Semantic (aka soft) types

DRAFT VERSION

Joshua Chen

Computational Logic
Department of Computer Science
University of Innsbruck

November 15, 2019

Abstract

This short expository note discusses abstractly the concept of *semantic* or *"soft" types*, to make plain that they are in fact already ubiquitously used by mathematicians, and to cast them as a general framework applicable to any logical system, including type theory itself.

1 Introduction

The notion of a "soft type" is prevalent in some formal proof circles [5, 6, 7, 8, 12], where it arose out of the need to organize the complex hierarchies and ontologies of objects considered in mathematical practice.

Consider the following example. In contrast to type theory, in set theory there is only one logical type, namely the type of sets. However, in practice we distinguish particular sets according to chosen properties. For instance, in ZFC we may make the following definitions:

Soft type		First-order definition
A set n is a **natural number**	iff	$n \in \omega$, where ω is the first nonzero limit ordinal.
A set x is an **ordered triple**	iff	x is of the form $(a,(b,c))$, where we use the Kuratowski definition of the ordered pair.
An ordered triple $(G,\cdot,e)$ is a **group**	iff	$(\cdot : G \times G \to G) \wedge (\text{associativity of } \cdot) \wedge \dots$.
A group $(G,\cdot,e)$ is **abelian**	iff	$\forall g,h \in G\ (g \cdot h = h \cdot g)$.

In the definitions above, n, x, $(G,\cdot,e)$ etc. are all sets, but if they satisfy their defining properties we consider them instances of a particular *kind* of set. These "kinds" are what we refer to as soft types.

Idea. A **soft type** T in a mathematical/logical system is a classification given to objects of the system that satisfy the defining property of T, which is formulated as a predicate in the logic of the system. T may be viewed as a *collection* of particular kinds of objects, or as a *label* applied to such objects.

We stress that, as opposed to types in type theory, soft types are not built into the underlying logical formalism of the system. Instead they essentially correspond to logical predicates, which may or may not hold for an object of the system. In particular, a term is allowed to be a member of (or, is allowed to be assigned) multiple soft types, a case which frequently arises in practice.

Observe that while we have introduced soft types in the context of set theory and first-order logic, there is nothing limiting us to any particular logic or theory. We may equally well formulate soft types for theories of higher-order predicate logic, simple type theory, or even dependent type theory.

We end this section apophatically and say what soft types, as described here, are *not*:

- They do not have any relation to "weak typing" as the term is applied to programming languages.
- Though much of the spirit is the same, a soft type is not strictly a type in the sense of the term as used in type theory. In particular, as we have already seen, the strict formal properties of type-theoretic types need not apply to soft types.

As searching online for "soft type" mostly turns up results along the lines of the first point above, I propose we also call soft types *semantic types*[1].

2 Why semantic types?

Semantic types are already handled implicitly and ubiquitously by working mathematicians, who recognize when an object of study X is simultaneously an instance of multiple types $T_1, T_2, \ldots$, and freely switch between viewing X now as a T_i-instance, then as a T_j-instance, and working with it accordingly. We simply introduce an abstract framework for formalizing what happens in practice. In a sense, all we are doing is book-keeping, albeit of a common and important form, which is also of particular help in making proof assistants more user-friendly for daily mathematical work.

Why not type theory, then? Dependent type theory already natively allows the creation of types defined directly via logical predicates, since every predicate is a type via the Curry-Howard correspondence. Indeed, type theory is both powerful and theoretically elegant, so while not wishing to detract from its approach, we offer the following advantages of semantic types:

1. Power and flexibility — Semantic types are not bound to the Curry-Howard viewpoint, and their formulation as a general framework may use any logic of choice. The type discipline allows for more flexibility: terms may have multiple types, and there are as many ways to form types as there are ways to form predicates. In a sufficiently powerful theory, one gets all the usual types of type theory and possibly more, for example the well-founded recursion theorem of ZFC gives us semantic types corresponding to inductive and more general recursive types.
2. Simplicity and naturalness — Constructions like subtyping and intersection types, which take some work to define properly in type theory [1, 3, 10], are almost trivial in first-order logic semantic type systems. From a formalization-centric point of view, dependent type theory is often a disincentive for mathematicians—among other things, it takes time to learn to use, proof terms can create some overhead, and its logic is not natively classical. By tailoring the system to fit existing mathematical practice, first-order set-theoretic systems with semantic typing may motivate greater adoption of proof assistants.

The distinction between the type-theoretic and semantic type approaches may be loosely compared to that between strongly and weakly-typed programming languages. Once one learns to use strong typing well, good things are guaranteed, at the cost of some flexibility and extra effort. Weakly-typed languages may occasionally feel more "hacky", with fewer nice theoretical properties, but they allow one to jump in and get things done quickly and directly.

3 Semantic types in set theory

In traditional set-theoretic foundations of mathematics, the semantic types correspond to subclasses

$$T = \{x \mid \phi(x)\}$$

of the universe of sets, for first-order predicates ϕ. These types can be very rich, encompassing such constructions as subtypes, intersection types, dependent types, and inductive types. Yet, as the following table illustrates, in many cases this rich system arises directly and naturally out of simple properties of first-order logic. In the following table, let $A = \{x \mid \phi(x)\}$ and $B = \{x \mid \psi(x)\}$.

[1] In contrast with "syntactic type", which are types that are part of the syntax of the underlying logic. I'm not actually entirely happy with this terminology; other suggestions are welcome.

Type construction	Defining property	Formulation as semantic types
Subtypes $A < B$	$t\colon B$ whenever $t\colon A$.	$A < B$ if and only if $\forall x\,(\phi(x) \longrightarrow \psi(x))$ holds.
Intersection types $A \cap B$	$t\colon A \cap B$ if and only if $t\colon A$ and $t\colon B$.	$A \cap B = \{x \mid \phi(x) \wedge \psi(x)\}$.
Dependent types $P(x_1\colon X_1, \ldots, x_n\colon X_n)$	A type P which depends on terms of type X_1, …, X_n.	$P = \{x \mid \phi(x; x_1, \ldots, x_n)\}$, where the defining predicate ϕ of P has parameters $x_i \in X_i$.
W-types	Well-founded inductive types T	$T = \bigcup\{F(z) \mid z \in Z\}$, where Z is a well-founded set and $F\colon Z \to V$ is defined by well-founded recursion.

It is an illustrative exercise to show how to define the semantic type $\mathrm{List}\,\mathbb{N}$ of finite lists of natural numbers.

Note that the the semantic type constructions of dependent and inductive types may be more general than the versions common in dependent type theory. In particular, dependent semantic types need not just be dependent functions or sums, and inductive semantic types may be defined over well-founded sets Z that are not isomorphic to $(\mathbb{N}, <)$, and hence need not be countable.

4 Semantic types in type theory

Semantic types allow us to go beyond the type system of simple type theory (also known as higher-order logic) and build dependent types on top. This is an approach described in [4] and used in the Isabelle proof assistant [11] implementations of constructive type theory (described in Chapter 5 of [9]) and homotopy type theory [2].

References and further reading

[1] Ali Assaf. "A calculus of constructions with explicit subtyping". In: *20th International Conference on Types for Proofs and Programs (TYPES 2014)*. Ed. by Hugo Herbelin, Pierre Letouzey, and Matthieu Sozeau. Vol. 39. LIPICS. Institut Henri Poincaré, Paris, France, May 2014. URL: https://hal.archives-ouvertes.fr/hal-01097401.

[2] Joshua Chen. *Isabelle/HoTT*. GitHub repository. 2019. URL: https://github.com/jaycech3n/Isabelle-HoTT.

[3] Joshua Dunfield. "Annotations for Intersection Typechecking". In: *arXiv e-prints*, arXiv:1307.8204 (2013), arXiv:1307.8204. arXiv: 1307.8204 [cs.PL].

[4] Bart Jacobs and Tom Melham. "Translating Dependent Type Theory into Higher Order Logic". In: *Typed Lambda Calculi and Applications*. Ed. by Marc Bezem and Jan Friso Groote. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 209–229. ISBN: 978-3-540-47586-6.

[5] Alexander Krauss. *Adding Soft Types to Isabelle*. https://www21.in.tum.de/~krauss/papers/soft-types-notes.pdf. Unpublished note. 2010.

[6] Leslie Lamport and Lawrence C. Paulson. "Should Your Specification Language Be Typed". In: *ACM Trans. Program. Lang. Syst.* 21.3 (May 1999), pp. 502–526. ISSN: 0164-0925. DOI: 10.1145/319301.319317. URL: http://doi.acm.org/10.1145/319301.319317.

[7] Adam Naumowicz and Josef Urban. *A Guide to the Mizar Soft Type System*. http://alioth.uwb.edu.pl/~adamn/types2016_slides.pdf. Talk given at TYPES 2016.

[8] Adam Naumowicz and Josef Urban. *A Guide to the Mizar Soft Type System*. 2016. URL: https://www.semanticscholar.org/paper/A-Guide-to-the-Mizar-Soft-Type-System-%E2%88%97-Naumowicz-Urban/82f2dfa4437b79a35aaf72ce71e85a919103b88f.

[9] Lawrence C. Paulson and contributors. *Isabelle's Logics*. 2018. URL: https://isabelle.in.tum.de/dist/Isabelle2018/doc/logics.pdf.

[10] Giovanni Sambin and Silvio Valentini. "Building up a toolbox for Martin-Löf's type theory. Part I: subset theory". In: *Oxford Logic Guides* (Jan. 1998).

[11] University of Cambridge, Technische Universität München, and contributors. *Isabelle*. 2019. URL: https://isabelle.in.tum.de.

[12] Freek Wiedijk. *The next generation of proof assistants: ten questions.* `https://www.cs.ru.nl/~freek/talks/lsfa.pdf`. Talk given at LSFA 2010.