Canonical Bidirectional Typing via Polarised System L

Zanzi Mihejevs

Glasgow Lab for AI Verification

1 Abstract

What is the relationship between polarity and bidirectional typing? It has long been observed that there is a connection between the two[Kri], but the precise relationship has remained unclear. Moreover, it has been argued [McB] that the link itself is a red herring, and that bidirectional typing is better explained not by polarity but by chirality - the duality between producers and consumers.

Polarised System L [Dow17] is a type theory that combines both dualities - the positive fragment is driven by a cut between a primitive producer and a pattern, and the negative fragment is driven by a cut between a primitive consumer and a co-pattern.

Remarkably, linear System L admits a canonical bidirectional typing discipline based on a combination of ideas from both standard and co-contextual typing, giving us a "bi-contextual" typing algorithm.

This lets us equip a type system based on classical linear logic - containing all four connectives and derivable implication and co-implication - with a bidirectional discipline where all typing annotations are exclusively limited to shifts between sythesisable and checkable expressions.

2 Introduction

Bidirectional typing has established itself as an effective approach to developing type-driven type-checking algorithms[DK21], however equipping a type system with a bidirectional discipline can still be more of an art than a science. This is especially true when it comes to type systems with non-trivial elimination forms, such as those involving binding. Prior work [MRK22] on polarised approaches to bidirectional typing has shown that utilising polarity in algorithmic type-checking can significantly simplify the type inference algorithm. We show how using System L rather than CBPV allows us to take this even further and give a decidable type-inference algorithm for a type-system based on full linear logic.

The advantage of System L over CBPV is that while both calculi have a notion of polarity, CBPV's split between values and computations is tied to the polarity of each type - value judgements correspond to positive types, and computation judgements correspond to negative types. On the other hand, in polarised System L both positive and negative types each have a pair of judgements corresponding to their own notion of producer and consumer terms. Moreover, each polarity has its own notion of focused judgement, which we call its 'principal chirality' - positive types are focused on producer terms, while negative types are focused on consumers.

3 Principal Chirality

The key idea of our approach is that each polarity has a 'principal chirality' corresponding to the focused judgement, and the 'auxiliary chirality' corresponding to the unfocused judgement.

Type-checking each polarity proceeds by first synthesizing the type of the judgement corresponding to the principal chirality, then checking the type of the judgement of the auxiliary chirality against the synthesized type. The crucial difference between the two phases is in the way that information gets propagated in the principal and auxiliary contexts. The principal context is built-up bottom-up using the co-contextual [EBK+15] approach (which means that positive variables and negative co-variables are both checkable), while the auxiliary contexts are built-up top-down (so negative variables and positive co-variables are synthesised). This combination of contextual and co-contextual typing is why we call this a 'bi-contextual' typing discipline.

4 Typing Algorithm

Remarkably, once we adopt the bi-contextual typing discipline, we discover that System L requires almost no further modifications to make it fit into this framework. The checkable/synthesisable discipline of each connective is fully determined by its canonical place in polaraised System L, and the only modification that we need to do is to add annotations on two shifts.

To type-check the positive fragment, we start with a cut between a producer and a pattern.

$$\frac{\Gamma \vdash \mathit{producer} \Leftarrow A | \Delta \quad \Gamma' | \mathit{pattern} \Rightarrow A \vdash \Delta'}{<\mathit{producer}|\mathit{pattern}> : (\Gamma, \Gamma' \vdash \Delta, \Delta')}(\text{Cut})$$

We first synthesise the type of the pattern, then check the type of the producer against the synthesised type, after which we are done. This covers all rules in the positive fragment.

Dually, to type-check the negative fragment, we start with a cut between a consumer and a corresponding co-pattern.

$$\frac{\Gamma, \mathit{copattern} \Rightarrow A \vdash \Delta \quad \Gamma' \vdash \mathit{consumer} \Leftarrow A, \Delta'}{<\mathit{copattern}|\mathit{consumer}> : (\Gamma, \Gamma' \vdash \Delta, \Delta')}(\text{Cut})$$

We first synthesise the type of a co-pattern, then check the type of the producer against the synthesised type, after which we are done. This covers the entirety of the negative fragment.

Finally, in order to type the full calculus, we need to say what happens to negation and the shifts. Curiously, negation inverts the polarity of each connective, but it does not change their principal chirality - so a value becomes a covalue and vice-versa, and patterns swap with copatterns. This requires no additional annotational burden as it means that checkable judgements remain checkable, and synthesisable judgements remain synthesisable.

The most interesting thing happens when we consider the shifts. The shifts amount to swapping *both* the polarities and principal chiralities of a judgement. On one side, a copattern embeds into a value, and a pattern embeds into a covalue. This corresponds to embedding a synthesisable judgement into a checkable one, and can be done with no issue. But the other way around - embedding a (co)value into a (co)pattern requires us to synthesise the type of a checkable judgement, which is precisely the only two places where we need to place a typing annotation.

References

[DK21] Jana Dunfield and Neel Krishnaswami. Bidirectional typing. *ACM Computing Surveys (CSUR)*, 54(5):1–38, 2021.

[Dow17] Paul Downen. *Sequent Calculus: A Logic and a Language for Computation and Duality.* PhD thesis, University of Oregon, 2017.

[EBK+15] Sebastian Erdweg, Oliver Bračevac, Edlira Kuci, Matthias Krebs, and Mira Mezini. A co-contextual formulation of type rules and its application to incremental type checking. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 880–897, 2015.

[Kri] Neel Krishnaswami. Polarity and bidirectional typechecking. Available at ̈https://semantic-domain.blogspot.com/2018/08/polarity-and-bidirectional-typechecking.html".

[McB] Conor McBride. Basics of bidirectionalism. Available at ̈https://pigworker.wordpress.com/2018/08/06/basics-of-bidirectionalism/".

[MRK22] Henry Mercer, Cameron Ramsay, and Neel Krishnaswami. Implicit polarized f: local type inference for impredicativity. *arXiv preprint arXiv:2203.01835*, 2022.