Semantic subtyping between coinductive mutable record types with unions and intersections

Davide Ancona[1] Andrea Corradi[2]

DIBRIS
Università di Genova
Genoa, Italy

Abstract

Semantic subtyping between coinductive record types supports accurate type analysis of object-oriented languages, by introducing Boolean type constructors and modeling cyclic objects.
In previous work, a sound and complete algorithm for semantic subtyping has been proposed, but only for coinductive immutable record types with unions.
In this work we address the issue of extending the previous results to the more challenging case where records can be mutable, and, besides union, also intersection types are supported.
We provide a set of subtyping rules on top of which a sound decision algorithm can be defined.

1 Introduction

Record types and subtyping are essential notions for defining flexible type systems for object-oriented languages.

In simple type systems the subtyping relation can be defined axiomatically by a set of inference rules; however, when more complex types are considered, the axiomatic approach may fail to convey the right intuition behind subtyping, and, in general, it is not simple to prove that the subtyping rules are complete.

Semantic subtyping has been proposed as a possible solution to these problems for XDuce [8], ℂDuce [7], the π-calculus [6], and for recursive object types with unions [3]. Semantic subtyping interprets subtyping as set inclusion between type interpretations (that is, set of values). In this way, the definition of subtyping is more intuitive, and several properties can be easily deduced (for instance, transitivity always holds trivially). Semantic subtyping is particularly suited to naturally supports Boolean type constructors; for instance, the interpretation of union

[1] Email: davide.ancona@unige.it
[2] Email: andrea.corradi@dibris.unige.it

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

and intersection types coincide with set-theoretic union and intersection, respectively. Furthermore, semantic subtyping helps reasoning on recursive types. Let us consider, for instance, the following recursive definition of the record type τ: $\tau = \langle elem{:}int, next{:}\tau \rangle$.

In the semantic subtyping approach, the syntactic equation above is turned into a semantic equation, which, in general, can be interpreted either inductively or coinductively. In languages as $\mathbb{C}$Duce where values are always inductive (as trees), types are always interpreted inductively, therefore the least solution of the semantic equation $[\![\tau]\!] = [\![\langle elem{:}int, next{:}\tau \rangle]\!]$ is considered, and the type τ is not inhabited since it denotes the empty set.

In object-oriented languages objects are allowed to contain cycles, therefore recursive types are interpreted coinductively [3]; the interpretation of τ corresponds to the greatest solution of $[\![\tau]\!] = [\![\langle elem{:}int, next{:}\tau \rangle]\!]$, hence τ denotes the set of all integer circular lists.

In previous work [3,5] we have studied semantic subtyping for coinductive immutable record types and unions, and defined a practical sound and complete top-down algorithm for deciding it.

In this paper we try to partially extend our previous results to the more challenging setting of mutable records with unions and intersections, and variance annotations for fields, to make subtyping more flexible.

The paper is organized as follows: Section 2 introduces and motivates the notion of mutable record type, and variance annotation, and show their interplay with union and intersection types. After some preliminary definitions given in Section 3, Section 4 defines the semantics of types, whereas Section 5 provides a set of subtyping rules which are sound w.r.t. semantic subtyping. Finally, conclusion and directions for future work are discussed in Section 6.

2 Mutable record types with variance annotations

Record types and the corresponding subtyping relation [1] are at the basis of structural type systems for precise type analysis of object-oriented languages. It is well known that depth subtyping between record types (that is, record subtyping is covariant w.r.t. the types of the fields) is sound only when fields are immutable, while subtyping between mutable records must be invariant w.r.t. the types of the fields.

To make record subtyping more flexible, annotations can be introduced [1] to specify whether subtyping w.r.t. a given field is covariant (that is, the field is readable), contravariant (that is, the field is writable), or invariant (that is, the field is both readable and writable).

For instance, the following functions can be typed more precisely by using type annotations:

N `get(`$\langle v^{+}{:}N \rangle$ `x){return x.v;}`
`void set(`$\langle v^{-}{:}N \rangle$ `x,` N `n){x.v=n;}`

By subtyping, function `get` can be safely applied to an argument having type $\langle v^{\nu}{:}N', \ldots \rangle$, where $\nu = +$ (v is readable) or $\nu = \circ$ (v is both readable, and writable), $N' \leq N$, and $\ldots$ indicates that the record can contain more fields. The fact that

$\langle v^{\circ}{:}N\rangle \leq \langle v^{+}{:}N\rangle$ suggests that annotations of field v statically limit the use of v for records managed in a certain context, to make subtyping more flexible, rather than specifying the runtime behavior of a record. For instance, if `{v = 42}` denotes a mutable record literal, then each of the three different types $\langle v^{\circ}{:}int\rangle$, $\langle v^{-}{:}int\rangle$, and $\langle v^{+}{:}int\rangle$ can be correctly deduced for it; however, the immutable record literal `{v`$^{+}$ `= 42}` can only have type $\langle v^{+}{:}int\rangle$.

Similarly, function `put` can be safely applied to arguments having type $\langle v^{\nu}{:}N', \ldots\rangle$ and N, respectively, where $\nu = -$ (v is writable) or $\nu = \circ$ (v is both readable and writable), $N \leq N'$, and $\ldots$ indicates that the record can contain more fields.

The annotation $\circ$ is used in contexts where a field must be both readable and writable; however in these cases the subtyping relation is less flexible. With intersection types the annotation $\circ$ is no longer required, and the subtyping relation is more flexible.

3 Types and trees

In this section we define coinductive mutable record types with unions and intersections, and present the definitions and the general results that will be used in the rest of the paper.

Definition 3.1 A regular tree is a possibly infinite tree containing a finite set of subtrees. A type is regular if it is a term that corresponds to a regular tree, that is, it has a finite set of subterms.

Proposition 3.2 *Every regular tree t can be represented by a system of guarded equations.*

We define types as all regular terms coinductively defined as follows:

$$\tau \ ::= \mathbf{0} \mid \mathbf{1} \mid \mathit{int} \mid \mathit{null} \mid \langle\rangle \mid \langle f^{-}{:}\tau\rangle \mid \langle f^{+}{:}\tau\rangle \mid \tau_1 \vee \tau_2 \mid \tau_1 \wedge \tau_2$$

The type $\langle\rangle$ represents all record values. The singleton record types $\langle f^{+}{:}\tau\rangle$ and $\langle f^{-}{:}\tau\rangle$ represent all record values containing the readable (hence, covariant) and the writable (hence, contravariant) field f, respectively.

Union types $\tau_1 \vee \tau_2$ intuitively represent the union of the values of τ_1 and τ_2, symmetrically, intersection types $\tau_1 \wedge \tau_2$ represent the intersection of the values of τ_1 and τ_2 [4,9]. Types $\mathbf{0}$ and $\mathbf{1}$ are the empty (a.k.a. bottom), and the universe (a.k.a. top) type, *int* represents the set $\mathbb{Z}$, and *null* denotes the singleton set containing the null reference.

The intersection type $\langle f^{+}{:}\tau\rangle \wedge \langle g^{+}{:}\tau'\rangle$ represents the type whose values are records which must have both the covariant fields f and g, of types τ and τ', respectively, hence corresponds to the record type usually written $\langle f^{+}{:}\tau, g^{+}{:}\tau'\rangle$. The intersection type $\langle f^{+}{:}\tau\rangle \wedge \langle f^{-}{:}\tau'\rangle$ represents all record values with the invariant field f of type τ, hence corresponds to the record type usually written $\langle f^{\circ}{:}\tau\rangle$.

We now introduce the notion of *contractive* type, which allows us to reject all those types whose intended interpretation is not coinductive.

Definition 3.3 A type is *contractive* if all its infinite paths contain a node corresponding to a record type.

Remark In the rest of the paper all types are restricted to be regular and contractive.

4 Semantic subtyping

In this section we define the semantics of types.

We interpret types as set of values. Values are all finite and infinite (but regular) terms coinductively defined as follows (where $i \in \mathbb{Z}$):

$$v \ ::= i \mid \text{null} \mid \langle f_1 \mapsto (\kappa_1^+, \kappa_1^-), \ldots, f_n \mapsto (\kappa_n^+, \kappa_n^-)\rangle$$

$$\kappa^+ \ ::= \{\} \mid \{v\} \qquad \kappa^- \ ::= \{v_1, \ldots, v_n\}$$

Record values are finite maps from fields to pairs, and we implicitly assume that field names are distinct and their order is immaterial. The first element of the pair, κ^+, is a set of cardinality ≤ 1; when empty, it means that it is not possible to read from the corresponding field. If it is the singleton $\{v\}$, then it is possible to read the value v. The second element of the pair is a set of arbitrary cardinality that contains the values that *cannot* be written in the corresponding field. An empty set means that any value can be written.

The interpretation of types is defined in Figure 1 by means of the membership relation. The right-hand-side ellipsis in record values indicates that the value is allowed to contain more fields. The definition in Figure 1 should be coinductive,

$$v \in_0 \tau \Leftrightarrow \text{true (approx } \in)$$

$$\text{with } n > 0\text{:}$$

$$v \in_n \mathbf{1} \Leftrightarrow \text{true (any } \in) \qquad \text{null} \in_n null \Leftrightarrow \text{true (null } \in)$$

$$v \in_n \tau_1 \vee \tau_2 \Leftrightarrow v \in_{n-1} \tau_1 \text{ or } v \in_{n-1} \tau_2 \text{ (or } \in) \qquad i \in_n int \Leftrightarrow \text{true (int } \in)$$

$$v \in_n \tau_1 \wedge \tau_2 \Leftrightarrow v \in_{n-1} \tau_1 \text{ and } v \in_{n-1} \tau_2 \text{ (and } \in) \qquad \langle \ldots \rangle \in_n \langle\rangle \Leftrightarrow \text{true (rec } \in)$$

$$\langle f \mapsto (\kappa^+, \kappa^-), \ldots \rangle \in_n \langle f^- : \tau \rangle \Leftrightarrow \forall v'. v' \in_{n-1} \tau \Rightarrow v' \notin \kappa^- \ (\text{rec}^- \in)$$

$$\langle f \mapsto (\{v\}, \kappa^-), \ldots \rangle \in_n \langle f^+ : \tau \rangle \Leftrightarrow v \in_{n-1} \tau \text{ and } \forall v'. v' \notin \kappa^- \Rightarrow v' \in_{n-1} \tau \ (\text{rec}^+ \in)$$

Fig. 1. Value membership

and the Tarski-Knaster theorem guarantees the existence of the greatest fixed-point if the corresponding one step inference function is monotone. Unfortunately in rule (rec$^-$ $\in$) we have a negative occurrence of the membership relation itself, ($v' \in_{n-1} \tau \Rightarrow v' \notin \kappa^-$ is equivalent to $v' \notin_{n-1} \tau$ or $v' \notin \kappa^-$), therefore the corresponding one step inference function is not monotone.

To overcome this problem we define an approximation of the $\in$ relation by introducing an approximation rule (approx $\in$) [2] stating that a value belongs to any type. Membership is equipped with an index ranging over natural numbers and representing the minimum height in the proof tree at which the approximation rule could be applied. In this way it is possible to reason on the membership relation in terms of arithmetic induction.

We say that a value v belongs to a type τ iff $\forall . n \in \mathbb{N}\ v \in_n \tau$ holds.

Definition 4.1 The interpretation of τ, is defined by $[\![\tau]\!] = \{v \mid \forall n \in \mathbb{N}. v \in_n \tau\}$.

We now provide and prove some useful lemmas, to show which equalities/inequalities are expected to hold and which are not.

Lemma 4.2 $[\![\langle f^+{:}\tau_1\rangle \wedge \langle f^-{:}\tau_2\rangle]\!] \neq \emptyset$ *iff* $[\![\tau_2]\!] \subseteq [\![\tau_1]\!]$.

Lemma 4.3 $[\![\langle f^+{:}\tau\rangle]\!] = \emptyset$ *iff* $[\![\tau]\!] = \emptyset$

Lemma 4.4 $[\![\langle f^+{:}\tau_1 \wedge \tau_2\rangle]\!] = [\![\langle f^+{:}\tau_1\rangle \wedge \langle f^+{:}\tau_2\rangle]\!]$ *and*
$[\![\langle f^-{:}\tau_1\rangle \wedge \langle f^-{:}\tau_2\rangle]\!] = [\![\langle f^-{:}\tau_1 \vee \tau_2\rangle]\!]$ *and* $[\![\langle f^-{:}\tau_1\rangle \vee \langle f^-{:}\tau_2\rangle]\!] = [\![\langle f^-{:}\tau_1 \wedge \tau_2\rangle]\!]$.

Lemma 4.5 $[\![\langle f^+{:}\tau_1\rangle \vee \langle f^+{:}\tau_2\rangle]\!] \subseteq [\![\langle f^+{:}\tau_1 \vee \tau_2\rangle]\!]$ *and*
$[\![\langle f^+{:}\tau_1 \vee \tau_2\rangle]\!] \not\subseteq [\![\langle f^+{:}\tau_1\rangle \vee \langle f^+{:}\tau_2\rangle]\!]$.

5 A sound set of subtyping rules

In this section we define a system of coinductive subtyping rules.

Similarly as done for immutable record types [3], the following identity is exploited: $A \subseteq B \Leftrightarrow A \setminus B = \emptyset \Leftrightarrow A \wedge \neg B = \emptyset$

In order to use this property we must be able to express the complement of a type. The extended types π, defined in Figure 2, include the negation as a possible type.

$$\pi \ ::= \mathbf{0} \mid \mathbf{1} \mid \mathit{int} \mid \mathit{null} \mid \langle\rangle \mid \langle f^-{:}\pi\rangle \mid \langle f^+{:}\pi\rangle \mid \vee\{\pi_1,\ldots,\pi_n\} \mid \wedge\{\pi_1,\ldots,\pi_n\} \mid \neg\pi$$

Fig. 2. Extended types

Beside adding negation types $\neg\pi$ we have also introduced or-sets $\vee\{\pi_1,\ldots,\pi_n\}$, representing the union of the types $\pi_1,\ldots,\pi_n$, and and-sets $\wedge\{\pi_1,\ldots,\pi_n\}$, representing the intersection of the types $\pi_1,\ldots,\pi_n$.

This representation simplifies some of the rules that will be presented in this section.

The function $\xi(\tau)$ transforms the type τ in the corresponding extended type; $\delta(\pi)$ put π in Disjunctive Normal Form (DNF for short) and it uses equivalences of Lemma 4.4 to simplify the types. In DNFs positive literals are all record types and the primitive types $\mathbf{0}, \mathbf{1}, \mathit{null}, \mathit{int}$, while negative literals are $\neg\langle\rangle$, $\neg\langle f^+{:}\tau\rangle$, $\neg\langle f^-{:}\tau\rangle$, $\neg\mathit{null}$, and $\neg\mathit{int}$.

Subtyping is defined as follows:

Definition 5.1 $\tau_1 \leq \tau_2$ iff $\delta(\wedge\{\xi(\tau_1), \neg\xi(\tau_2)\}) \cong \emptyset$ holds.

To check whether $\tau_1 \leq \tau_2$ holds, both types are transformed in the corresponding extended types then the type $\wedge\{\xi(\tau_1), \neg\xi(\tau_2)\}$ is put in DNF, simplified, and, finally, it is checked whether it is empty.

The inductive rules for checking emptiness of a type are defined in Figure 3.

Rule (rec$\cong\emptyset$) corresponds to Lemma 4.3. Rule (prim$\cong\emptyset$) deals with intersection of disjoint primitive types (*int*,*null*). The intersection between $\neg\langle\rangle$ and any record type (rule ($\langle\rangle\cong\emptyset$)) is empty. The remaining rules deal with complementation between record types. Rule (r\r$\cong\emptyset$) is the classical rule on covariant record types, that is, $\langle f^+{:}\pi\rangle \leq \langle f^+{:}\pi'\rangle$ iff $\pi \leq \pi'$; rule (w\w$\cong\emptyset$) is the symmetric one.

$$(\text{empty}\cong\emptyset)\frac{}{\wedge\{\ldots,\mathbf{0},\ldots\}\cong\emptyset} \qquad (\text{rec}\cong\emptyset)\frac{\delta(\pi)\cong\emptyset}{\wedge\{\ldots,\langle f^+{:}\pi\rangle,\ldots\}\cong\emptyset}$$

$$(\text{prim}\cong\emptyset)\frac{\pi\in\{\mathbf{0},int,null\}\quad \pi\neq\mathbf{0}\Rightarrow\pi\neq\pi'}{\wedge\{\ldots,\pi,\pi',\ldots\}\cong\emptyset} \qquad (\vee\cong\emptyset)\frac{\pi_i\cong\emptyset\ \forall i\in\{1\ldots,m\}}{\vee\{\pi_1,\ldots,\pi_m\}\cong\emptyset}$$

$$(\langle\rangle\cong\emptyset)\frac{\pi\in\{\mathbf{0},\langle\rangle,\langle f^+{:}\pi'\rangle,\langle f^-{:}\pi'\rangle\}}{\wedge\{\ldots,\pi,\neg\langle\rangle,\ldots\}\cong\emptyset} \qquad (\text{r}\backslash\text{r}\cong\emptyset)\frac{\delta(\wedge\{\pi,\neg\pi'\})\cong\emptyset}{\wedge\{\ldots,\langle f^+{:}\pi\rangle,\neg\langle f^+{:}\pi'\rangle,\ldots\}\cong\emptyset}$$

$$(\text{w}\backslash\text{w}\cong\emptyset)\frac{\delta(\wedge\{\pi',\neg\pi\})\cong\emptyset}{\wedge\{\ldots,\langle f^-{:}\pi\rangle,\neg\langle f^-{:}\pi'\rangle,\ldots\}\cong\emptyset} \qquad (\text{r}\backslash\text{w}\cong\emptyset)\frac{\delta(\pi')\cong\emptyset}{\wedge\{\ldots,\langle f^+{:}\pi\rangle,\neg\langle f^-{:}\pi'\rangle,\ldots\}\cong\emptyset}$$

Fig. 3. Emptiness of types

The type $\langle f^-{:}\mathbf{0}\rangle$ is the greatest contravariant type with field f; in fact, looking at the semantics, we have that $\langle f\mapsto(\kappa^+,\kappa^-),\ldots\rangle\in_n\langle f^-{:}\mathbf{0}\rangle$ iff $\forall n.\forall v'.v'\in_{n-1}\mathbf{0}\Rightarrow v'\notin\kappa^-$ then, since $v'\in_{n-1}\mathbf{0}$ does not hold, we have that $\langle f\mapsto(\kappa^+,\kappa^-),\ldots\rangle\in_n\langle f^-{:}\mathbf{0}\rangle$ always holds. Keeping in mind this reasoning, rule (r\w$\cong\emptyset$) states that $\wedge\{\langle f^+{:}\pi\rangle,\neg\langle f^-{:}\pi'\rangle\}\cong\emptyset$ holds when $\delta(\pi')\cong\emptyset$ holds.

The emptiness judgment does not have a rule for the case $\wedge\{\langle f^+{:}\pi\rangle,\langle f^-{:}\pi'\rangle\}$; this type, by Lemma 4.2, is empty when $\pi'\leq\pi$ does not hold. This would require the introduction of a judgment for the negation of subtyping which we do not know whether is decidable.

There is no rule symmetric to (r\w$\cong\emptyset$) because $\wedge\{\langle f^-{:}\pi\rangle,\neg\langle f^+{:}\pi'\rangle\}\cong\emptyset$ can never be empty, since values of shape $\langle f\mapsto(\{\},\kappa^-)\rangle$ (that is, records where field f is not readable), can not be removed by subtracting a covariant record type.

Claim 5.2 (Soundness) *If $\tau\leq\tau'$ holds then $[\![\tau]\!]\subseteq[\![\tau']\!]$.*

6 Conclusion

In this paper we have studied semantic subtyping for mutable record types with variance annotations, union and intersection types.

We have provided an interpretation of types which accommodates variance annotations with the semantic subtyping approach, so that subtyping corresponds to set inclusion between type interpretations.

Such a model allowed us to study the main properties underlying subtyping between mutable record types with variance annotations, unions and intersections.

Furthermore we defined a set of rules to decide subtyping, that we claim to be sound w.r.t. semantic subtyping.

For what concerns future development, besides completing the proof of soundness of subtyping rules, we are planning to investigate whether a practical algorithm can be derived from the rules, as done for the case of immutable records [3]. Finally, it would be interesting to study whether there exists a complete procedure for semantic subtyping for mutable record types with variance annotations, union and intersection types, or if the problem is undecidable.

References

[1] Abadi, M. and L. Cardelli, "A Theory of Objects," Monographs in Computer Science, Springer, 1996.

[2] Ancona, D., *How to Prove Type Soundness of Java-like Languages Without Forgoing Big-step Semantics*, in: *Formal techniques for Java-like programs (FTfJP14)* (2014), pp. 1:1–1:6.

[3] Ancona, D. and A. Corradi, *Sound and complete subtyping between coinductive types for object-oriented languages*, in: *ECOOP 2014 - Object-Oriented Programming - 28th European Conference*, Lecture Notes in Computer Science **8586** (2014), pp. 282–307.

[4] Barbanera, F., M. Dezani-Ciancaglini and U. De'Liguoro, *Intersection and union types: Syntax and semantics*, Information and Computation **119** (1995), pp. 202–230.

[5] Bonsangue, M., J. Rot, D. Ancona, F. de Boer and J. Rutten, *A coalgebraic foundation for coinductive union types*, in: *ICALP 2014 - 41st International Colloquium on Automata, Languages and Programming*, 2014, pp. 62–73.

[6] Castagna, G., R. De Nicola and D. Varacca, *Semantic subtyping for the π-calculus*, Theoretical Computer Science **398** (2008), pp. 217–242.

[7] Frisch, A., G. Castagna and V. Benzaken, *Semantic subtyping: Dealing set-theoretically with function, union, intersection, and negation types*, J. ACM **55** (2008).

[8] Hosoya, H. and B. C. Pierce, *XDuce: A statically typed XML processing language*, ACM Trans. Internet Techn. **3** (2003), pp. 117–148.

[9] Igarashi, A. and H. Nagira, *Union types for object-oriented programming*, Journ. of Object Technology **6** (2007), pp. 47–68.