BULL: A Library for Learning Algorithms of Boolean Functions[⋆]

Yu-Fang Chen and Bow-Yaw Wang

Academia Sinica, Taiwan
`http://bull.iis.sinica.edu.tw/`

Abstract. We present the tool BULL (Boolean fUnction Learning Library), the first publicly available implementation of learning algorithms for Boolean functions. The tool is implemented in C with interfaces to C++, JAVA and OCAML. Experimental results show significant advantages of Boolean function learning algorithms over all variants of the L^* learning algorithm for regular languages.

1 Introduction

BULL is the first publicly available implementation of learning algorithms for Boolean functions. Three algorithms are implemented in the library. The classical CDNF algorithm infers Boolean functions over a fixed number of variables. The incremental CDNF+ and CDNF++ algorithms infers Boolean functions over an indefinitely number of variables. The library is implemented in C with C++, JAVA, and OCAML interfaces. Sample codes of C, C++, JAVA, and OCAML are distributed with the library. Users can adopt BULL by modifying them.

What Is It?. Learning algorithms for Boolean functions can be viewed as an efficient procedure to generate a *target* Boolean function only known to a teacher. This type of learning algorithms assume a teacher who answers queries about the target Boolean function. The learning algorithms acquire information from the answers to queries and organize them in a systematic way. In the worst case, learning algorithms will infer a target Boolean function within a polynomial number of queries in the CNF and DNF formula sizes of the target function.

Learning in Formal Verification. Since the work in [8], algorithmic learning has been applied to formal verification techniques such as specification synthesis [8], automated compositional verification [5], and regular model checking [6]. Most applications are based on the L^* automata learning algorithm for regular languages. The learning algorithm enumerates states explicitly. Its applications are hence inherently explicit [5], or use explicit automata as implicit representations of state spaces [6].

⋆ The project was supported in part by the National Science Council of Taiwan project no. NSC-101-2221-E-001-006 and no. NSC-101-2221-E-001-007.

Why Use Boolean Learning. Implicit algorithms (e.g., SAT-based model checking) can greatly improve the capacity of various verification techniques. Similar improvements have also been reported in applications of the CDNF learning algorithm for Boolean functions. In [3], the learning algorithm is adopted to infer implicit contextual assumptions in automated compositional reasoning. It is shown that learning implicitly can tackle certain hard problems unattainable by traditional explicit algorithms. The CDNF algorithm is also applied to loop invariant generation. The learning-based framework can be much more efficient than conventional static analysis algorithms [7].

For regular languages, the learning algorithms are available in veteran tools (such as libalf [1] and learnlib [11,10,9]). Implementations of learning algorithms for Boolean functions however are still missing. Since it would take a considerable amount of time to understand and implement learning algorithms for Boolean functions, lack of publicly available tools could be an obstacle to develop related techniques in the research community. In order to lower the barrier to entry, we decide to develop the BULL library.

The Position of the Paper. The Boolean learning project starts in 2009 and since then we tested different variants of the algorithms and data structures. Several of them indeed dramatically improved the performance, e.g., non-membership queries are introduced partly for performance reasons. However, since boolean learning is a new technique to most people in the community. We decided to spend the pages for a general introduction instead of technical details.

2 The BULL Library

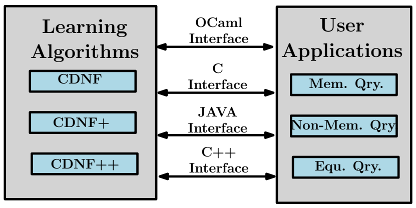

Fig. 1. System Architecture

Figure 1 shows the architecture of the BULL library. The core library contains three learning algorithms implemented in C. They are the CDNF [2], the CDNF+ [4], and the CDNF++ [4] algorithms. The CDNF algorithm assumes that the number of variables in the target Boolean function is known. The CDNF+ and the CDNF++ algorithms do not have this assumption. In addition to the learning algorithms, we also provide C++, JAVA (via JNI), and OCaml interfaces.

2.1 How to Use the Package

In order to adopt the learning algorithms in BULL, users have to play the teacher and answer queries posed by the algorithms. For the sake of presentation, let us assume that $f(x, y, z) = (x \wedge \neg y) \vee (x \wedge z)$ is the target Boolean function over variables x, y, and z. Consider the following sample queries from the learning algorithms:

1. A *membership* query on a partial assignment $\{(x, \mathit{false})\}$. On a membership query, the teacher checks if the target is satisfiable under the given assignment. Here the teacher answers *no* since $f(\mathit{false}, y, z)$ is not satisfiable.
2. A *non-membership* query on a parital assignment $\{(y, \mathit{true})\}$. On a non-membership query, the teacher checks if the negation of the target is satisfiable under the assignment. For this example, the teacher answers *yes*.
3. An *equivalence* query on a conjecture $f'(x, y, z) = x \wedge y$. On an equivalence query, the teacher answer *yes* if the given formula is equivalent to the target. Otherwise, she returns an assignment as a counterexample. For this example, the teacher may return the assignment $\{(x, \mathit{true}), (y, \mathit{true}), (z, \mathit{false})\}$ since $f'(\mathit{true}, \mathit{true}, \mathit{false}) \neq f(\mathit{true}, \mathit{true}, \mathit{false})$.

Table 1. Features of Algorithms

	Num. of Vars.	Mem. Qry,	Non-Mem. Qry.	Equ. Qry.
CDNF	known	√		√
CDNF+	unknown	√		√
CDNF++	unknown	√	√	√

Different learning algorithms pose different types of queries. Table 1 shows the differences among the three learning algorithms in BULL. The CDNF algorithm assumes the number of variables in the target Boolean function is known. The CDNF+ algorithm does not know the number of variables. Both algorithms only pose membership and equivalence queries. The CDNF++ algorithm does not presume the number of variables is known. It however poses membership, non-memberhip, and equivalence queries.

BULL defines the interfaces to the three types of queries. If all queries can be answered automatically, users can implement a mechanical teacher to answer queries through the interface. Learning algorithms in BULL will invoke the mechanical teacher and infer unknown target functions automatically. We refer interested users to our full version (`http://bull.iis.sinica.edu.tw/`) which contains a detailed demonstration of how to implement the above query functions and connect them to BULL.

2.2 Users of BULL

The BULL library targets the formal verification research community. As far as we know, several people in the field are interested in the applications of learning algorithms for Boolean functions. The library has already been used by the verification group in Oxford University (Learning-based Compositional Probabilistic Model Checking), the software trustability and verification group in Tsinghua University (Learning-Based Compositional Verification), and the static analysis group in Seoul National University (Loop Invariant Inference). Several other groups have shown their interests and asked for the source code.

2.3 Potential Applications

The CDNF algorithm has been applied to synthesize contextual assumptions in assume-guarantee reasoning. It has also been used to infer a loop invariant in program verification. These applications share common characteristics. First, computing contextual assumptions or loop invariants without learning is possible but expensive. It is however easy to verify if purported contextual assumptions or loop invariants work. Moreover, contextual assumptions or loop invariants are by no mean unique. It suffices to compute but one contextual assumption or loop invariant in these applications. From our experience, we believe that learning is most suitable for problems with the aforementioned characteristics.

For interested reader, a step-by-step tutorial of how to use the BULL library to find loop invariants is provided in our full version (`http://bull.iis.sinica.edu.tw/`). We hope it may give some insights to more applications of the library.

3 Experimental Results

Since the target application of BULL is verification, in the first experiment, we decide to pick a classical example, n-bit counter, as the target for learning (Table 2). In Table 3, we show a different version where the n-bit counter model can be non-deterministically reset to 0 from any state. In the second experiment, we compare the performance of the Boolean learning algorithms using random 3SAT formulae of n variables. In those formulae, the ratio of the number of variables to the number of clauses is 1/4.[1] We use a timeout period of 10 minutes. In Figure 2, we show the average execution time of the first 50 non-trivial instances (satisfiable and all algorithms finished within the timeout period). In Table 4, we show the number of timeout cases out of 180 instances.

Table 2. Comparison of Boolean function learning algorithms: using n-bit counter as the example

	2	3	4	5	6	7	8	9	10	11	12
CDNF	0.02	0.02	0.05	0.11	0.35	1.03	2.29	4.3	9.8	23.6	66.2
CDNF+	0.01	0.02	0.04	0.09	0.27	0.77	1.5	2.4	5.7	14.1	40.3
CDNF++	0.01	0.02	0.04	0.09	0.25	0.77	1.5	2.4	5.6	13.8	39.8

At the first glance, CDNF learning algorithm has the best performance among the three. However, it is not a fair interpretation for two reasons. First, CDNF makes use of some information (number of variables in the target function) that is not known by the other two algorithms. More importantly, in particular for

[1] This ratio is very close to satisfiability threshold of 3SAT formulae. Hence the chance of getting a satisfiable formula is 50%.

Table 3. Comparison of Boolean function learning algorithms: using n-bit counter with non-deterministic reset as the example

	2	3	4	5	6	7	8	9	10	11	12
CDNF	0.00	0.02	0.07	0.24	0.75	2.83	12.13	32.01	112	451	1374
CDNF+	0.01	0.02	0.06	0.21	0.67	2.63	12.1	36.8	144	637	1671
CDNF++	0.01	0.02	0.06	0.21	0.62	2.63	12.08	36.88	145	582	1632

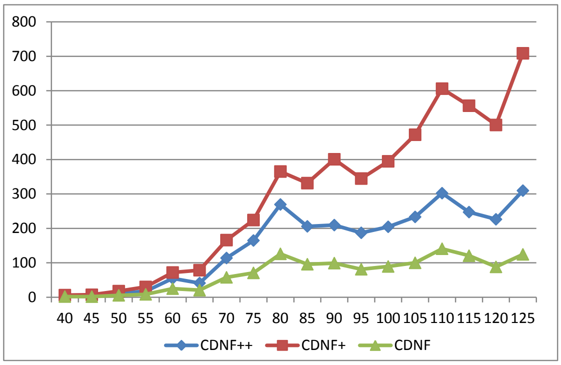

Fig. 2. Comparison of Boolean learning algorithms, using random 3SAT formulae as the benchmark. The vertical axis is the average execution time in seconds and the horizontal axis is the number of variables in the formula. Each point is the average results of 50 instances.

the case of randomly generated formulae, almost all the variables will be added to the final result. Hence the benefit obtained from incremental learning is not significant in such type of examples. In fact, the CDNF+ and CDNF++ algorithms are particularly useful in formal verification applications [4] such as those based on predicate abstraction and interpolation-based refinement. Typically in these applications, a boolean variable is used to indicate the truth of a predicate in certain points of program executions. Since the number of predicates in use would increase in each refinement step, there is no *a prior* known upper bound of needed variables.

Table 4. The number of timeout cases out of 180 instances

Num. of Var.	40	45	50	55	60	65	70	75	80	85	90	95	100	105	110	115	120	125	130	135
CDNF	0	0	0	0	0	0	1	2	0	14	7	24	28	31	48	51	70	76	90	93
CDNF+	0	0	0	0	0	0	1	6	5	21	19	42	48	51	83	80	88	99	118	122
CDNF++	0	0	0	0	0	0	2	4	6	19	16	32	40	45	69	69	82	90	106	109

References

1. Bollig, B., Katoen, J.-P., Kern, C., Leucker, M., Neider, D., Piegdon, D.R.: `libalf`: The Automata Learning Framework. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 360–364. Springer, Heidelberg (2010)
2. Bshouty, N.H.: Exact learning Boolean function via the monotone theory. Information and Computation 123(1), 146–153 (1995)
3. Chen, Y.-F., Clarke, E.M., Farzan, A., Tsai, M.-H., Tsay, Y.-K., Wang, B.-Y.: Automated Assume-Guarantee Reasoning through Implicit Learning. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 511–526. Springer, Heidelberg (2010)
4. Chen, Y.-F., Wang, B.-Y.: Learning Boolean Functions Incrementally. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 55–70. Springer, Heidelberg (2012)
5. Cobleigh, J.M., Giannakopoulou, D., Păsăreanu, C.S.: Learning Assumptions for Compositional Verification. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 331–346. Springer, Heidelberg (2003)
6. Habermehl, P., Vojnar, T.: Regular model checking using inference of regular languages. In: ENTCS, pp. 21–36 (2005)
7. Jung, Y., Kong, S., Wang, B.-Y., Yi, K.: Deriving Invariants by Algorithmic Learning, Decision Procedures, and Predicate Abstraction. In: Barthe, G., Hermenegildo, M. (eds.) VMCAI 2010. LNCS, vol. 5944, pp. 180–196. Springer, Heidelberg (2010)
8. Maler, O., Pnueli, A.: On the learnability of infinitary regular sets. Information and Computation 118(2), 316–326 (1995)
9. Merten, M., Howar, F., Steffen, B., Cassel, S., Jonsson, B.: Demonstrating Learning of Register Automata. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 466–471. Springer, Heidelberg (2012)
10. Merten, M., Steffen, B., Howar, F., Margaria, T.: Next Generation LearnLib. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 220–223. Springer, Heidelberg (2011)
11. Raffelt, H., Steffen, B., Berg, T., Margaria, T.: Learnlib: a framework for extrapolating behavioral models. STTT 11(5), 393–407 (2009)