INDUCTION OF STATE MACHINES FROM TEMPORAL LOGIC CONSTRAINTS

Marek Perkowski †, Anatoly Chebotarev +, and Alan Mishchenko †, +

† Portland State University,
Department of Electrical and Computer Engineering, Portland, USA

+ Glushkov Institute of Cybernetics, Kiev, Ukraine

EVOLVABLE HARDWARE OR LEARNING HARDWARE?

- Evolvable Hardware is Genetic Algorithm (GA) plus reconfigurable hardware.
- One may ask: "Why **Genetic** Algorithm"?
- We question the usefulness of GA as a sole learning method to reconfigure binary FPGAs.
- We propose the "Learning Hardware" approach.
- Creating a sequential network based on feedback from the environment (for instance, positive and negative examples from the trainer), and realizing this network in an array of Field Programmable Gate Arrays (FPGAs).

INDUCTION OF STATE MACHINES FROM TEMPORAL LOGIC CONSTRAINTS

1. Training on examples.
2. Constraints solving.
3. Finite State Machine (FSM) minimization.
4. Structural mapping of machine to Regular Automata.
5. Functional decomposition of multi-valued logic functions and relations to Regular Layout.
6. Final mapping of Regular Automata and Layout to FPGA resources.

LEARNING ON A HIGHER LEVEL

- Learning on the level of constraints acquisition and functional decomposition rather than on the low level of programming binary switches.
- Occam's Razor learning that allows for generalization and discovery.
- Fast operations on complex logic expressions and solving NP-complete problems such as satisfiability.
- Algorithms realized in hardware to obtain the necessary speed-ups.
- Fast prototyping tool, the DEC-PERLE-1 board based on an array of Xilinx FPGAs.

SOFT COMPUTING AND MACHINE LEARNING VERSUS HARDWARE DESIGN

- Artificial Neural Nets (ANNs), Cellular Neural Nets (CNN), Fuzzy Logic, Rough Sets, Genetic Algorithms (GA), Genetic and Evolutionary Programming, Artificial Life, solving problems by analogy to Nature, decision making, knowledge acquisition, new approaches to intelligent robotics.
- Learning, adapting, modifying, evolving or emerging.
- Mixed approaches.
- The computer is taught on examples rather than completely programmed (instructed) what to do.
- **Machine Learning** becomes a new and most general system design paradigm unifying these previously disconnected research areas.
- It starts to become a **new hardware construction paradigm** as well.

EVOLVABLE HARDWARE

- DeGaris - Evolvable Hardware is realization of genetic algorithm (GA) in reconfigurable hardware.
- Brain Builder CBM (DeGaris), ROBOKONEKO.
- Neural Nets PLUS Genetic Algorithm.
- The Genetic Algorithm is a simple and practically blind mechanism of Nature.
- It is easily realizable in hardware.
- Although it is relatively easy to do crossover and mutation in hardware, the fitness function evaluation is difficult.

UNIVERSAL LOGIC MACHINE

- Started in Poland, 1977. Logic Design Machine. (TTL logic model): Satisfiability, Petrick Function (ICCAD'85).
- Tsutomu Sasao, 1985: Tautology Engine in EPLDs (ICCD'85).
- Cube Calculus Machine, since 1990. (realization in FPGAs). (Sendai'92).
- Decomposition Machine, since 1997, (DEC-PERLE-1), (Lousanne'98, ICCD'98, Sendai'99).
- Temporal Constraints Machine - new ideas presented here for the first time (reduce to Satisfiability, Tautology, Decision Functions, and Boolean/Multi-Valued Logic Equations.

LOGIC ALGORITHMS IN HARDWARE

- Logic algorithms draw upon human knowledge.
- Logic algorithms are optimal and mathematically sophisticated.
- Logic algorithms lead to high quality learning results: knowledge generalization, discovery, no overfitting, small learning errors (Ross, Abu Mostafa, DFC, COLT).
- Their software realizations use very complex data structures and controls.
- It is difficult to realize them in hardware.

LEARNING HARDWARE

- Learning understood very broadly, as any mechanism that leads to the improvement of operation.
- Evolution-based learning is thus included in it.
- Combinational or sequential network is constructed that stores the knowledge acquired in the learning phase.
- The learned network is next run on old or new data.
- The responses may be correct or erroneous. The network's behavior is then evaluated by some fitness (cost) functions and the learning and running phases are alternating.

TWO PHASES OF LEARNING IN HARDWARE

- **The phase of learning**, which is, constructing and tuning the network.
- **The phase of acting**. Using knowledge, running the network for data sets.
- Comparing to the process of developing and using a computer, the first stage could be compared to the entire process of conceptualizing, designing, and optimizing a computer, and the second stage to using this computer to perform calculations.
- You cannot redesign standard computer hardware, however, when it cannot solve the problem correctly.
- The Learning Hardware will redesign itself automatically using new learning examples given to it.

Logic rather than evolutionary methods for learning

- Michie makes distinction between black-box and knowledge-oriented concept learning systems by introducing concepts of **weak** and **strong** criteria.
- The system satisfies a **weak criterium** if it uses sample data to generate an updated basis for improved performance on subsequent data. A **strong criterion** is satisfied if the system communicates concepts learned in symbolic form.
- ANNs satisfy only the weak criterium while our approach satisfies the strong criterium. Our approach operates on higher and more natural symbolic representation levels.
- The built-in mathematical optimization techniques (such as graph coloring or satisfiability) support the Occam's Razor Principle.
- Solutions are provably good in the sense of Computational Learning

Theory (COLT).

Importance of Functional Decomposition

- **Functional Decomposition** is used in many applications: FPGA mapping, custom VLSI design, regular arrays, Machine Learning, Data Mining and Knowledge Discovery in Data Bases (KDD).
- **Exact** decomposition programs are slow.
- **Approximate** programs may give inferior quality solutions.
- How to create a decomposer that will be **both effective** and **efficient**?.
- ANSWER: Software/Hardware Co-Design.

We do not like Genetic Algorithms. Any Discussions?

- In our experience, especially poor results on logic approaches are obtained using the genetic algorithms.
- The same was true based on literature.
- In our approach we want to make use of this accumulated human experience, rather than to "reinvent" algorithms using GA.

The ULM approach to Learning Hardware

1. Based on sets of examples specified in our input language **L**, we create a Reactive State Machine (RSM), in particular, a (combinational) function or a relation with no temporal variables.
2. The description consists in input-output specification, initial state specifications and global environment constraints.
3. This machine is usually non-deterministic, but is state-minimal from construction with respect to all its variables as state variables.
4. The machine can be determinized (converted from non-deterministic to deterministic form).
5. Next the machine is state-minimized (with respect to the new set of state variables, which are a subset of initial input/output variables).
6. The machine is mapped to constrained structural resources which we call Regular Automata (RA).

7. The time-based MV logic expressions of Regular Automata are decomposed. (Ashenhurst-Curtis decomposition). The timed variables, and the multi-valued variables, are converted to new binary variables.

8. The (quasi)optimally constructed network is logically mapped to standard FPGA CLBs and realized using standard partitioning, placement, and routing with the help of EDA tools from Xilinx or other companies. Thus, each RSM is converted to a binary pattern of programming switches in FPGA.

9. The knowledge of the machine is stored in binary memory patterns representing final FPGA reconfigurable information. Under supervision of the software program, the hardware switches between a number of evolved circuits, depending on rules that can also be acquired automatically.

10. As the network solves new problems, the new data sets and training decisions are accumulated and the network is repeatedly redesigned.

Induction of Reactive State Machines from Temporal Logic Constraints

- Our state machine design integrates methods developed in USA and former USSR.
- The use of temporal logic as the input specification.
- The use of Regular Automata for structural design.

The Input Language to Represent the Learning Data

- High-level constraint temporal language.
- temporal logic (Chebotarev), regular expressions (Glushkov,Kleene), Petri nets, State Machine tables, tabular representation of data (Codd), binary and Multi-Valued Cube Calculus (Dietmeyer), Decision Tables, Rough Sets (Pawlak), and Labeled Rough Partitions (Grygiel).

Small Example

	x_1	x_2	y_1	y_2
a	0,2	1	-	2
b	0,1	0	0,2	1
c	2	0	1,2	0
d	1	1	1,2	2

Table 1: Multi-Valued multi-output (combinational) relation in tabular form.

Now we add time

Introducing variables that depend on discrete time.

For instance, the example a (row a) from Table 1 can be rewritten to our language as follows:

$$x_1[0,2](t) \;\wedge\; x_2[1](t) \;\Rightarrow y_2[2](t)$$

because both input variables $x_1(t)$, $x_2(t)$ and the output variable $y_2(t)$ are defined in the same moment of time. By allowing previous or next ticks of time, for instance:

$$x_1[0,2](t-2) \;\wedge\; x_2[1](t-3) \;\Rightarrow y_2[2](t+3)$$

we can specify arbitrary regular grammars, regular expressions, sequential netlists, or state machines with multi-valued inputs and outputs.

EXAMPLE: "Man, Wolf, Goat and Cabbage"

- At the beginning, all of them are on the left bank of the river.
- The man should transport them to the right bank.
- The wolf and the goat, as well as the goat and the cabbage, cannot be left alone on the same bank without man.
- The boat is navigated by the man, who can take only one object with him.
- Boolean variables are first-order predicates depending on discrete time.
- $M(t)$ is true when the man in the left bank and false when the man in on the right bank.
- The same applies to variables $W(t), G(t), C(t)$, which denote the wolf, the goat, and the cabbage respectively.

Temporal logic specification of constraints for "Wolf" (1)

1) The wolf and the goat cannot stay on the left bank and on the right bank without the man:

$(W(t)\&G(t)) \Rightarrow M(t). \quad (\sim W(t)\& \sim G(t)) \Rightarrow \sim M(t).$

The goat and the cabbage cannot stay on the left bank and on the right bank without the man:

$(G(t)\&C(t)) \Rightarrow M(t). \quad (\sim G(t)\& \sim C(t)) \Rightarrow \sim M(t).$

2) If the wolf is on the left (right) bank, it means that either the wolf stayed there or the man has brought it there from the right (left) bank one unit of time before:

$W(t) \Rightarrow (W(t-1) \mid \sim W(t-1)\& \sim M(t-1)\&M(t)). \quad \sim W(t) \Rightarrow (\sim W(t-1) \mid W(t-1)\&M(t-1)\& \sim M(t)).$

Temporal logic specification of constraints for "Wolf" (2)

The same for the goat and the cabbage:

$(G(t) \Rightarrow (G(t-1) \mid \sim G(t-1) \& \sim M(t-1) \& M(t)).\quad \sim G(t) \Rightarrow (\sim G(t-1) \mid G(t-1) \& M(t-1) \& \sim M(t)).$

$(C(t) \Rightarrow (C(t-1) \mid \sim C(t-1) \& \sim M(t-1) \& M(t)).\quad \sim C(t) \Rightarrow (\sim C(t-1) \mid C(t-1) \& M(t-1) \& \sim M(t)\ ).$

3) Any two objects cannot be brought across the river at the same time:

$(W(t) \Leftrightarrow W(t-1)) \mid (G(t) \Leftrightarrow G(t-1)).$

$(W(t) \Leftrightarrow W(t-1)) \mid (C(t) \Leftrightarrow C(t-1)).$

$(G(t) \Leftrightarrow G(t-1)) \mid (C(t) \Leftrightarrow C(t-1)).$

4) The man is not lazy:

$M(t-1) \Leftrightarrow \sim M(t).$

Reactive State Machine for "Wolf"

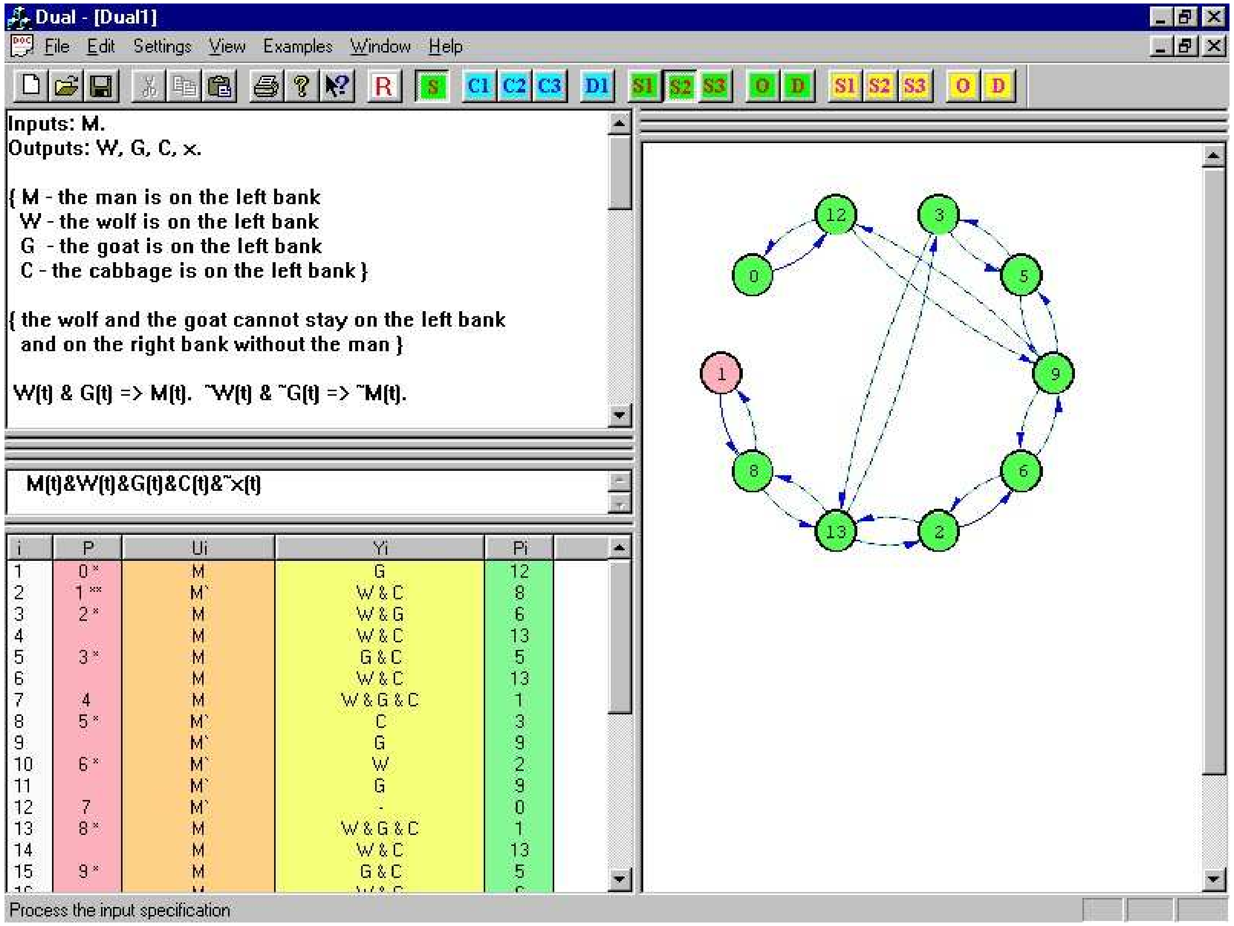

Figure 1: DUAL interface for RSM acquisition and synthesis ("man, wolf, goat and cabbage" problem).

Reactive State Machine for "Wolf"

- The state transition graph given in dotted lines corresponds to the non-deterministic algorithm, which the man should follow to safely transport all the objects to the right bank.
- This algorithm consists of seven steps, with two possible variations, depending on whether he chooses to take the cabbage or the wolf when the goat is already trasported to the right bank.
- Observe that the designer who gives the language **L** expressions to the learning system, does not know the state machine.
- He only specifies constraints on the behavior of the system or sample sequences of input and output values.

DEC-PERLE-1 Board for Fast Prototyping

- The DEC-PERLE-1 board (Vuillemin) is organized around a central computational matrix made up of 16 Xilinx XC3090 LCAs surrounded by a four 1MB RAM banks, and 7 other LCAs to implement switching and controlling functions.
- Cube Calculus Machine (CCM) on DEC-PERLE.
- The DEC-PERLE-1 board advocates **regular design styles** without many control signals.
- Good for small SIMD processors, pipelining, systolic processors, Cellular Automata or complex (decomposed) Boolean functions.
- The basic design principle is: **“map two-dimensional tables to two-dimensional logic resource arrays”**.
- Leads to the introduction of the concept of **Regular Automata**.

Demonstration of Learning Hardware in Robotics

- "An Improved Furby".
- Humanoid arms/head robot.
- OWI, Robix and LEGO Masterminds robotic kits.
- Sensors (touch, light, sound, temperature).
- Speech recognition.
- Image Recognition.

Demonstration of Learning Hardware in Robotics

- Learning is done with the human as the feedback loop.
- The set of sequences is incomplete, so the machine performs the **generalization** automatically. Adding or removing new rules, by the human supervisor or automatically/randomly, will change the behavior.
- Mimique the human's behaviors seen by the camera
- Like the Furby toy, but with real learning.
- Capable of building its own "world model" and internal model with unlimited behaviors.
- The new states corresponding to changed behavior will be created using our RSM approach.

Brain Builder (De Garis) versus Universal Logic Machine.

\ *Approach to* *Aspect of* \ *"Learning Hardware"* *Comparison.*\	Brain Builder	Universal Logic Machine
Model of Learning	Artificial Neural Net	Reactive State Machine
Training Data	Sample Vectors	Multi-Valued Temporal Logic Constraint Language **L**
How the net is constructed	Genetic Algorithm, ANN Training	Reactive State Machines construction and minimization, Multi-Valued Logic Synthesis
Virtual intermediate representation	Cellular Automata (CA)	Regular Automata (RA)
What is learned:	CA tables	Binary sequential logic nets
Net construction realized in:	Hardware (intrinsic EHW)	Hardware and software (software-hardware codesign)
Mapped to:	Array of binary FPGAs	Array of binary FPGAs
Hardware platform:	Xilinx 6000 series CBM	Xilinx 3090 + on-board memory DEC-PERLE-1 board + DEC workstation