# REVISITING THE INTERPLAY BETWEEN MACHINE LEARNING AND PROGRAMMING LANGUAGES

**Edward Berman** [1 2]

**ABSTRACT**

In this paper, I revisit a 2018 article detailing the relationship between Machine Learning and Programming Languages, and respond directly to a blog published in opposition to it.

## 1 INTRODUCTION

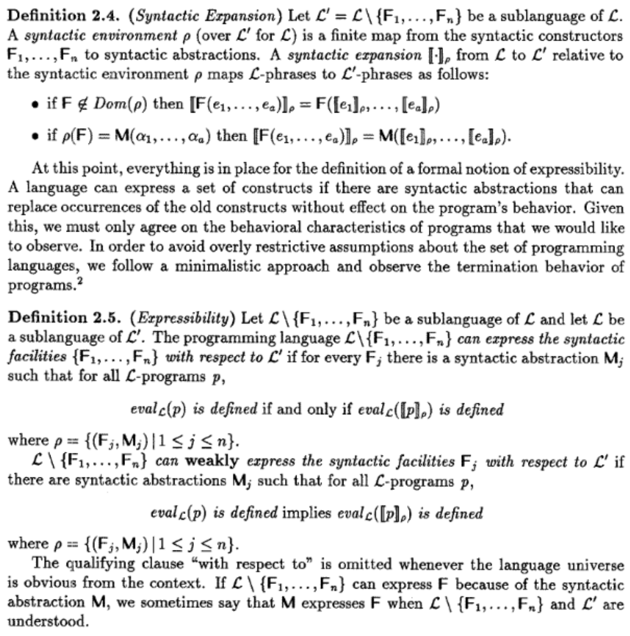
Our journey starts with (Innes et al., 2018), a paper aguing the merits of JULIA, FLUX.JL, and related software. Innes describes the necessity of a Machine Learning Language Paradigm and details how the Julia ecosystem is paving the way for this sort of machine learning specific language environment. The issue with these arguments, as described in (Ginesin, 2023), is that Innes talks about the advantages of different programming languages as containing fundamentally different *capabilities*, when its more appropriate to describe these advantages as *ergonomics*. If I wanted to, I could write the same programs in JULIA, RUST, LEAN, PYTHON, C, etc. The advantage of Julia or any idealized machine learning programming language isn't the features, its the helpful abstractions. In Julia's case, it's the ease of dealling with problems pertaining to numerical linear algebra/analysis and optimization.

In this paper, I attempt to pivot toward a more meaningful discussion of programming language paradigms, express-ability, and their relevance to machine learning.

## 2 ON EXPRESSABILITY

### 2.1 What is Expressability?

Matthias Felleisen gave us the following definitions for *expansion* and *expressability* (see Figure 1 and (Felleisen, 1991)):



**Definition 2.4.** (*Syntactic Expansion*) Let $\mathcal{L}' = \mathcal{L} \setminus \{\mathsf{F}_1, \dots, \mathsf{F}_n\}$ be a sublanguage of $\mathcal{L}$. A *syntactic environment* $\rho$ (over $\mathcal{L}'$ for $\mathcal{L}$) is a finite map from the syntactic constructors $\mathsf{F}_1, \dots, \mathsf{F}_n$ to syntactic abstractions. A *syntactic expansion* $[\![\cdot]\!]_\rho$ from $\mathcal{L}$ to $\mathcal{L}'$ relative to the syntactic environment $\rho$ maps $\mathcal{L}$-phrases to $\mathcal{L}'$-phrases as follows:

- if $\mathsf{F} \notin Dom(\rho)$ then $[\![\mathsf{F}(e_1, \dots, e_a)]\!]_\rho = \mathsf{F}([\![e_1]\!]_\rho, \dots, [\![e_a]\!]_\rho)$
- if $\rho(\mathsf{F}) = \mathsf{M}(\alpha_1, \dots, \alpha_a)$ then $[\![\mathsf{F}(e_1, \dots, e_a)]\!]_\rho = \mathsf{M}([\![e_1]\!]_\rho, \dots, [\![e_a]\!]_\rho)$.

At this point, everything is in place for the definition of a formal notion of expressibility. A language can express a set of constructs if there are syntactic abstractions that can replace occurrences of the old constructs without effect on the program's behavior. Given this, we must only agree on the behavioral characteristics of programs that we would like to observe. In order to avoid overly restrictive assumptions about the set of programming languages, we follow a minimalistic approach and observe the termination behavior of programs.[2]

**Definition 2.5.** (*Expressibility*) Let $\mathcal{L} \setminus \{\mathsf{F}_1, \dots, \mathsf{F}_n\}$ be a sublanguage of $\mathcal{L}$ and let $\mathcal{L}$ be a sublanguage of $\mathcal{L}'$. The programming language $\mathcal{L} \setminus \{\mathsf{F}_1, \dots, \mathsf{F}_n\}$ *can express the syntactic facilities* $\{\mathsf{F}_1, \dots, \mathsf{F}_n\}$ *with respect to* $\mathcal{L}'$ if for every $\mathsf{F}_j$ there is a syntactic abstraction $\mathsf{M}_j$ such that for all $\mathcal{L}$-programs $p$,

$$eval_{\mathcal{L}}(p) \textit{ is defined} \text{ if and only if } eval_{\mathcal{L}}([\![p]\!]_\rho) \textit{ is defined}$$

where $\rho = \{(\mathsf{F}_j, \mathsf{M}_j) \mid 1 \le j \le n\}$.

$\mathcal{L} \setminus \{\mathsf{F}_1, \dots, \mathsf{F}_n\}$ *can* **weakly** *express the syntactic facilities* $\mathsf{F}_j$ *with respect to* $\mathcal{L}'$ if there are syntactic abstractions $\mathsf{M}_j$ such that for all $\mathcal{L}$-programs $p$,

$$eval_{\mathcal{L}}(p) \textit{ is defined} \text{ implies } eval_{\mathcal{L}}([\![p]\!]_\rho) \textit{ is defined}$$

where $\rho = \{(\mathsf{F}_j, \mathsf{M}_j) \mid 1 \le j \le n\}$.

The qualifying clause "with respect to" is omitted whenever the language universe is obvious from the context. If $\mathcal{L} \setminus \{\mathsf{F}_1, \dots, \mathsf{F}_n\}$ can express $\mathsf{F}$ because of the syntactic abstraction $\mathsf{M}$, we sometimes say that $\mathsf{M}$ expresses $\mathsf{F}$ when $\mathcal{L} \setminus \{\mathsf{F}_1, \dots, \mathsf{F}_n\}$ and $\mathcal{L}'$ are understood.

*Figure 1.* Definitions of Expansion and Expressability

Sure, if you would like to be really pedantic, FLUX.JL with JULIA can be thought of as its own language with JULIA as a sublanguage. I can expand the functions of FLUX.JL with JULIA into primitives, and then JULIA itself into primitives, and JULIA will be a sublanguage. But what does saying this really accomplish. This is one of these **definition of the law** but not **spirit of the law** type things. That's not to say this isn't a useful definition, but using it to claim that your machine learning is equivalent to making a new language and should be treated as such is a massive stretch. FLUX.JL is not meaningful because it's a new language, it's meaningful because it makes useful **abstractions**.

### 2.2 What Abstractions we would like to Express?

As (Ginesin, 2023) asked, what abstractions are meaningful. FLUX.JL with JULIA answers this question very well. While FLUX.JL is not unique in it's ability to let you play with different descent methods, it's by far the most robust and

[1]Department of Mathematics, Northeastern University [2]Department of Physics, Northeastern University. Correspondence to: Edward Berman <berman.ed@northeastern.edu>.

*Proceedings of the 5th MLSys Conference*, Santa Clara, CA, USA, 2024. Copyright 2024 by the author(s).

can handle things that other libraries can't. For example, recurssions. Beyond just the abstractions, everything is functional, statically typed, and is written on top of a dedicated numerical analysis tool with Zygote (Van Merriënboer et al., 2018; Innes, 2019). JULIA and FLUX.JL were *designed* to handle hard numerical problems in machine learning and beyond.

## 3 ON PARADIGMS

I also wanted to quickly mention that the original paper also missuses the term probabilistic paradigm. I would saying TURING.JL is a probabilistic paradigm, but not JULIA or FLUX.JL in it of themselves. TURING.JL is an actual language superset of JULIA, designed for the ease of writing and solving bayesian estimation problems. Another great example of a probalistic programming is Steven Holtzen's DICE (Holtzen et al., 2020).

You can have great choices and conventions while not making a whole new paradigm. Such is the case with FLUX.JL. It's not emblamatic of a new paradigm, it just executes a (functional) one very very well.

## 4 CONCLUSION

JULIA and FLUX.JL are great. Not because they are new languages or paradigms, because they make meaningful abstractions that other libraries and languages do not afford.

## REFERENCES

Felleisen, M. On the expressive power of programming languages. *Science of Computer Programming*, 17(1):35–75, 1991. ISSN 0167-6423. doi: https://doi.org/10.1016/0167-6423(91)90036-W. URL https://www.sciencedirect.com/science/article/pii/016764239190036W.

Ginesin, J. abstraction, November 2023. URL https://jakegines.in/blog/2023/abstraction. Accessed: 2023-11-28.

Holtzen, S., Van den Broeck, G., and Millstein, T. Scaling exact inference for discrete probabilistic programs. *Proc. ACM Program. Lang.*, 4(OOPSLA), nov 2020. doi: 10.1145/3428208. URL https://doi.org/10.1145/3428208.

Innes, M. Don't unroll adjoint: Differentiating ssa-form programs, 2019.

Innes, M., Karpinski, S., Shah, V., Barber, D., Saito Stenetorp, P., Besard, T., Bradbury, J., Churavy, V., Danisch, S., Edelman, A., et al. On machine learning and programming languages. Association for Computing Machinery (ACM), 2018.

Van Merriënboer, B., Breuleux, O., Bergeron, A., and Lamblin, P. Automatic differentiation in ml: Where we are and where we should be going. *Advances in neural information processing systems*, 31, 2018.