

Complexity and Theoretical Foundations of Learning and Program Synthesis.

Gabriel Bathie, Nathanaël Fijalkow

gabriel.bathie@gmail.com, nathanael.fijalkow@gmail.com

LaBRI, Université de Bordeaux,

Programming by Example and Program synthesis

It is often much easier to describe the behavior of a function using examples of input-output pairs than to spell out its exact behavior. “Programming by Example”, or Program Synthesis, is a research field that aims to create systems that *derive* the code of a function or *learn* the logical formulation of a specification from a small set of input-output examples.

These systems have many useful applications. For example, by learning a logical formula that separates normal system logs from anomalous ones, we obtain an interpretable *logical explanation* of the anomaly.

Another example is *code deobfuscation*. Obfuscation is the process of hiding the behavior of a program by making it hard to read, for example by writing $z = (x \wedge y) + ((x \& y) << 1)$; instead of $z = x + y$; (the two are equivalent) in the final assembly. Obfuscated code can still be executed, which allows us to generate input-output pairs. Using program synthesis, we can recover the original readable code from these examples. Code deobfuscation is for example helpful in malware reverse engineering.

Theoretical foundations and computational complexity

The computational complexity of program synthesis depends on many factors, such as the structure of the target programming language (in a wide sense, it could be a formal logic; in general we call these Domain Specific Languages (DSLs)). For example, we know that for the Linear Temporal Logic (LTL) DSL, the problem is NP-hard even for restricted sub-DSLs [1], and these results were recently extended to Signal Temporal Logic [2].

This internship aims to further study the computational complexity of Program Synthesis, both from the point of view of hardness and of algorithms: NP-hardness, W[1]-hardness, inapproximability, approximation and FPT algorithms. Good starting points include the LTL [3] and the Mixed Boolean Arithmetic [4] DSLs. If we obtain interesting algorithms and if the intern is interested in writing code, we propose to write a proof-of-concept implementation of these algorithms in our existing tools such as Bolt [3] or SiMBA.

Contact:

- Gabriel Bathie: gabriel.bathie@gmail.com
- Nathanaël Fijalkow : nathanael.fijalkow@gmail.com

The lab can cover your travel costs to and from Bordeaux for the internship, and help you find an accommodation. If you do not have an ENS salary, we have funding available to pay you for the duration of the internship.

References:

- [1] N. Fijalkow and G. Lagarde, “The complexity of learning linear temporal formulas from examples,” in *International conference on grammatical inference*, PMLR, 2021, pp. 237–250.
- [2] B. Bordais, D. Neider, and R. Roy, “The complexity of learning temporal properties,” *arXiv preprint arXiv:2408.04486*, 2024.
- [3] G. Bathie, N. Fijalkow, T. Matricon, B. Mouillon, and P. Vandenhover, “ LTL_f Learning Meets Boolean Set Cover,” in *TACAS*, Springer, 2026, pp. 480–501.
- [4] G. Bathie, B. Mouillon, and N. Fijalkow, “GPU-accelerated synthesis of mixed-boolean arithmetic: Beyond caching.” 2026. Available: <https://arxiv.org/abs/2605.08243>