Polymorphic Gradual Typing: A Set-Theoretic Perspective

Giuseppe Castagna[1], Victor Lanvin[1], Tommaso Petrucciani[1,2], and Jeremy G. Siek[3]

[1] Université Paris Diderot, France [2] Università di Genova, Italy [3] Indiana University, USA

The aim of this work is to combine gradual typing, as introduced by Siek and Taha [3], with polymorphic union and intersection types as defined in the semantic subtyping approach [2].

Semantic subtyping is a technique by Frisch et al. [2] to define a type theory for union, intersection, and negation type connectives, in particular in the presence of higher-order functions. It consists in defining a semantic interpretation of types as sets (e.g., sets of values of some language) and then defining the subtyping relation as set-containment of the interpretations, whence the name of *set-theoretic types*. The advantage is that, by definition, types satisfy natural distribution laws (e.g., $(t \times s_1) \vee (t \times s_2)$ and $t \times (s_1 \vee s_2)$ are equivalent and so are $(s \to t_1) \wedge (s \to t_2)$ and $s \to (t_1 \wedge t_2)$).

Gradual typing is a recent approach that combines the safety guarantees of static typing with the flexibility and development speed of dynamic typing [3]. The idea behind it is to introduce an *unknown* type, often denoted by "`?`", used to inform the compiler that additional type checks may need to be performed at run time. Programmers can add type annotations to a program *gradually* and control precisely how much checking is done statically versus dynamically. The typechecker ensures that the parts of the program that are typed with *static types*—i.e., types that do not contain "`?`"—enjoy the type safety guarantees of static typing (well-typed expressions never get stuck), while the parts annotated with *gradual types*—i.e., types in which `?` occurs—enjoy the same property modulo the possibility to fail on some dynamic type check.

In this work we explore a new idea to interpret gradual types, namely, that the unknown type `?` acts like a type variable, but a peculiar one since each occurrence of `?` can be substituted by a different type. More precisely: a semantics of gradual types can be given by considering *each occurrence of* `?` *as a placeholder for a possibly distinct type variable*. We believe this idea to be the essence of gradual typing, and we formalize it by defining an operation of *discrimination* (denoted by $\odot$) which replaces each occurrence of `?` in a gradual type by a type variable.

Discrimination is the cornerstone of our semantics for gradual types: by applying discrimination we map a gradual type into a polymorphic set-theoretic type; then we use the semantic interpretation of the latter into a set, to interpret, indirectly, our initial gradual type. We use this semantic interpretation to revisit some notions from the gradual typing literature: we restate some of them, make new connections between them, and introduce new concepts. In particular, we use discrimination to define two preorders on gradual types: the *subtyping relation* (by which $\tau_1 \leq \tau_2$ implies that an expression of type τ_1 can be safely used where one of type τ_2 is expected) and the *materialization relation* ($\tau_1 \preccurlyeq \tau_2$ iff τ_2 is more precise—i.e., it has the same form but fewer occurrences of `?`—than τ_1). Using these preorders, we can define a gradual type system in a declarative form where terms are typed with standard rules (e.g., those of Hindley-Milner type systems) plus two non-syntax-directed rules corresponding to the two relations: subsumption (for subtyping) and materialization. The simplicity of this extension contrasts with current literature where gradual typing is obtained by embedding tests of consistency or of consistent subtyping (two non-transitive relations) in elimination rules.

Finally, our formalization partially brings to light the logical meaning of the cast language, the target language used by the compiler to add the dynamic type checks that ensure type safety. In our framework, expressions of the cast language encode (modulo the use of subsumption) the type derivations for the gradually-typed language. As such, the cast language looks like the missing ingredient in the Curry-Howard isomorphism for gradual typing disciplines. An intriguing direction for future work is to study the logic associated to these expressions.

A glimpse of formalization. We define subtyping on gradual types by *discrimination*, that is, by converting occurrences of ? to type variables to obtain static types (i.e., types without ?). Then, we reuse the existing semantic subtyping relation for polymorphic set-theoretic static types. That is, we define subtyping as

$$\tau_1 \leq \tau_2 \stackrel{\text{def}}{\iff} \exists t_1 \in \odot(\tau_1), t_2 \in \odot(\tau_2).\, t_1 \leq_t t_2$$

where $\odot(\tau)$ is the set of the possible discriminations of τ and where $\leq_t$ is the subtyping relation on polymorphic set-theoretic types defined in [1] (we use the meta-variables τ and t for gradual and static types, respectively). It turns out that in order to check subtyping, it is not necessary to consider all the possible discriminations of the two types. It suffices to check that the relation holds when we replace in both types all covariant occurrences of ? by one variable and all contravariant occurrences by a second variable different from the first (this property holds only for deciding subtyping and not, say, for typing: functions of type $? \to ? \to \mathsf{Int}$ and of type $\alpha \to \alpha \to \mathsf{Int}$ are completely different beasts).

The *materialization* relation on gradual types $\tau_1 \preccurlyeq \tau_2$ is defined as follows:

$$\tau_1 \preccurlyeq \tau_2 \stackrel{\text{def}}{\iff} \exists t_1 \in \odot(\tau_1). \exists \theta.\, t_1\theta = \tau_2$$

where θ is a substitution mapping the variables inserted by discrimination to gradual types. It turns out that this new definition characterizes the inverse relation of the "precision" relation of [4], it extends it to set-theoretic types and generalizes it since it is (type) syntax agnostic. We use these typing rules:

$$\frac{\Gamma(x) = \forall \vec{\alpha}.\, \tau}{\Gamma \vdash x\colon \tau[\vec{t}/\vec{\alpha}]} \qquad \frac{\Gamma, x\colon t \vdash e\colon \tau}{\Gamma \vdash (\lambda x.\, e)\colon t \to \tau} \qquad \frac{\Gamma, x\colon \tau' \vdash e\colon \tau}{\Gamma \vdash (\lambda x\colon \tau'.\, e)\colon \tau' \to \tau} \qquad \frac{\Gamma \vdash e_1\colon \tau' \to \tau \qquad \Gamma \vdash e_2\colon \tau'}{\Gamma \vdash e_1\, e_2\colon \tau}$$

$$\frac{\Gamma \vdash e_1\colon \tau_1 \qquad \Gamma, x\colon \forall \vec{\alpha}.\, \tau_1 \vdash e_2\colon \tau}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2\colon \tau}\ \vec{\alpha} \sharp \Gamma \qquad [\leq]\ \frac{\Gamma \vdash e\colon \tau'}{\Gamma \vdash e\colon \tau}\ \tau' \leq \tau \qquad [\preccurlyeq]\ \frac{\Gamma \vdash e\colon \tau'}{\Gamma \vdash e\colon \tau}\ \tau' \preccurlyeq \tau$$

The first five rules are standard rules for Hindley-Milner type systems, with the subtlety that function parameters can be assigned gradual types only if they are explicitly annotated.

The type system is declarative insofar as the two key relations on gradual types, subtyping and materialization, are added to the system by two specific non-syntax-directed rules, $[\leq]$ and $[\preccurlyeq]$, stating that these relations can be used in any context. This is a novelty and one of the contributions of our work. Hitherto, gradual typing was obtained by inserting checks in elimination rules; although this describes the algorithmic aspects of the implementation, it hides the logical meaning of "graduality". Rule $[\leq]$ is standard (but uses a new subtyping relation), and states that the type of every well-typed expression can be subsumed to a supertype. Rule $[\preccurlyeq]$ is new (but uses a relation already existing in the literature on gradual types) and is the counterpart of subsumption for the precision relation. It states that we can subsume every well-typed expression to a more precise type obtained by replacing some occurrences of ? by a gradual type: this is a declarative way to embed consistency checks in the type system.

[1] G. Castagna and Z. Xu. Set-theoretic foundation of parametric polymorphism and subtyping. In *ICFP '11*, 2011.

[2] A. Frisch, G. Castagna, and V. Benzaken. Semantic subtyping: dealing set-theoretically with function, union, intersection, and negation types. *Journal of the ACM*, 55(4), 2008.

[3] J. G. Siek and W. Taha. Gradual typing for functional languages. In *Proceedings of Scheme and Functional Programming Workshop*, 2006.

[4] J. G. Siek, M. M. Vitousek, M. Cimini, and J. T. Boyland. Refined criteria for gradual typing. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 32, 2015.