Evolutionary construction of derivations in classical propositional logic using a symbolic-connectionist representation

Maciej Komosinski Adam Kups

Technical Report RA–3/17

Institute of Computing Science
Poznan University of Technology
May 2017

Abstract

This report introduces a way derivations in classical propositional logic can be constructed using evolutionary algorithms. The derivations are represented by connectionist systems. There are three kinds of nodes constituting these systems: formula nodes that generate signal in the form of strings of symbols, *modus ponens* nodes that transform incoming signal according to the *modus ponens* rule, and substitution nodes that transform incoming signal by applying the substitution rule. This work presents initial research on an approach that is a part of our quest for efficient construction of derivations using various logics and constrained in various ways. The final part of this report outlines limitations encountered in our initial experiments and the ways the proposed approach can be improved.

1 Introduction

This report describes a technique that may be used to automatically obtain derivations expressed in the language of classical propositional logic. The ultimate goal of this research is to apply this technique to generate a special kind of derivations which are axiomatic proofs. The difference between proofs and derivations is such that proofs are based on a specific kind of formulas called axioms, which allow to derive the entire classic propositional logic (CPL); see [11] for examples of axiomatic proofs. The approach explicated in this report covers:

1. the representation of the set of source formulas as a group of input nodes that output a signal in the symbolic form,
2. the representation of rules transforming formulas into other formulas as nodes transforming input symbols into output symbols,
3. the representation of derived formulas as output nodes,
4. the application of evolutionary optimisation to build derivations of selected formulas from sets of formulas.

2 Derivations in CPL

A derivation $d(A, X)$ of a formula A from the set X is a sequence s of formulas such that each formula of the sequence either belongs to X, or is obtained from previous formulas of the sequence by applying the *modus ponens* (MP) rule or the *substitution* (SUB). The last element of s is A.

More formally,

$$s =< X_1, \ldots, X_n, R_1, \ldots, R_m, A > \tag{1}$$

where $X_1, \ldots, X_n$ are formulas from X and $R_1, \ldots, R_m$ are formulas obtained by the application of the rules to the formulas occurring earlier in the sequence.

The MP is a rule that allows to derive the the formula B from the implication $A \rightarrow B$ and the formula A.

More formally,

$$A \rightarrow B, A \vdash B \tag{2}$$

The operation of substitution can be expressed as a function $SUB(A, x, B)$ that takes three arguments: the formula A which is the subject of substitution, the variable x, and the formula B. The substitution operation returns a formula C which differs from A only in that in each place where the variable x occurs in A, the formula B is put instead. When A does not contain x, A and C are identical.

SUB is a rule that allows to derive the formula $SUB(A, x, B)$ from the formula A, and arbitrary x and B.

Formally,

$$A \vdash SUB(A, x, B) \tag{3}$$

It is easy to prove that both MP and SUB preserve a property of being valid in classical propositional logic. This means that if the premises of those rules are tautologies, then the conclusions are also tautologies.

3 Derivations in a connectionist environment

In order to express a logic as a connectionist system, various approaches are possible [5, 1]. In this report we propose a hybrid connectionist-symbolic approach where there are two kinds of nodes: the *formula* nodes and the *rule* nodes, and the signal flowing between nodes consists of strings of symbols that represent formulas.

Formula nodes (FNs) are initial nodes in the sense that no signal flows into them – they only output strings of symbols. Formula nodes are characterized by one parameter, f, which is a string of symbols that represents a formula from the set X.

Rule nodes are of two types: substitution nodes (SUBNs) and modus ponens nodes (MPNs). SUBNs have one input and they output a string of symbols that represents formulas obtained by the application of the substitution rule to the formula represented by the string of symbols input to the node. SUBs have two parameters: x that represents a propositional variable which is to be replaced during substitution, and s that represents a formula that replaces the variable represented by x. MPNs are nodes with two inputs. The implementation of an MPN is the most problematic out of all of the node types because it requires the decision of what kind of signal should be flowing out of these nodes when the input signals to those nodes do not represent the implication $A \rightarrow B$ and the antecedent A. Should the output signal represent the implication, the antecedent, or should it be some kind of a “null” signal, or should some other “default” formula be sent out of such a node? When networks of derivations are constructed manually, the selected approach depends on the conveniences/inconveniences of the human designer. However, when the automated, optimization-oriented constructive approach is employed, various choices regarding the MPNs output may result in vastly different outcomes and different quality of obtained derivations. Research comparing such outcomes is one of our future tasks. In the current implementation, the “improperly wired” MPNs output a signal that represents an empty string of symbols.

Each node can have a branching output, i.e., the output can be connected to many other nodes at the same time, with identical signal flowing through all these connections. This facilitates parallel processing and indicates that constructed networks are in fact families of (possibly multi-) connected derivations. Note, however, that cycles in such networks should not exist.

4 Building of derivations as an optimization task

For computational experiments, Framsticks software is used [9, 8]. In the preliminary version of automated building of the derivation network, the evolutionary algorithm is employed [2]. Only mutation operator is in use, and it can randomly change several attributes of derivation networks:

- it can add and delete new nodes,
- it can change parameters of nodes,
- it can add and remove connections between nodes.

If needed, selected nodes, parameters or connections can be marked as immutable to preserve the desired characteristics of networks across subsequent generations of derivations networks.

While in this work we report results of a directed evolutionary algorithm, where the goal is the maximization of objective, single-criterion fitness, another interesting approach would be to randomly generate derivation networks (e.g. by random modifications of some initial network) and then analyze the behavior of the set of these randomly modified networks. Such an analysis could provide useful data concerning etymology and patterns in the set of derivations, and to further obtain knowledge about how to construct efficient fitness functions [12] and about heuristics needed to arrive at particular formulas during derivation.

The optimization is performed according to a standard procedure in evolutionary algorithms, in particular:

1. it starts with one individual in the gene pool – a genotype that represents a collection of FNs,
2. in each cycle, one selected genotype is mutated and translated to a phenotype that is simulated for several simulation steps,
3. after this, the value of the fitness function for this phenotype is calculated,
4. if gene pool capacity is exceeded, selected genotypes are removed.

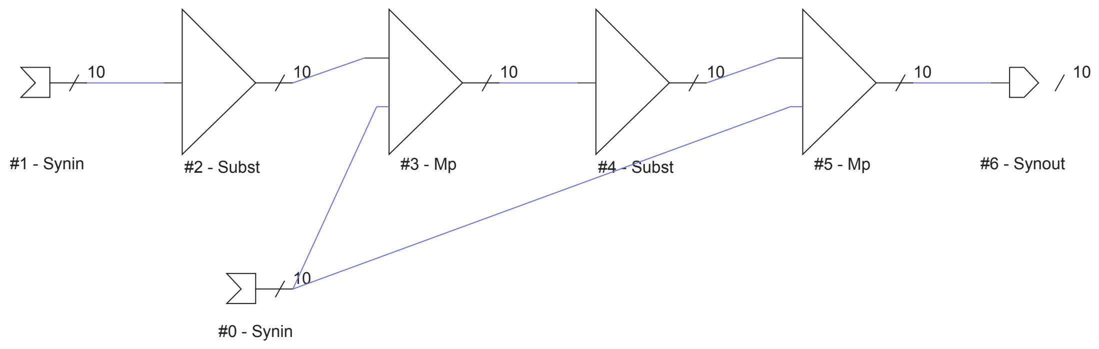

Figure 1: A sample derivation generated by the evolutionary algorithm described in Sect. 4.

In the most naive approach, the fitness function can be defined as the similarity between the specified formula and the output of the nodes in the tested phenotypes. Following this approach, each node of the network was examined and the one with the most similar formula determined the fitness value of the entire derivation. More precisely, the similarity was expressed as the number of identical symbols at identical positions in pairs of formulas divided by the number of symbols in the specified formula. For example, let us assume that we try to construct a derivation of the formula $p \to p$ and there is some node that outputs the formula $p \to q$. The value of this node would be $v = \frac{2}{3} \approx 0.67$, because two symbols in the formula produced by the node match their respective position in the formula for which we try to build a derivation. Were this value the highest of all nodes in the network, then this value would be the fitness value of the network. In other words, the fitness value is the maximum of the function which maps the nodes into values representing similarity between the target formula and the formula produced by the respective nodes. The motivation for such a fitness function was to obtain a derivation of a particular formula from the specified set of formulas represented by immutable FN nodes.

For this naively defined fitness function, various evolutionary selection strategies were tested (such as "only the best/worst", roulette, tournament, etc.) for both positive (mutation) and negative (removal) selection stages. This kind of optimization-based derivation building, although sometimes successful, was not particularly efficient. One reason for that is that the fitness function did not yield a smooth landscape, and additionally it was highly probable that a trend of the increasing fitness value lead to a local plateau. Another reason for difficulties in optimization is that changing parameter values of SUBNs was unrestricted (i.e., any randomly generated formula could have been put in the premise-formula) which resulted in many low-quality networks. Several improvement strategies that may help overcome these difficulties are discussed in Sect. 5.

A sample generated derivation of formula $p \to p$ from the set $\{p \to (q \to p), (p \to (q \to r)) \to ((p \to q) \to (p \to r))\}$ is presented in Fig. 1. The translation of this simple network to a form of a derivation is provided below. While, technically speaking, a derivation is a sequence of formulas, here we present the list of formulas and operations performed on them accompanied by the names of the nodes (in square brackets) to clearly demonstrate how the generated network works. The names of the nodes correspond to the # numbers in Fig. 1, and the types of nodes provided in square brackets correspond to the previously introduced rule descriptions.

1. $(p \to (q \to r)) \to ((p \to q) \to (p \to r))$ [FN #0],
2. $p \to (q \to p)$ [FN #1],
3. $(p \to (q \to p)) \to ((p \to q) \to (p \to p))$ [SUBN $x : r$, $s : p$, #2],
4. $(p \to q) \to (p \to p)$ [MPN, #3, #2],
5. $(p \to (q \to p)) \to (p \to p)$ [SUBN $x : q$, $s : q \to p$, #4],
6. $p \to p$ [MPN, #5, #2],
7. $p \to p$ [output].

5 Conclusions and future work

This report described one approach to the representation and the automated generation of classical propositional logic derivations as hybrid symbolic-connectionist systems. Automated generation is performed by means of evolutionary optimisation. Preliminary results are mildly satisfying, but more importantly they provide information on what should be done to improve efficiency of the approach introduced in this work. First of all, the fitness function should be adjusted to yield a more smooth fitness landscape. This can be done by taking into consideration more pieces of information than just the precise similarity between the expected derived formula and the formulas that are currently derived by the connectionist system. Hints on what are these additional pieces of information can be obtained by performing a meta research, comparing different optimization landscapes that correspond to different fitness functions in order to establish what kind of characteristics of the fitness function is most informative with respect to the global optimum. Secondly, the mutation operator should be improved to avoid producing an abundance of neutral or bad solutions. This is especially the case for the mutation of substitutions nodes – this kind of mutation should be more sophisticated. Additional information on how this kind of mutation should be enriched can be established during a meta research similar to the one proposed for adjusting the fitness function.

Apart from the modification of the procedure, future tasks also include the extension of the research scope. This includes research concerning the generation of axiomatic proofs for CPL. The next step would be to extend the logic used in the research from the CPL (which serves as a testbed) to more interesting epistemic propositional logic [3, 13]. Yet another interesting analysis may concern the genesis of the considered derivations systems during the evolutionary optimization, and comparing such attempts to obtain proper derivations with the behavior of a logician when performing the same task. This kind of analysis could provide interesting knowledge useful both for the experts (logicians) and for the optimization process. Finally, future plans also include modifying the procedure to solve abduction tasks. The abduction-oriented approach [4, 10] can be understood as a reinterpretation of the process of finding a derivation – where the central question is "which formulas were added to construct a derivation", and not "is it possible to (efficiently) construct a derivation". As abduction in this approach is the process of finding formulas, called abducibles, that fill the deductive gap between some set of formulas and another formula (called the abductive goal) [7, 6], the procedure would require careful analyses of the signal (that represents formulas) in the generated derivation to establish which formulas serve as abducibles. A more sophisticated and dynamic approach could operate not on formulas, but on derivation rules that are needed to derive a formula. This kind of research would concentrate on the assessment of the efficiency of such rules in solving an abductive problem, which would however exceed the classical understanding of abduction.

References

[1] Artur S Avila Garcez and Gerson Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence*, 11(1):59–77, 1999.

[2] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[3] Jaakko Hintikka. Individuals, possible worlds, and epistemic logic. *Nous*, pages 33–62, 1967.

[4] Jaakko Hintikka. What is abduction? The fundamental problem of contemporary epistemology. In *Inquiry as Inquiry: A Logic of Scientific Discovery*, pages 91–113. Springer, 1999.

[5] Pascal Hitzler, Steffen Hölldobler, and Anthony Karel Seda. Logic programs and connectionist networks. *Journal of Applied Logic*, 2(3):245–272, 2004.

[6] Maciej Komosinski, Adam Kups, Dorota Leszczyńska-Jasion, and Mariusz Urbański. Identifying efficient abductive hypotheses using multi-criteria dominance relation. *ACM Transactions on Computational Logic*, 15(4):28:1–28:20, 2014. URL: `http://doi.acm.org/10.1145/2629669`, `doi:10.1145/2629669`.

[7] Maciej Komosinski, Adam Kups, and Mariusz Urbanski. Multi-criteria evaluation of abductive hypotheses: towards efficient optimization in proof theory. In *Proceedings of the 18th International Conference on Soft Computing*, pages 320–325, Brno, Czech Republic, 2012.

[8] Maciej Komosinski and Szymon Ulatowski. Framsticks: Creating and understanding complexity of life. In Maciej Komosinski and Andrew Adamatzky, editors, *Artificial Life Models in Software*, chapter 5, pages 107–148. Springer, London, 2nd edition edition, 2009.

[9] Maciej Komosinski and Szymon Ulatowski. Framsticks web site, 2017. URL: `http://www.framsticks.com`.

[10] Marta Cialdea Mayer and Fiora Pirri. Propositional abduction in modal logic. *Logic Journal of IGPL*, 3(6):907–919, 1995.

[11] Carew A. Meredith, A. N. Prior, et al. Notes on the axiomatics of the propositional calculus. *Notre Dame Journal of Formal Logic*, 4(3):171–187, 1963.

[12] Franz Rothlauf. 3.3 Locality. In *Representations for genetic and evolutionary algorithms*, pages 73–95. Springer-Verlag, 2006.

[13] Hans Van Ditmarsch, Wiebe van Der Hoek, and Barteld Kooi. *Dynamic epistemic logic*, volume 337. Springer Science & Business Media, 2007.