : s_n\}$ is a *record* type expression. We make the syntactic assumption that the order of the labels is irrelevant. If s and t are type expressions then $\forall a \leq s.\, t$ is a *bounded quantification* which binds free occurrences of the variable a in the type expression t (but not in s). Similarly, $\mu a.\, t$ is a *recursive* type expression in which the type variable a is bound in the type expression t. Intuitively, $\mu a.\, t$ is the solution of the equation $a = t$. We will use $[s/a]t$ for substitution. The *raw* terms of the language include (term) variables x, applications $d(e)$ and lambda abstractions $\lambda x{:}t.\, e$. An expression $\{l_1 = e_1, \ldots, l_n = e_n\}$ is called a record with *fields* $l_1, \ldots, l_n$ and the expression $e.l$ is the *selection* of the field l. Again, we assume that the order of the fields of a record is irrelevant, but the labels must all be distinct. We also have bounded type abstraction $\Lambda a \leq t.\, e$ and the corresponding application $e(t)$. To form terms of recursive type $\mu a.\, t$ we have *intro* expressions $\mathsf{intro}[\mu a.\, t]e$ and they are eliminated from the recursion by *elim* expressions $\mathsf{elim}\ e$. See Appendix A to find a grammar for the type expressions and raw terms of the fragment.

Raw terms are type-checked by deriving *typing judgements*, of the form $\Gamma \vdash e : t$. where Γ is a context. *Contexts* are defined recursively as follows: $\emptyset$ is a context; if Γ is a context which does not declare a, and the free variables of t are declared in Γ, then $\Gamma, a \leq t$ is a context; if Γ is a context which does not declare x, and the free variables of t are declared in Γ, then $\Gamma, x{:}t$ is a context. The proof system for deriving typing judgments is the relevant fragment of the corresponding proof system for Fun (see [CW85] on pages 519–520) enriched with two type-checking rules for the introduction and elimination of recursive types [CGW87]. A complete list of these proof rules is in Appendix A under the heading **Fragment.**

Among these proof rules, the following two illustrate the effect of inheritance on type-checking:

[INH] $$\frac{\Gamma \vdash e : s \qquad \widehat{\Gamma} \vdash s \leq t}{\Gamma \vdash e : t}$$

[B-SPEC] $$\frac{\Gamma \vdash e : \forall a {\leq} s.\, t \qquad \widehat{\Gamma} \vdash r \leq s}{\Gamma \vdash e(r) : [r/a]t}$$

They make use of *inheritance judgements* which have the form $C \vdash s \leq t$ where C is an inheritance context. *Inheritance contexts* are contexts in which only declarations of the form $a \leq t$ appear. If Γ is a context, we denote-by $\widehat{\Gamma}$ teh inheritance context obtained from Γ by erasing the declarations of the form $x{:}t$. The proof system for deriving inheritance judgments is, with the exception of one rule, the same as the relevant fragment of the corresponding proof system for Fun (see [CW85], on page 519). In this paper we do not attempt to enrich it with any rule deriving inheritances *between* recursive types. A discussion of this issue appears in our conclusions. The Appendix contains a complete list of these proof rules too.

In comparison with Fun, we would like to strengthen the rule deriving inheritances between bounded generics, and we are able to do so for some of our results. Where Fun had just

(W-FORALL)
$$\frac{C, a{\le}t \;\vdash\; u \;\le\; v}{C \;\vdash\; \forall a{\le}t.\, u \;\le\; \forall a{\le}t.\, v}$$

we will consider

(FORALL)
$$\frac{C \;\vdash\; s \;\le\; t \qquad C, a{\le}s \;\vdash\; u \;\le\; v}{C \;\vdash\; \forall a{\le}t.\, u \;\le\; \forall a{\le}s.\, v}$$

This makes the system strictly stronger, allowing more inheritances to be derived, and thus more terms to type-check.

Originally, we believed that coherence could be proved for a system that includes variants and the stronger rule (FORALL) [BCGS89]. In dealing with the *case* construct for variant types, however, our coherence proof uses an order-theoretic property (see Lemma 11) which fails for the stronger system for deriving inheritances that uses (FORALL) (for a counterexample, see Giorgio Gelli's dissertation [Ghe90]). Thus, we prove the coherence of the translation of variants (Theorem 13) only for the weaker system with (W-FORALL). Note, however, that we prove coherence in the presence of (FORALL) for the system without variants (Theorem 4) and for the system for deriving inheritances between types, including variant types (Lemma 9).

Remark. Decidability of type-checking in the stronger system is a non-trivial question. The question whether an algorithm of Luca Cardelli will decide the provability of judgements in this calculus has only recently been settled by Ghelli [Ghe90].

The salient feature of bringing inheritance into a type system is that (in given contexts) terms will *not* have a unique type any more. For example, due to the rule

(TOP)
$$C \;\vdash\; t \;\le\; \mathit{Top}$$

where the free variables of t are declared in C, by [INH], all terms that type-check with some type will also type-check with type *Top*. This makes it possible to define ordinary generics as syntactic sugar: $\forall a.\, t \overset{\text{def}}{=} \forall a{\le}\mathit{Top}.\, t$.

The proof system for **SOURCE**, while quite intuitive, allows for the following complication: there may be more than one derivation of the same typing judgement. In fact, we only need record types, (RECD), [VAR], [SEL] and [INH] (see Appendix) to provide such an example: in the context $x\colon \{l_1\colon \mathit{Top}, l_2\colon \mathit{Top}\}$, we can either directly derive by [SEL] $x.l_1 : \mathit{Top}$, or we can derive by [VAR] $x : \{l_1\colon \mathit{Top}, l_2\colon \mathit{Top}\}$, then by (RECD) and [INH] $x : \{l_1\colon \mathit{Top}\}$, and finally by [SEL] $x.l_1 : \mathit{Top}$. In view of this, for any semantics given by "induction on the rules", one needs to prove that derivations of the same judgement have the same meaning.

The target calculus. As mentioned before, **TARGET** is the Girard-Reynolds polymorphic lambda calculus, enriched with record and recursive types [CGW87, BC88, CGW89]. Here, we present it as a simplification of **SOURCE**. Types are given by

$$a \mid s \to t \mid \forall a.\, t \mid \{l_1\colon s_1, \ldots, l_n\colon s_n\} \mid \mu a.\, t$$

and terms by

$$x \mid M(N) \mid \lambda x{:}t.\, M \mid \Lambda a.\, M \mid M(t) \mid \{l_1 = M_1, \ldots, l_n = M_n\} \mid M.l \mid \mathsf{intro}[\mu a.\, t]M \mid \mathsf{elim}\ M$$

For $n = 0$ we get the the *empty record type* $\mathbf{1} \overset{\text{def}}{=} \{\}$ and the *empty record*, for which we will keep the notation $\{\}$. *Typing contexts* are the obvious simplification of contexts in which only typing judgements occur (there is no inheritance relation in **TARGET**). The rules for deriving typing judgements in the fragment of **TARGET** discussed in this section can be found in Appendix B. The following is a well-known fact:

Proposition 1 *In* **TARGET**, *derivations of typing judgements are unique.*

Proof: All the "elimination" rules, [APPL], [SEL], [SPEC], and [R-ELIM] are "cut" rules, in the sense that there is information in the premisses that does not appear in the conclusion. Consequently, they should in principle cause problems for the uniqueness of derivations. However, the lost information is always in the type part, and types "should" be unique. This suggests the strengthening of the induction hypothesis, which then passes trivially through these "cut" rules.

One proves therefore that for any two derivations Δ_1 and Δ_2, if Δ_1 ends in $\Upsilon \vdash M : t_1$ and Δ_2 ends in $\Upsilon \vdash M : t_2$ then $\Delta_1 \equiv \Delta_2$ (in particular, $t_1 \equiv t_2$).

The proof can be done straightforwardly, either by induction on the maximum of the heights of Δ_1 and Δ_2, or on the sum of those heights, or even on the structure of M (with a bit of reformulation). ▮

A technical point: it turns out that type decorations are unnecessary on "elimination" constructs, but they are in fact necessary on some "introduction" constructs, such as lambda abstraction and the recursive type construct intro[]. Later on, with the addition of variants in section 4, we will find that we need to differ with [CW85], and decorate with types the constructs that "inject" into variant types (see Appendix B).

Equations are derived by a proof system (see [CGW87, BC88, CGW89]) which contains rules like reflexivity, symmetry, transitivity, congruence with respect to function application, closure under functional abstraction (ξ), congruence with respect to application to types, closure with respect to type abstraction (type ξ). There are also the {BETA} and {ETA} rules for both functional and type abstraction, rules saying that intro[ ] and elim are inverse to each other, as well as

{RECD-BETA} $\qquad\qquad \{l_1 = M_1, \ldots, l_n = M_n\}.l_i \;=\; M_i$

where $n \geq 1$, and

{RECD-ETA}. $\qquad\qquad \{l_1 = M.l_1, \ldots, l_n = M.l_n\} \;=\; M$

where $M : \{l_1{:}\, s_1, \ldots, l_n{:}\, s_n\}$ .The last rule gives, for $n = 0$, the equation $\{\} = M$ which makes $\mathbf{1}$ into a terminator. Under our interpretation, the type *Top* will be nothing like a "universal domain" which can be used to interpret *Type:Type* [CGW89, GJ90]. On the contrary, it will be interpreted as a one point domain in the models we list below!

The translation. For any **SOURCE** `item` we will denote by `item`* its translation into **TARGET**. We begin with the types. Note the translation of bounded generics and of *Top*.

$$\begin{array}{rcl@{\qquad}rcl} a^* & \stackrel{\text{def}}{=} & a & \{l_1\colon s_1, \ldots, l_n\colon s_n\}^* & \stackrel{\text{def}}{=} & \{l_1\colon s_1^*, \ldots, l_n\colon s_n^*\} \\ \mathit{Top}^* & \stackrel{\text{def}}{=} & \mathbf{1} & (\forall a \leq s.\, t)^* & \stackrel{\text{def}}{=} & \forall a.\, (a \to s^*) \to t^* \\ (s \to t)^* & \stackrel{\text{def}}{=} & s^* \to t^* & (\mu a.\, t)^* & \stackrel{\text{def}}{=} & \mu a.\, t^* \end{array}$$

One shows immediately that $([s/a]t)^* \equiv [s^*/a]t^*$. We extend this to contexts and inheritance contexts, which translate into just typing contexts in **TARGET**.

$$\begin{array}{rcl@{\qquad}rcl} \emptyset^* & \stackrel{\text{def}}{=} & \emptyset & \emptyset^* & \stackrel{\text{def}}{=} & \emptyset \\ (\Gamma,\, a \leq t)^* & \stackrel{\text{def}}{=} & \Gamma^*,\, a,\, f\colon a \to t^* & (C,\, a \leq t)^* & \stackrel{\text{def}}{=} & C^*,\, a,\, f\colon a \to t^* \\ & & & (\Gamma,\, x\colon t)^* & \stackrel{\text{def}}{=} & \Gamma^*,\, x\colon t^* \end{array}$$

where f is a *fresh* variable for each a.

Next we will describe how we translate the derivations of judgments of **SOURCE**. The translation is defined by recursion on the structure of the derivation trees. Since these are freely generated by the derivation rules, it is sufficient to provide for each derivation rule of **SOURCE** a corresponding rule on trees of **TARGET** judgments. It will be a lemma (Lemma 2 to be precise) that these corresponding rules are *directly derivable* in **TARGET**, therefore the translation takes derivations in **SOURCE** into derivations in **TARGET**.

A **SOURCE** derivation yielding an inheritance judgment $C \vdash s \leq t$ is translated as a tree of **TARGET** judgments yielding $C^* \vdash P : s^* \to t^*$. We present three of the rules here; the full list for the fragment appears in Appendix C. The coercion into *Top* is simply the constant map:

(TOP)* $$C^* \vdash \lambda x\colon t^*.\, \{\} : t^* \to \mathbf{1}$$

To see how coercion works on types, assume that we are given a coercion $P\colon s \to t$ from s into t and a coercion $Q\colon u \to v$ from u into v. Then it is possible to coerce a function $f\colon t \to u$ into a function from s to v as follows. Given an argument of type s, coerce it (using P) into an argument of type t. Apply the function f to get a value of type u. Now coerce this value in u into a value in v by applying Q. This describes a function of the desired type. More formally, we translate the (ARROW) rule by

(ARROW)* $$\frac{C^* \vdash P : s^* \to t^* \qquad C^* \vdash Q : u^* \to v^*}{C^* \vdash R : (t^* \to u^*) \to (s^* \to v^*)}$$

where $R \stackrel{\text{def}}{=} \lambda z\colon t^* \to u^*.\, P; z; Q$. (We use ; as shorthand for *composition.* For example, $P; z; Q$ above stands for $\lambda x\colon s^*.\, Q(z(P(x)))$ where x is fresh.) Now, to translate the rule (FORALL) which describes the inheritance relation for the bounded quantification we view the quantification as ranging over a type together with a coercion from that type into the bound:

(FORALL)* $$\frac{C^* \vdash P : s^* \to t^* \qquad C,\, a,\, f\colon a \to s^* \vdash Q : u^* \to v^*}{C^* \vdash R : (\forall a.\, (a \to t^*) \to u^*) \to (\forall a.\, (a \to s^*) \to v^*)}$$

where $R \stackrel{\text{def}}{=} \lambda z\colon (\forall a.\, (a \to t^*) \to u^*).\, \Lambda a.\, \lambda f\colon a \to s^*.\, Q(z(a)(f; P))$

Now, a **SOURCE** derivation yielding an typing judgment $\Gamma \vdash e : t$ is translated as a tree of **TARGET** judgments yielding $\Gamma^* \vdash M : t^*$. For example, the inheritance rule is translated by simply making the inheritance coercion "explicit":

$$[\text{INH}]^* \qquad \frac{\Gamma^* \vdash M : s^* \qquad \widehat{\Gamma}^* \vdash P : s^* \rightarrow t^*}{\Gamma^* \vdash P(M) : t^*}$$

The specialization of a bounded quantification is more subtle. The variable is instantiated by substituting the type expression to which the abstraction is applied, but then the coercion from the argument type to the bound type must be passed as an argument to the resulting function:

$$[\text{B-SPEC}]^* \qquad \frac{\Gamma^* \vdash M : \forall a.\,(a \rightarrow s^*) \rightarrow t^* \qquad \widehat{\Gamma}^* \vdash P : r^* \rightarrow s^*}{\Gamma^* \vdash M(r^*)(P) : [r^*/a]t^*}$$

The remaining rules for translating the fragment are given in Appendix C. It is possible to check that the translated rules are derivable in the target language:

Lemma 2 *The rules* (TOP)* – (TRANS)* *and* [VAR]* – [INH]* *are directly derivable in* TARGET. ∎

Coherence of the translation. For any derivation Δ in **SOURCE**, let Δ^* be the **TARGET** derivation into which it is translated. The central result about *inheritance* judgements says that, given a judgement $s \leq t$ and a pair of proofs Δ_1 and Δ_2 of this judgement, the coercions induced by these two proofs are provably equal in the equational theory of **TARGET**. More formally, we have the following:

Lemma 3 (Coherence of the translation of inheritance) *Let Δ_1 and Δ_2 be two* SOURCE *derivations of the same inheritance judgement,* $C \vdash s \leq t$. *Let Δ_1^*, Δ_2^* yield (coercion) terms P_1, P_2. Then,* $P_1 = P_2$ *is provable in* TARGET.

The central result about *typing* judgements says that, given a judgement $e{:}\,t$ and a pair of proofs Δ_1 and Δ_2 of this judgement, the translations of these proofs end in sequents (translations of $e{:}\,t$) which are provably equal in the equational theory of **TARGET**, *i.e.* we have:

Theorem 4 (Coherence) *Let Δ_1 and Δ_2 be two* SOURCE *derivations yielding the same typing judgement,* $\Gamma \vdash e : t$. *Let Δ_1^*, Δ_2^* yield terms M_1, M_2. Then,* $M_1 = M_2$ *is provable in* TARGET.

The proofs of the lemma and theorem are almost as difficult as the ones we shall give for the corresponding results in the full language. Since the proofs of these results for the fragment follow similar lines to the proofs for the full language we omit the proofs of Lemma 3 and Theorem 4 in favor of the proofs of Lemma 9 and Theorem 13 below.

4 Between incoherence and inconsistency: adding variants

The calculus described so far does not deal with a crucial type constructor: variants. In particular, it is very useful to have a combination of variant types with recursive types. On the other hand, the combination of these operators in the same calculus is also problematic, especially for the equational theory. The situation is familiar from both domain theory and proof theory. In this section we propose an approach which will suffice to prove the coherence theorem which we need to show that our semantic function is well-defined.

We extend the type formation rules of **SOURCE** by adding *variant* type expressions: $[l_1\!:t_1,\ldots,l_n\!:t_n]$ where $n \geq 1$. We also extend the term formation rule by the formation of variant terms $[l_1\!:t_1,\ldots,l_i = e,\ldots,l_n\!:t_n]$ and the *case statement:*

$$\textsf{case}\ e\ \textsf{of}\ l_1 \Rightarrow f_1,\ldots,l_n \Rightarrow f_n$$

The inheritance judgement derivation rules are extended correspondingly with the rule:

(VART) $$\frac{C\ \vdash\ s_1\ \leq\ t_1\quad\cdots\quad C\ \vdash\ s_p\ \leq\ t_p}{C\ \vdash\ [l_1\!:s_1,\ldots,l_p\!:s_p]\ \leq\ [l_1\!:t_1,\ldots,l_p\!:t_p,\ldots,l_q\!:t_q]}$$

Note the "duality" between this rule and the inheritance rule (RECD) for records (see Appendix A). While a record subtype has more fields, a variant subtype has fewer variations (summands).

Like before, we intend to translate this calculus into a calculus without inheritance and, naturally, we extend **TARGET** with variants (see Appendix B). Note how the syntax of variant injections differs from [CW85]. This is in order for the resulting system to enjoy the property of having unique type derivations: the proof of Proposition 1 extends immediately to the variant constructs. Most importantly, we must extend the equational theory of **TARGET** in a manner that insures the coherence of our translation. It is here that we encounter an interesting problem which readers who know domain theory will find familiar. The following two axioms hold in a variety of models:

{VART-BETA} $$\textsf{case}\ \textsf{inj}_{l_i}(M_i)\ \textsf{of}\ l_1 \Rightarrow F_1,\ldots,l_n \Rightarrow F_n\ =\ F_i(M_i)$$

where $F_1 : t_1 \to t,\ldots,F_n : t_n \to t$, $M_i : t_i$ and $\textsf{inj}_{l_i}$ is shorthand for $\lambda x\!:t_i.\,[l_1\!:t_1,\ldots,l_i = x,\ldots,l_n\!:t_n]$.

{VART-ETA} $$\textsf{case}\ M\ \textsf{of}\ l_1 \Rightarrow \textsf{inj}_{l_1},\ldots,l_n \Rightarrow \textsf{inj}_{l_n}\ =\ M$$

where $M : [l_1\!:t_1,\ldots,l_n\!:t_n]$. Unfortunately, these two axioms do not suffice to prove all the identifications required by the coherence of our translation!

To see the problem, we start with an example. In **SOURCE**, suppose that $t\ \leq\ s$ is derivable in the context $\widehat{\Gamma}$, and that we have a derivation Δ of $\Gamma \vdash e\ :\ [l_1\!:t_1, l_2\!:t_2]$ and derivations Δ_i of $\Gamma\ \vdash\ f_i\ :\ t_i \to t$, $i\ =\ 1,2$. Consider then the following two **SOURCE** derivations of the typing judgement $\Gamma \vdash \textsf{case}\ e\ \textsf{of}\ l_1 \Rightarrow f_1, l_2 \Rightarrow f_2\ :\ s$.

1. by Δ, Δ_1, Δ_2 and the rule [CASE], one deduces $\Gamma\ \vdash\ \textsf{case}\ e\ \textsf{of}\ l_1 \Rightarrow f_1, l_2 \Rightarrow f_2\ :\ t$. Since $\widehat{\Gamma}\ \vdash\ t\ \leq\ s$ by hypothesis, one infers by inheritance $\Gamma\ \vdash\ \textsf{case}\ e\ \textsf{of}\ l_1 \Rightarrow f_1, l_2 \Rightarrow f_2\ :\ s$.
2. from $\widehat{\Gamma}\ \vdash\ t\ \leq\ s$ we can deduce $\widehat{\Gamma}\ \vdash\ (t_i \to t)\ \leq\ (t_i \to s)$. Hence, by inheritance from Δ_i, one deduces $\Gamma\ \vdash\ f_i\ :\ t_i \to s$. Then, from Δ and by the rule [CASE], one deduces $\Gamma\ \vdash\ \textsf{case}\ e\ \textsf{of}\ l_1 \Rightarrow f_1, l_2 \Rightarrow f_2\ :\ s$.

The coherence property requires that these two derivations have provably equal translations. With the obvious translation for the variant type constructor and the rules [VART] and [CASE] (see Appendix C) and with the translation of the rules [INH], (ARROW) and (REFL) as in Section 3, this comes down to the following identity

$$P(\mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow F_1, l_2 \Rightarrow F_2) \ = \ \mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow (F_1; P), l_2 \Rightarrow (F_2; P)$$

where $P : t^* \to s^*$ is a "coercion term", $M : [l_1{:}\,t_1^*, l_2{:}\,t_2^*]$, $F_i : t_i^* \to t^*$, $i = 1,2$. Thus, we are tempted to postulate

{VART-CRN?} $\quad P(\mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow F_1, \ldots, l_n \Rightarrow F_n) \ = \ \mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow F_1; P, \ldots, l_n \Rightarrow F_n; P$

where $M{:}\,[l_1{:}\,t_1, \ldots, l_n{:}\,t_n]$, $F_1{:}\,t_1 \to t, \ldots, F_n{:}\,t_n \to t$, $P{:}\,t \to s$. This equation follows from the equation that axiomatizes variants analogously to coproducts:

{VART-COP?} $\quad Q(M) \ = \ \mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow (\mathsf{inj}_{l_1}; Q), \ldots, l_n \Rightarrow (\mathsf{inj}_{l_n}; Q)$

where $M{:}\,[l_1{:}\,t_1, \ldots, l_n{:}\,t_n]$, $Q{:}\,[l_1{:}\,t_1, \ldots, l_n{:}\,t_n] \to t$. More precisely, it is possible to check that the system {VART-BETA}+{VART-COP} is equivalent to {VART-BETA}+{VART-CRN}+{VART-ETA}. However, it is known [Law69, HP89a] that {VART-BETA}+{VART-COP} is inconsistent with the existence of fixed-points. In fact, this may be refined:

Proposition 5 *The system* {VART-BETA}+{VART-CRN} *is (equationally) inconsistent with the existence of fixed-points.*

Proof: The "categorical" equation { VART-COP } may be thought of as an "induction" principle on a sum: it reduces the proof of an equation $P(M) = Q(M)$, $M{:}\,[l_1{:}\,t_1, l_2{:}\,t_2]$, to the proofs of $P(\mathsf{inj}_{l_1}(x)) = Q(\mathsf{inj}_{l_1}(x))$, for $x{:}\,t_1$ and $P(\mathsf{inj}_{l_2}(x)) = Q(\mathsf{inj}_{l_2}(x))$, for $x{:}\,t_2$. Indeed, we have $P(M) = \mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow \lambda x.\ P(\mathsf{inj}_{l_1}(x)), l_2 \Rightarrow \lambda x.\ P(\mathsf{inj}_{l_2}(x))$ and $Q(M) = \mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow \lambda x.\ Q(\mathsf{inj}_{l_1}(x)), l_2 \Rightarrow \lambda x.\ Q(\mathsf{inj}_{l_2}(x))$. Given a type t, it is possible to define a "negation-like" operation on $[l_1{:}\,t, l_2{:}\,t]$ by $\mathsf{neg}(M) = \mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow \lambda x.\mathsf{inj}_{l_2}(x), l_2 \Rightarrow \lambda x.\mathsf{inj}_{l_1}(x)$. Given $x, y{:}\,t$, it is easy enough to define an operation $f(M,N){:}\,t$, for $M,N{:}\,[l_1{:}\,t, l_2{:}\,t]$ in such a way that $f(\mathsf{inj}_{l_1}(u), \mathsf{inj}_{l_1}(u)) = f(\mathsf{inj}_{l_2}(v), \mathsf{inj}_{l_2}(v)) = x$, and $f(\mathsf{inj}_{l_1}(u), \mathsf{inj}_{l_2}(v)) = f(\mathsf{inj}_{l_2}(v), \mathsf{inj}_{l_1}(u)) = y$. We deduce then from the "induction principle" that $f(M,M) = x$, and $f(M, \mathsf{neg}(M)) = y$, identically for $M{:}\,[l_1{:}\,t, l_2{:}\,t]$, hence the (equational) inconsistency when we have a fixed-point combinator.

The fact that we can use instead of {VART-COP?} + {VART-BETA} the weaker system {VART-BETA} + {VART-CRN?} comes simply from the fact that we can "relativise" this reasoning to the elements of $[l_1{:}\,t, l_2{:}\,t]$ of the form $\mathsf{case}\ M\ \mathsf{of}\ \mathsf{inj}_{l_1}\mathsf{inj}_{l_2}$, elements that satisfy the equation { VART-ETA }. ∎

Thus, a naive approach gives us an unattractive choice between incoherence and inconsistency! We are saved from this by the observation that, at least in the example above, we do not seem to need the "full" usage of {VART-CRN} but only those instances in which P is a term coming out of a translation of an inheritance judgement, *i.e.,* a “coercion term”. Such terms are much simpler than general terms. In particular, we note that in models based on continuous maps, such terms denote *strict* maps, and in models based on stable maps, they denote *linear* maps. Appropriate constructions for interpreting variants can be given in both cases, such that {VART-CRN} is sound, as long as P ranges only over strict (or linear) maps.

Maintaining the same philosophy to our approach as in Section 3 we will try to *abstractly* embody in **TARGET** a sufficient amount of formalism to insure the provable coherence of our translation. Thus, the previous discussion of variants leads us to introduce a new type constructor $s \multimap t$, the type of “coercions” from s to t. Consequently, the coercion assumptions $a \leq t$ that occur in inheritance contexts must translate to variables ranging over types of coercions $f\colon a \multimap t^*$. As a consequence, the translation of bounded quantification must change:

$$(\forall a \leq s.\, t)^* \overset{\text{def}}{=} \forall a.\, ((a \multimap s^*) \rightarrow t^*)$$

In order to express the correct versions of {VART-CRN}, we introduce a family of constants in **TARGET**

$$\iota_{s,t} : (s \multimap t) \rightarrow (s \rightarrow t)$$

called *coercion-coercion combinators*. With this, we have

$$\{\text{VART-CRN}\}\quad \iota(P)(\mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow F_1, \ldots, l_n \Rightarrow F_n) \;=\; \mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow F_1; \iota(P), \ldots, l_n \Rightarrow F_n; \iota(P)$$

$$\text{where}\quad M\colon [l_1\colon t_1, \ldots, l_n\colon t_n],\ F_1\colon t_1 \rightarrow t, \ldots, F_n\colon t_n \rightarrow t,\ P\colon t \multimap s\,.$$

(the complete list is in Appendix B).

In order to translate all inheritance judgements into coercion terms, we add a special set of constants (coercion combinators) that “compute” the translations of the rules for deriving inheritance judgements. To prove coherence, we axiomatize the behavior of the ι-images of these combinators. For example, the coercion combinator for the rule (ARROW) takes a pair of coercions as arguments and yields a new coercion as value:

$$\mathsf{arrow}[s, t, u, v] : (s \multimap t) \rightarrow (u \multimap v) \rightarrow ((t \rightarrow u) \multimap (s \rightarrow v))$$

Since (ARROW) is a rule *scheme*, we naturally have a *family* of such combinators, indexed by types. To simplify the notation, these types will be omitted whenever possible. The equational property of the arrow combinator is given in terms of the coercion coercer:

$$\iota(\mathsf{arrow}(P)(Q)) \;=\; \lambda z\colon t \rightarrow u.\, (\iota(P)); z; (\iota(Q))$$

where $P\colon s \multimap t,\ Q\colon u \multimap v$. For the rule (TRANS), we introduce

$$\mathsf{trans}[r, s, t] : (r \multimap s) \rightarrow (s \multimap t) \rightarrow (r \multimap t)$$

which, of course, behaves like composition, modulo the coercion coercer:

$$\iota(\mathsf{trans}(P)(Q)) \;=\; \iota(P); \iota(Q)$$

where $P\colon r \mathrel{\circ\!\!\rightarrow} s,\ Q\colon s \mathrel{\circ\!\!\rightarrow} t$. The combinator for the rule (FORALL) is the most involved:

$$\mathsf{forall}[s,t,a,u,v] : (s \mathrel{\circ\!\!\rightarrow} t) \rightarrow \forall a.\, ((a \mathrel{\circ\!\!\rightarrow} s) \rightarrow (u \mathrel{\circ\!\!\rightarrow} v)) \rightarrow (\forall a.\, ((a \mathrel{\circ\!\!\rightarrow} t) \rightarrow u) \mathrel{\circ\!\!\rightarrow} \forall a.\, ((a \mathrel{\circ\!\!\rightarrow} s) \rightarrow v))$$

with the equational axiomatization

$$\iota(\mathsf{forall}(P)(W)) \;=\; \lambda z\colon (\forall a.\, (a \mathrel{\circ\!\!\rightarrow} t) \rightarrow u).\, \Lambda a.\, \lambda f\colon a \mathrel{\circ\!\!\rightarrow} s.\, \iota(W(a)(f))(z(a)(\mathsf{trans}(f)(P)))$$

where $P\colon s \mathrel{\circ\!\!\rightarrow} t,\ W\colon \forall a.\, (a \mathrel{\circ\!\!\rightarrow} s) \rightarrow (u \mathrel{\circ\!\!\rightarrow} v)$. Of course, we have gone to the extra inconvenience of introducing the type of coercions in order to provide a satisfactory account of variants. These require a scheme of combinators having the types:

$$\mathsf{vart}[s_1,\ldots,s_p,t_1,\ldots,t_q] : (s_1 \mathrel{\circ\!\!\rightarrow} t_1) \rightarrow \cdots \rightarrow (s_p \mathrel{\circ\!\!\rightarrow} t_p) \rightarrow ([l_1\colon s_1,\ldots,l_p\colon s_p] \mathrel{\circ\!\!\rightarrow} [l_1\colon t_1,\ldots,l_p\colon t_p,\ldots,l_q\colon t_q])$$

And it is now possible to assert a consistent equation for these combinators:

$$\iota(\mathsf{vart}(R_1)\cdots(R_p)) \;=\; \lambda w\colon [l_1\colon s_1,\ldots,l_p\colon s_p].\, \mathsf{case}\ w\ \mathsf{of}\ l_1 \Rightarrow \iota(R_1); \mathsf{inj}_{l_1}, \ldots, l_p \Rightarrow \iota(R_p); \mathsf{inj}_{l_p}$$

where $R_1\colon s_1 \mathrel{\circ\!\!\rightarrow} t_1, \ldots, R_p\colon s_p \mathrel{\circ\!\!\rightarrow} t_p$. In order to prove equalities between terms of coercion type one uses the following rule:

{IOTA-INJ} $$\frac{\iota(P) \;=\; \iota(Q)}{P \;=\; Q}$$

which asserts that ι is an injection. In fact, all of the models we give below will interpret ι as an inclusion. It is natural to ask whether the coercion coercer ι could have been omitted from the calculus in favor of a rule:

$$\frac{P\colon s \mathrel{\circ\!\!\rightarrow} t}{P\colon s \rightarrow t}.$$

This would have the unfortunate consequence that a typing judgement $e\colon s$ would no longer uniquely encode its proof and the coherence question would therefore arise again! The other combinators and their equational properties are described in Appendix B.

We are now ready to explain how to translate our full language **SOURCE** (complete with variants) into the language **TARGET** (with the coercion coercer and combinators). For starters, the inheritance judgement for the function space is simply translated using the arrow combinator:

(ARROW)* $$\frac{C^* \;\vdash\; P : s^* \mathrel{\circ\!\!\rightarrow} t^* \qquad C^* \;\vdash\; Q : u^* \mathrel{\circ\!\!\rightarrow} v^*}{C^* \;\vdash\; \mathsf{arrow}(P)(Q) : (t^* \rightarrow u^*) \mathrel{\circ\!\!\rightarrow} (s^* \rightarrow v^*)}$$

The translation of an inheritance between quantified types takes the induced coercion and a polymorphic function as its arguments:

(FORALL)* $$\frac{C^* \;\vdash\; P : s^* \mathrel{\circ\!\!\rightarrow} t^* \qquad C^*,\, a,\, f\colon a \mathrel{\circ\!\!\rightarrow} s^* \;\vdash\; Q : u^* \mathrel{\circ\!\!\rightarrow} v^*}{C^* \;\vdash\; \mathsf{forall}(P)(\Lambda a.\, \lambda f\colon a \mathrel{\circ\!\!\rightarrow} s^*.\, Q) : \forall a.\, ((a \mathrel{\circ\!\!\rightarrow} t^*) \rightarrow u^*) \mathrel{\circ\!\!\rightarrow} \forall a.\, ((a \mathrel{\circ\!\!\rightarrow} s^*) \rightarrow v^*)}$$

Other inheritance judgements are similarly translated. The real work is being done by equational properties of the combinators.

The proofs of typing judgements are translated in a manner quite similar to how they were translated in the fragment. For example,

[B-SPEC]*
$$\frac{\Gamma^* \vdash M : \forall a.\,((a \mathrel{\circ\!\!\rightarrow} s^*) \rightarrow t^*) \qquad \widehat{\Gamma}^* \vdash P : r^* \mathrel{\circ\!\!\rightarrow} s^*}{\Gamma^* \vdash M(r^*)(P) : [r^*/a]t^*}$$

is affected only by indicating that the map into the bound must be a coercion. The inheritance rule is translated by

[INH]*
$$\frac{\Gamma^* \vdash M : s^* \qquad \widehat{\Gamma}^* \vdash P : s^* \mathrel{\circ\!\!\rightarrow} t^*}{\Gamma^* \vdash \iota(P)(M) : t^*}$$

since a coercion cannot be applied until it is made into a function by an application of the coercion coercer. The full description of the translation of the full language is given in Appendix C. We now turn to the proof of the central technical results of the paper.

5 Coherence of the translation for the full calculus

In this section we prove first the coherence of the translation of inheritance judgements. This result is then used to show the coherence of the translation of typing judgements.

The main cause for having distinct derivations of the same inheritance judgements is the rule (TRANS). Our strategy is to show that the usage of (TRANS) can be coherently postponed to the end of derivations (Lemma 6), and then to prove the coherence of the translation of (TRANS)-postponed derivations (Lemma 8).

We introduce some convenient notations for the rest of this section. For any derivation Δ in **SOURCE**, let Δ^* be the **TARGET** derivation into which it is translated. We will write $C \vdash r_0 \leq \cdots \leq r_n$ instead of $C \vdash r_0 \leq r_1, \ldots, C \vdash r_{n-1} \leq r_n$. The composition of coercions given by trans occurs so often that we will write $P \odot Q$ instead of $\mathsf{trans}(P)(Q)$. It is easy to see, making essential use of the rule {IOTA-INJ}, that $\odot$ is provably *associative*. We will take advantage of this to unclutter the notation. We will also write I instead of refl. Again it is easy to see that I is provably an identity for $\odot$, that is, $I \odot M = M \odot I = M$ is provable in **TARGET**.

Lemma 6 *For any* **SOURCE** *derivation Δ yielding the inheritance judgement $C \vdash s \leq t$, there exist types $r_0, \ldots, r_n$ such that $s \equiv r_0$, $r_n \equiv t$, and (TRANS)-free derivations $\Delta_1, \ldots, \Delta_n$ yielding respectively*

$$C \vdash r_0 \leq \cdots \leq r_n$$

Moreover, if the translations $\Delta^*, \Delta_1^*, \ldots, \Delta_n^*$ yield respectively the (coercion) terms $C^* \vdash P : s^* \mathrel{\circ\!\!\rightarrow} t^*$, $C^* \vdash P_1 : r_0^* \mathrel{\circ\!\!\rightarrow} r_1^*, \ldots, C^* \vdash P_n : r_{n-1}^* \mathrel{\circ\!\!\rightarrow} r_n^*$ then

$$C^* \vdash P = P_1 \odot \cdots \odot P_n$$

is provable in **TARGET**.

Proof: By induction on the height of the derivation Δ. The base is trivial since derivations consisting of instances of (TOP), (VAR), or (REFL) are already (TRANS)-free. We present the more interesting cases of the induction step.

Suppose Δ ends with an application of (ARROW). By induction hypothesis there are (TRANS)-free derivations for

$$s \equiv r_0 \ \leq \ \cdots \ \leq \ r_m \equiv t \quad \text{and} \quad u \equiv w_0 \ \leq \ \cdots \ \leq \ w_n \equiv v$$

(for simplicity, we omit the context). From these, using (REFL) and (ARROW) we get (TRANS)-free derivations for

$$t \rightarrow u \equiv r_m \rightarrow u \ \leq \ \cdots \ \leq \ r_0 \rightarrow u \equiv s \rightarrow w_0 \ \leq \ \cdots \ \leq \ s \rightarrow w_n \equiv s \rightarrow v \ .$$

(This is not most economical: one can get a derivation requiring only $\max(m, n)$, rather than $m + n$, steps of (TRANS) at the end.) Proving the equality of the corresponding translations uses the associativity of $\odot$ and the fact that I acts like an identity, as well as

$$\text{(1)} \qquad \mathsf{arrow}(P)(Q) \odot \mathsf{arrow}(R)(S) \ = \ \mathsf{arrow}(R \odot P)(Q \odot S)$$

which can be verified, in view of {IOTA-INJ}, by applying ι to both sides, resulting in a simple {BETA}-conversion.

Suppose Δ ends with an application of (FORALL). By induction hypothesis there are (TRANS)-free derivations for

$$C \ \vdash \ s \equiv r_0 \ \leq \ \cdots \ \leq \ r_m \equiv t \quad \text{and} \quad C, a {\leq} s \ \vdash \ u \equiv w_0 \ \leq \ \cdots \ \leq \ w_n \equiv v$$

From these, using (REFL) and (FORALL) we get (TRANS)-free derivations for

$$C \ \vdash \ \forall a {\leq} t.u \equiv \forall a {\leq} r_m.u \ \leq \ \cdots \ \leq \ \forall a {\leq} r_0.u \equiv \forall a {\leq} s.u \equiv \forall a {\leq} s.w_0 \ \leq \ \cdots \ \leq \ \forall a {\leq} s.w_n \equiv \forall a {\leq} s.v \ .$$

Proving the equality of the corresponding translations uses

$$\begin{aligned} \text{(2)} \qquad & \mathsf{forall}(P)(\Lambda a.\, \lambda f{:}\, a \multimap s.\, Q) \odot \mathsf{forall}(R)(\Lambda a.\, \lambda g{:}\, a \multimap t.\, S) \ = \\ & = \ \mathsf{forall}(R \odot P)(\Lambda a.\, \lambda g{:}\, a \multimap t.\, [g \odot R/f]Q \odot S) \end{aligned}$$

and which can be verified by applying ι to both sides.

Suppose Δ ends with an application of (VART). By induction hypothesis there are (TRANS)-free derivations for

$$\begin{gathered} s_1 \equiv r_0^1 \ \leq \ \cdots \ \leq \ r_{n_1}^1 \equiv t_1 \\ \vdots \\ s_p \equiv r_0^p \ \leq \ \cdots \ \leq \ r_{n_p}^p \equiv t_p \end{gathered}$$

(for simplicity, we omit the context). From these, using (REFL) and (VART) we get (TRANS)-free derivations for

$$[l_1{:}\, s_1, \ldots, l_p{:}\, s_p] \equiv [l_1{:}\, r_0^1, \ldots, l_p{:}\, s_p] \ \leq \ \cdots \ \leq \ [l_1{:}\, r_{n_1}^1, \ldots, l_p{:}\, s_p] \ \leq \ \cdots \ \leq \ [l_1{:}\, r_{n_1}^1, \ldots, l_p{:}\, r_0^p] \ \leq \ \cdots$$

$$\cdots \;\leq\; [l_1\!: r^1_{n_1}, \ldots, l_p\!: r^p_{n_p}] \equiv [l_1\!: t_1, \ldots, l_p\!: t_p] \;\leq\; [l_1\!: t_1, \ldots, l_p\!: t_p, \ldots, l_q\!: t_q] \;.$$

Proving the equality of the corresponding translations uses

$$\text{(3)} \qquad \mathsf{vart}(P_1)\cdots(P_p) \odot \mathsf{vart}(Q_1)\cdots(Q_q) \;=\; \mathsf{vart}(P_1 \odot Q_1)\cdots(P_p \odot Q_p) \quad (p \leq q).$$

To verify this, let L be the left hand side of the equation, R the right hand side and let w be a fresh variable. By extensionality (or {ETA} and {XI}) and by {IOTA-INJ}, it is sufficient to show $\iota(L)(w) \;=\; \iota(R)(w)$. By {VART-COP}, this follows from

$$\mathsf{case}\ w\ \mathsf{of}\ l_1 \Rightarrow (\mathsf{inj}_{l_1}; \iota(L)), \ldots, l_p \Rightarrow (\mathsf{inj}_{l_p}; \iota(L)) \;=\; \mathsf{case}\ w\ \mathsf{of}\ l_1 \Rightarrow (\mathsf{inj}_{l_1}; \iota(R)), \ldots, l_p \Rightarrow (\mathsf{inj}_{l_p}; \iota(R))$$

which is readily verified.

When Δ ends with (TRANS), we just concatenate the chains of (TRANS)-free derivations and the equality of the translations is an immediate consequence of the associativity of $\odot$. ▌

The following is used to handle one of the cases in Lemma 8 below.

Lemma 7 *For any two derivations, Δ yielding $C \vdash s \leq t$ and Θ yielding $C, a \leq t \vdash u \leq v$, there exists a derivation Σ yielding $C, a \leq s \vdash u \leq v$ such that $\mathit{height}(\Sigma) = \max(\mathit{height}(\Delta), \mathit{height}(\Theta))$. Moreover, if the translations $\Delta^*, \Theta^*, \Sigma^*$ yield respectively*

$$C^* \vdash P : s^* \multimap t^* \;,\; C^*,\, a,\, g\!: a \multimap t^* \vdash Q : u^* \multimap v^* \;,\; C^*,\, a,\, f\!: a \multimap s^* \vdash R : u^* \multimap v^*$$

then

$$C^*,\, a,\, f\!: a \multimap s^* \vdash R \;=\; [f \odot P/g]Q$$

is provable in **TARGET**.

Proof: By induction on the height of Θ. ▌

Lemma 8 *Let $\Delta_1, \ldots, \Delta_m$ be (TRANS)-free derivations in* **SOURCE** *yielding respectively $C \vdash s_0 \leq \cdots \leq s_m$ and $\Theta_1, \ldots, \Theta_n$ be (TRANS)-free derivations yielding respectively $C \vdash t_0 \leq \cdots \leq t_n$. Let the translations $\Delta^*_1, \ldots, \Delta^*_m, \Theta^*_1, \ldots, \Theta^*_n$ yield respectively the (coercion) terms*

$$C^* \vdash P_1 : s^*_0 \multimap s^*_1 \;, \ldots,\; C^* \vdash P_m : s^*_{m-1} \multimap s^*_m \;,\; C^* \vdash Q_1 : t^*_0 \multimap t^*_1 \;, \ldots,\; C^* \vdash Q_n : t^*_{n-1} \multimap t^*_n \;.$$

If $s_0 \equiv t_0$ and $s_m \equiv t_n$ then

$$C^* \vdash P_1 \odot \cdots \odot P_m \;=\; Q_1 \odot \cdots \odot Q_n$$

is provable in **TARGET**.

Proof: We begin with the following remarks:

- If one of $s_0, \ldots, s_m, t_0, \ldots, t_n$ is *Top* then the desired equality holds. Indeed, then $s_m \equiv \mathit{Top} \equiv t_n$ and the equality follows from the identity

  $$\text{(4)} \qquad P \;\leq\; \mathsf{top}$$

  which is verified by applying ι to both sides (recall that **1** is a terminator).

- Those derivations among $\Delta_1, \ldots, \Delta_m, \Theta_1, \ldots, \Theta_n$ which consist entirely of one application of (REFL) can be eliminated without loss of generality. Indeed, the corresponding coercion term is I which acts as an identity for $\odot$.

- If none of the derivations among $\Delta_1, \ldots, \Delta_m, \Theta_1, \ldots, \Theta_n$ consists of just (TOP), then those derivations which consist of just (VAR) can also be eliminated without loss of generality. Indeed, once we have eliminated the (REFL)'s, the (VAR)'s must form an initial segment of both $\Delta_1, \ldots, \Delta_m$ and $\Theta_1, \ldots, \Theta_n$ because whenever $s \leq a$ is derivable, s must also be a type variable. Let's say that $s_0 \equiv a_0, \ldots, s_p \equiv a_{p-1}$, $(p \leq m)$, where $\Delta_1, \ldots, \Delta_p$ are *all* the derivations consisting of just (VAR), and also that $t_0 \equiv b_0, \ldots, t_q \equiv b_{q-1}$, $(q \leq n)$, where $\Theta_1, \ldots, \Theta_q$ are all the derivations consisting of just of (VAR). Then, $a_0 \leq a_1, \ldots, a_{p-1} \leq s_p$ as well as $b_0 \leq b_1, \ldots, b_{q-1} \leq t_q$ must all occur in C. But $a_0 \equiv s_0 \equiv t_0 \equiv b_0$ so by the uniqueness of declarations in contexts, $a_1 \equiv b_1, \ldots$, *etc.* Suppose $p < q$. Then, $s_p \equiv b_p$ is a variable. Since Δ_{p+1} can't be just a (REFL) or a (TOP) is must be a (VAR) contradicting the maximality of p. Thus $p = q$ and $s_p \equiv t_q$ and the (VAR)'s can be eliminated.

We proceed to prove the lemma by induction on the maximum of the heights of the derivations $\Delta_1, \ldots, \Delta_m, \Theta_1, \ldots, \Theta_n$. The basis of the induction is an immediate consequence of the remarks above.

For the induction step, in the view of the remarks above, we can assume without loss of generality that none of the derivations is just a (TOP), (VAR), or (REFL). Consequently, $\Delta_1, \ldots, \Delta_m, \Theta_1, \ldots, \Theta_n$ must all end with the same rule, depending on the type construction used in $s_0 \equiv t_0$.

If all derivations end in (ARROW), the desired equality follows from the induction hypothesis, the associativity of $\odot$ and the equation (1). Similarly for (VART) using the equation (3). The desired equality in the case (FORALL) follows from the induction hypothesis using Lemma 7, from the associativity of $\odot$ and from the equation (2). The remaining cases are straight-forward. ▌

This gives us the coherence of the translation of inheritance judgements. To state it we need some terminology. We say that two **SOURCE** derivations which yield the same judgement are *congruent* if their translations in **TARGET** yield provably equal terms. We will write $\Delta_1 \cong \Delta_2$ for congruence of derivations. It is easy to check that $\cong$ *is* in fact a congruence with respect to the operations on derivations induced by the rules.

Lemma 9 (Coherence of the translation of inheritance) *If Δ_1 and Δ_2 are two* **SOURCE** *derivations yielding the same inheritance judgement then $\Delta_1 \cong \Delta_2$ (their translations yield provably equal terms in* **TARGET***).*

Proof: Immediate consequence of Lemmas 6 and 8 ▌

Before we turn to the coherence of the translation of typing judgements, we will note a few facts about inheritance judgements that follow from Lemma 6 and that will be invoked subsequently. These facts are closely related to the remarks opening the proof of Lemma 8.

Remark 10 *If $C \vdash s \leq t$ is derivable, $s \equiv a$, a type variable, and $t \not\equiv a$ then*

- *if* $t \equiv b$ *, also a type variable, there must exist type variables* $a_0, \ldots, a_n$, $n \geq 1$ *such that* $a \equiv a_0$, $b \equiv a_n$ *, and* $a_{i-1} \leq a_i \in C$,$i = 1, \ldots, n$ *;*
- *if t is not a type variable, there must exist type variables* $a_0, \ldots, a_n$, $n \geq 0$ *and a type u such that* $a \equiv a_0$, $a_{i-1} \leq a_i \in C$,$i = 1, \ldots, n$, $a_n \leq u \in C$ *, and* $C \vdash u \leq t$ *(of course, this is trivial when* $t \equiv Top$ *);*

If $C \vdash s \leq t$ *is derivable, and s is not a type variable variable, then t cannot be a type variable, and if moreover* $t \not\equiv Top$ *, then s and t must both have the "same" outermost type constructor (as detailed exhaustively below) and*

- *if* $s \equiv s_1 \rightarrow s_2$ *and* $t \equiv t_1 \rightarrow t_2$ *then* $C \vdash t_1 \leq s_1$ *and* $C \vdash s_2 \leq t_2$ *;*
- *if* $s \equiv \{l_1 : s_1, \ldots, l_q : s_q\}$ *and* $t \equiv \{l_1 : t_1, \ldots, l_p : t_p\}$ *then* $p \leq q$ *and* $C \vdash s_1 \leq t_1$, ... ,$C \vdash s_p \leq t_p$ *;*
- *if* $s \equiv \forall a \leq s_1 . s_2$ *and* $t \equiv \forall a \leq t_1 . t_2$ *then* $C \vdash t_1 \leq s_1$ *and* $C, a \leq t_1 \vdash s_2 \leq t_2$ *;*
- *if s and t are both recursive types then they must be identical;*
- *if* $s \equiv [l_1 : s_1, \ldots, l_p : s_p]$ *and* $t \equiv [l_1 : t_1, \ldots, l_q : t_q]$ *then* $p \leq q$ *and* $C \vdash s_1 \leq t_1$, ... ,$C \vdash s_p \leq t_p$.

We turn now to the coherence of the translation of typing judgements, which is the central technical result of the paper. As explained in section 3, we weaken the system by replacing the rule (FORALL) with (W-FORALL) (see Appendix A). With this, we have the following order-theoretic property about the inheritance judgments, which fails in the presence of (FORALL). The property asserts the existence of conditional greatest lower bounds and of least upper bounds.

Lemma 11 *Replace (FORALL) with (W-FORALL). Let C be an inheritance context and let* t_1, t_2 *be types.*

1. *If there is an r with* $C \vdash r \leq t_i$, $(i = 1, 2)$ *, then there exists a type* $t_1 \sqcap t_2$ *such that*
   - $C \vdash t_1 \sqcap t_2 \leq t_i$, $(i = 1, 2)$ *and*
   - *for any s such that* $C \vdash s \leq t_i$, $(i = 1, 2)$ *we have* $C \vdash s \leq t_1 \sqcap t_2$. ▌
2. *There is a type* $t_1 \sqcup t_2$ *such that*
   - $C \vdash t_i \leq t_1 \sqcup t_2$, $(i = 1, 2)$ *and*
   - *for any s such that* $C \vdash t_i \leq s$, $(i = 1, 2)$ *we have* $C \vdash t_1 \sqcup t_2 \leq s$. ▌

Proof: Because of the contravariance property of the first argument of the function space operator manifest in the rule (ARROW), we will prove items 1 and 2 simultaneously. In view of Lemma 6, it is sufficient to work with proofs where all instances of (TRANS) appear at the end. Since moreover any two types have a common upper bound, *Top*, the statement of the lemma is equivalent to the following formulation:

For any $\Delta_1, \ldots, \Delta_m$*, (TRANS)-free derivations in* **SOURCE** *yielding respectively* $C \vdash u_0 \leq \cdots \leq u_m$ *and any* $\Theta_1, \ldots, \Theta_n$*, (TRANS)-free derivations yielding respectively* $C \vdash v_0 \leq \cdots \leq v_n$,

1. *if $u_0 \equiv v_0$, and let $t_1 \equiv u_m$ and $t_2 \equiv v_n$, then there is a type $t_1 \sqcap t_2$ having the properties in item 1 of the lemma;*
2. *if $u_m \equiv v_n$, and let $t_1 \equiv u_0$ and $t_2 \equiv v_0$, then there is a type $t_1 \sqcup t_2$ having the properties in item 2 of the lemma.*

This is shown by induction on the maximum of m, n and of the heights of $\Delta_1, \ldots, \Delta_m, \Theta_1, \ldots, \Theta_n$. To be able to apply the induction hypothesis, a case analysis is performed, depending on the structure of t_1 and t_2. We will only look at a few illustrative cases. The facts listed in Remark 10 and the reasoning that produced these facts as well as the remarks opening the proof of Lemma 8 are used throughout.

For example, if t_1 is a type variable in item 1, then u_i is also a type variable for each i, and $u_{i-1} \le u_i \in C$, $i = 1, \ldots, n$. Then, one of $C \vdash u_0 \le \cdots \le u_m$ or $C \vdash v_0 \le \cdots \le v_n$, must be an initial segment of the other, so t_1 and t_2 are comparable and $t_1 \sqcap t_2$ can be taken as the smaller among them. For item 2, if t_1 is a type variable, then $u_0 \le u_1 \in C$ and, by induction hypothesis (m decreases), $t_1 \sqcup t_2$ can be taken to be $u_1 \sqcup t_2$.

As another example, suppose that in item 1 t_1 has the form $\forall a \le s.\, r_1$. If $u_0 \equiv v_0$ is a type variable, then $u_0 \le u_1 \in C$ and $v_0 \le v_1 \in C$ hence $u_1 \equiv v_1$ and we can apply the induction hypothesis by eliminating Δ_1, Θ_1. Assume that $u_0 \equiv v_0$ is not a type variable. By Remark 10 (simplified to take into account the weakening of (FORALL)), it must have the form $\forall a \le s.\, r$. Again by Remark 10 t_2 is either *Top* or has the form $\forall a \le s.\, r_2$. If $t_2 \equiv \mathit{Top}$ then $t_1 \sqcap t_2$ can be taken to be t_1. Otherwise, there are (TRANS)-free derivations $\Delta'_1, \ldots, \Delta'_m$ yielding $C, a \le s \vdash u'_0 \le \cdots \le u'_m$ and $\Theta'_1, \ldots, \Theta'_n$ yielding respectively $C, a \le u \vdash v'_0 \le \cdots \le v'_n$ where $u'_0 \equiv v'_0$ and $u'_m \equiv r_1$ and $v'_n \equiv r_2$, and where each of these derivations has strictly smaller height than the corresponding one among $\Delta_1, \ldots, \Delta_m, \Theta_1, \ldots, \Theta_n$. By induction hypothesis we get a type $r_1 \sqcap r_2$, and we can then take $t_1 \sqcap t_2$ to be $\forall a \le s.\, r_1 \sqcap r_2$. This calculation makes clear where our proof breaks down if we were to use the more general rule (FORALL) instead of (W-FORALL). Indeed, if the bounds on the type variables were allowed to differ, as in the more general case, we would be unable to apply the induction hypothesis since the two contexts would differ between the Θ's and the Δ's.

We omit the remaining cases, which use similar ideas. ▌

We will use this property in the proof of Lemma 12, which is a slightly stronger result than the actual coherence of the translation of typing judgements. Of course, the strengthening is exploited in a proof by induction. First we introduce a definition and more convenient notations. For derivations yielding typing judgements we define the ***essential height*** which is computed as the usual height, with the proviso that [INH] and the rules yielding inheritance judgements do *not* increase it. We will also use a special notation for describing "composition" of derivations via the rules. We explain this notation through two examples. If Σ yields $\Gamma \vdash e : s$ and Θ yields $\widehat{\Gamma} \vdash s \le t$, then $[\mathrm{INH}]\langle \Sigma, \Theta \rangle$ yields $\Gamma \vdash e : t$. If Δ yields $\Gamma, x{:}s \vdash e : t$ then $[\mathrm{ABS}]\langle \Delta \rangle$ yields $\Gamma \vdash \lambda x{:}s.\, e : s \to t$.

In preparation for the proof of the next lemma, we have two remarks.

- We have the following congruence

$$[\mathrm{INH}]\langle [\mathrm{INH}]\langle \Sigma, \Theta_1 \rangle, \Theta_2 \rangle \cong [\mathrm{INH}]\langle \Sigma, (\mathrm{TRANS})\langle \Theta_1, \Theta_2 \rangle \rangle \ .$$

This follows from the fact that $\iota(Q)(\iota(P)(M)) \;=\; \iota(P \odot Q)(M)$ which is immediately verified.

- Any **SOURCE** derivation is congruent to a derivation of the form $[\text{INH}]\langle \Delta , \Theta\rangle$ where Δ does *not* end with an application of the [INH] rule. This follows from the previous remark and, in the case when the original derivation did not end in [INH], from

$$\Delta \cong [\text{INH}]\langle \Delta , (\text{REFL})\rangle$$

which in turn follows from $M \;=\; \iota(I)(M)$.

Lemma 12 *Replace (FORALL) with (W-FORALL). For any two* **SOURCE** *derivations, Δ_i yielding $\Gamma \;\vdash\; e \,:\, t_i$, $(i = 1,2)$, there exists a type s, a derivation Σ yielding $\Gamma \;\vdash\; e \,:\, s$ and two derivations Θ_i yielding $\widehat{\Gamma} \;\vdash\; s \;\leq\; t_i$, $(i = 1,2)$ such that*

$$\Delta_i \cong [\text{INH}]\langle \Sigma , \Theta_i\rangle \;,\;\; (i = 1,2).$$

Proof: By induction on the maximum of the essential heights of Δ_1, Δ_2. In view of the previous remarks, it is sufficient to prove the statement of the lemma assuming that neither Δ_1 nor Δ_2 ends in [INH] (but we retain the actual statement of the lemma in the induction hypothesis). For such derivations, Δ_1 and Δ_2 must end with the same rule (which rule, depends on the structure of e). We do a case analysis according to this last rule, and we include here only the cases which we believe are important for the understanding of the result (even if their treatment is straightforward) as well as some cases which are particularly complex. We will call the type s , whose existence is the essence of the result, the *common type*.

Rule [VAR]. It must be the case that $t_1 \equiv t_2 \equiv r$ where $x\!:\!r$ occurs in Γ. Consequently, the treatment of this rule is trivial: take the common type to be r , $\Sigma \equiv [\text{VAR}]$, and $\Theta_1 \equiv \Theta_2 \equiv (\text{REFL})$.

The introduction rules are quite simple and we illustrate them with the **rule** [ABS]. Suppose that $\Delta_i \equiv [\text{ABS}]\langle \Delta_i' \rangle$ and that Δ_i yields $\Gamma \;\vdash\; \lambda x\!:\!s.\, e \,:\, s \to t_i$ (s is the same since it appears in the term), thus Δ_i' yields $\Gamma,\, x\!:\!s \;\vdash\; e \,:\, t_i$, $(i = 1,2)$. Apply the induction hypothesis to Δ_1', Δ_2' obtaining $r, \Sigma', \Theta_1', \Theta_2'$. Also by induction hypothesis,

$$\Delta_i \cong [\text{ABS}]\langle [\text{INH}]\langle \Sigma' , \Theta_i'\rangle \rangle \;,\;\; (i = 1,2).$$

We claim that the right hand side is congruent to

$$[\text{INH}]\langle [\text{ABS}]\langle \Sigma' \rangle , (\text{ARROW})\langle (\text{REFL}) , \Theta_i'\rangle\rangle \,.$$

This implies that the statement of the lemma holds for Δ_1, Δ_2, with common type $s \to r$, with $\Sigma \equiv [\text{ABS}]\langle \Sigma' \rangle$, and with $\Theta_i \equiv (\text{ARROW})\langle (\text{REFL}) , \Theta_i'\rangle$, $(i = 1,2)$. The congruence claim follows from

$$\lambda x\!:\!s.\, \iota(P)(M) \;=\; \iota(\mathsf{arrow}(I)(P)(\lambda x\!:\!s.\, M)$$

which is readily verified.

Rule[B-SPEC]. To simplify the notation, we omit the contexts. Suppose that $\Delta_i \equiv [\text{B} - \text{SPEC}]\langle \Delta_i' , \Xi_i\rangle$ and that Δ_i yields $e(r) \,:\, [r/a]t_i$ (r is the same since it appears in the term and we can take the bound variable to be the same without loss of generality), thus Δ_i' yields $e : \forall a \leq s_i .\, t_i$ and Ξ_i yields $r \ \leq \ s_i$, $(i = 1,2)$. Apply the induction hypothesis to Δ_1', Δ_2' obtaining $w, \Sigma', \Theta_1', \Theta_2'$. Also by induction hypothesis,

$$(5) \qquad \Delta_i \cong [\mathrm{B-SPEC}] \langle\, [\mathrm{INH}] \langle\, \Sigma' \,,\, \Theta_i' \rangle \,,\, \Xi_i \rangle \ , \ (i = 1,2).$$

Since $w \ \leq \ \forall a \leq s_i .\, t_i$, $(i = 1,2)$ it follows from Remark 10 (simplified to take into account the weakening of (FORALL)) that there must exist types u, v such that $s_i \equiv u$, $a \leq s_i \ \vdash \ v \ \leq \ t_i$, $(i = 1,2)$ and $w \ \leq \ \forall a \leq u .\, v$ are derivable. It follows that $r \ \leq \ u$, and, by Lemma 7, that $a \leq r \ \vdash \ v \ \leq \ t_i$, $(i = 1,2)$ are derivable. Next, we will use the following sublemma:

> **Sublemma** For any derivation Δ yielding $C, a \leq r \ \vdash \ s \ \leq \ t$ there exists a derivation Σ yielding $C \ \vdash \ [r/a]s \ \leq \ [r/a]t$ such that, if the translations Δ^*, Σ^* yield respectively
>
> $$C^*,\, a,\, f\!: a \mathrel{\circ\!\!\rightarrow} r^* \ \vdash \ P : s^* \mathrel{\circ\!\!\rightarrow} t^* \ , \ C^* \ \vdash \ Q : [r^*/a]s^* \mathrel{\circ\!\!\rightarrow} [r^*/a]t^*$$
>
> then
>
> $$C^* \ \vdash \ Q \ = \ (\Lambda a.\, \lambda f\!: a \mathrel{\circ\!\!\rightarrow} r^* .\, P)(r^*)(I)$$
>
> is provable in **TARGET.** ▮

The sublemma is proved by induction on the height of Δ and is omitted. The sublemma allows us to obtain $[r/a]v \ \leq \ [r/a]t_i$ from $a \leq r \ \vdash \ v \ \leq \ t_i$, $(i = 1,2)$. Let Θ_i be some derivation of $[r/a]v \ \leq \ [r/a]t_i$, $(i = 1,2)$. Let Ξ be some derivation of $r \ \leq \ u$. Let Ω be some derivation of $w \ \leq \ \forall a \leq u .\, v$. One can readily verify that the right hand side of (5) is congruent to

$$[\mathrm{INH}] \langle\, [\mathrm{B-SPEC}] \langle\, [\mathrm{INH}] \langle\, \Sigma' \,,\, \Omega \rangle \,,\, \Xi \rangle \,,\, \Theta_i \rangle$$

This implies that the statement of the lemma holds for Δ_1, Δ_2, with common type $[r/a]v$, with $\Sigma \equiv [\mathrm{B-SPEC}] \langle\, [\mathrm{INH}] \langle\, \Sigma' \,,\, \Omega \rangle \,,\, \Xi \rangle$, and with Θ_i being just Θ_i, $(i = 1,2)$. (**Note.** There is no difficulty in dealing with (FORALL) instead of (W-FORALL) here: $s_i \equiv u$ would be simply replaced by $s_i \ \leq \ u$.)

Rule[R-ELIM]. Suppose that $\Delta_i \ \equiv \ [\mathrm{R-ELIM}] \langle\, \Delta_i' \,\rangle$ and that Δ_i yields $\Gamma \ \vdash \ \mathsf{elim}\ e \ : \ [\mu a_i .\, t_i / a_i] t_i$, thus Δ_i' yields $\Gamma \ \vdash \ e \ : \ \mu a_i .\, t_i$, $(i = 1,2)$;. Apply the induction hypothesis to Δ_1', Δ_2' obtaining $s', \Sigma', \Theta_1', \Theta_2'$. Also by induction hypothesis,

$$\Delta_i \cong [\mathrm{R-ELIM}] \langle\, [\mathrm{INH}] \langle\, \Sigma' \,,\, \Theta_i' \rangle \,\rangle \ , \ (i = 1,2).$$

Since $s' \ \leq \ \mu a_i .\, t_i$, $(i = 1,2)$ are derivable, it follows from Remark 10 that there must exist a, t such that $\mu a_i .\, t_i \equiv \mu a .\, t$, $(i = 1,2)$ and $s' \ \leq \ \mu a .\, t$ are derivable. Let Θ' be any derivation of $s' \ \leq \ \mu a .\, t$. Since by Lemma 9, $\Theta_1' \cong \Theta_2' \cong \Theta'$, the statement of the lemma holds with common type $[\mu a .\, t / a]t$, with $\Sigma \equiv [\mathrm{R-ELIM}] \langle\, [\mathrm{INH}] \langle\, \Sigma' \,,\, \Theta' \rangle \,\rangle$, and with $\Theta_i \equiv (\mathrm{REFL})$, $(i = 1,2)$.

Rule[CASE]. Again, to simplify the notation, we omit the contexts. Suppose that $\Delta_i \ \equiv \ [\mathrm{CASE}] \langle\, \Delta_i' \,,\, \Delta_{1i}' \,, \ldots ,\, \Delta_{ni}' \rangle$ and that Δ_i yields $\mathsf{case}\ e\ \mathsf{of}\ l_1 \Rightarrow f_1, \ldots, l_n \Rightarrow f_n \ : \ t_i$, thus Δ_i'

yields $e \,:\, [l_1\!:t_{1i},\ldots,l_n\!:t_{ni}]$, and Δ'_{ji} yield $f_j \,:\, t_{ji}\to t_i$, $(j = 1,\ldots,n)$, $(i = 1,2)$. Apply the induction hypothesis to Δ'_1, Δ'_2 obtaining $s, \Sigma', \Theta'_1, \Theta'_2$. Also apply the induction hypothesis, to $\Delta'_{j1}, \Delta'_{j2}$ obtaining $s_j, \Sigma'_j, \Theta'_{j1}, \Theta'_{j2}$, $(j = 1,\ldots,n)$. By induction hypothesis,

(6) $\quad \Delta_i \cong [\mathrm{CASE}]\langle\,[\mathrm{INH}]\langle\,\Sigma',\,\Theta'_i\rangle,\,[\mathrm{INH}]\langle\,\Sigma'_1,\,\Theta'_{1i}\rangle,\ldots,\,[\mathrm{INH}]\langle\,\Sigma'_n,\,\Theta'_{ni}\rangle\rangle$, $(i = 1,2)$.

Since $s \;\le\; [l_1\!:t_{1i},\ldots,l_n\!:t_{ni}]$, $(i = 1,2)$ are derivable, it follows again from Remark 10 that there must exist $m \le n$ and types $r_1,\ldots,r_m$ such that $r_1 \;\le\; t_{1i},\ldots,r_m \;\le\; t_{mi}$, $(i = 1,2)$ and $s \;\le\; [l_1\!:r_1,\ldots,l_m\!:r_m]$ are derivable. Again similarly, for each of $j = 1,\ldots,n,$, since $s_j \;\le\; t_{ji}\to t_i$, $(i = 1,2)$ are derivable, there must exist u_j, v_j such that $t_{ji} \;\le\; u_j$ and $v_j \;\le\; t_i$, $(i = 1,2)$ as well as $s_j \;\le\; u_j\to v_j$ are derivable. Thus, we can derive $r_j \;\le\; t_{ji} \;\le\; u_j$,$(j = 1,\ldots,n)$, $(i = 1,2)$. However, the fact that the v_j's may be distinct causes a problem when we want to apply [CASE]. This is resolved by Lemma 11. Since $n \ge 1$, there exists a common lower bound of t_1 and t_2 (say v_1) hence $v \equiv t_1 \sqcap t_2$ exists and we can derive $v_j \;\le\; v \;\le\; t_i$,$(j = 1,\ldots,n)$, $(i = 1,2)$. We conclude that there exists a derivation Θ'' of $s \;\le\; [l_1\!:u_1,\ldots,l_n\!:u_n]$, that there exist derivations Θ''_j of $s_j \;\le\; u_j\to v$, $(j = 1,\ldots,n)$ and that there exist derivations Θ_i of $v \;\le\; t_i$, $(i = 1,2)$. With these, we claim that the right hand side of (6) is congruent to

$$[\mathrm{INH}]\langle\,[\mathrm{CASE}]\langle\,[\mathrm{INH}]\langle\,\Sigma',\,\Theta''\rangle,\,[\mathrm{INH}]\langle\,\Sigma'_1,\,\Theta''_1\rangle,\ldots,\,[\mathrm{INH}]\langle\,\Sigma'_n,\,\Theta''_n\rangle\rangle,\,\Theta_i\rangle\,,$$

This implies that the statement of the lemma holds for Δ_1, Δ_2, with common type v , with $\Sigma \equiv [\mathrm{CASE}]\langle\,[\mathrm{INH}]\langle\,\Sigma',\,\Theta''\rangle,\,[\mathrm{INH}]\langle\,\Sigma'_1,\,\Theta''_1\rangle,\ldots,\,[\mathrm{INH}]\langle\,\Sigma'_n,\,\Theta''_n\rangle\rangle$, and with Θ_i being just Θ_i, $(i = 1,2)$.

To prove the congruence claim we introduce notations for certain derivations of inheritance judgements whose existence we have established. For each $j = 1,\ldots,n$, $i = 1,2$, let Ξ_{ji} be some derivation for $t_{ji} \;\le\; u_j$. Then, $(\mathrm{ARROW})\langle\,\Xi_{ji},\,\Theta_i\rangle$ is a derivation for $u_j\to v \;\le\; t_{ji}\to t_i$. By Lemma 9 we have

(7) $$\Theta'_{ji} \cong (\mathrm{TRANS})\langle\,\Theta''_j,\,(\mathrm{ARROW})\langle\,\Xi_{ji},\,\Theta_i\rangle\rangle$$

Let Ξ be some derivation of $s \;\le\; [l_1\!:r_1,\ldots,l_m\!:r_m]$. For each $j = 1,\ldots,m$, $i = 1,2$, let Ω_{ji} be some derivation for $r_j \;\le\; t_{ji}$. By Lemma 9 we have

(8) $$\Theta'_i \cong (\mathrm{TRANS})\langle\,\Xi,\,(\mathrm{VART})\langle\,\Omega_{1i},\ldots,\,\Omega_{mi}\,\rangle\rangle$$

and

(9) $\quad \Theta'' \cong (\mathrm{TRANS})\langle\,\Xi,\,(\mathrm{TRANS})\langle\,(\mathrm{VART})\langle\,\Omega_{1i},\ldots,\,\Omega_{mi}\,\rangle,\,(\mathrm{VART})\langle\,\Xi_{1i},\ldots,\,\Xi_{ni}\,\rangle\rangle\rangle$.

With these, the congruence claim follows from

$$\mathsf{case}\ \iota(P\odot\mathsf{vart}(Q_1)\cdots(Q_m))(M)\ \mathsf{of}\ l_1\Rightarrow\iota(R_1\odot\mathsf{arrow}(S_1)(T))(F_1),\ldots,l_n\Rightarrow\iota(R_n\odot\mathsf{arrow}(S_n)(T))(F_n) \;=$$

$$=\; \iota(T)(\mathsf{case}\ \iota(P\odot\mathsf{vart}(Q_1)\cdots(Q_m)\odot\mathsf{vart}(S_1)\cdots(S_n))(M)\ \mathsf{of}\ l_1\Rightarrow\iota(R_1)(F_1),\ldots,l_n\Rightarrow\iota(R_n)(F_n))\ .$$

By (3) and {VART-CRN} the right hand side equals

$$\mathsf{case}\ \iota(P\odot\mathsf{vart}(Q_1\odot S_1)\cdots(Q_m\odot S_m))(M)\ \mathsf{of}\ l_1\Rightarrow\iota(R_1)(F_1);\iota(T),\ldots,l_n\Rightarrow\iota(R_n)(F_n);\iota(T)$$

and the equality is readily verified. ▌

Theorem 13 (Coherence) *Replace (FORALL) with (W-FORALL). If Δ_1 and Δ_2 are two* **SOURCE** *derivations yielding the same typing judgement then $\Delta_1 \cong \Delta_2$ (their translations yield provably equal terms in* **TARGET***).*

Proof: Take $t_1 \equiv t_2$ in Lemma 12. By Lemma 9, $\Theta_1 \cong \Theta_2$. It follows that $\Delta_1 \cong \Delta_2$. ▮

6 Models

So far we have not actually given a model for the language **SOURCE**. In this section we correct this omission. However, it is a central point of this paper that there is ***basically nothing new that we need to do in this section,*** since calculi satisfying the equational theory of **TARGET** have been thoroughly studied in the literature on the semantics of type systems. Domain-theoretic semantics suggests natural candidates for a special class of maps with the properties needed to interpret the operators $\rightarrow$ and $\multimap$. Here we present list some of these semantic solutions; all of which apply to abstract types as well as to variants. A syntactic version could also be given by a syntactic translation into an extension of the target calculus of section 2, which expresses the properties mentioned above and the consistency of which is ensured by our semantic considerations.

The domain-theoretic interpretations that we have examined so far are summarized in the following table. The necessary properties for all but the last row can be found in [TT87, HP89b], [CGW89],[ABL86], [CGW87], and [Gir87] respectively. The properties needed for the last row can be checked in a manner similar to [Gir87].

TYPES	TERMS	COERCIONS	VARIANTS
Algebraic lattices	continuous maps	bistrict maps	sep sum of lattices
Scott domains		strict maps	separated sums
Finitary projections			
dI domains	stable maps	strict stable maps	
coherent spaces		linear maps	$!A \oplus !B$
dI domains			

By a bistrict map of lattices we mean a continuous map which preserves both bottom and top elements. A separated sum of lattices L and M is the disjoint sum of L and M together with new top and bottom elements. Note that the category of Scott domains (finitary projections, respectively) and strict maps does have finite coproducts, given by coalesced sums of domains, and this implies that the required equation

$$\{\text{VART-CRN?}\} \qquad P(\mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow F_1, \ldots, l_n \Rightarrow F_n) \;=\; \mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow F_1; P, \ldots, l_n \Rightarrow F_n; P$$

holds if P is a strict map (in fact, a separated sum of domains A and B is just the coalesced sum of the lifted domains $A_\perp$ and $B_\perp$). Furthermore, it may be checked that strictness is preserved by the formation of coercion maps from given ones according to the coercion rules given in section 3 and at the beginning of this section. This model satisfies also {VART-BETA}+{VART-ETA}. An important property used in the case of Scott domains (finitary projections, respectively) is that the continuous maps from C to D

are in one-to-one correspondence with the strict maps from $C_\perp$ to D. Analogous remarks hold for stable maps and linear maps, with $!C$ instead of $C_\perp$ (see [Gir89], Chapter 8).

From a category-theoretic point of view, the main point is that we are dealing with *two categories,* one a reflective subcategory of the other, i.e. the inclusion functor has a left adjoint. The subcategory contains all objects of the larger category. While the larger category is cartesian closed, the reflective subcategory (in which our coercions live) does have coproducts.

From a proof-theoretic point of view, it is interesting to note that our solution is similar to the treatment of proof-theoretic commutation rules for disjunction (see [Tro73], 4.1.3, on page 279 for a presentation of commutation rules). The so-called commutation rules for sums in proof theory are closely related to the equations {VART-CRN?} where P is an "evaluation" map (see the Appendix B of [Gir88]).

7 Conclusions and directions for further investigation

The development of calculi for the representation of inheritance polymorphism and the semantics of such calculi is a growing and dynamic area of research investigation in programming languages. We expect that the calculi considered in this paper are only a small sample of what is yet to be developed. In this section we will speculate on a few of the most important directions for further development which will play a significant role in future work of the authors of this paper in particular and the research community in general.

Partial Equivalence Relations. Much of the research on the semantics of the system which we have considered has been based on the use of PER's as described by Bruce and Longo [BL88]. It is therefore worthwhile to compare the approach in this paper to this alternative approach. There is an evident means of carrying out a technical comparison: since the PER model interprets the calculus **TARGET**, it also interprets **SOURCE** via our translation. But the semantics in [BL88] gives the interpretation (without recursion) directly using PER's. Could these two interpretations be the same? For a certain fragment of **SOURCE** (including recursion but not bounded quantification), Cardone has recently answered the question in the affirmative for his form of semantics [Car89b] (where coherence is not an issue because the interpretation of a judgement $e\colon s$ is given as the equivalence class, in s, of the interpretation of the erasure of e—hence the meaning is not defined inductively on a derivation). For the full calculus the answer is still unknown as this paper is being written. Amadio's thesis contains some results about the relationship between explicit coercions and PER inclusion [Ama91].

Equational Theory. The reader has probably noted that we have never offered an equational theory for **SOURCE**, only one for **TARGET**. At the current time, the proper equational theory for **SOURCE** is still a subject of active research. However, our translation does suggest an equational theory. One can prove that two terms of **SOURCE** are equal by showing that their translations are equivalent in the equational theory for **TARGET**. Any of the models we have proposed will satisfy the resulting equational theory. (Whether this is also true of the interpretation of [BL88] may follow if this interpretation is the same as ours.) Since our translation is computable, it follows that this reflected equational theory for **SOURCE** is recursively enumerable; it is natural to ask for a reasonable axiomatization of this theory. Note, for example, if $e = e'\colon s$ holds in **SOURCE** and $s \leq t$, then $e = e'\colon t$ also holds in the reflected theory. There are probably many similarly interesting derived equational rules.

Recursion Any attempt to provide a model for a calculus which combines inheritance and recursion must deal with the seemingly contradictory semantic characteristics of inheritance and recursion at higher types. Ordinarily, the rule for inheritance between exponentials (function spaces) is given as follows:

$$\frac{u \leq s \qquad t \leq v}{s \to t \leq u \to v}$$

where s, t, u, v are type expressions and $\leq$ is the relation of inheritance (reading $s \leq t$ as "s inherits from t"). Note, in particular, the *contra*variance in the first argument of the $\to$ operator. In contrast, semantic domains which solve recursive domain equations such as $D = D \to D$ are generally constructed using a technique—adjoint pairs to be precise—which make it possible to "order" types using a concept of approximation based on the rule

$$\frac{\phi\colon s \to u \qquad \psi\colon t \to v}{\phi \to \psi\colon (s \to t) \to (u \to v)}$$

where $\phi = \langle \phi^L, \phi^R \rangle$ and $\psi = \langle \psi^L, \psi^R \rangle$ are adjoint pairs and $\phi \to \psi$ is the adjoint pair $\langle \lambda f.\ \psi^L \circ f \circ \phi^R,\ \lambda f.\ \psi^R \circ f \circ \phi^L \rangle$. Note, for this case, the *co*variance in the first argument of the $\to$ operator. Because of this difference, models such as the PER interpretation of Bruce and Longo [BL88], which provides a semantics for inheritance and parametric polymorphism, do not evidently extend to a semantics for recursive types. To provide for recursive types under this interpretation M. Coppo and M. Zacchi [Cop85, CZ86] utilize an appeal to the structure of the underlying universal domain, which is itself an inverse limit which solves a recursive equation. R. Amadio [Ama89, Ama90] and F. Cardone [Car89b] have explored this approach in considerable detail. There has also been progress on understanding the solution of recursive equations over domains internally to the PER model which should provide further insights [FMRS89, Fre89]. On the other hand, models such as those of Girard [Gir86] and Coquand, Gunter and Winskel [CGW87, CGW89], which handle parametric polymorphism and recursive types, do not provide an evident interpretation for inheritance. It has been the purpose of this paper to resolve this problem by an appeal to the paradigm of "inheritance and implicit coercion". However, this leaves open the question of how recursive types can be treated with this technique if one is to include a more powerful set of rules for deriving inheritance judgements between recursive types.

One complicating problem is to decide exactly what form of inheritance between recursive types is desired. For example, it seems very reasonable that if s is a subtype of t then the type of lists of s's should be a subtype of lists of t's. This is not actually derivable in the inheritance system described in this paper since there are no rules for inheritance between recursive types. But care must be taken: if s is a subtype if t then is the solution of the equations $a = a \to s$ be a subtype of the solution of $a = a \to t$? There are several possible approaches to answering this question. The PER interpretation provides a good guide: we can ask whether the solutions of these two equations have the desired relation in the PER model. Concerning the coercions approach we are forced to ask whether there is any intuitive coercion between these two types. If there is, we have not seen it! It is reasonable to conjecture that inheritance relations derived using the following rule will be acceptable:

$$\text{(REC)} \qquad \frac{C,\, a \leq \mathit{Top} \ \vdash \ s \ \leq \ t}{C \ \vdash \ \mu a.\, s \ \leq \ \mu a.\, t}$$

where types s and t have only *positive* occurrences of the variable a. Unfortunately, this misses many interesting inheritance relations that one would like to settle. Discussions of this problem will appear in several future publications on this subject. A rather satisfactory treatment using coercions has been described in [BGS89] by using the "Amber rule" of Cardelli [Car86].

Operational semantics. Despite its importance there is virtually no literature on theoretical issues concerning the operational semantics of languages with inheritance polymorphism. In particular, at the time we are writing there are no published discussions of the relationship (if any!) of the denotational models which have been studied to the intended operational semantics of a programming language based on the models. In fact, the operational semantics of no existing "practical" programming language is based on the kind of semantics discussed in this or any of the other papers on the semantics of Fun. This is because there is a divergence between the "traditional" style of semantics for the λ-calculus and the way the evaluation mechanisms of modern functional programming languages actually work. In particular, no functional programming language in common use evaluates past a lambda abstraction. Hence the identification of the constantly divergent function with the divergent element will cause the denotational semantics to fail to be computationally adequate with respect to the evaluation. Another related problem concerns the use of the β-rule and call-by-value evaluation. Many of the functional programming languages now in use evaluate all actual function parameters. This evaluation strategy immediately causes the full β-rule to fail. For example, the application of a constant function to a divergent argument will diverge in general. Semantically, this means that terms of higher type must be interpreted as *strict* functions. In a subsequent paper [BGS90], three of the authors of the current document have explored the operational semantics of inheritance with a coercion semantics in a call-by-value setting. The results there are intuitively pleasing, but there is much more that needs to be done. This direction of investigation offers several opportunities for practical applications of the specification and implementation of compilers and interpreters for new languages with inheritance.

Existentials. We have omitted discussion of existentials in this paper. We believe that the coherence results we have described will extend to a suitable interpretation of the existential types using the equational theory for weak sums, but did not choose to involve ourselves in additional cases that this would mean for our proofs.

Order-sorted algebra. The use of coercions in a first-order setting has been investigated in work of J. A. Goguen, J-P. Jouannaud and J. Meseguer on order-sorted algebras [GJM85, GM]. In particular, the implementation of OBJ2 utilized a form of "inheritance as implicit coercion" approach. Related work by Bruce and Wegner appears in [BW90].

Abstract coherence. Since there are many different calculi for which a coherence theorem is interesting, it is very useful to have a more abstract theory from which special instances of coherence can be derived, thus making coherence a more routine part of a semantic theory for an inheritance calculus such as the one we have discussed. We mentioned earlier that coherence was an issue in category theory and this might provide a framework for a more general theory. (Although, the results on coherence in the category theory literature are insufficient for the results of this paper so further extensions will be needed). Using rewriting techniques, Curien and Ghelli have developed a type-theoretic approach to the abstract coherence problem for $F_{\leq}$ which is a subsystem of **SOURCE** featuring only function and bounded generic types [CG90]. It would be interesting to see this technique extended to all of **SOURCE**, especially in view of the complications we encountered with variants.

Subtyping of bounded quantification. Our main coherence result was proved for a weaker version of the system, one that uses the rule (W-FORALL) instead of (FORALL) (see Appendix A). We believe that this is only a technical restriction that arose from our particular proof, and that coherence holds for the stronger system. A proof would however require a way to circumvent the usage of Lemma 11 in the treatment of the [CASE] rule in Lemma 12, since Lemma 11 fails when (FORALL) is postulated (for a counterexample, see Giorgio Gelli's dissertation [Ghe90]). Perhaps greatest lower bounds and least upper bounds can be replaced by some canonical choice of lower and upper bounds, a choice that may result from the derivation of the typing judgement itself.

Record update. For practical applications of calculi such as Fun, a particularly important problem concerns the semantics of "record update". The idea is this: given a function $f\colon s \to t$ and a record e with a field l of type s, we would like to modify or update the l field of e by replacing $e.l$ by $f(e.l)$ *without losing or modifying any of the other fields of* e. The development of calculi which can deal with this form of polymorphism and the ways in which Fun and related languages can be used to represent similar techniques are an object of considerable current investigation. One recent effort in this direction is [CM89] but several other efforts are under way. Despite its importance we have not explored this issue in this paper since the discussion about it is very unsettled and it will merit independent treatment at a later date.

We believe that the "inheritance as implicit coercion" method is quite robust. For example, it easily extends to accommodate "constant" inheritances between base types, such as *int* $\leq$ *real* , as long as coherence conditions similar to the ones arising in the proofs of the relevant lemmas in this paper hold between the the constant coercions which interpret these inheritances. Moreover, we expect that our methods will extend to the functional part of Quest [Car89a] and to the language described in [CM89], using the techniques of Coquand [Coq88] and Lamarche [Lam88]. Current work on inheritance and subtyping such as [CHC90] and [Mit90] will provide new challenges. We *do not claim* that every interesting aspect of inheritance can necessarily be handled in this way. However, our treatment, by showing that inheritance can be uniformly eliminated in favor of definable coercion, provides a challenge to formalisms which purport to introduce inheritance as a fundamentally new concept. Moreover, our basic approach to the semantics of inheritance should provide a useful contrast with other approaches.

8 Acknowledgements.

Breazu-Tannen's research was partially supported by ARO Grant DAAG29-84-K-0061 and ONR Grant NOOO14-88-K-0634. Many of the results of this paper were obtained during Coquand's visit to the University of Pennsylvania, partially sponsored by the Natural Sciences Association. Gunter's research was partially supported by ARO Grant DAAG29-84-K-0061 and ONR Grant NOOO14-88-K-0557. Scedrov's research was partially supported by NSF Grant CCR-87-05596, by ONR Grant NOOO14-88-K-0635, and by the 1987 Young Scientist Award from the Natural Sciences Association of the University of Pennsylvania.

Appendix A : The language SOURCE

Type expressions:

Fragment: $a \mid Top \mid s \to t \mid \{l_1 : s_1, \ldots, l_m : s_m\} \mid \forall a \leq s.\, t \mid \mu a.\, t$

Variants: $\mid [l_1 : t_1, \ldots, l_n : t_n]$

where a ranges over type variables, $m, n \geq 1$, and, in $\forall a \leq s.\, t$, a cannot be free in s. We will use $[s/a]t$ for substitution.

Raw terms:

Fragment:

$$x \mid d(e) \mid \lambda x{:}t.\, e \mid \{l_1 = e_1, \ldots, l_m = e_m\} \mid e.l \mid \Lambda a \leq t.\, e \mid e(t) \mid \mathsf{intro}[\mu a.\, t]e \mid \mathsf{elim}\ e$$

Variants:

$$\mid [l_1 : t_1, \ldots, l_i = e, \ldots, l_n : t_n] \mid \mathsf{case}\ e\ \mathsf{of}\ l_1 \Rightarrow f_1, \ldots, l_n \Rightarrow f_n$$

where x ranges over (term) variables and $m, n \geq 1$. (Note the type decorations on variant "injections"; this is necessary for the uniqueness of type derivations in the inheritance-less system and it differs from [CW85].)

Raw terms are type-checked by deriving *typing judgements*, of the form $\Gamma \vdash e : t$. where Γ is a context. *Contexts* are defined recursively as follows: $\emptyset$ is a context; if Γ is a context which does not declare a, and the free variables of t are declared in Γ, then $\Gamma, a \leq t$ is a context; if Γ is a context which does not declare x, and the free variables of t are declared in Γ, then $\Gamma, x{:}t$ is a context. The proof system for deriving typing judgements makes use of ***inheritance judgements*** which have the form $C \vdash s \leq t$ where C is an inheritance context. *Inheritance contexts* are contexts in which only declarations of the form $a \leq t$ appear. If Γ is a context, we denoted by $\widehat{\Gamma}$ the inheritance context obtained from Γ by erasing the declarations of the form $x{:}t$.

Rules for deriving inheritance judgements:

Fragment:

(TOP) $$C \vdash t \leq Top$$

where the free variables of t are declared in C

(VAR) $$C_1, a \leq t, C_2 \vdash a \leq t$$

(ARROW) $$\frac{C \vdash s \leq t \qquad C \vdash u \leq v}{C \vdash t \to u \leq s \to v}$$

(RECD) $$\frac{C \vdash s_1 \le t_1 \quad \cdots \quad C \vdash s_p \le t_p}{C \vdash \{l_1\colon s_1, \ldots, l_p\colon s_p, \ldots, l_q\colon s_q\} \le \{l_1\colon t_1, \ldots, l_p\colon t_p\}}$$

(FORALL) $$\frac{C \vdash s \le t \qquad C, a{\le}s \vdash u \le v}{C \vdash \forall a{\le}t.\, u \le \forall a{\le}s.\, v}$$

For Lemmas 11 and 12, and for Theorem 13 this is replaced with the weaker

(W-FORALL) $$\frac{C, a{\le}t \vdash u \le v}{C \vdash \forall a{\le}t.\, u \le \forall a{\le}t.\, v}$$

(REFL) $$C \vdash t \le t$$

where the free variables of t are declared in C

(TRANS) $$\frac{C \vdash r \le s \qquad C \vdash s \le t}{C \vdash r \le t}$$

Variants:

(VART) $$\frac{C \vdash s_1 \le t_1 \quad \cdots \quad C \vdash s_p \le t_p}{C \vdash [l_1\colon s_1, \ldots, l_p\colon s_p] \le [l_1\colon t_1, \ldots, l_p\colon t_p, \ldots, l_q\colon t_q]}$$

Rules for deriving typing judgements:

Fragment:

[VAR] $$\Gamma_1, x{:}t, \Gamma_2 \vdash x : t$$

[ABS] $$\frac{\Gamma, x{:}s \vdash e : t}{\Gamma \vdash \lambda x{:}s.\, e : s \to t}$$

[APPL] $$\frac{\Gamma \vdash d : s \to t \qquad \Gamma \vdash e : s}{\Gamma \vdash d(e) : t}$$

[RECD]
$$\frac{\Gamma \vdash e_1 : t_1 \quad \cdots \quad \Gamma \vdash e_m : t_m}{\Gamma \vdash \{l_1 = e_1, \ldots, l_m = e_m\} : \{l_1 : t_1, \ldots, l_m : t_m\}}$$

[SEL]
$$\frac{\Gamma \vdash e : \{l_1 : t_1, \ldots, l_m : t_m\}}{\Gamma \vdash e.l_i : t_i}$$

[B-GEN]
$$\frac{\Gamma, a \leq s \vdash e : t}{\Gamma \vdash \Lambda a \leq s.\, e : \forall a \leq s.\, t}$$

[B-SPEC]
$$\frac{\Gamma \vdash e : \forall a \leq s.\, t \qquad \widehat{\Gamma} \vdash r \leq s}{\Gamma \vdash e(r) : [r/a]t}$$

[R-INTRO]
$$\frac{\Gamma \vdash e : [\mu a.\, t/a]t}{\Gamma \vdash \textsf{intro}[\mu a.\, t]e : \mu a.\, t}$$

[R-ELIM]
$$\frac{\Gamma \vdash e : \mu a.\, t}{\Gamma \vdash \textsf{elim}\, e : [\mu a.\, t/a]t}$$

[INH]
$$\frac{\Gamma \vdash e : s \qquad \widehat{\Gamma} \vdash s \leq t}{\Gamma \vdash e : t}$$

Variants:

[VART]
$$\frac{\Gamma \vdash e : t_i}{\Gamma \vdash [l_1 : t_1, \ldots, l_i = e, \ldots, l_n : t_n] : [l_1 : t_1, \ldots, l_i : t_i, \ldots, l_n : t_n]}$$

[CASE]
$$\frac{\Gamma \vdash e : [l_1 : t_1, \ldots, l_n : t_n] \qquad \Gamma \vdash f_1 : t_1 \rightarrow t \quad \cdots \quad \Gamma \vdash f_n : t_n \rightarrow t}{\Gamma \vdash \textsf{case}\ e\ \textsf{of}\ l_1 \Rightarrow f_1, \ldots, l_n \Rightarrow f_n : t}$$

Appendix B: The language TARGET

Type expressions:

Fragment: $a \mid s \rightarrow t \mid \{l_1 : s_1, \ldots, l_m : s_m\} \mid \forall a.\, t \mid \mu a.\, t$

Variants: $\mid [l_1 : t_1, \ldots, l_n : t_n]$

Coercion space: $\mid s \multimap t$

where a ranges over type variables and $n \geq 1$. For $m = 0$ we get the *empty record type* $\mathbf{1} \overset{\text{def}}{=} \{\}$.

Raw terms:

Fragment:

$$x \mid M(N) \mid \lambda x : t.\, M \mid \{l_1 = M_1, \ldots, l_m = M_m\} \mid M.l \mid \Lambda a.\, M \mid M(t) \mid \mathsf{intro}[\mu a.\, t]M \mid \mathsf{elim}\ M$$

Variants:

$$\mid [l_1 : t_1, \ldots, l_i = M, \ldots, l_n : t_n] \mid \mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow F_1, \ldots, l_n \Rightarrow F_n$$

Coercion-coercion combinator:

$$\mid \iota_{s,t}$$

Coercion combinators:

$$\mid \mathsf{top}[t] \mid \mathsf{arrow}[s, t, u, v] \mid \mathsf{recd}[s_1, \ldots, s_q, t_1, \ldots, t_p] \mid \mathsf{forall}[s, t, a, u, v] \mid$$

$$\mathsf{vart}[s_1, \ldots, s_p, t_1, \ldots, t_q] \mid \mathsf{refl}[t] \mid \mathsf{trans}[r, s, t]$$

where x ranges over (term) variables and $n \geq 1$. For $m = 0$ we get the *empty record*, for which we will keep the notation $\{\}$. We will usually omit the cumbersome type tags on the coercion(-coercion) combinators. We use $[N/x]M$ for substitution.

Typing judgements, have the form $\Upsilon \vdash M : t$, where Υ is a typing context. *Typing contexts* are defined recursively as follows: $\emptyset$ is a context; if Υ is a context which does not declare a, then Υ, a is a typing context; if Υ is a context which does not declare x, and the free variables of t are declared in Υ, then $\Upsilon, x : t$ is a typing context.

Rules for deriving typing judgements:

Fragment:

Same as in Appendix A: [VAR] , [ABS] , [APPL] , [RECD] (in particular, for $n = 0$, $\Upsilon \vdash \{\} : \mathbf{1}$) , [SEL].

[GEN] $$\frac{\Upsilon, a \vdash M : t}{\Upsilon \vdash \Lambda a.\, M : \forall a.\, t}$$

[SPEC] $$\frac{\Upsilon \vdash M : \forall a.\, t}{\Upsilon \vdash M(s) : [s/a]t}$$

Same as in Appendix A: [R-INTRO] , [R-ELIM].

Variants:

Same as in Appendix A: [VART] , [CASE].

Coercion(-coercion) combinators:
We omit the typing contexts to simplify the notation.

$$\iota_{s,t} : (s \circ\!\!\rightarrow t) \rightarrow (s \rightarrow t)$$

$$\mathsf{top}[t] : t \circ\!\!\rightarrow \mathbf{1}$$

$$\mathsf{arrow}[s,t,u,v] : (s \circ\!\!\rightarrow t) \rightarrow (u \circ\!\!\rightarrow v) \rightarrow ((t \rightarrow u) \circ\!\!\rightarrow (s \rightarrow v))$$

$$\mathsf{recd}[s_1,\ldots,s_q,t_1,\ldots,t_p] : (s_1 \circ\!\!\rightarrow t_1) \rightarrow \cdots \rightarrow (s_p \circ\!\!\rightarrow t_p) \rightarrow (\{l_1\!: s_1,\ldots,l_p\!: s_p,\ldots,l_q\!: s_q\} \circ\!\!\rightarrow \{l_1\!: t_1,\ldots,l_p\!: t_p\})$$

$$\mathsf{forall}[s,t,a,u,v] : (s \circ\!\!\rightarrow t) \rightarrow \forall a.\,((a \circ\!\!\rightarrow s) \rightarrow (u \circ\!\!\rightarrow v)) \rightarrow (\forall a.\,((a \circ\!\!\rightarrow t) \rightarrow u) \circ\!\!\rightarrow \forall a.\,((a \circ\!\!\rightarrow s) \rightarrow v))$$

$$\mathsf{vart}[s_1,\ldots,s_p,t_1,\ldots,t_q] : (s_1 \circ\!\!\rightarrow t_1) \rightarrow \cdots \rightarrow (s_p \circ\!\!\rightarrow t_p) \rightarrow ([l_1\!: s_1,\ldots,l_p\!: s_p] \circ\!\!\rightarrow [l_1\!: t_1,\ldots,l_p\!: t_p,\ldots,l_q\!: t_q])$$

$$\mathsf{refl}[t] : t \circ\!\!\rightarrow t$$

$$\mathsf{trans}[r,s,t] : (r \circ\!\!\rightarrow s) \rightarrow (s \circ\!\!\rightarrow t) \rightarrow (r \circ\!\!\rightarrow t)$$

Equational theory:
Technically, equational judgements should all contain a typing context under which both terms in the equation typecheck with the same type [CGW87, BC88, CGW89]. To simplify the notation, we will in most cases omit these contexts.

Fragment:
We omit the simple rules for reflexivity, symmetry, transitivity, and congruence with respect to function application, record formation, field selection, application to types, recursive type introduction, and recursive type elimination.

{XI}
$$\frac{\Upsilon,\, x{:}s \;\vdash\; M \;=\; N}{\Upsilon \;\vdash\; \lambda x{:}s.\, M \;=\; \lambda x{:}s.\, N}$$

{TYPE-XI}
$$\frac{\Upsilon,\, a \;\vdash\; M \;=\; N}{\Upsilon \;\vdash\; \Lambda a.\, M \;=\; \Lambda a.\, N}$$

{BETA}
$$(\lambda x{:}s.\, M)(N) \;=\; [N/x]M$$

where $N : s$.

{ETA}
$$\lambda x{:}s.\, M(x) \;=\; M$$

where $M : s \to t$ and x not free in M.

{RECD-BETA}
$$\{l_1 = M_1, \ldots, l_m = M_m\}.l_i \;=\; M_i$$

where $m \geq 1$, $M_1{:}t_1, \ldots, M_m{:}t_m$.

{RECD-ETA}
$$\{l_1 = M.l_1, \ldots, l_m = M.l_m\} \;=\; M$$

where $M : \{l_1{:}t_1, \ldots, l_m{:}t_m\}$. For $m = 0$, this rule gives $\{\} \;=\; M$ which makes **1** into a terminator.

{FORALL-BETA}
$$(\Lambda a.\, M)(r) \;=\; [r/a]M$$

{FORALL-ETA}
$$\Lambda a.\, M(a) \;=\; M$$

where $M : \forall a.\, t$ and a not free in M.

{R-BETA}
$$\mathsf{elim}\,(\mathsf{intro}[\mu a.\, t]M) \;=\; M$$

where $M : \mu a.\, t$.

{R-ETA} $$\mathsf{intro}[\mu a.\, t](\mathsf{elim}\ M) \ = \ M$$

where $M : [\mu a.\, t/a]t$.

Variants:

We omit the simple rules for congruence with respect to variant formation, and case analysis.

{VART-BETA} $$\mathsf{case}\ \mathsf{inj}_{l_i}(M_i)\ \mathsf{of}\ l_1 \Rightarrow F_1, \ldots, l_n \Rightarrow F_n \ = \ F_i(M_i)$$

where $F_1 : t_1 \rightarrow t, \ldots, F_n : t_n \rightarrow t,\ M_i : t_i$ and inj_{l_i} is shorthand for $\lambda x{:}\, t_i.\, [l_1{:}\, t_1, \ldots, l_i = x, \ldots, l_n{:}\, t_n]$.

{VART-ETA} $$\mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow \mathsf{inj}_{l_1}, \ldots, l_n \Rightarrow \mathsf{inj}_{l_n} \ = \ M$$

where $M{:}\, [l_1{:}\, t_1, \ldots, l_n{:}\, t_n]$.

{VART-CRN} $$\iota(P)(\mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow F_1, \ldots, l_n \Rightarrow F_n) \ = \ \mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow F_1; \iota(P), \ldots, l_n \Rightarrow F_n; \iota(P)$$

where $M{:}\, [l_1{:}\, t_1, \ldots, l_n{:}\, t_n],\ F_1{:}\, t_1 \rightarrow t, \ldots, F_n{:}\, t_n \rightarrow t,\ P{:}\, t \mathrel{\circ\!\!\rightarrow} s$.

Alternatively, we could require instead of { VART-ETA } + { VART-CRN }:

{VART-COP} $$\iota(Q)(M) \ = \ \mathsf{case}\ M\ \mathsf{of}\ l_1 \Rightarrow (\mathsf{inj}_{l_1}; \iota(Q)), \ldots, l_n \Rightarrow (\mathsf{inj}_{l_n}; \iota(Q))$$

where $M{:}\, [l_1{:}\, t_1, \ldots, l_n{:}\, t_n],\ Q{:}\, [l_1{:}\, t_1, \ldots, l_n{:}\, t_n] \mathrel{\circ\!\!\rightarrow} t$.

Coercion(-coercion) combinators:

$$\iota(\mathsf{top}) \ = \ \lambda x{:}\, t.\ \{\}$$

$$\iota(\mathsf{arrow}(P)(Q)) \ = \ \lambda z{:}\, t \rightarrow u.\, (\iota(P)); z; (\iota(Q))$$

where $P{:}\, s \mathrel{\circ\!\!\rightarrow} t,\ Q{:}\, u \mathrel{\circ\!\!\rightarrow} v$.

$$\iota(\mathsf{recd}(R_1) \cdots (R_p)) \ = \ \lambda w{:}\, \{l_1{:}\, s_1, \ldots, l_p{:}\, s_p, \ldots, l_q{:}\, s_q\}.\ \{l_1{:}\, \iota(R_1)(w.l_1), \ldots, l_p{:}\, \iota(R_p)(w.l_p)\}$$

$$\text{where}\ \ R_1\colon s_1 \multimap t_1, \ldots, R_p\colon s_p \multimap t_p\,.$$

$$\iota(\mathsf{forall}(P)(W)) \;=\; \lambda z\colon(\forall a.\,(a \multimap t) \to u).\,\Lambda a.\,\lambda f\colon a \multimap s.\,\iota(W(a)(f))(z(a)(\mathsf{trans}(f)(P)))$$

$$\text{where}\ \ P\colon s \multimap t,\ W\colon\forall a.\,(a \multimap s) \to (u \multimap v).$$

$$\iota(\mathsf{vart}(R_1)\cdots(R_p)) \;=\; \lambda w\colon[l_1\colon s_1,\ldots,l_p\colon s_p].\,\mathsf{case}\ w\ \mathsf{of}\ l_1 \Rightarrow \iota(R_1);\mathsf{inj}_{l_1},\ldots,l_p \Rightarrow \iota(R_p);\mathsf{inj}_{l_p}$$

$$\text{where}\ \ R_1\colon s_1 \multimap t_1, \ldots, R_p\colon s_p \multimap t_p\,.$$

$$\iota(\mathsf{refl}) \;=\; \lambda x\colon t.\,x$$

$$\iota(\mathsf{trans}(P)(Q)) \;=\; \iota(P);\iota(Q)$$

$$\text{where}\ \ P\colon r \multimap s,\ Q\colon s \multimap t.$$

{IOTA-INJ}

$$\frac{\iota(P) \;=\; \iota(Q)}{P \;=\; Q}$$

Appendix C: The translation

We present first the remaining of the translation of the fragment discussed in section 3.

(VAR)*
$$C_1^*,\, a,\, f{:}\,a\to t^*,\, C_2^* \;\vdash\; f : a\to t^*$$

(RECD)*
$$\frac{C^* \;\vdash\; P_1 : s_1^*\to t_1^* \quad \cdots \quad C^* \;\vdash\; P_p : s_p^*\to t_p^*}{C^* \;\vdash\; R : \to\{l_1{:}\,s_1^*,\ldots,l_p{:}\,s_p^*,\ldots,l_q{:}\,s_q^*\}\{l_1{:}\,t_1^*,\ldots,l_p{:}\,t_p^*\}}$$

where $R \overset{\text{def}}{=} \lambda w{:}\,\{l_1{:}\,s_1^*,\ldots,l_p{:}\,s_p^*,\ldots,l_q{:}\,s_q^*\}.\,\{l_1{:}\,P_1(w.l_1),\ldots,l_p{:}\,P_p(w.l_p)\}$

(REFL)*
$$C^* \;\vdash\; \lambda x{:}\,t^*.\,x : t^*\to t^*$$

where the free variables of t^* are declared in C^*

(TRANS)*
$$\frac{C^* \;\vdash\; P : r^*\to s^* \qquad C^* \;\vdash\; Q : s^*\to t^*}{C^* \;\vdash\; P;Q : r^*\to t^*}$$

The rules [VAR] , [ABS] , [APPL] , [RECD] , [SEL] , [R-INTRO] , [R-ELIM] are translated straightforwardly, see below. Here is the translation of the only other rule left (the translations of the other rules appears in section 3).

[B-GEN]
$$\frac{\Gamma^*,\, a,\, f{:}\,a\to s^* \;\vdash\; M : t^*}{\Gamma^* \;\vdash\; \Lambda a.\,\lambda f{:}\,a\to s^*.\,M : \forall a.\,((a\to s^*)\to t^*)}$$

In the following, we present the translation for the full calculus. As before, for any **SOURCE** `item` we will denote by `item`* its translation into **TARGET** . We begin with the types. Note the translation of bounded generics and of *Top*.

$$\begin{array}{rcl@{\qquad}rcl}
a^* & \overset{\text{def}}{=} & a & (\forall a\leq s.\,t)^* & \overset{\text{def}}{=} & \forall a.\,((a \circ\!\!\rightarrow s^*)\to t^*) \\
Top^* & \overset{\text{def}}{=} & \mathbf{1} & (\mu a.\,t)^* & \overset{\text{def}}{=} & \mu a.\,t^* \\
(s\to t)^* & \overset{\text{def}}{=} & s^*\to t^* & [l_1{:}\,s_1,\ldots,l_n{:}\,s_n]^* & \overset{\text{def}}{=} & [l_1{:}\,s_1^*,\ldots,l_n{:}\,s_n^*] \\
\{l_1{:}\,s_1,\ldots,l_m{:}\,s_m\}^* & \overset{\text{def}}{=} & \{l_1{:}\,s_1^*,\ldots,l_m{:}\,s_m^*\} & & &
\end{array}$$

where $s\times t \overset{\text{def}}{=} \{\mathit{left}{:}\,s,\mathit{right}{:}\,t\}$.

One shows immediately that $([s/a]t)^* \equiv [s^*/a]t^*$. We extend this to contexts and inheritance contexts, which translate into just typing contexts in **TARGET** .

$$\begin{array}{rcl@{\qquad}rcl}
\emptyset^* & \stackrel{\text{def}}{=} & \emptyset & \emptyset^* & \stackrel{\text{def}}{=} & \emptyset \\
(\Gamma,\, a \leq t)^* & \stackrel{\text{def}}{=} & \Gamma^*,\, a,\, f\!: a \mathrel{\circ\!\!\to} t^* & (C,\, a \leq t)^* & \stackrel{\text{def}}{=} & C^*,\, a,\, f\!: a \mathrel{\circ\!\!\to} t^* \\
(\Gamma,\, x\!: t)^* & \stackrel{\text{def}}{=} & \Gamma^*,\, x\!: t^* & & &
\end{array}$$

where f is a *fresh* variable for each (a, f).

Next we will describe how we translate the derivations of judgments of **SOURCE** . The translation is defined by recursion on the structure of the derivation trees. Since these are freely generated by the derivation rules, it is sufficient to provide for each derivation rule of **SOURCE** a corresponding rule on trees of **TARGET** judgments. One then checks that these corresponding rules are *directly derivable* in **TARGET** (Lemma 14 below), therefore the translation takes derivations in **SOURCE** into derivations in **TARGET** .

A **SOURCE** derivation yielding an inheritance judgment $C \;\vdash\; s \;\leq\; t$ is translated as a tree of **TARGET** judgments yielding $C^* \;\vdash\; P : s^* \mathrel{\circ\!\!\to} t^*$. Here are the **TARGET** rules that correspond to the rules for deriving inheritance judgements in **SOURCE**.

(TOP)*

$$C^* \;\vdash\; \mathsf{top} : t^* \mathrel{\circ\!\!\to} \mathbf{1}$$

(VAR)*

$$C_1^*,\, a,\, f\!: a \mathrel{\circ\!\!\to} t^*,\, C_2^* \;\vdash\; f : a \mathrel{\circ\!\!\to} t^*$$

(ARROW)*

$$\frac{C^* \;\vdash\; P : s^* \mathrel{\circ\!\!\to} t^* \qquad C^* \;\vdash\; Q : u^* \mathrel{\circ\!\!\to} v^*}{C^* \;\vdash\; \mathsf{arrow}(P)(Q) : (t^* \to u^*) \mathrel{\circ\!\!\to} (s^* \to v^*)}$$

(RECD)*

$$\frac{C^* \;\vdash\; P_1 : s_1^* \mathrel{\circ\!\!\to} t_1^* \quad \cdots \quad C^* \;\vdash\; P_p : s_p^* \mathrel{\circ\!\!\to} t_p^*}{C^* \;\vdash\; \mathsf{recd}(P_1)\cdots(P_p) : \{l_1\!: s_1^*, \ldots, l_p\!: s_p^*, \ldots, l_q\!: s_q^*\} \mathrel{\circ\!\!\to} \{l_1\!: t_1^*, \ldots, l_p\!: t_p^*\}}$$

(FORALL)*

$$\frac{C^* \;\vdash\; P : s^* \mathrel{\circ\!\!\to} t^* \qquad C^*,\, a,\, f\!: a \mathrel{\circ\!\!\to} s^* \;\vdash\; Q : u^* \mathrel{\circ\!\!\to} v^*}{C^* \;\vdash\; \mathsf{forall}(P)(\Lambda a.\, \lambda f\!: a \mathrel{\circ\!\!\to} s^*.\, Q) : \forall a.\, ((a \mathrel{\circ\!\!\to} t^*) \to u^*) \mathrel{\circ\!\!\to} \forall a.\, ((a \mathrel{\circ\!\!\to} s^*) \to v^*)}$$

(VART)*

$$\frac{C^* \;\vdash\; P_1 : s_1^* \mathrel{\circ\!\!\to} t_1^* \quad \cdots \quad C^* \;\vdash\; P_p : s_p^* \mathrel{\circ\!\!\to} t_p^*}{C^* \;\vdash\; \mathsf{vart}(P_1)\cdots(P_p) : [l_1\!: s_1^*, \ldots, l_p\!: s_p^*] \mathrel{\circ\!\!\to} [l_1\!: t_1^*, \ldots, l_p\!: t_p^*, \ldots, l_q\!: t_q^*]}$$

(REFL)*

$$C^* \vdash \mathsf{refl} : t^* \mathrel{\circ\!\!\rightarrow} t^*$$

where the free variables of t^* are declared in C^*

(TRANS)*

$$\frac{C^* \vdash P : r^* \mathrel{\circ\!\!\rightarrow} s^* \qquad C^* \vdash Q : s^* \mathrel{\circ\!\!\rightarrow} t^*}{C^* \vdash \mathsf{trans}(P)(Q) : r^* \mathrel{\circ\!\!\rightarrow} t^*}$$

A **SOURCE** derivation yielding an typing judgment $\Gamma \vdash e : t$ is translated as a tree of **TARGET** judgments yielding $\Gamma^* \vdash M : t^*$. Here are the **TARGET** rules that correspond to the rules for deriving typing judgements in **SOURCE**.

The rules [VAR] , [ABS] , [APPL] , [RECD] , [SEL] , [R-INTRO] , [R-ELIM] , [VART] , [CASE] all have direct correspondents in **TARGET** so their translation is straightforward. We ilustrate it with two examples.

[VAR]*

$$\Gamma_1^*, x{:}t^*, \Gamma_2^* \vdash x : t^*$$

[ABS]*

$$\frac{\Gamma^*, x{:}s^* \vdash M : t^*}{\Gamma^* \vdash \lambda x{:}s^*.\, M : s^* \rightarrow t^*}$$

Here is the translation of the other three rules.

[B-GEN]

$$\frac{\Gamma^*, a, f{:}a \mathrel{\circ\!\!\rightarrow} s^* \vdash M : t^*}{\Gamma^* \vdash \Lambda a.\, \lambda f{:}a \mathrel{\circ\!\!\rightarrow} s^*.\, M : \forall a.\, ((a \mathrel{\circ\!\!\rightarrow} s^*) \rightarrow t^*)}$$

[B-SPEC]*

$$\frac{\Gamma^* \vdash M : \forall a.\, ((a \mathrel{\circ\!\!\rightarrow} s^*) \rightarrow t^*) \qquad \widehat{\Gamma}^* \vdash P : r^* \mathrel{\circ\!\!\rightarrow} s^*}{\Gamma^* \vdash M(r^*)(P) : [r^*/a]t^*}$$

[INH]*

$$\frac{\Gamma^* \vdash M : s^* \qquad \widehat{\Gamma}^* \vdash P : s^* \mathrel{\circ\!\!\rightarrow} t^*}{\Gamma^* \vdash \iota(P)(M) : t^*}$$

Lemma 14 *The rules* (TOP)* – (TRANS)* *and* [VAR]* – [INH]* *are directly derivable in* **TARGET** . ▮

References

[ABL86] R. Amadio, K. B. Bruce, and G. Longo. The finitary projection model for second order lambda calculus and solutions to higher order domain equations. In A. Meyer, editor, *Logic in Computer Science*, pages 122–130, IEEE Computer Society Press, 1986.

[Ama89] R. Amadio. *Recursion over realizability structures.* Research Report TR 1/89, Universitá di Pisa, January 1989.

[Ama90] R. Amadio. Recursion over realizability structures. *Information and Computation*, ??:??–??, 1990. To appear.

[Ama91] R. Amadio. *Recursion and subtyping in lambda calculi.* PhD thesis, University of Pisa, 1991.

[Bar84] H. Barendregt. *The Lambda Calculus: Its syntax and Semantics.* Volume 103 of *Studies in Logic and the Foundations of Mathematics*, Elsevier, revised edition, 1984.

[BBG88] V. Breazu-Tannen, P. Buneman, and C. A. Gunter. Typed functional programming for the rapid development of reliable software. In J. E. Gaffney, editor, *Productivity: Progress, Prospects and Payoff*, pages 115–125, Association for Computing Machinery, June 1988.

[BC88] V. Breazu-Tannen and T. Coquand. Extensional models for polymorphism. *Theoretical Computer Science*, 59:85–114, 1988.

[BCGS89] V. Breazu-Tannen, T. Coquand, C. A. Gunter, and A. Scedrov. Inheritance and explict coercion (preliminary report). In R. Parikh, editor, *Logic in Computer Science*, pages 112–134, IEEE Computer Society, June 1989.

[BGS89] V. Breazu-Tannen, C. Gunter, and A. Scedrov. *Denotational Semantics for Subtyping between Recursive Types.* Research Report MS-CIS-89-63/Logic & Computation 12, Department of Computer and Information Science, University of Pennsylvania, 1989.

[BGS90] V. Breazu-Tannen, C. Gunter, and A. Scedrov. Computing with coercions. In M. Wand, editor, *Lisp and Functional Programming*, pages 44–60, ACM, 1990.

[BL88] K. B. Bruce and G. Longo. A modest model of records, inheritance and bounded quantification. In Y. Gurevich, editor, *Logic in Computer Science*, pages 38–50, IEEE Computer Society, July 1988.

[BW90] K. Bruce and P. Wegner. An algebraic model of subtype and inheritance. In F. Bancilhon and P. Buneman, editors, *Advances in database programming languages*, pages 75–96, ACM Press and Addison-Wesley, New York, 1990.

[Car84] L. Cardelli. A semantics of multiple inheritance. In G. Kahn, D. B. MacQueen, and G. D. Plotkin, editors, *Semantics of Data Types*, pages 51–67, *Lecture Notes in Computer Science vol. 173,* Springer, 1984.

[Car85] R. Cartwright. Types as intervals. In B. K. Reid, editor, *Symposium on Principles of Programming Languages*, pages 22–36, ACM, 1985.

[Car86] L. Cardelli. Amber. In G. Cousineau, P.-L. Curien, and B. Robinet, editors, *Combinators and Functional Programming Languages*, pages 21–47, *Lecture Notes in Computer Science vol. 242,* Springer, 1986.

[Car88a] L. Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76:138–164, 1988.

[Car88b] L. Cardelli. Structural subtyping and the notion of power type. In J. Ferrante and P. Mager, editors, *Symposium on Principles of Programming Languages*, pages 70–79, ACM, 1988.

[Car89a] L. Cardelli. *Typeful programming*. Research Report 45, DEC Systems, Palo Alto, May 1989.

[Car89b] F. Cardone. Relational semantics for recursive types and bounded quantification. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *International Colloquium on Automata, Languages and Programs*, pages 164–178, *Lecture Notes in Computer Science vol. 372,* Springer, July 1989.

[CG90] P.-L. Curien and G. Ghelli. Coherence of subsumption. In *Proceedings CAAP'90, LNCS 431*, pages ??–??, 1990. Full version to appear in *Mathematical Structures in Computer Science.*

[CGW87] T. Coquand, C. A. Gunter, and Glynn Winskel. DI-domains as a model of polymorphism. In M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Language Semantics*, pages 344–363, *Lecture Notes in Computer Science vol. 298,* Springer, April 1987.

[CGW89] T. Coquand, C. A. Gunter, and G. Winskel. Domain theoretic models of polymorphism. *Information and Computation.*, 81:123–167, 1989.

[CH88] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.

[CHC90] W. R. Cook, W. L. Hill, and P. S. Canning. Inheritance is not subtyping. In P. Hudak, editor, *Principles of Programming Languages*, pages ??–??, ACM, 1990.

[CM89] L. Cardelli and J. Mitchell. Operations on records. In M. Mislove, editor, *Mathematical Foundations of Programming Semantics*, pages ??–??, *Lecture Notes in Computer Science vol. ??,* Springer, March 1989.

[Cop85] M. Coppo. A completeness theorem for recursively defined types. In W. Brauer, editor, *International Colloquium on Automata, Languages and Programs*, pages 120–129, *Lecture Notes in Computer Science vol. 194,* Springer, 1985.

[Coq88] T. Coquand. Categories of embeddings. In Y. Gurevich, editor, *Logic in Computer Science*, pages 256–263, IEEE Computer Society, July 1988.

[CW85] L. Cardelli and P. Wegner. On understanding types, data abstraction and polymorphism. *ACM Computing Surveys*, 17(4):471–522, 1985.

[CZ86] M. Coppo and M. Zacchi. Type inference and logical relations. In A. Meyer, editor, *Symposium on Logic in Computer Science*, pages 218–226, ACM, 1986.

[FMRS89] P. Freyd, P. Mulry, G. Rosolini, and D.S. Scott. Domains in PER. 1989. Unpublished manuscript.

[Fre89] P. Freyd. Recursive types. 1989. Unpublished manuscript.

[Ghe90] G. Ghelli. *Proof-Theoretic studies about a minimal type system integrating inclusion and parametric polymorphism.* PhD thesis, University of Pisa, 1990.

[Gir86] J. Y. Girard. The system F of variable types: fifteen years later. *Theoretical Computer Science*, 45:159–192, 1986.

[Gir87] J. Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir88] J. Y. Girard. Normal functors, power series, and λ-calculus. *Annals of Pure and Applied Logic*, 37:129–177, 1988.

[Gir89] J. Y. Girard. *Proofs and Types.* Cambridge University Press, 1989.

[GJ90] C. A. Gunter and A. Jung. Coherence and consistency in domains (extended outline). *Journal of Pure and Appled Algebra*, 63:49–66, 1990.

[GJM85] J. A. Goguen, J-P. Jouannaud, and J. Meseguer. Operational semantics for order-sorted algebra. In W. Brauer, editor, *International Colloquium on Automata, Languages and Programs*, pages 221–231, *Lecture Notes in Computer Science vol. 194,* Springer, July 1985.

[GM] J. A. Goguen and J. Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. Unpublished manuscript.

[GR83] A. Goldberg and D. Robson. *Smalltalk-80: the language and its implementation.* Addison-Wesley, Reading, MA, 1983.

[HP89a] H. Huwig and A. Poigné. A note on inconsistencies caused by fixpoints in a cartesian closed category. *Theoretical Computer Science*, ??:??–??, 1989. To appear.

[HP89b] J. M . E. Hyland and A. Pitts. The theory of constructions: categorical semantics and topos-theoretic models. In J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, pages 137–199, ACM, 1989.

[JM88] L. Jategaonkar and J. C. Mitchell. ML with extended pattern matching and subtypes. In R. Cartwright, editor, *Symposium on LISP and Functional Programming*, pages 198–211, ACM, 1988.

[KL71] G. M. Kelly and S. Mac Lane. Coherence in closed categories. *J. Pure Appl. Algebra*, 1:97–140, 1971. Erratum ibid. 2(1971), p. 219.

[Koy82] C. Koymans. Models of the lambda calculus. *Information and Control*, 52:306–332, 1982.

[Lam88] F. Lamarche. *Modelling Polymorphism with Categories.* PhD thesis, McGill University, 1988.

[Law69] F. W. Lawvere. Diagonal arguments and cartesian closed categories. In *Category theory, homology theory, and their applications II*, pages 134–145, *Lecture Notes in Mathematics*, Vol. 92, Springer-Verlag, 1969.

[LP85] S. Mac Lane and R. Pare. Coherence for bicategories and indexed categories. *Journal of Pure and Appled Algebra*, 37:59–80, 1985.

[Mar84] P. Martin-Löf. *Intutionistic Type Theory. Studies in Proof Theory*, Bibliopolis, 1984.

[Mar88] S. Martini. Bounded quantifiers have interval models. In R. Cartwright, editor, *Symposium on LISP and Functional Programming*, pages 164–173, ACM, 1988.

[Mey82] A. R. Meyer. What is a model of the lambda calculus? *Information and Control*, 52:87–122, 1982.

[Mit90] J. Mitchell. Toward a typed foundation for method specialization and inheritance. In P. Hudak, editor, *Principles of Programming Languages*, pages ??–??, ACM, 1990.

[OB88] A. Ohori and P. Buneman. Type inference in a database programming language. In R. Cartwright, editor, *Symposium on LISP and Functional Programming*, pages 174–183, ACM, New York, 1988.

[Rey80] J. C. Reynolds. Using category theory to design implicit conversions and generic operators. In N. D. Jones, editor, *Semantics-Directed Compiler Generation*, pages 211–258, *Lecture Notes in Computer Science vol. 94*, Springer, 1980.

[Sal88] A. Salvesen. Polymorphism and monomorphism in Martin-Löf's Type Theory. In *Logic Colloquium'88*, pages ??–??, 1988. To appear.

[Sco80] D. S. Scott. Relating theories of the lambda calculus. In J. R. Hindley, editor, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 403–450, Academic Press, 1980.

[Sta88] R. Stansifer. Type inference with subtypes. In J. Ferrante and P. Mager, editors, *Symposium on Principles of Programming Languages*, pages 88–97, ACM, 1988.

[Str88] T. Streicher. *Correctness and completeness of a categorical semantics of the Calculus of Constructions*. PhD thesis, Passau University, 1988.

[Tro73] A. S. Troelstra. *Metamathematical Investigations of Intuitionistic Arithmetic and Analysis. Lecture Notes in Mathematics vol. 344*, Springer, 1973.

[TT87] T. Coquand and T. Ehrhard. An equational presentation of higher-order logic. In D. H. Pitt, A. Poigné, and D. E. Rydeheard, editors, *Category Theory and Computer Science*, pages 40–56, *Lecture Notes in Computer Science vol. 283*, Springer, 1987.

[Wan87] M. Wand. Complete type inference for simple objects. In D. Gries, editor, *Symposium on Logic in Computer Science*, pages 37–46, IEEE Computer Society Press, Ithaca, New York, June 1987.