Types as a Specification Language for Creativity

YOUYOU CONG, Institute of Science Tokyo, Japan

Types allow us to write programs that are correct by construction. This technique has recently been applied to creative domains such as music and textiles. We aim to develop a general typed programming language for creativity. The main challenge involved is how to express soft rules that are pervasive in creative domains. As a potential solution, we propose to build a type system in which an expression inhabits a type to a certain degree. We hope that our work will foster interactions among creative fields and bring new insights into programming languages research.

Additional Key Words and Phrases: type systems, creative computing

1 Introduction

Types are often used as a specification language for programs. In particular, advanced types such as refinement types and dependent types allow us to develop programs that are *correct by construction*. Previous work has used this technique to enforce safe usage of resources in concurrent programming [Brady and Hammond 2010], and to guarantee grammatical correctness of strings produced by a pretty printer [Danielsson 2013].

The idea of using types to ensure correctness has also been applied to several creative domains. In the context of music, Szamozvancev and Gale [2017] and Cong and Leo [2019] express the composition rules of counterpoint using dependent types, allowing one to check the correctness of their composition via the type checker of the language. In the context of textiles, Clark and Bohrer [2023] descibe operations on sewn quilts using concepts from Homotopy Type Theory, making it possible to statically determine the precise construction of pieces.

These applications of types suggest a research direction: designing a typed programming language in which one can compose works in *any* creative domain. Such a language would serve as a unified framework for creativity, potentially making it easier for people in different fields to learn from and collaborate with each other. It could also be a source of inspiration for programming languages researchers, in that the demands arising from creativity would lead to unique language design that has never been considered before.

In this paper, we present our ongoing work on designing a general typed programming language for creativity. We first observe that creative domains commonly have hard and soft rules, and then discuss how to express both kinds of rules as types.

2 Hard and Soft Rules in Creative Domains

When composing works in a creative domain, one generally follows guidelines established in that domain. These guidelines include *hard rules*, which enforce certain styles and ensure physical feasibility, and *soft rules*, which focus more on aesthetic aspects. Below, we give examples of hard and soft rules in three creative domains: music, textiles, and dance.

Species Counterpoint. Species counterpoint is a method for combining multiple melodies. Cong [2023] formalizes two-voice species counterpoint using a weighted variant of refinement types that can represent the following rules.

Hard rules

- All intervals on strong beats must be consonant.
- Adjacent intervals must not form direct fifth or octave.

HATRA '25, Singapore
2025.

Soft rules

- Imperfect intervals are preferred to perfect intervals.
- Contrary motion is preferred to direct motion.

Rug Patterns. Traditional rugs usually have patterns unique to a specific region. Dalvandi et al. [2010] design rug patterns with spirals and flowers using a genetic algorithm, which incorporates the following rules.

Hard rules

- The distances between spiral layers must be constant.
- Connected spirals must revolve in opposite directions.

Soft rules

- Spirals should not have too many or too few layers.
- Flowers should be distributed evenly.

Ballet Motions. Ballet dancers start their day with a series of warm-up exercises called barre exercises. LaViers et al. [2011b] and LaViers et al. [2011a] descibe the moves of barre exercises using linear temporal logic based on the following rules.

Hard rules

- Two legs in different positions must never be both off the ground.
- Jumps must not happen from a non-flat foot.

Soft rules

- When one foot is flat, the other leg leaves the floor.
- In a quick, lively movement, jumps occur frequently.

3 Design Ideas for a Typed Language for Creativity

Having seen examples of hard and soft rules in creative domains, we explore the design space of a programming language in which those rules are enforced via types. Our criteria for the language are as follows.

- The language rejects works that break hard rules but accepts works that break soft rules. This guarantees the minimum quality of composed works while allowing room for creativity.
- The language can express to what degree a work is correct based on which soft rules are satisfied or broken. This information is useful for ranking works, whose results are in turn useful for implementing creativity support in the form of code suggestion and synthesis.

Below, we describe our current ideas of how to design such a language, focusing on its type system.

3.1 Syntax of Types

As an interface for expressing hard and soft rules, we use a variation of refinement types that have two sets of refinement predicates. Thus, a refinement type in our language takes the form $\{b \mid \phi_h \mid \phi_s\}$, where b is a base type, ϕ_h is a predicate representing hard rules (henceforth *hard predicate*), and ϕ_s is a predicate representing soft rules (henceforth *soft predicate*). Both kinds of predicates are a conjunction of boolean-valued expressions that contains a distinguished variable representing the inhabitant of the overall type.

3.2 Typing Judgment

Given an expression e that fully satisfies the hard predicate of a refinement type τ but not the soft predicate, we consider e as inhabiting τ to a certain degree. To represent this relation, we augment a typing judgment with a number d as in $\Gamma \vdash e : \tau \,\&\, d$. The value of d is determined by which soft rules are satisfied (if we take d as rewards) or broken (if we take d as penalties), and how important those rules are. This kind of graded typing judgment has been used previously to formalize opinion dynamics [Arya et al. 2022] and to perform probabilistic reasoning [Cooper et al. 2015].

3.3 Typing Rules

Naturally, we would wish to compute the correctness degree of a non-atomic expression from the correctness degrees of its constituents. This leads to typing rules resembling those found in an effect system. We are thinking of modeling our type system after the call-by-name effect system of McDermott and Mycroft [McDermott and Mycroft 2018], where the effect of a function argument is duplicated if it is used multiple times in the function body.

4 Conclusion and Open Questions

We presented our ongoing work on designing a typed programming language for creativity. The key idea is to use a graded notion of typing to account for soft rules.

There are many questions left open. For example, what creativity support can we provide using types? Also, would the ability to express soft rules be useful in general programming as well? We look forward to discussing these questions with the HATRA audience.

References

Shreya Arya, Greta Coraglia, Paige North, Sean O'Connor, Hans Riess, and Ana Tenório. 2022. Fuzzy Type Theory for Opinion Dynamics. Presented at the Applied Category Theory Conference 2022.

Edwin Brady and Kevin Hammond. 2010. Correct-by-construction concurrency: Using dependent types to verify implementations of effectful resource usage protocols. *Fundamenta Informaticae* 102, 2 (2010), 145–176.

Charlotte Clark and Rose Bohrer. 2023. Homotopy Type Theory for Sewn Quilts. In *Proceedings of the 11th ACM SIGPLAN International Workshop on Functional Art, Music, Modelling, and Design* (Seattle, WA, USA) *(FARM 2023)*. Association for Computing Machinery, New York, NY, USA, 32–43. doi:10.1145/3609023.3609803

Youyou Cong. 2023. Weighted Refinement Types for Counterpoint Composition. In *Proceedings of the 11th ACM SIGPLAN International Workshop on Functional Art, Music, Modelling, and Design* (Seattle, WA, USA) *(FARM 2023)*. Association for Computing Machinery, New York, NY, USA, 2–7. doi:10.1145/3609023.3609804

Youyou Cong and John Leo. 2019. Demo: Counterpoint by Construction. In *Proceedings of the 7th ACM SIGPLAN International Workshop on Functional Art, Music, Modeling, and Design* (Berlin, Germany) *(FARM '19)*. Association for Computing Machinery, New York, NY, USA, 22–24. doi:10.1145/3331543.3342578

Robin Cooper, Simon Dobnik, Shalom Lappin, and Staffan Larsson. 2015. Probabilistic Type Theory and Natural Language Semantics. In *Linguistic Issues in Language Technology, Volume 10*.

Arefe Dalvandi, Pooya Amini Behbahani, and Steve DiPaola. 2010. Exploring Persian rug design using a computational evolutionary approach. In *Electronic Visualisation and the Arts (EVA 2010)*. BCS Learning & Development.

Nils Anders Danielsson. 2013. Correct-by-construction pretty-printing. In *Proceedings of the 2013 ACM SIGPLAN Workshop on Dependently-typed Programming*. 1–12.

Amy LaViers, Yushan Chen, Calin Belta, and Magnus Egerstedt. 2011a. Automatic sequencing of ballet poses. *IEEE robotics & automation magazine* 18, 3 (2011), 87–95.

Amy LaViers, Magnus Egerstedt, Yushan Chen, and Calin Belta. 2011b. Automatic generation of balletic motions. In *2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*. IEEE, 13–21.

Dylan McDermott and Alan Mycroft. 2018. Call-by-need effects via coeffects. *Open Computer Science* 8, 1 (2018), 93–108.

Dmitrij Szamozvancev and Michael B. Gale. 2017. Well-Typed Music Does Not Sound Wrong (Experience Report). In *Proceedings of the 10th ACM SIGPLAN International Symposium on Haskell* (Oxford, UK) *(Haskell 2017)*. Association for Computing Machinery, New York, NY, USA, 99–104. doi:10.1145/3122955.3122964