Gradual Type Precision as Retraction

Max S. New
Northeastern University
maxnew@ccs.neu.edu

POPL 2017 *Student Research Competition* Extended Abstract

1 Introduction

Gradually typed programming languages allow for a mix of precision of static type information, allowing advanced type features to be added to existing languages, while still supporting interoperability with legacy code. The advantages of gradual typing are enticing to researchers and practitioners alike, but a general theory of gradually typed languages is only beginning to emerge after a decade of research.

It has long been noted that there is much similarity between work on contracts and gradual typing, and the work of Scott [1976, 1980] using retracts in domain theory. Here we take this connection seriously and consider how judgments in modern gradually typed languages can be framed in terms of retractions. While retractions in programming languages were originally studied in terms of denotational semantics in domains, our presentation will use only the most basic elements of category theory: composition, identity and equality of terms, so our formulation is equally applicable to axiomatic or operational semantics.

We propose a semantic criterion for the notion of *precision* of gradual types, a common judgment in gradually typed languages (sometimes called *naïve subtyping* for historical reasons). We relate it to a previous definition from Wadler and Findler [2009] (henceforth WF09) that defines type precision in terms of blame. We show that our definition decomposes in a similar way into "positive" and "negative" type precision, but without depending on a specific notion of blame in the language.

2 Compatibility and Precision

One of the central judgments of gradually typed languages in the style of [Siek and Taha, 2006] (as opposed to the style of Typed Racket [Tobin-Hochstadt and Felleisen, 2006]) is that of *type compatibility* (also known as *consistency*), usually denoted by $\sim$. Informally, $A \sim B$ means that A and B, which are partially dynamic types, agree on the statically known components of their type. A complementary semantic intuition is that A and B have non-empty intersection. For example, let ? be the "fully dynamic" type and $\mathbb{B}$ the "fully static" type of booleans. Then $? \to \mathbb{B} \sim \mathbb{B} \to ?$ because they agree on the statically known portion $\to$ and the mismatches are all with a dynamic type.

The judgment is used pervasively in gradual typing rules for instance:

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A' \qquad A \sim A'}{\Gamma \vdash tu : B}$$

We allow this to typecheck because we cannot locally determine that there will be an error. In this way, gradual typing enables a mix of static and dynamic reasoning.

Originally Siek and Taha [2006] defined $\sim$ syntactically, but Siek and Wadler [2010] presented a semantic definition in terms of *type precision*, denoted here by $\sqsubseteq$. Intuitively, $A \sqsubseteq B$ means that A is "less dynamic" than B. For example in the poset of types ordered by precision, ? is the top element. Then $A \sim B$ means that A, B have a non-trivial meet. Therefore we can define type compatibility by type precision.

One striking feature of this relation is that the $\to$ type constructor is *covariant*, unlike the typical subtyping rule:

$$\frac{A \sqsubseteq A' \qquad B \sqsubseteq B'}{(A \to B) \sqsubseteq (A' \to B')}$$

WF09 define precision using the idea of *blame* from contracts [Findler and Felleisen, 2002]. They use this definition to "decompose" type precision into two interrelated judgments that recover the contravariance of the $\to$ constructor. There they have a gradually typed language with casts from any type to any other, denoted $A \Rightarrow B$. They characterize type precision in terms of two subsidiary judgments: positive $A \leq^+ B$ and negative $A \leq^- B$ subtyping. A positive subtyping judgment $A \leq^+ B$ means the cast $A \Rightarrow B$ never blames the positive party (the value), while a negative subtyping judgment $A \leq^- B$ means the cast never blames the negative party (the continuation). Then a type precision judgment $A \sqsubseteq B$ is equivalent to $A \leq^+ B \wedge B \leq^- A$, i.e., casts between A and B never blame the A side.

The downside of this definition is that it depends on the definition of *blame*, which has no generally agreed upon definition. Additionally, gradual type systems defined without a notion of blame cannot benefit from this definition.

3 Precision as Retraction

Instead, we can define precision in terms of section-retraction pairs. Unfortunately the terminology here is fraught. In Findler and Blume [2006], following Scott [1976], the word retraction was used to mean what in more modern terminology is called an *idempotent.* However, the more appropriate formalism for typed casts is the closely related notion of *section-retraction pair.* A section-retraction pair is a pair of opposing arrows:

$$A \overset{s}{\rightarrowtail} B \overset{r}{\twoheadrightarrow} A$$

such that $r \circ s = \mathrm{id}_A$. We argue that this captures an idea of safe typed cast, the section s is an "up-cast", embedding the smaller, more precise type into the bigger, more dynamic type, whereas the retraction r is a "down-cast", coercing terms of type B to have type A. The condition $r \circ s = \mathrm{id}_A$ guarantees that we don't lose any information about an A value or continuation by "casting" to the more dynamic type B.

With this definition in hand, we propose the following alternative definition of the judgment $A \sqsubseteq B$, for a language in the style of WF09, where there is a dynamic type ? and casts between arbitrary types. First, since ? is the most dynamic type, the casts $A \Rightarrow ? \Rightarrow A$ should form a section-retraction pair for every gradual type A. So casting any term to dynamic type and back should result in an equivalent term. Second, $A \sqsubseteq B$ if the section-retraction pair A into ? is a "subretraction" of the section-retraction pair of B into ?, that is, the casts $A \Rightarrow B \Rightarrow A$ form a section-retraction pair and the following triangles of casts commute[1]:

$$\begin{tikzcd} A \arrow[rr, Rightarrow] \arrow[dr, Rightarrow] & & B \arrow[dl, Rightarrow] \\ & ? & \end{tikzcd} \qquad \begin{tikzcd} A & & B \arrow[ll, Rightarrow] \\ & ? \arrow[ul, Rightarrow] \arrow[ur, Rightarrow] & \end{tikzcd}$$

Which makes A act like a subset of B, since its embedding into ? passes through B and its enforcement as well.

This also "recovers" the contravariance of the arrow typing rule by viewing $A \sqsubseteq B$ as 2 judgments, one from which we can extract the section, and the other from which we extract the retraction. More precisely, we can define $A \leq^+ B$ to mean the cast $A \Rightarrow B$ is a section (with opposite cast its retraction) and $A \leq^- B$ to mean the cast $A \Rightarrow B$ is a retraction (with the opposite cast its section). Syntactically, this is the same as defining $A \leq^+ B = A \sqsubseteq B$ and $A \leq^- B = B \sqsubseteq A$, but constructively it reflects the action of the $\to$ type on sections and retractions. We can read the rule from WF09:

$$\frac{A' \leq^- A \qquad B \leq^+ B'}{A \to B \leq^+ A' \to B'}$$

constructively, which means to construct a section $s_\to : (A \to B) \rightarrowtail (A' \to B')$, it is sufficient to have a section $s_B : B \rightarrowtail B'$ and a *retraction* $r_A : A' \twoheadrightarrow A$:

$$s_\to = \lambda f. s_B \circ f \circ r_A$$

The retraction $r_\to$ can be constructed using the dual rule and the fact that $s_\to, r_\to$ form a section-retraction pair is a simple calculation.

4 Blame vs Retraction

It is difficult to make a general comparison between the definition in terms of retraction and the definition using blame, because blame is defined as part of the language and doesn't have a general definition. This is an advantage of our definition, because it requires no additional structure in the language. We can at least compare directly WF09's language. While their syntactic rules for type precision are sound for our definition, their definitions of positive and negative subtyping are more permissive than ours, i.e., strictly more judgments hold according to their definition than ours. In particular, in their system any "ground" type such as $\mathbb{B}$, the judgment $\mathbb{B} \leq^- ?$ holds since the cast $\mathbb{B} \Rightarrow ?$ never raises blame as defined in their language. By our definition this judgment does not hold, because of course the composition $? \Rightarrow \mathbb{B} \Rightarrow ?$ is not the identity.

One possible interpretation is that their definition of *blame* is at fault, and it *should* be possible for the cast $\mathbb{B} \Rightarrow ?$ to raise negative blame. For instance consider a continuation $K : ? \to \bot$ that casts to a type of numbers $\mathbb{N}$. Then composing K with the cast $\mathbb{B} \Rightarrow ?$ and any $\mathbb{B}$ value such as

$$\top \xrightarrow{\texttt{true}} \mathbb{B} \Rightarrow ? \xrightarrow{K} \bot$$

reduces to an error. It seems reasonable here to assign blame to K for not living up to the interface $\mathbb{B}$.

This suggests a change in the definition of blame that we hope to consider in future work. We also plan to investigate if this is a failure of complete monitoring in the sense of Dimoulas et al. [2012].

References

C. Dimoulas, S. Tobin-Hochstadt, and M. Felleisen. Complete monitors for behavioral contracts. In *Eu-*

[1]This is a slice category into ? of a category of section-retraction pairs of casts.

ropean Symposium on Programming (ESOP), Mar. 2012.

R. Findler and M. Blume. Contracts as pairs of projections. In *International Symposium on Functional and Logic Programming (FLOPS)*, Apr. 2006.

R. B. Findler and M. Felleisen. Contracts for higher-order functions. In *International Conference on Functional Programming (ICFP), Pittsburgh, Pennsylvania*, pages 48–59, Sept. 2002.

D. Scott. Data types as lattices. *Siam Journal on computing*, 5(3):522–587, 1976.

D. S. Scott. Relating theories of the lambda-calculus. In J. Seldin and J. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 403–450, 1980.

J. G. Siek and W. Taha. Gradual typing for functional languages. In *Scheme and Functional Programming Workshop (Scheme)*, pages 81–92, Sept. 2006.

J. G. Siek and P. Wadler. Threesomes, with and without blame. In *ACM Symposium on Principles of Programming Languages (POPL), Madrid, Spain*, pages 365–376, 2010.

S. Tobin-Hochstadt and M. Felleisen. Interlanguage migration: From scripts to programs. In *Dynamic Languages Symposium (DLS)*, pages 964–974, Oct. 2006.

P. Wadler and R. B. Findler. Well-typed programs can't be blamed. In *European Symposium on Programming (ESOP)*, pages 1–16, Mar. 2009.