Noninvasive Polarized Subtyping for Inductive Types

Théo Laurent[1] and Kenji Maillard[2]

[1] Inria Paris, Prosecco Team
[2] Inria Rennes-Bretagne Atlantique, Gallinette Team

Abstract

Polarized subtyping extends a type system with annotations tracking the variance of type constructors with respect to subtyping. Type system with polarized subtyping are highly expressive at the cost of additional information decorating types and contexts. In the dependently typed setting, for instance inside proof assistants, this pervasive complexity obstruct its integration within existing implementations. We advocate a less invasive approach, extending the Calculus of Inductive Constructions (CIC) with a schema for polarized inductive types while retaining vanilla typing jugdments for most of the theory. The design of our extension is guided by a model exhibiting subtyping as a particular order structure on types.

Type Theory and Polarized Subtyping

The Calculus of Inductive Constructions (CIC) is both a powerful logic, used as the basis for many proof assistants, e.g. Coq, and an expressive type system underlying programming languages such as F* [9]. Dependent types, introduced notably though dependent product types $\Pi(x : A).\, B$ and a hierarchy of universes $\square_{i\in\mathbb{N}}$, provide a uniform language for programs as well as specifications. In practice, implementations of CIC sometimes extend the theory with different forms of subtyping to make it more flexible, for instance cumulativity in Coq [10] or refinement types in F*. Both of these instances however lack from structural forms of subtyping [5] for general inductive types. A structural subtyping discipline allows one to derive subtyping relationships between inductive types from their structure and assumptions on their parameters. For instance, in such a system one can derive that $\texttt{List}\, A$ is a subtype of $\texttt{List}\, B$ whenever A is a subtype of B. Said otherwise, $\texttt{List} : \square_i \to \square_i$ is monotone for the subtyping order endowing $\square_i$. *Polarities* [1, 7] endow types with annotations that capture their monotonicity. As an example, the function type constructor $(\to)$ can be given the annotated type $\Pi(A :_- \square_i).\Pi(B :_+ \square_i).\, \square_i$ expressing that it is antitone in its domain A and monotone in its codomain B with respect to subtyping. In a dependently typed setting, tracking these polarities pervasively in types and terms result in complex type theories that seems unlikely to be practical.

External Polarities for Inductive Types

We advocate for a pragmatic approach suitable for adoption by proof assistants. In order to stay closer to CIC, we restrict polarities to where they are most relevant in practice: on parameters of inductive types. This restriction allow us to keep polarities as *external* entities of the type theory. Following the practice of [8], we propose to collect inductive declarations in a signature Σ and only alter context and types so that they can be decorated with additional variance information. Our system feature four kind of variance: discrete (`dis`), covariant $(+)$, contravariant $(-)$ and codiscrete (`cod`). Continuing with the example of parametrized Lists, this inductive is represented as the following entry in the signature:

$$\Sigma_{\texttt{List}} := \{\, \texttt{List}\,(A :_+ \square_i) := \texttt{nil} \mid \texttt{cons}\,(hd : A, tl : \texttt{List}\, A)\,\}.$$

The parameter $A : \square_i$ is annotated with a polarity $+$ indicating that the type constructor `List` is monotonic with regard to subtyping in this parameter. For an inductive entry to be well formed with respect to its polarity annotations, we must ensure that the parameters are used adequately in the constructors. For `List`, the two instances of A in `cons` are indeed in positive positions. For general inductives, we use polarized typing judgments inspired by directed type theories [4, 7] to track adequately these polarities and formulate the well-formedness conditions.

The monotonicity informations captured through polarities in the signature induce corresponding structural subtyping rules and coercions [6]. In the case of `List`, this pschema generate the following rule:

$$\frac{\Sigma_{\texttt{List}}; \Gamma \vdash \texttt{List} : \Pi(A :_+ \square_i).\, \square_i \qquad \Sigma_{\texttt{List}}; \Gamma \vdash A \preceq B}{\Sigma_{\texttt{List}}; \Gamma \vdash \texttt{List}\, A \preceq \texttt{List}\, B} \tag{1}$$

Backing Up our Approach with a Syntactic Translation

To support the design of our type theory and establish its metatheory, we build upon the syntactic models of [3]. We construct a model for our theory by compiling it to Coq, translating a typing judgement $\Gamma \vdash t : A$ to $[\![\Gamma]\!] \vdash [t] : [\![A]\!]$. A peculiarity of our model is the use of a second translation $[\![-]\!]_\varepsilon$ for the interpretation of subtyping judgements. This translation equips any closed type A with a binary relation $[\![A]\!]_\varepsilon : [\![A]\!] \to [\![A]\!] \to \square_i$ in a fashion reminiscent of binary parametricity [2]. In particular, the subtyping order arises as the relation $[\![\square_i]\!]_\varepsilon$ on the universe. The stratification of the model into two distinct interpretations illustrate the external nature of polarities: indeed, $[\![-]\!]$ merely erases polarities and only $[\![-]\!]_\varepsilon$ take polarities into account.

The pair of translations $[-], [\![-]\!]$ is relatively simple, preserving all term and type construction from CIC homomorphically but universes $\square_i$ and subtyping coercions. Indeed, in order to maintain a stratification between typing and subtyping translations, we interpret subtyping coercions as functions defined by case analysis on types. We do so using universes of codes to reify types, following the approach of ad-hoc polymorphism in [3]. As a consequence, these coercions can be computed by structural induction on these codes, only relying on the mere existence of subtyping relations to discard impossible cases.

Conversely, $[-]_\varepsilon, [\![-]\!]_\varepsilon$ crucially uses the polarity-annotated types in the signature to ensure that the interpretation of type constructors are adequately monotone. In particular, the structural subtyping rules attached to each inductive is validated by this translation. Continuing with the example of `List`, the translation give us

$$\begin{aligned}
[\![\Pi(A :_+ \square_i).\, \square_i]\!]_\varepsilon\, [\texttt{List}]\ [\texttt{List}] &:= \Pi(A_1 : [\![\square_i]\!])(A_2 : [\![\square_i]\!]) \\
&\quad (A_\varepsilon : [\![\square_i]\!]_\varepsilon\, A_1\, A_2).\, [\![\square_i]\!]_\varepsilon\, ([\texttt{List}]\, A_1)\, ([\texttt{List}]\, A_2) \\
[\![A \preceq B]\!] &:= [\![\square_i]\!]_\varepsilon\, A\, B \\
[\![\texttt{List}\, A \preceq \texttt{List}\, B]\!] &:= [\![\square_i]\!]_\varepsilon\, ([\texttt{List}]\, A)\, ([\texttt{List}]\, B)
\end{aligned}$$

Hence, from the term $[\texttt{List}]_\varepsilon : [\![\Pi(A :_+ \square_i).\, \square_i]\!]_\varepsilon\, [\texttt{List}]\ [\texttt{List}]$ and a witness $[A \preceq B] : [\![A \preceq B]\!]$ we are able to construct $[\texttt{List}]_\varepsilon\, [A]\, [B]\, [A \preceq B] : [\![\texttt{List}\, A \preceq \texttt{List}\, B]\!]$, validating the rule (1).

References

[1] Andreas Abel. Polarised subtyping for sized types. *Math. Struct. Comput. Sci.*, 18(5):797–822, 2008.

[2] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Proofs for free — parametricity for dependent types. *Journal of Functional Programming*, 22(02):107–152, 2012.

[3] Simon Boulier, Pierre-Marie Pédrot, and Nicolas Tabareau. The next 700 syntactical models of type theory. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, pages 182–194, 2017.

[4] Daniel R. Licata and Robert Harper. 2-dimensional directed type theory. In *Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics, MFPS 2011, Pittsburgh, PA, USA, May 25-28, 2011*, pages 263–289, 2011.

[5] Zhaohui Luo and Robin Adams. Structural subtyping for inductive types with functorial equality rules. *Math. Struct. Comput. Sci.*, 18(5):931–972, 2008.

[6] Zhaohui Luo, Sergei Soloviev, and Tao Xue. Coercive subtyping: Theory and implementation. *Inf. Comput.*, 223:18–42, 2013.

[7] Andreas Nuyts. Toward a directed homotopy type theory based on 4 kinds of variance. Master's thesis, Katholieke Universiteit Leuven, 2015.

[8] Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. The metacoq project. *J. Autom. Reason.*, 64(5):947–999, 2020.

[9] Nikhil Swamy, Cătălin Hriţcu, Chantal Keller, Aseem Rastogi, Antoine Delignat-Lavaud, Simon Forest, Karthikeyan Bhargavan, Cédric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, Jean-Karim Zinzindohoué, and Santiago Zanella-Béguelin. Dependent types and multi-monadic effects in F*. In *43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 256–270. ACM, January 2016.

[10] Amin Timany and Matthieu Sozeau. Cumulative inductive types in Coq. In *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, pages 29:1–29:16, 2018.