Performance Upgrade of Sequence Detector Evolution Using Grammatical Evolution and Lexicase Parent Selection Method

Bilal Majeed[1(✉)], Samuel Carvalho[2], Douglas Mota Dias[1,3], Ayman Youssef[4], Aidan Murphy[5], and Conor Ryan[1]

[1] University of Limerick, Limerick, Ireland
bilal.majeed@ul.ie
[2] Technological University of the Shannon: Midlands Midwest, Limerick, Ireland
[3] Rio de Janeiro State University, Rio de Janeiro, Brazil
[4] Electronics Research Institute, Cairo, Egypt
[5] Trinity College Dublin, Dublin, Ireland

Abstract. Quickly designing correct and efficient digital circuits is a crucial need for the electronics industry. Several Electronic Design Automation tools are used for this task. Still, they often lack the diversity of designs that search-based techniques can offer, such as our system producing three different designs for a 5-bit '11011' Sequence Detector. Sequence Detectors are some of the most crucial digital sequential circuits evolved in this work using Grammatical Evolution, a Machine Learning technique based on Evolutionary Computation. Compared to the literature, a reasonably small training data set is used to generate diverse solutions/circuits. A comparison is delivered of the results of the evolved circuits using two different parent selection techniques, tournament selection and lexicase selection. It is shown that the evolved circuits using a small training data set have shown a hundred percent test accuracy on a vast amount of test data sets, and the performance of lexicase selection is much better than tournament selection while evolving these circuits.

Keywords: Electronic Design Automation · Evolvable Hardware · Grammatical Evolution · Sequence Detector · Lexicase Selection

1 Introduction

Designing digital circuits is a time-consuming and complex task for designers, which involves passing crucial tests to the system to validate the circuit. It can also involve rigorous verification in the environment in which the circuit will perform. Today, Hardware Description Languages (HDLs) are used throughout the industry to aid in designing circuits [3]. Electronic Design Automation (EDA) tools are used by designers to efficiently use their time and resources to design

P. Collet et al. (Eds.): CCE 2023, LNCS 13927, pp. 90–103, 2023.
https://doi.org/10.1007/978-3-031-44355-8_7

digital circuits correctly [5]. Some industrial tools use machine learning, such as [24], while some others are based on synthetic intelligence, such as [7] and [4]. Despite having such tools available, digital circuit design involves a lot of human effort. Intelligent and smart EDA tools can make it easier and quicker for the designers to test the circuit on Field-Programmable Gate Arrays (FPGA) before implementing the Application-Specific Integrated Circuit (ASIC). None of the available EDA tools currently use Evolutionary Computation (EC), a search-based technique inspired by the Darwinian theory of evolution. EC had its roots in 1948 when Alan Turing proposed the genetic search method for the first time in history. Since its introduction, the EC research field has expanded significantly and has shown consistent results in different application areas. There are various types of EC methods nowadays, including Genetic Algorithms (GA) [15], Genetic Programming (GP) [8], Evolutionary Strategies (ES) [21], etc. The field of study dedicated to the evolution of circuits or electronic hardware is called Evolvable Hardware (EH). EH is divided into two major categories: intrinsic evolution [34] and extrinsic evolution [6]. In intrinsic evolution, the circuit is evolved and tested on hardware such as an FPGA, while in extrinsic evolution, it is evolved and tested through simulations. This work uses Grammatical Evolution (GE) to create sequential circuits. GE is a GP technique using context-free Backus-Naur Form (BNF) grammars for genotype-to-phenotype mapping. This allows the designers to write the rules of the problem according to the circuit's structure and design requirement in grammar and let the system evolve the solutions from there.

Very High-Speed Integrated Circuit Hardware Description Language (VHDL) [18], and Verilog [3] are the most commonly used HDLs. System-Verilog (SV) [25], a superset of Verilog, is used in this work. SV has some significant advantages over Verilog; in addition to being an HDL, it is also Hardware Verification Language (HVL).

Digital circuits are typically divided into two types: combinational circuits and sequential circuits. Combinational circuits are not time-dependent and provide the output solely based on the current input of the system. They give the output as soon as the input changes. In contrast, a sequential circuit's output is dependent not only on the current input but the previous inputs of the system as well. Unlike combinational circuits, a sequential circuit follows a pattern of inputs to reach a specific output. This pattern is shown in terms of the states of the system. The sequential circuits have a memory element attached to save these states.

A sequential circuit can be represented as a Finite State Machine (FSM) [16]. An FSM is a pictorial representation of the states of a sequential circuit (can be seen here) and shows the state transition based on the system's current state and/or input. It also shows the output of the system against each transition. There are two types of FSMs, Mealy and Moore. The output of the Mealy machine depends upon the current state and input of the system, while in the case of a Moore machine, the output solely depends upon the system's current state.

This paper proposes a single-stage automatic design of some hard-to-design sequential circuits named Sequence Detectors (SDs). SDs are specific kinds of sequential circuits that detect a binary input sequence (such as 3-bit '101') and give '1' at the output on the positive edge of the clock when the sequence is detected. SDs are used in many systems, particularly critical systems such as alarm generation on sensing a specific sequence of events. Throughout this paper, we refer to a human-made circuit which solves the problem as the *gold circuit*, and such a circuit does not involve any evolution or automation. For example, the comparison of the gold circuit vs evolved 6-bit '111000' is shown here. Overlapping means it can detect two consecutive sequences, such as '111000111000'.

2 Related Work on Digital Circuits Evolution

The first work ever presented on the evolution of SD was by Ali et al. in 2004 [1], where they evolved 4-bit ('1010') and 6-bit ('011011') SDs using GA and four different evolutionary stages. The proposed 6-bit SD differs from the 6-bit SD given in our work here as it combines two '011' 3-bit SDs, which is far easier to evolve than evolving the 6-bit SD. Another work, presented in 2005 by Popa et al., evolved the same hardware using GA [20]. It achieved much better and optimised solutions while using fewer computational resources.

Presented by Yao et al. in 2007, a 3-bit SD ('110') evolved using an incremental, evolutionary approach with GA at its base [32], which evolves the basic modules of the large circuits with a greedy search in a small search space. Due to the evolution of basic modules in a small search space, this system is too focused on the evolution of specific circuits and not generalized. Another system generated to evolve the overlapping 3-bit SD is presented by Xiong et al. [31], which could detect separate or overlapping '101' and '100' sequences. This is evolved intrinsically over hardware. The exact lead author presented an applicable version of this work to the cardiovascular system in [27], where they evolved the same SD using GA-based extrinsic and intrinsic evolution and showed their work's implementation on FPGA. In both works, for 3-bit (not complex at all to evolve), the computational resources used are massive compared to ours.

In another work presented by Tao et al. [28] in 2012, a 4-bit SD is evolved using a GA. They first evolved the basic circuits (small modules), which are then used to generate the whole circuit of SD. The proposed method comprises three evolutionary stages, which is a lot to evolve such a small SD. In our work, we evolved the 4-bit SD using far fewer computational resources and just one evolutionary stage.

In all the presented works above, SDs only evolved at the gate level. In our previous work [11], 3, 4, 5 and 6-bit SDs are evolved on the behavioural level using the tournament selection as the parent selection method. GE is used as the base for the evolution of SDs. 3, 4, and 5-bit SDs are successfully evolved using just one evolutionary stage, a performance upgrade compared to all previous works. However, 6-bit SD could not be evolved with the single-stage automated system. Hence, the grammar encapsulation was used, in which grammar is fed

with the best individual achieved so far and given the system a chance to evolve from there (detail in the [11]) to evolve 6-bit SD. Following the scheme proposed in [12], 50-bits long 1,000 training sequences are used (500 with desired seq. + 500 without desired seq). However, in this work, we conjectured that fewer cases could be used and started with the minimum bits required in the training sequences for each SD.

The system presented here also evolves SDs on the behavioural level. Behavioural level designs are recommended for the complex circuits [13] since parts of gate level codes such as *transif0* and *rpmos* cannot be synthesised. Also, gate-level codes do not scale flexibly as behavioural-level codes. Although evolving circuits on the behavioural level are challenging as well [23] due to its usage of highly expressive statements such as *for* or *while* loops and conditional statements such as *if-else*, but they can be synthesised.

Another challenge in evolving sequential circuits such as SDs at the behavioural level is deciding the structure and amount of training and test data required for evolution. Compared to [11], in this work, we have used far fewer training sequences, and the length of each sequence is also less than that used in that work (see Sect. 5 for detail). When compared with [11], it can be seen that tournament selection has shown almost the same results using more extensive and minimum bit length training sequences. However, our significant progress in this work is that Lexicase selection has shown outstanding results using minimum bit length training sequences compared to Tournament selection with identical length sequences.

3 Grammatical Evolution

GE [22] is a grammar-based Genetic Programming (GP) technique which uses rules written in Backus-Naur form (BNF) grammars for genotype to phenotype mapping. It typically uses context-free grammar and can evolve structures/objects written in any language. It has shown success in combinational [33] as well as sequential circuit designing [11], symbolic regression [2], and classification [17]. GE can generate HDL codes for circuits which then can be analysed for their efficiency and power/hardware consumption. A sample of genotype to phenotype mapping on gate level for HDL circuit design is shown in Fig. 1 here. BNF grammars used in GE comprise a set of four tuples: Starting symbol (S), Production rule (P) Non-terminal (N) and Terminal (T). S is a part of the first P and is an N where the mapping starts. In the shown example, it is $< var >$ where the first list of N starts on the Right Hand Side (RHS) of the first P. N are the parts of grammar which can be found on either the right or left side of equality such as $< var >$ (found on the RHS of first P and is also found on the Left Hand Side (LHS) of third P in shown example) and are further mapped to T. In contrast, T can not be further mapped to anything and can only be found on the right side of the equation, such as '!' and '&', which can only be seen on the RHS of the second P.

It can be seen in Fig. 1 that each chunk of 8 bits is first converted into decimals, and then each decimal value is used to expand the P of grammar until

we get to an expression having just T in them. This example shows the generation of logic gates between two variables x and y, which are the circuit's inputs and gives three gates in the options named AND, OR, and NOT. Please note that since the NOT gate can not be implied between two inputs (x and y) as it is an operation of inversion. It can only be used with one of the two variables, such as $!x$, using the second N of the first P. When the mapping finishes in the example given, the final expression implies the OR gate between the two input variables of the circuit, i.e., x and y. A detailed step-by-step explanation of this example can be seen in [11].

4 Tournament Selection vs Lexicase Selection

Parent selection is a significant part of the evolutionary cycle. It selects the individuals used before crossover and mutation to create offspring for the next generation. These parents are usually among the fittest individuals from the current generation, which can lay strong offspring for the next generation. The fitness score of each individual is assigned to it based on its performance on the training data set.

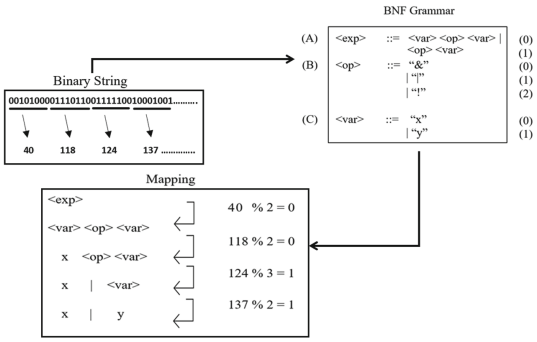

Fig. 1. Genotype to Phenotype Mapping in GE to Create SV Expression With Variables (x,y) And Logic Gates (& (AND), — (OR), and ! (NOT)).

Tournament Selection (TS) [14] runs a number of tournaments among n-individuals and selects the best based on its fitness value. In our case, the value of n is set to two. For example, two individuals are randomly selected with fitness scores of 320 and 321, and the TS method will choose the individual with a score of 321 as a parent. In our experiments, an individual's fitness value is its absolute fitness value computed after it is evaluated for all the test cases.

In contrast, Lexicase Selection (LS) [26] does not pick an individual based on its aggregated (across all test cases) fitness value. Instead, it picks individuals based on their performance on certain test cases, which means this method picks specialists rather than generalists. For example, only those individuals who solve a specific test case can be selected. If there is a tie between two individuals, they can be chosen.

LS has shown excellent results in solving problems related to different fields, such as regression problems [19] and evolutionary robotics [9]. It has especially shown outstanding performance in the evolution of digital circuits [23,29,30] as well.

5 Data Set Generation

The size of the training data set was seen to influence the system's performance significantly. Different schemes for selecting training data set size were devised and experimented with for both Tournament and Lexicase Selection. In contrast with the length of sequences as defined in [11] (can be seen here), where they followed the technique proposed in [12], we use training sequences with the minimum required bits to evolve the SDs to get a correct solution. The word 'correct' implies the performance of a circuit if it is evolved using the training data set with minimum bit sequences and gives a hundred per cent test accuracy on a more extensive test data set (50-bit 1,000 sequences) used in [11].

Table 1. Training Data Set Generation Scheme Using Minimum Number of TVs Required.

Sequences For Overlapping SDs Accordingly	Minimum Bits Required Per Sequence	Total Number of Sequences	Sequences With '1' in Their Output	Resulting Distributed TVs
3-bit '101' ==>'10101'	5	$2^5 = 32$	11/32	$32 \times 5 = 160$
4-bit '1101' ==>'1101101'	7	$2^7 = 128$	31/128	$128 \times 7 = 896$
5-bit '11011' ==>'11011011'	8	$2^8 = 256$	31/256	$256 \times 8 = 2048$
6-bit '111000' ==>'111000111000'	6	$2^6 = 64$	1/64	$64 \times 6 = 384$

In this work, Mealy machines are evolved, which respond quickly and use fewer hardware resources than Moore machines. We have evolved overlapping FSMs for these circuits to detect the desired sequences coming adjacent to each other at the input, resulting in a highly responsive and generic SD. It can be seen in Table 1 that the minimum bits required per sequence are higher than the bits of the sequence under detection, as it is an overlapping SD. However, this condition does not apply to the 6-bit '111000' SD since it does not matter if it uses overlapping sequence detection; the detection of the new incoming sequences will always start from the first state, i.e. $S0$, which is usually a case in human-made FSMs. So, in this case, the minimum length for each sequence for 6-bit '111000' SD is six.

As the SDs are designed to detect the binary sequences, the minimum number of producible sequences for any n-bits per sequence is 2^n. It can be seen in the third column of Table 1 that the total number of sequences required for training the system is far less than the 1,000 sequences (500 with desired seq. + 500 without desired seq) used in [11]. The number of n-bit sequences having the

actual sequence required to be detected (such as 3-bit '101') out of a total of 2^n sequences can also be seen in the orange colour, far less than 500 used in [11]. However, it can be seen that for the 6-bit SD, we have one sequence out of 64 for the 6-bit '111000' sequence. We will discuss the effect of it later in Sect. 6. As discussed in [11], for n-bit SD, each 50-bit sequence is distributed in 50 TVs for each of the 1,000 sequences resulting in a total Test Vectors (TV) and overall fitness of 50,000. In the proposed method, total TVs and fitness are far less than 50,000, as shown in the last column of Table 1.

6 Experiments and Results

6.1 Experimental Setup, Tools and Evolutionary Parameters

BNF grammars used in this work are the same as those used in [11]. Figure 2 shows the 3-bit '101' SD sample grammar from [11]. The structure of this grammar creates the SV module for the circuit by combining the parameter list indicated by <parameters> with the sequential (`always`) block, which is comprised of conditional (`if/else`) statements. These (`if/else`) statements are used to determine the current state and input of the system, as well as to set the next state and output accordingly. In grammar, the current states are hard-coded, while the following states are evolved according to the current state and input. These evolvable states and outputs which belong to T come in the $<$ *states_block* $>$ from the last two N, i.e., <state> and <out> respectively.

Table 2. Evolutionary Parameters.

Parameter	Value	Parameter	Value
No. Of Runs	30	Crossover Probability	0.9 (One Point)
Population Size	1,000		
No. Of Generations	30	Selection	Tournament
Initialisation	Sensible	Elitism	Yes
Mutation Probability	0.01	Test Vectors	1,000

To evolve the SD using this grammar, we used Grammatical Algorithms in Python for Evolution (GRAPE) [10], a Python-based implementation of GE. In this system, we also use Icarus Verilog, a Verilog/SystemVerilog simulator used here to evaluate individuals. All the experiments are run here on Dell OptiPlex 5070 desktop computer. This system comprises a single RAM of 16 GB, 1 TB HDD, 256 GB SSD, and a 64-bit quad-core 9th generation i7 processor with a 12 MB cache. The primary frequency of this used processor is 3.0 GHz, which can reach 4.7 GHz when required.

6.2 SD Evolution Using Minimum and Increased Length Sequences

Given the experimental setup, evolutionary parameters shown in Table 2 and minimum TVs taken from length sequences, 3-bit '101', 4-bit '1101', 5-bit '11011', and 6-bit '111000' SDs are evolved using Tournament and Lexicase parent selection methods. The results can be seen in Table 3. As one might expect, as we approach the more complex SDs, the success rates decrease. It can be seen in Table 3 that Lexicase performed better than Tournament for 4-bit '1101' and 5-bit '11011' SDs. Since lexicase apparently did not do better for 3-bit '101' and 6-bit '111000', Wilcoxon statistical significance tests were run on the values taken from these experiments. For 3-bit '101' and 6-bit '111000', using the fitness values of the best-resulting individual in each run, we find p-values above our threshold of 0.05, demonstrating no statistically significant difference between the two setups since the difference between the success rates and the achieved max fitness values (383/384 vs 384/384 for 6-bit SD) is extremely low. However, for the 4-bit '1101' SD, the p-value is 0.0256 which is less than 0.05 and indicates that there is a significant difference between the results of these two setups

```
<final>         ::= <parameters> \n <sequential>
<parameters>    ::= "reg [1:0]state = 2'b00;\n
                             parameter S0 = 2'b00;\n
                             parameter S1 = 2'b01;\n
                             parameter S2 = 2'b11;\n "
<sequential>    ::= "always @ (posedge clk) begin \n
                            if (rst == 1)begin \n
                                state <= S0; \n
                                out <= 0; \n end \n
                            else if (rst == 0) \n begin \n
                                if (state == S0) \n begin \n
                                    "<states_block> \n" end \n
                                else if (state == S1) \n begin \n
                                    "<states_block> \n" end \n
                                else if (state == S2) \n begin \n
                                    "<states_block> \n" end \n
                            end \n end \n"
<states_block>  ::= "if (inp==1)\n begin \n
                        state <= " <state> ";\n
                        out <= "<out>"; end \n
                    else if(inp == 0) \n begin \n
                        state <= " <state> ";\n
                        out <= "<out>"; end"
<state>         ::= "S0"|"S1"|"S2"
<out>           ::= "0"|"1"
```

Fig. 2. BNF grammar to evolve SV if/else statements deciding next state and output of 3-bit '101' SD.

and the success rates (21/30 vs 04/30) backing this claim, tells that Lexicase is performing much better than Tournament here. The graph representing the mean of the best fitness across all thirty runs for thirty generations for 4-bit SD can be seen here.

Table 3. Comparison Of Success Rates Of Evolved SDs With Lexicase vs Tournament Selection Using Minimum Length Sequences.

Seq. Detector	Success Rate (Tournament)	Success Rate (Lexicase)	Wilcox Test Results Based On Max. Fitness (P-Value)	Wilcox Test Results Based On Mean Of Avg. Fitness (P-Value)
3-bits '101'	**30**/30	**28**/30	0.1607	$\ll$0.001
4-bits '1101'	**04**/30	**21**/30	0.0256	$\ll$0.001
5-bits '11011'	**01**/30	**03**/30	0.3784	$\ll$0.001
6-bits '111000'	**01**/30	**Zero**/30	0.3337	$\ll$0.001

Since these SDs are evolved using significantly less training data set compared to the previous work when we tested them against the generalised test data used in [11], i.e., 50-bits 1,000 sequences (500 with desired seq. + 500 without desired seq.), 3-bit '101' and 4-bit '1101', SDs gave us a hundred per cent success rate for both Tournament selection and Lexicase selection as shown in Table 4 and Table 5. We did not get such great results for 5-bit '11011' (02/03 for Lexicase and 01/01 for Tournament); however, we obtained the generalised solutions of 5-bit '11011' SD with both parent selection techniques.

Table 4. Performances Of Sequence Detectors Evolved Using Lexicase Selection On Generalised Test Data Set.

Seq Detector	Pop. Size Taken For Evol.	Success Rate (Lexicase)	Performance On Test Set (1,000 Sequences Each Of 50-bits)
3-bits '101'	1,000	**28**/30	**28**/28
4-bits '1101'	1,000	**21**/30	**21**/21
5-bits '11011'	1,000	**03**/30	**02**/03
6-bits '111000'	1,000	**Zero**/30	–
6-bits '111000'	2,000	**Zero**/30	–
6-bits '111000'	5,000	**01**/30	**Zero**/01

For the 6-bit '111000' SD, no success was achieved using Lexicase selection with the evolutionary parameters used in any experiments. We increased the population size from 1,000 to 2,000 and then 5,000. With 5,000, a single solution was found, which did not perform perfectly on the test data, as shown in Table 4.

For the same SD using Tournament selection, one success was achieved with a population size of 1,000. Still, the experiments were run with a population size of 2,000 and 5,000 to compare the results fairly, as shown in Table 5. One correct solution was discovered when using a population size of 2,000 but, when the population was increased to 5,000, no solutions were found. Such results are not to be unexpected when dealing with such small numbers of successes with stochastic systems, but we reran the 2,000 and 5,000 population sizes, this time with 100 repetitions. The results are in the bottom half of Table 5.

Table 5. Performances Of Sequence Detectors Evolved Using Tournament Selection On Generalised Test Data Set.

Seq Detector	Pop. Size Taken For Evol.	Success Rate (Tournament)	Performance On Test Set (1,000 Sequences Each Of 50-bits)
3-bits '101'	1,000	**30**/30	**30**/30
4-bits '1101'	1,000	**04**/30	**04**/04
5-bits '11011'	1,000	**01**/30	**01**/01
6-bits '111000'	1,000	**01**/30	**Zero**/01
6-bits '111000'	2,000	**01**/30	**Zero**/01
6-bits '111000'	5,000	**Zero**/30	–
6-bits '111000'	2,000	**01**/100	**Zero**/01
6-bits '111000'	5,000	**05**/100	**Zero**/05

Thus far, no successful candidate has evolved which could perform perfectly on the test data set for the 6-bit SD. We believe the key reason for this is that the training set is too biased, as it contains just a single positive case since just one out of 64 training sequences has 6-bit '111000' in them. Rather than doubling the number of bits for each sequence, we incrementally increased the length and selected an equal number of sequences with and without'111000'. For example, in 7-bit sequences, only four out of 128 sequences contained the key '111000' pattern. So, in addition to selecting these four sequences, four more sequences are randomly selected, which do not have 6-bit '111000' in them. From these, distributed TVs were generated using the same technique used for the minimum length sequences as shown in the last column of Table 3. Following this increase in the bits of each training sequence, only one successful generalized solution was found while evolving with 9-bit sequences and using the Lexicase selection method as shown in Table 6 while no generalized solution could be found with Tournament selection as seen in Table 6. Although for 7-bit training sequences, Tournament performed a little better than Lexicase selection (30/30 vs 29/30), the overall performance of Lexicase is much better than Tournament, which can be seen in Table 6. Following the P-value, it can be said that both selection methods' performance is the same for 7-bit training sequences since Lexicase is falling behind by just one run out of 30.

The evolved FSM of 6-bit '111000' SD can be seen in comparison with human-made gold FSM here which is entirely different from all the solutions evolved in [11].

Table 6. Performance Comparison Of 6-bit SD Evolved Using Equal And Lengthened Seqs. With Lexicase And Tournament Selection on Generalised Test Data Set.

Sequence Bits ==>Seqs.	Equal No. Of TVs (With Seq. + Without Seq.)	Success Rate (Lexicase)	Performance On Test Set (Lexicase)	Success Rate (Tournament)	Performance On Test Set (Tournament)
7 ==>128	4 + 4 = 8	**29**/30	**Zero**/29	**30**/30	**Zero**/30
8 ==>256	12 + 12 = 24	**07**/30	**Zero**/07	**01**/30	**Zero**/01
9 ==>512	32 + 32 = 64	**04**/30	**01**/04	**Zero**/30	–

7 Conclusion

This paper uses the Grammatical Evolution and Lexicase Parent Selection methods to present an enhanced performance in the successful evolution of 3-bit, 4-bit, 5-bit, and 6-bit Sequence Detectors. It is shown that Lexicase has performed much better than the Tournament Parent Selection method. Another achievement is the successful evolution of generalized solutions using a minimal number and length of training sequences compared to the literature for 3-bit, 4-bit and 5-bit Sequence Detectors. However, for 6-bit, we had to increase the size of the sequences. These generalized solutions have performed excellently on a vast amount of test data sets. Since the evolved circuits here are the single input and single output Sequence Detectors, it is planned to extend this work to evolve multi-input and/or multi-output Sequence Detectors. It is also planned to evolve the current states of the Finite State Machines in addition to the next states and outputs (evolved in this work) of the presented circuits.

Acknowledgments. This work is supported by the Science Foundation Ireland grant #16/IA/4605. The third author is also financed by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES), Finance Code 001, and the Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ).

Scientific Validation. This paper has benefited from the remarks of the following reviewers:

- Malcolm Heywood, Dalhousie University, Canada
- Ting Hu, Queen's University, Canada
- Karim Tout, Uqudo, UAE

The conference organisers wish to thank them for their highly appreciated effort and contribution.

References

1. Ali, B., Almaini, A.E.A., Kalganova, T.: Evolutionary algorithms and theirs use in the design of sequential logic circuits. Genet. Program. Evolvable Mach. **5**, 11–29 (2004)
2. Ali, M., Kshirsagar, M., Naredo, E., Ryan, C.: Towards automatic grammatical evolution for real-world symbolic regression. In: Proceedings of the 13th International Joint Conference on Computational Intelligence - Volume 1: ECTA, pp. 68–78. INSTICC (2021)
3. Ciletti, M.D.: Advanced Digital Design with the Verilog HDL, 2nd edn. Prentice Hall Press, Hoboken (2010)
4. Eagle: Eagle by autodesk (1988). https://www.autodesk.com/products/eagle/overview. Accessed 1 Nov 2022
5. Farrahi, A., Hathaway, D., Wang, M., Sarrafzadeh, M.: Quality of EDA CAD tools: definitions, metrics and directions. In: Proceedings IEEE 2000 First International Symposium on Quality Electronic Design (Cat. No. PR00525), pp. 395–405. IEEE, San Jose, CA, USA (2000)
6. Kalganova, T.: An extrinsic function-level evolvable hardware approach. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) Genetic Programming, pp. 60–75. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-540-46239-2_5
7. KiCad: KiCad electronic design automation (1992). https://www.kicad.org/. Accessed 1 Nov 2022
8. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
9. La Cava, W., Moore, J.: Behavioral search drivers and the role of elitism in soft robotics. In: ALIFE 2018: The 2018 Conference on Artificial Life, pp. 206–213 (2018)
10. de Lima, A., Carvalho, S., Dias, D.M., Naredo, E., Sullivan, J.P., Ryan, C.: GRAPE: grammatical algorithms in Python for evolution. Signals **3**(3), 642–663 (2022)
11. Majeed., B., et al.: Evolving behavioural level sequence detectors in systemverilog using grammatical evolution. In: Proceedings of the 15th International Conference on Agents and Artificial Intelligence - Volume 3: ICAART, pp. 475–483 (2023)
12. Manovit, C., Aporntewan, C., Chongstitvatana, P.: Synthesis of synchronous sequential logic circuits from partial input/output sequences. In: Sipper, M., Mange, D., Pérez-Uribe, A. (eds.) ICES 1998. LNCS, vol. 1478, pp. 98–105. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0057611
13. Mealy, B., Tappero, F.: Free Range VHDL. Free Range Factory (2013); eBook (2018), USA (2018)
14. Miller, B.L., Goldberg, D.E.: Genetic algorithms, tournament selection, and the effects of noise. Complex Syst. **9**, 193–212 (1995)
15. Mirjalili, S.: Genetic Algorithm, pp. 43–55. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-93025-1_4
16. Morris, M., Ciletti, M.D.: Digital Design. Pearson Prentice Hall, Upper Saddle River (2007)
17. Murphy, A., Murphy, G., Amaral, J., Mota Dias, D., Naredo, E., Ryan, C.: Towards incorporating human knowledge in fuzzy pattern tree evolution. In: Hu, T., Lourenco, N., Medvet, E. (eds.) European Conference on Genetic Programming (Part of EvoStar), vol. 12691, pp. 66–81. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72812-0_5

18. Navabi, Z.: VHDL: Modular Design and Synthesis of Cores and Systems. McGraw-Hill, New York (2007)
19. Orzechowski, P., La Cava, W., Moore, J.H.: Where are we now? A large benchmark study of recent symbolic regression methods. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1183–1190. Association for Computing Machinery (2018)
20. Popa, R., Aiordăchioaie, D., Sîrbu, G.: Evolvable hardware in Xilinx Spartan-3 FPGA. In: Proceedings of the 2005 WSEAS International Conference on Dynamical Systems and Control (ICDSC), pp. 66–71 (2005)
21. Rudolph, G.: Evolutionary Strategies, pp. 673–698. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-540-92910-9_22
22. Ryan, C., Collins, J.J., Neill, M.O.: Grammatical evolution: evolving programs for an arbitrary language. In: Banzhaf, W., Poli, R., Schoenauer, M., Fogarty, T.C. (eds.) EuroGP 1998. LNCS, vol. 1391, pp. 83–96. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055930
23. Ryan., C., Tetteh., M.K., Dias., D.M.: Behavioural modelling of digital circuits in system verilog using grammatical evolution. In: Proceedings of the 12th International Joint Conference on Computational Intelligence - ECTA, pp. 28–39. INSTICC, SciTePress (2020)
24. Solido: Solido design solutions (2005). https://eda.sw.siemens.com/en-US/ic/solido/. Accessed 1 Nov 2022
25. Spear, C.: SystemVerilog for Verification. A Guide to Learning the Testbench Language Features, 2nd edn. Springer, New York (2008). https://doi.org/10.1007/978-1-4614-0715-7
26. Spector, L.: Assessment of problem modality by differential performance of lexicase selection in genetic programming: A preliminary report. In: Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation. p. 401–408. GECCO '12, Association for Computing Machinery, New York, NY, USA (2012)
27. Tani, F.I.J.U., Tani, M.M.: An evolutionary circuit model for cardiovascular system: an FPGA approach. Int. J. Comput. Inf. Technol. Eng. (2011)
28. Tao, Y., Cao, J., Zhang, Y., Lin, J., Li, M.: Using module-level evolvable hardware approach in design of sequential logic circuits. In: 2012 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8. IEEE, New York (2012)
29. Tetteh, M., Dias, D.M., Ryan, C.: Grammatical evolution of complex digital circuits in SystemVerilog. SN Comput. Sci. **3**(3), 188 (2022)
30. Tetteh, M.K., Mota Dias, D., Ryan, C.: Evolution of complex combinational logic circuits using grammatical evolution with SystemVerilog. In: Hu, T., Lourenço, N., Medvet, E. (eds.) Genetic Programming, pp. 146–161. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72812-0_10
31. Xiong, F., Rafla, N.I.: On-chip intrinsic evolution methodology for sequential logic circuit design. In: 2009 52nd IEEE International Midwest Symposium on Circuits and Systems, pp. 200–203. IEEE, New York (2009)
32. Yao, R., Wang, Y., Yu, S., Gao, G.: Research on the online evaluation approach for the digital evolvable hardware. In: Kang, L., Liu, Y., Zeng, S. (eds.) ICES 2007. LNCS, vol. 4684, pp. 57–66. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74626-3_6
33. Youssef, A., Majeed, B., Ryan, C.: Optimizing combinational logic circuits using grammatical evolution. In: 2021 3rd Novel Intelligent and Leading Emerging Sciences Conference (NILES), pp. 87–92. IEEE, New York (2021)

34. Zhang, Y., Smith, S., Tyrrell, A.: Digital circuit design using intrinsic evolvable hardware. In: Proceedings of 2004 NASA/DoD Conference on Evolvable Hardware, pp. 55–62. IEEE, New York (2004)