1 Expressing category theory through Haskell

1.1 Category theory as OOP with *all* data private

I used to think that the relationship between functional and object oriented programming was something like "functional programming is like "ordered set theory," basically everything amounts to processing lists, with OOP being what you graduate to when you want to work with object sin other categories, with all their internal structure.

Functional is actually the more category theoretic frame: in functional programming, no internals of any object are exposed, because objects have no internals. Arrows are everything.

It's often possible to recover lots of information about the private attributes and methods of some class by investigating how instances of the class behave.

2 A

[...]

2.1 Typed functional programming as a foundation for ML models

Problem 2.1.1. Systems like *OpenAI Codex* ↗ [CTJ+21], and *CodeT5* ↗ [WWJH21, LWG+22, WLG+23], and *AlphaCode* ↗ [LCC+22] generate code as plain text. This leads to (i) syntactically valid but semantically broken code, (ii) type errors, and (iii) violations of safety constraints.

Possible solution: A typed lambda calculus foundation provides rich static guarantees, via *purity*, *types*, *resource constraints*, *better priors for code generation*, *type-directed completion.*

2.2 Typed Code Models

Models trained on type annotations (e.g., Haskell, OCaml, Rust) can perform type-aware generation.

2.3 Neural Program Synthesis with Types

e.g., DeepCoder and successors use type information as features to guide synthesis.

2.4 Monads and Effect Systems in ML Safety

a. Monads as Models of Effects Monads are widely used in Haskell to isolate side effects (IO, randomness, state), and to model pipelines.

ML models, especially in reinforcement learning and simulation, deal with effects (e.g., randomness, stateful environments). Using monadic abstractions could:

make dataflow pipelines compositional and safe,

allow reasoning about effects (e.g., logging, uncertainty),

embed constraints in types (e.g., a function that only accesses a read-only dataset).

This is one of the core ideas in:

Probabilistic Programming Languages (PPLs) like Hakaru, Anglican, and Pyro, where the probabilistic model is itself a monadic computation.

Effect systems in ML IRs, such as Google's Relay (part of TVM), which tracks purity/effects.

2.5 Category Theory and Differentiable Programming

Category theory underpins both monads and many concepts in automatic differentiation and differentiable programming.

Projects like:

DiffSharp (F#-based autodiff using category theory), JAX and Haskell's backprop, and

Conal Elliott's work on denotational AD using categories,

explore categorical formulations of AD as functorial transformations, leading to better compositionality and correctness.

2.6 Safe ML via Typed DSLs

Inspired by Haskell MLIR (Multi-Level IR) and TVM's Relay explore typed, functional-style IRs for safe optimization and compilation.

TensorFlow Eager and JAX both adopt a functional, pure style to enable more robust transformations (e.g., JIT, vectorization).

Haskell-based DSLs like Grenade, TorchHask, and DeepLearningHaskell offer ways to build ML pipelines with compile-time guarantees.

2.7 Practical ML Projects Using Haskell

Although niche, some serious Haskell ML libraries exist:

Hasktorch: Haskell bindings to LibTorch with typed tensors and differentiable programming.

Grenade: A fully Haskell-native DL framework with type-safe layer composition.

Edward Kmett's category-theoretic autodiff experiments.

These show how GADT-based types, dependent typing, and monads can be used to enforce shape, operation safety, and effect constraints in ML pipelines.

2.8 S

ummary: Why This is Gaining Traction The motivation is this:

Modern ML models are too brittle, opaque, and unrestricted. Haskell-like systems offer tools (types, purity, monads, category theory) to impose structure, improve interpretability, and enable safer code synthesis and reasoning.

This isn't just academic—industry is gradually adopting functional and type-theoretic ideas into production pipelines, even if not full Haskell.

Would you like a deeper dive into any of these (e.g., how monads model probabilistic computations, or how types guide neural program synthesis)?

References

[CTJ$^+$21] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

[LCC$^+$22] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, December 2022.

[LWG+22] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. In *NeurIPS*, 2022.

[WLG+23] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi D.Q. Bui, Junnan Li, and Steven C. H. Hoi. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint*, 2023.

[WWJH21] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *EMNLP*, 2021.