# Conservativity of Two-Level Type Theory Corresponds to Staged Compilation

András Kovács

Eötvös Loránd University, Budapest, Hungary
kovacsandras@inf.elte.hu

Two-level type theory (2LTT) [1] was originally intended as a tool to make metatheoretical reasoning about constructions in homotopy type theory more convenient, by internalizing an extensional metatheory in a syntactic layer.

**2LTT as a two-stage language.** First, we observe that basic variants of 2LTT formalize two-stage compilation, where the meta-level syntactic fragment contains static (compile-time) computations, and *staging* is the algorithm which performs all static computation, producing object-theoretic syntax as output. The basic rules are the following. There are two universes, $\mathsf{U}_0$ and $\mathsf{U}_1$, where $\mathsf{U}_0$ classifies object-level (runtime) types and $\mathsf{U}_1$ classifies meta-level (compile time) types. There are three *staging operations*.

*Lifting:* for $A : \mathsf{U}_0$, we have $\Uparrow A : \mathsf{U}_1$. From the staging point of view, $\Uparrow A$ is the type of metaprograms which compute runtime expressions of type $A$.

*Quoting:* for $A : \mathsf{U}_0$ and $t : A$, we have $\langle t \rangle : \Uparrow A$. A quoted term $\langle t \rangle$ represents the metaprogram which immediately yields $t$.

*Splicing:* for $A : \mathsf{U}_0$ and $t : \Uparrow A$, we have $\sim t : A$. During staging, the metaprogram in the splice is executed, and the resulting expression is inserted into the output.

Quoting and splicing are definitional inverses. Also, the above operations are the *only* way of crossing between stages; all type formers stay within a single stage, and in particular we cannot eliminate from one stage to a different one. The staging interpretation of 2LTT remains valid with arbitrary assumed type formers. Note that all three operations correspond to features in existing staged systems such as MetaOCaml [4] or typed Template Haskell [8], although none of the existing systems support staging with dependent types.

**Conservativity as staging.** By conservativity we mean the following. There is an embedding morphism $\ulcorner - \urcorner$ which maps from the object theory to the object-level syntactic fragment of 2LTT. 2LTT is conservative if $\ulcorner - \urcorner$ is bijective on types and terms, i.e. $\mathsf{Ty}_{\mathsf{Obj}}\,\Gamma \simeq \mathsf{Tm}_{\mathsf{2LTT}}\,\ulcorner\Gamma\urcorner\,\mathsf{U}_0$ and $\mathsf{Tm}_{\mathsf{Obj}}\,\Gamma\,A \simeq \mathsf{Tm}_{\mathsf{2LTT}}\,\ulcorner\Gamma\urcorner\,\ulcorner A\urcorner$.

A staging algorithm consists of functions $\mathsf{Stage} : \mathsf{Tm}_{\mathsf{2LTT}}\,\ulcorner\Gamma\urcorner\,\mathsf{U}_0 \to \mathsf{Ty}_{\mathsf{Obj}}\,\Gamma$ and $\mathsf{Stage} : \mathsf{Tm}_{\mathsf{2LTT}}\,\ulcorner\Gamma\urcorner\,\ulcorner A\urcorner \to \mathsf{Tm}_{\mathsf{Obj}}\,\Gamma\,A$. We call $\mathsf{Stage}$ *stable* if $\mathsf{Stage} \circ \ulcorner - \urcorner = \mathsf{id}$, and *sound* if $\ulcorner - \urcorner \circ \mathsf{Stage} = \mathsf{id}$. Hence, conservativity is the same as having a sound and stable staging algorithm. In [1], only a weak form of conservativity is shown, which corresponds to staging without soundness.

**Staging by evaluation.** We define $\mathsf{Stage}$ as the evaluation of 2LTT types and terms in the presheaf model over the syntactic category of the object theory. We call this model $\widehat{\mathsf{Obj}}$. The object-level 2LTT fragment is interpreted using sets of types and terms in the object theory. Operationally, this yields closed evaluation for the meta-level 2LTT fragment, and we get naive weakening for object-theoretic terms. Naive weakening can be inefficient, but in practice it can be optimized using De Bruijn levels and delayed variable renamings. The same efficiency issue arises in presheaf-based normalization-by-evaluation, and the same solution applies there.

However, staging can be overall more efficient, because meta-level evaluation is closed (no free variables occur in values).

**Soundness.** Stability of staging follows by straightforward induction on the object theory; soundness requires more effort. We define a *restriction* morphism, which maps from 2LTT to $\widehat{\mathsf{Obj}}$, restricting the meta-level syntactic fragment so that it can only depend on object-level typing contexts, that is, contexts given as $\ulcorner\Gamma\urcorner$ for some $\Gamma$. Then, we show soundness of staging by a proof-relevant logical relation between the evaluation morphism from 2LTT to $\widehat{\mathsf{Obj}}$ and the restriction morphism. We define this logical relation in the internal language of $\widehat{\mathsf{Obj}}$, to avoid the deluge of boilerplate for showing stability under object-theoretic substitution.

Alternatively, staging together with its soundness can be established in a single step, by gluing along the restriction morphism. This interpretation can be more compactly defined using synthetic Tait computability [7].

**Intensional analysis.** This means analyzing the internal structure of object-level terms, i.e. values with type $\Uparrow A$. While intensional analysis can be often simulated with deeply embedded inductive syntaxes at the meta level, it may be more concise and convenient to use native intensional analysis features instead. We consider the interpretation of such features in the presheaf models.

If the object theory has parallel substitutions as syntactic morphisms, then $\widehat{\mathsf{Obj}}$ does not support intensional analysis. For illustration, consider decidable equality of $\Uparrow A$ as an intensional meta-level axiom; this is essentially decidability of definitional equality of object terms. This axiom does not hold in $\widehat{\mathsf{Obj}}$, because inequality of object-level terms is not stable under substitution: inequal variables can be mapped to equal terms.

However, we may choose to only have *weakenings* as morphisms in the object syntax. In this case, decidable equaity for $\Uparrow A$ holds in $\widehat{\mathsf{Obj}}$, since term inequality is stable under weakening. As a trade-off, if there is no notion of substitution in the specification of the object theory, it is not possible to specify dependent or polymorphic types there. This is still sufficient for many practical use cases, for example when the object theory is simply-typed, in which case staging also performs *monomorphization*. In this setup, it makes sense to only have weakening in the equational theory of the object theory, but no $\beta\eta$-rules and no substitution. The reason is that we do not want to equate programs with different performance characteristics, when we do staging in order to improve runtime code performance.

**Future work.** A major line of future work is to connect 2LTT to existing literature on staged compilation. This would involve formalizing existing tricks and techniques, such as let-insertion techniques [5], fusion, various binding-time improvements and CPS conversions [3]. Another line is fleshing out practical details for staging and intensional analysis. In staging, a production-strength solution should include some form of caching, to reduce code duplication. In intensional analysis, some form of induction or pattern matching would be more ergonomic than plain decidability of conversion.

Also, 2LTT could be extended to more general *multimodal* [2] type theories, where modalities represent morphisms between different object-theoretic syntactic categories, in possibly different object theories. A simple example is the closed or "crisp" modality [6] which can be used to represent closed object terms.

# References

[1] Danil Annenkov, Paolo Capriotti, Nicolai Kraus, and Christian Sattler. Two-level type theory and applications. *ArXiv e-prints*, may 2019.

[2] Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal dependent type theory. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 492–506. ACM, 2020.

[3] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial evaluation and automatic program generation*. Prentice Hall international series in computer science. Prentice Hall, 1993.

[4] Oleg Kiselyov. The design and implementation of BER metaocaml - system description. In Michael Codish and Eijiro Sumii, editors, *Functional and Logic Programming - 12th International Symposium, FLOPS 2014, Kanazawa, Japan, June 4-6, 2014. Proceedings*, volume 8475 of *Lecture Notes in Computer Science*, pages 86–102. Springer, 2014.

[5] Oleg Kiselyov and Jeremy Yallop. let (rec) insertion without effects, lights or magic. *CoRR*, abs/2201.00495, 2022.

[6] Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. Internal universes in models of homotopy type theory. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 22:1–22:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[7] Jonathan Sterling. *First Steps in Synthetic Tait Computability*. PhD thesis, Carnegie Mellon University Pittsburgh, PA, 2021.

[8] Ningning Xie, Matthew Pickering, Andres Löh, Nicolas Wu, Jeremy Yallop, and Meng Wang. Staging with class: a specification for typed template haskell. *Proc. ACM Program. Lang.*, 6(POPL):1–30, 2022.