Evolvable Hardware or Learning Hardware?

Extrinsic Approach

MACHINE LEARNING IS DESIGNING A NETWORK

Evolving in hardware versus learning in hardware.

- In recent years rapid developments in the area of soft computing occurred, including : Artificial Neural Networks (ANNs), Cellular Neural Nets (CNNs), Fuzzy Logic, Rough Sets, Genetic Algorithms (GAs), Genetic and Evolutionary Programming.
  - Mixed methods of the above are used to solve complex or poorly defined problems. The common factor is that they propose a way of automatic learning by the system. The computer is taught by examples rather than completely programmed in what to do.
- This philosophy also dominates the areas of Artificial Life, solving problems by analogy to nature, decision making, knowledge acquisition.
- Machine Learning thus becomes a new and general system design paradigm unifying these previously disconnected research areas.

- Evolvable Hardware (**EHW**) means the realization of genetic algorithm (GA) in reconfigurable hadware.

- Learning hardware approach consists in using feedback from the environment to create a sequential network and subsequently realizing this network in FPGAs.

- Universal Logic Machine (**ULM)** proposes the creation of a learning machine based on logic principles, in particular temporal logic, constructive induction, and rough set theory.

- Learning Hardware as any mechanism that leads to the improvement of operation including evolution-based learning. In the process of learning a network (combinational or sequential) is constructed that stores the knowledge acquired in the learning phase. The learned network is then run on old or new data. Responses may be correct or erroneous. The network behavior is then evaluated by some cost -

Induction of State Machines from Temporal Logic Constraints

- Human thinking consists in abstract use of symbols rather than assigning numeric weights to neurons.
- The built-in mathematical optimization techniques such as graph coloring and satisfiability

LEARNING IS DESIGNING A NETWORK

- Network can be continuous or discrete.
  - Combinational or sequential. Synchronous or asynchronous.
  - Boolean, Multi-Valued, Fuzzy, Algebraic.
- There is some way of designing this network by positive (and negative) examples.
- Next some way of evaluating network's behavior on data sets.
- Thus, the structure of the network must be created, and also its elements must be designed, adapted, selected from a menu, or tuned in the learning process.
- The network can be realized in **software, in hardware** or as **software-hardware co-design.**
- Amdahl' Law is crucial for hardware and mixed designs.
  - Not reasonable to speed up only part of computation

TRANSFORMING THE NETWORK

- Once the network has been found, it can be transformed to another network, either completely equivalent to it or being its generalization.
  - For instance, an integer-based neural net or a multi-valued (MV) decision tree can be both compiled to binary logic gates.
- The net can be designed using constructive methods all at once from the complete set of examples (an approach used for diagnostic trees), or it can be **built incrementally** (like done for neural nets).
- Many new approaches can be created and investigated based on **combining** basic learning models and methods in various ways.
- For instance:
  - the ANN built in Brain Builder can be directly compiled to binary hardware without the intermediate medium of cellular automata.
  - A **learning algorithm** different than the genetic algorithm can be used to construct the ANN.

VARIOUS APPROACHES TO LEARNING

- When the system has been already taught, it is irrelevant how.
- Construction methods differ in:
  - convergence speeds, sizes of networks,
  - their learning errors, networks' speeds,
  - testabilities, power consumptions.
- Different methods to design Learning Hardware and Evolvable Hardware disagree in the network model selection and network construction methods.

Evolving or Learning in Hardware?

Machine Learning becomes a new and most general system design paradigm

It starts to become a new hardware construction paradigm as well

Evolvable Hardware is Genetic Algorithm PLUS reconfigurable hardware

We propose **Learning Hardware** as any learning algorithm PLUS **reconfigurable hardware**

Learning algorithm can be realized in software or in hardware.

PLAN OF THIS LECTURE

- Compare logic versus ANN and GA approaches to learning.
- Introduce the concept of Learning Hardware .
- Methods of ***knowledge representation*** in the Universal Logic Machine (ULM):
  - **variants of Cube Calculus.**
- Two different concepts of designing Learning Hardware using the DEC-PERLE-1 board.
  - A general-purpose computer with instructions specialized to operate on logic data: Cube Calculus Machine.
  - A processor for only one application: Curtis Decomposition Machine.

LOGIC APPROACHES TO LEARNING: OUR EXPERIENCE

- We compared various network structures for learning:
  - two-level AND/OR (Sum-of-Products (SOP), or DNF),
  - Exclusive-Or-Sum-of-Products (ESOP),
  - Three-Level NAND/AND/OR networks,
  - Three-Level AND/NOT (TANT) networks,
  - Decision trees (C4.5),
  - multi-level decomposition structures.
- We compared also various logic, non-logic and mixed optimization methods:
  - search, rule-based, set-covering, graph-coloring,
  - genetic algorithm (including mixtures of logic and GA approaches),
  - genetic programming, articial neural nets, and simulated annealing.

LOGIC APPROACHES TO LEARNING: THE CONCLUSIONS

- We compared our networks' results on:
  - their **complexity** (Occam's Razor),
  - various ways of calculating the **error of learning .**
- The Decomposed Function Cardinality (DFC) and its extensions for MV logic were used as common measures of complexity, because of their theoretically proven properties.
- Logic approaches and especially the MV decomposition techniques, are *superior* to other approaches with respect to smaller net complexity and learning error.
- They should be combined with smart heuristic strategies and good data representations.

GENETIC ALGORITHMS ARE POOR. SO WHAT?

- Based on small complexity and error, the especially poor results were obtained using the genetic algorithms.
- We do not know of any problem for which a GA-based algorithm would be superior to a logic algorithm.
- We want to make use of the accumulated human experience in our approach, rather than to "reinvent" algorithms using GA.
- On the other hand, for large data the logic algorithms are relatively slow, **hence must be speed-up in hardware** .

HARDWARE SPEED-UP OF LEARNING ALGORITHMS

- **Five approaches** to implementing the learning algorithms:
  - **A1.** Both learning and execution are done in software. This standard approach still dominates the Artificial Neural Nets, Constructive Induction, Data Mining, and the so-called "extrinsic" Evolvable Hardware (DeGaris-93).
  - **A2.** The learning phase is performed in software and the network is next downloaded to hardware for execution (fuzzy logic controllers).
  - **A3.** The learning phase is performed in hardware and the execution phase in software. This approach is thus a hardware-accelerated design of a knowledge-based expert system.
  - **A4.** Both the learning and the execution phase are performed in hardware. This is the area of classical ANNs, CNNs and "intrinsic" evolvable hardware.
  - **A5.** Software-hardware co-design in one or in both phases. Most prospective approach in our view.

TECHNOLOGIES FOR LEARNING HARDWARE

- Field Programmable Analog Arrays, Cellular Neural Nets, Artificial Neural Nets, Multi-Valued FPGAs, Fuzzy Logic Programmable Arrays, and Mixed FPAAs.
- **Binary Field Programmable Gate Arrays** - there are no other mass-scale hardware reconfigurable technologies available.
- A practical task should be then to compare FPGA-based realizations of various machine learning paradigms.
- Learning levels:
  - 1. arithmetic operations (ANNs, Fuzzy Logic functions) - too high.
  - 2. switching transistors for routing connection paths - too low.
- OUR PRINCIPLE: Because in binary FPGAs everything is realized on the level of binary logic gates and flip-flops, the learning process should be also performed on the level of logic gates.

OTHER MAIN PRINCIPLES

- Re-use powerful EDA tools that engineers have already developed in many years in the area of digital design automation, especially for reconfigurable computers: state machines, logic synthesis, technology mapping, placement and routing, partitioning, timing analysis, etc.
- Occam Razor principle = meaningful discoveries.
- We do not believe that the "purist strategies" to evolutionary hardware (Brain Builder Group) will be practically acceptable for most commercial applications of Learning Hardware.

OTHER MAIN PRINCIPLES (cont)

- We propose here the principles of Learning Hardware that will:
  - use previous human problem-solving experience
  - and apply many mathematical algorithms and problem-solving strategies
  - rather than rely on only <u>two generic methods</u> of Evolvable Hardware: ANNs and GA.
- Learning/evolution should still remain as the main principle of building new generation hardware,
  - but it should be restricted to high abstract levels.
- The variants evaluation/selection should be also performed at abstract levels, before mapping to low-level field-programmable resources, such as switches, for which chromosomes are long and the operation of GA is inefficient.

LEARNING HARDWARE METHODOLOGY

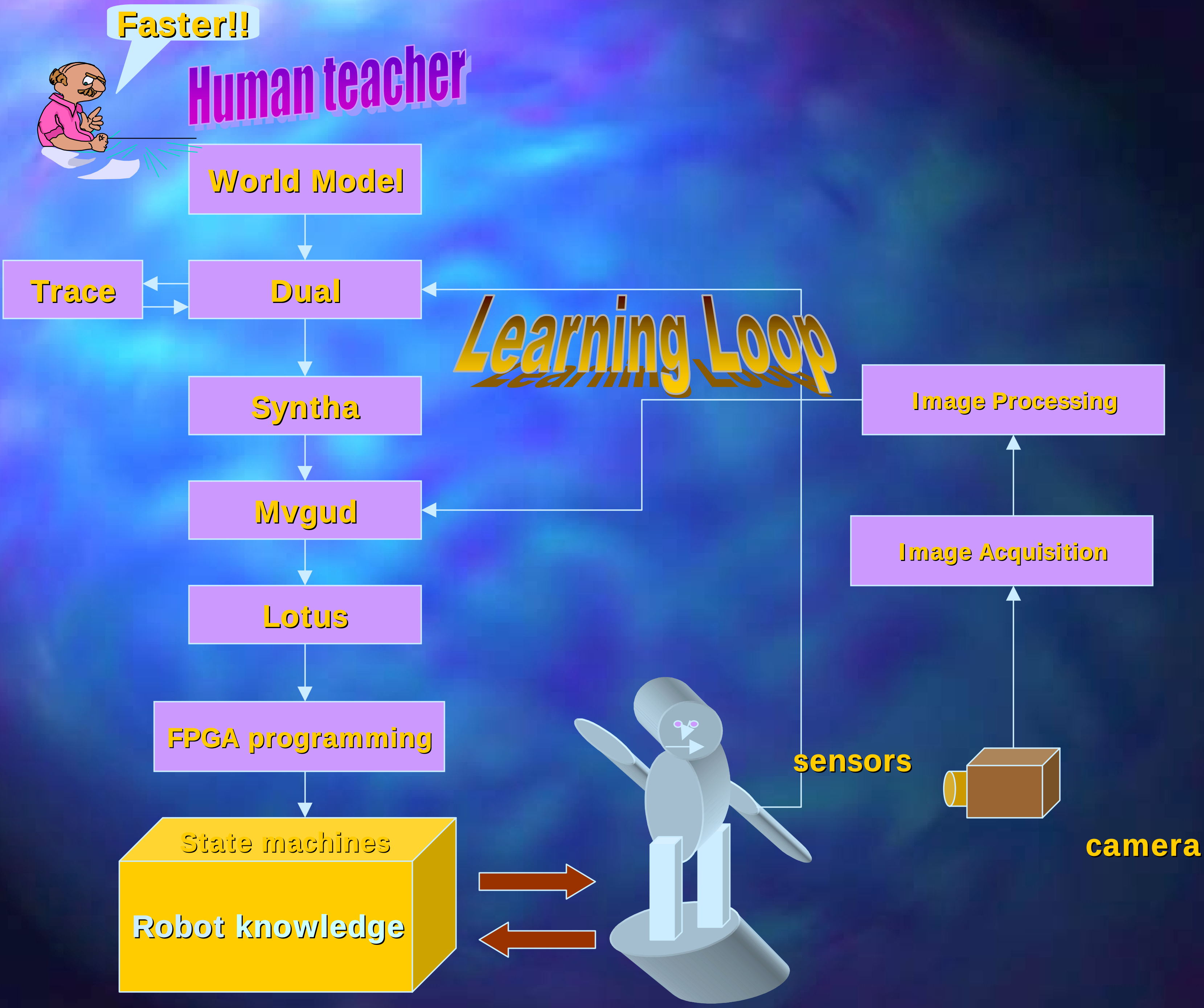
The phase of learning

- Based on sets of examples classified to several (at least two) categories, and various network requirements (background knowledge), the hardware processors, using logic/mathematical algorithms, create the logic network description.
- We use hardware realization of well-known logic synthesis algorithms such as: two-level AND/OR minimizers, two-level AND/EXOR minimizers, three-level OR/AND/OR minimizers, and functional (Ashenhurst-Curtis) decomposers.
- The (quasi)optimally constructed network is mapped to standard FPGAs and realized using standard EDA tools:
  - partitioning, placement and routing and other EDA tools from Xilinx and commercial EDA software companies.
- The knowledge of the machine is ***stored in memory patterns*** representing logic nets.

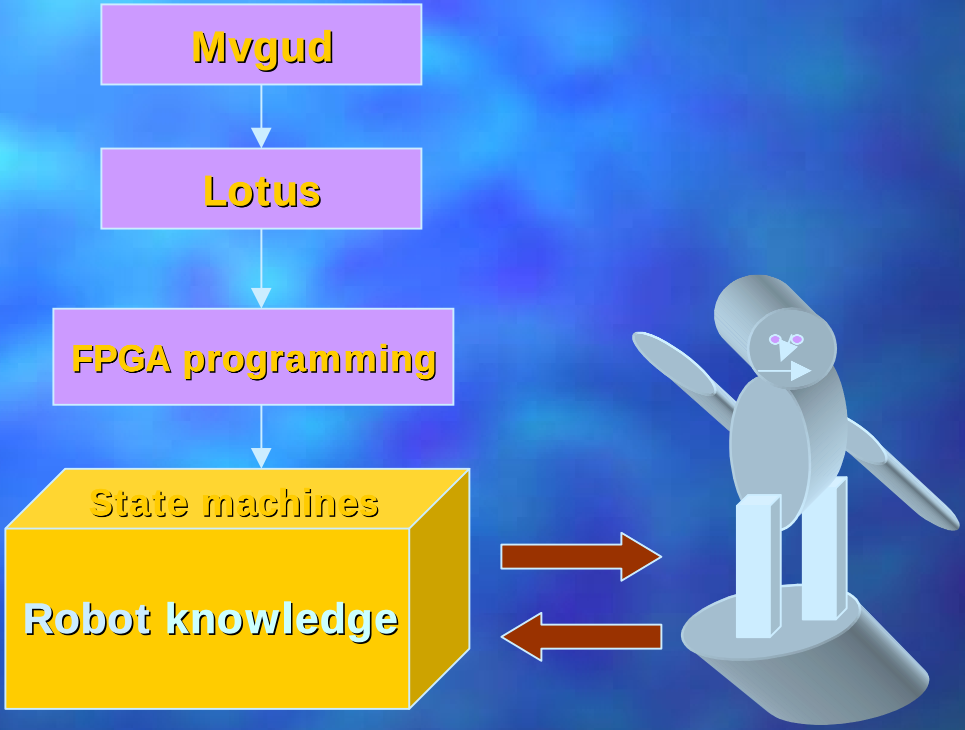
LEARNING HARDWARE METHODOLOGY

The phase of learning

- Under supervision of the software program, the *hardware multiplexes between various learned nets*, depending on rules that also can be acquired automatically.
- As the network solves new problems, the new data sets and training decisions are accumulated and the network is repeatedly automatically redesigned.
  - The old network can serve as a redesign plan for the new network, or the net is "redesigned from scratch".
- The process of evolving on all design levels used in EHW is replaced with the ULM model of learning at high level and next compiling to low level using standard FPGA-based tools.
- The same physical FPGA resources are multiplexed to implement:
  - the virtual human-designed "learning hardware"
  - and the automatically learned "data hardware".


Faster!!
Human teacher
World Model
Trace
Dual
Learning Loop
Syntha
Image Processing
Mvgud
Image Acquisition
Lotus
FPGA programming
sensors
State machines
Robot knowledge
camera

The "learning hardware" and the "data hardware”



- While the "learning hardware" is designed once by humans and cannot be changed, the "data hardware” can be permanently modified.

COMPARISON OF BRAIN BUILDER CAM-BRAIN MACHINE (CBM) AND UNIVERSAL LOGIC MACHINE

	Brain Builder	Universal Logic Machine
Model of Learning	Artificial Neural Net	Multi-Valued Logic Language
How the net is constructed	Genetic Algorithm, ANN Training	Multi-Valued Logic Synthesis, Constructive Induction, Rough Theory
Virtual intermediate representation	Cellular Automata	MV logic networks and state machines with arbitrary structures and arbitrary operators realized as look-up tables
What is learned:	automata tables	language expressions
Net construction	hardware	hardware and software
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

OUR GOALS

- ULM is the early prototype of **Data Mining machines** .
  - Such DM Machines will be able to mine data in on-line data bases, for instance from the Internet, production line, or battle-field.
- To develop a system for meaningful discoveries in narrowly defined areas.
- To speed-up *both* the learning and the execution phases of application software that is now used in Machine Learning, Knowledge Discovery from Databases, Data Mining, and robotics.

PRACTICAL GOALS OF THIS PROJECT

- Solve in few seconds and with error as small as the learning error of our current software, the following problems:
  - 1. Every data set from the U.C. Irvine Repository of Data Mining benchmarks, as well our benchmarks. Including all versions of large examples such as Breast Cancer, and Michalski's Trains with 30 trains (Grygiel). They take now up to 30 minutes in software.
  - 2. The recognition of cervical mucus fearning microscope images for ovulation prediction. It takes now minutes in software.
  - 3. The recognition of 3-dimensional images of rooms and corridors for mobile robot orientation (PSUBOT). This task takes now up to 17 minutes in software.

MULTI-VALUED LOGIC LANGUAGE TO REPRESENT THE LEARNING DATA IN HARDWARE

- A high-level language, in which expressions, the virtual nets, are automatically created, evaluated, selected and optimized, to be next realized as hardware FPGA nets by top-down automatic design methods.
- In ML, DM and KDD data are very strongly unspecied (99% of don't cares, or more).
- Operations should be easily realizable in hardware.
- Previous such languages: binary and Multi-Valued Cube Calculus (Dietmeyer), Decision Tables, Rough Sets (Pawlak), Rough Partitions (Luba), Labeled Rough Partitions (Grygiel), Binary and Multi-Valued Decision Diagrams.
- Tabular representation of data (Codd).

LOGIC PATTERNS IN TABLES

- Each row can be thought of as a record from a data base, or their set, or a collection of image features after image preprocessing.
- *Standard don't cares* versus generalized don't cares .
- Comparing rows and columns of such table can be done partially in parallel and can serve to nd patterns in data.
- Patterns used to:
  - generate prime implicants,
  - decompose function,
  - find a "bound set" or "free set" of variables for decomposition,
  - remove redundant variables,
  - find essential variables,
  - find essential implicants, etc.

LOGIC PATTERNS IN TABLES (2)

- Finding and analyzing patterns in such tables is a subject of Rough Sets Theory (Pawlak), Logic Synthesis, and Data Base Theory (Codd).
- Many similar or competing algorithms for the same task have been developed in these areas independently for the tabular data model.
- Basic **operations of algorithms**
  - **remove** rows or columns,
  - **copy and modify** them,
  - **merge** rows or columns, etc.

Induction of Logic Formulas from Examples

- **Multi-Valued multi-output (combinational) relation in tabular form**

	X1	X2	Y1	Y2
a	0,2	1	-	2
b	0,1	0	0,2	1
c	2	0	1,2	0
d	1	1	1,2	2

Record from data base

Input 1 input2 output1 output2

TWO- AND ONE-DIMENSIONAL DATA MODELS

- Two-dimensional model can be **regularly partitioned** to smaller parts.
- Smaller tables can be extracted as ***scannable windows*** in the big table, similarly as it is done in convolution-based DSP algorithms.
- Two-dimensional representations are **partitioned to one-dimensional representations.**
- This can be done vertically or horizontally.
- The main advantage of one-dimensional representations is that they can be efficiently processed in one-dimensional cellular automata, systolic, ping-pong, SIMD and pipelined hardware architectures.
- Regularity is the key to success in "Array of FPGAs" environment
- where routing long connections is the main design bottleneck.
- Regular Automata = generalization of Cellular Automata.

Regular automata

- This environment is similar to Cellular Automata (CA), but does not require all automata to be the same or to be entirely regularly connected.
- More flexibility exists thus in this model than in the CAs, which are a **very restricted design environment.**
- One-dimensional representations resemble **chromosomes** in Genetic Algorithms, which allows to use them in evolutionary computations.

TWO-DIMENSIONAL REPRESENTATIONS COMPOSED FROM ONE-DIMENSIONAL REPRESENTATIONS

- Standard Binary Cube Calculus (CC) - Roth, Karp, Dietmeyer.
- Multi-Valued Cube Calculus (MVCC) - Su, Brayton, Sasao, Perkowski.
- Generalized MV Cube Calculus (GMVCC) - Perkowski (Sendai92).
- Simplied Binary Cube Calculus (SBCC) - Decomposition Machine, Perkowski/Mattson.
- Simplied MV Cube Calculus (SMVCC) - Pawlak, Michalski, Codd - Rough Set Machine, Muraszkiewicz/Rybinski, Lewis/Perkowski.
- Spectral Representations. Reed-Muller (FPRM and GRM), Walsh, Linearly-Independent, Falkowski/Perkowski.
- Rough Partitions (Luba) and Labeled Rough Partitions
- (Perkowski/Grygiel).

Why we do not like the Genetic Algorithm?

- Our criticism about using GA for digital circuits
- No explanation is given by a GA
- It is very slow comparing to any EDA tool
- It makes no use of human knowledge and years of accumulated engineering/research experience
- Practically, it is not convergent
- It is difficult to control
- We do not know of any "real" success story of using GA in digital design

Logic Synthesis Approach to Learning

- Synthesis and Decision problems reduced to NP-hard combinational problems
- Combinational problems reduced to simple combinational problems:
  - such as graph coloring,
  - set covering, binate covering,
  - clique partitioning, satisfiability
  - or multi-valued relation/function manipulation

Logic Synthesis Approach to Learning

- Phase of learning (construction, synthesis)
- Phase of acting (function evaluation, state machine operation)
- You cannot redesign standard computer hardware when it cannot solve the problem correctly.
- The Learning Hardware redesigns itself using new learning examples given to it
- Michie makes distinction between black-box and knowledge-oriented learning systems
- Concepts of "weak" and "strong" criteria
- "The system satisfies a weak criterium if it uses data to generate an updated basis for improved performance on subsequent data" (Neural, Genetic)

Strong Criterium for Learning

- A strong criterium is satisfied if the system communicates in symbolic form concepts that it learned
- Constructive Induction (Michalski), Rough Set Theory (Pawlak), Decision Trees (Quinlan), Decision Diagrams, Disjunctive Normal Forms.
- Occam’s Razor Principle
- Learning on symbolic level is the first main point of our approach, learning on the level of logic gates is the second
- Our approach is based on decomposition of relations and functions and on synthesis of non-deterministic machines from declarative specifications
- “Do-not-knows” become “don’t-cares” for logic synthesis

Decomposition and Constructive Induction

- The high quality of decompositional techniques in Machine Learning, Data Mining and Knowledge Discovery areas was demonstrated by several authors; Ross (Wright Labs),Bohanec, Bratko/Zupan, Perkowski/Grygiel,Perkowski/Luba/Sadowska, Jozwiak, Luba, Goldman, Axtel.
- Small learning errors. Natural problem representation
- We compared the same problems using several methods: decomposition, decision trees, neural nets, and genetic algorithms
- Decomposition is clearly the winner but it is slow because the NP-complete problem of graph creation and coloring is repeated very many times.

Extrinsic Versus Intrinsic Approaches to Evolvable Hardware

- Thus, a special hardware is needed
- Our work goes in two directions: software and hardware
- In software approach, we developed several efficient algorithms for state machine design and relation/function decomposition and realization
- Our methods are tuned to data represented as relations (many “don’t-knows”)
- In hardware approach, we developed an accelerator for basic operations

Evolvable Hardware <ins>versus</ins> Learning Hardware

Technology:

The FPGAs

The Technology

How is Machine Learning Hardware Realized?

• FPGA’s provide a mass scale reconfigurable, relatively inexpensive, and widely available technology.

• The work of learning and reconfiguration is mainly at the software level and the result is data which defines the new configuration (or programming) of the FPGA.

• However, more steps are being take to place vital optimization routines such as decomposition and satisfiability theory into hardware.

•And who knows… Maybe with advances in technology, we’ll be able to dedicate part of a highly programmable chip with the programmer itself (in effect, the software is downloaded on to the same chip that programs itself).

The Process of Learning in Hardware

THE PROCESS OF LEARNING IN HARDWARE WITH FPGAs.

•Seven steps are used to perform an iteration of Machine Learning.

•The first step performs all the essential “learning” and the next 6 steps are for minimization and conversion into an FPGA.

•We’ll list these steps:

•1) Based on sets of examples… we create Reactive State Machines (RSM).

- These are usually non-deterministic state machines.
- The functional block contains no temporal variables (time relations).
- The description consists of the input/output specs, initial state specs, and environment constraints.

Steps Continued:

•2) The RSM is then made deterministic and state-minimized w.r.t. a partition of the original state variables.

•3) The new machine is mapped to constrained structural resources (a.k.a Regular Automata).

- This is the basic network which is a limited layout of regular sequential structures.
- Here we check for transition graph properties between the autonomous state graph and the RSM graph such as isomorphisms.
- (Technology mapping to FSM's could be used here).

•4) The Time based MV logic expressions or RA are decomposed (utilizing some form of MV functional decomposition).

- All variables are converted to binary representations.

5) The optimized network is then mapped to FPGA complex logic blocks (CLB) and realized with standard partitioning/placement/ routing tools. Each RSM is converted to a binary image of programming switches in the FPGA.

6) The knowledge of the machine is stored in binary memory patterns representing the FPGA programming information. The hardware then undergoes a transition by switching between a number of evolved circuits, depending on rules.

7) As the network solves new problems, the new data sets and training decisions are accumulated and the learning procedure is performed again… either over the existing knowledge or "from scratch" to avoid any bias.

- You may have noticed that we mention ***temporal logic*.**
- The idea behind temporal variables is briefly discussed next….

- Hardware describes combinational logic or finite state machines
- Simplified languages to describe them
- Generalized concepts (Petri Nets, Parallel Machines, Clauses, etc)

Simplified pseudo-Natural Languages For Learning

Temporal Logic Constraints

- Extension of predicate language can be done by introducing variables that depend on discrete time. For instance, row a from the previous table can be rewritten to the new language as follows :-

  X1[0,2] (t) & X2[1] (t) ⇒ Y2[2] (t)   ----------- at the same moment.

- We can specify arbitrary regular grammars, regular expressions, sequential netlists, or state machines with mulivalued I/O by allowing previous or next ticks of time, for instance :-

  X1[0,2] (t-2) & X2[1] (t-3) ⇒ Y2[2] (t+3) -----arbitrary I/O times.

- Timed binary variables are next internally converted to standard binary variables, for instance x(t) ⇒ x, ~ x(t-1) ⇒ x’, x(t-2) ⇒ x’’, etc. Similarly, timed MV variables are internally converted to standard binary variables.

Simplified pseudo-Natural Languages For Learning

- Time dynamical data can be very hard for a machine to "learn".
  - After all, how do teach a machine data based on time or events.
- Rather than using a large data-base of input/output pairs, we add another feature to our "learning language".
- A time relational feature. Temporal variables are the same variables we would see in our data-base, except they have that added time relation.
- This is well suited for creating state machines. This way, a machine can **learn** event-driven knowledge.

Simplified pseudo-Natural Languages For Learning

Famous examples of this are:

the **Man, Wolf, Goat, and Cabbage;**

the Missionaries and Cannibals;

models of Pavlov’s behaviors,

various real-time protocols,

other problems specified by temporal logic constraints.

and **the Dining Philosophers.**

We feed the machine basic rules which it could learn from a data base, such as ***“the wolf and the goat cannot be on the same side without the man”,*** etc.

Example of Description in Language L

Acquisition of Finite State Machines from Examples

Man, Wolf, Goat and Cabbage

- **Problem Statement** :- At the beginning all of them are on the left bank of the river, and 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 man who can take only one object with him.
- All the boolean variables are first order predicates depending on discrete time.
- The following is true when in the left bank of the river and false when on the right bank as follows :-

  M(t) : man, W(t) : wolf, G(t) : goat , C(t) : cabbage.
- The symbols ~ , &, |, ⇒ , ⇔ designate complemented variable, logic AND, logic OR, logic implication (~a&b), and logic equivalence (a&b/ ~a& ~b), respectively.

… But we can also add temporal information. Such as, "if the wolf is on the left bank, it means that either the wolf stayed there or the man has brought it there from the right bank a unit of time before."

A relational language was invented that allows us to express such rules including temporal relations.

The rule: "The wolf and the goat cannot stay on the same bank without the man" can be expressed as:

(W(t) & G(t)) => M(t)

And the expression we noted at the top of the page:

W(t) => (W(t-1) | ~W(t-1)&~M(t-1)&M(t))

The '~' is the negation of the relation and in this case refers to the "left" side of the bank.

So, **M(t) is true if the man is on the right side of the bank**

and **~M(t) is true if he's on the left.**

Temporal Logic specification of "Man, Wolf, Goat and Cabbage" Problem

- All the 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.

- 1. The wolf and the goat cannot stay on the left bank and on the right bank without the man:
- `(  W(t) &  G(t) ) => M(t).`
- `( ~W(t) & ~G(t) ) => ~M(t).`

Man, Wolf, Goat and Cabbage

- The goat and the cabbag cannot stay on the left bank and on the right bank without the man:
- `(   G(t) &   C(t) ) => M(t).`
- `( ~G(t) & ~C(t) ) => ~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) =>( W(t-1) |~W(t-1)&~M(t-1)& M(t) ).`
- `~W(t) =>(~W(t-1) | W(t-1)& M(t-1)&~M(t) ).`

Induction of State Machines from Temporal Logic Constraints

4. If the goat is on the left (right) bank, it means that either the goat stayed there or the man has brought it there from the right (left) bank one unit of time before, and the same for the cabbage :-

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

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

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

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

5. Any two objects cannot be brought across the river at the same time

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

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

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

6. The man is always on the move:-

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

Food for thought

- Can we apply these ideas to factory automation?

Learning from temporal formulas?

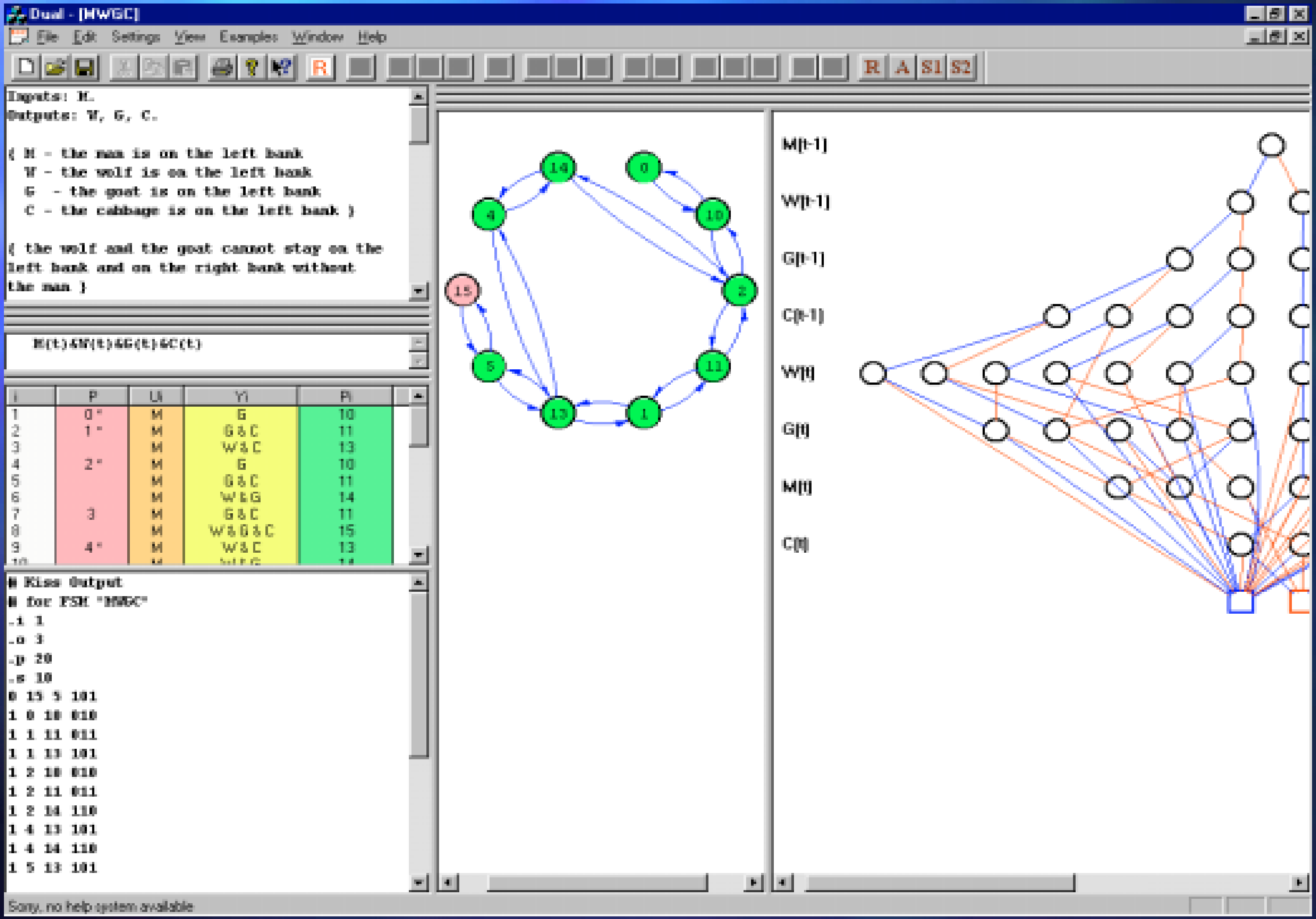
First Stage:

DUAL: Induction of State Machines from Temporal Logic Constraints

Dual: Synthesis from Declarative Specifications

- FSM specification using time constraints
- Internal data representation using BDDs
- Minimum-state FSM synthesis by construction
- Determinization and specification completion
- Verification using R-resolution (new method)

Declarative specification of temporal constraints is converted to a nondeterministic state machine



```
Inputs: M.
Outputs: W, G, C.

{ M - the man is on the left bank
  W - the wolf is on the left bank
  G  - the goat is on the left bank
  C - the cabbage is on the left bank }

{ the wolf and the goat cannot stay on the
left bank and on the right bank without
the man }
```

```
  M(t)&W(t)&G(t)&C(t)
```

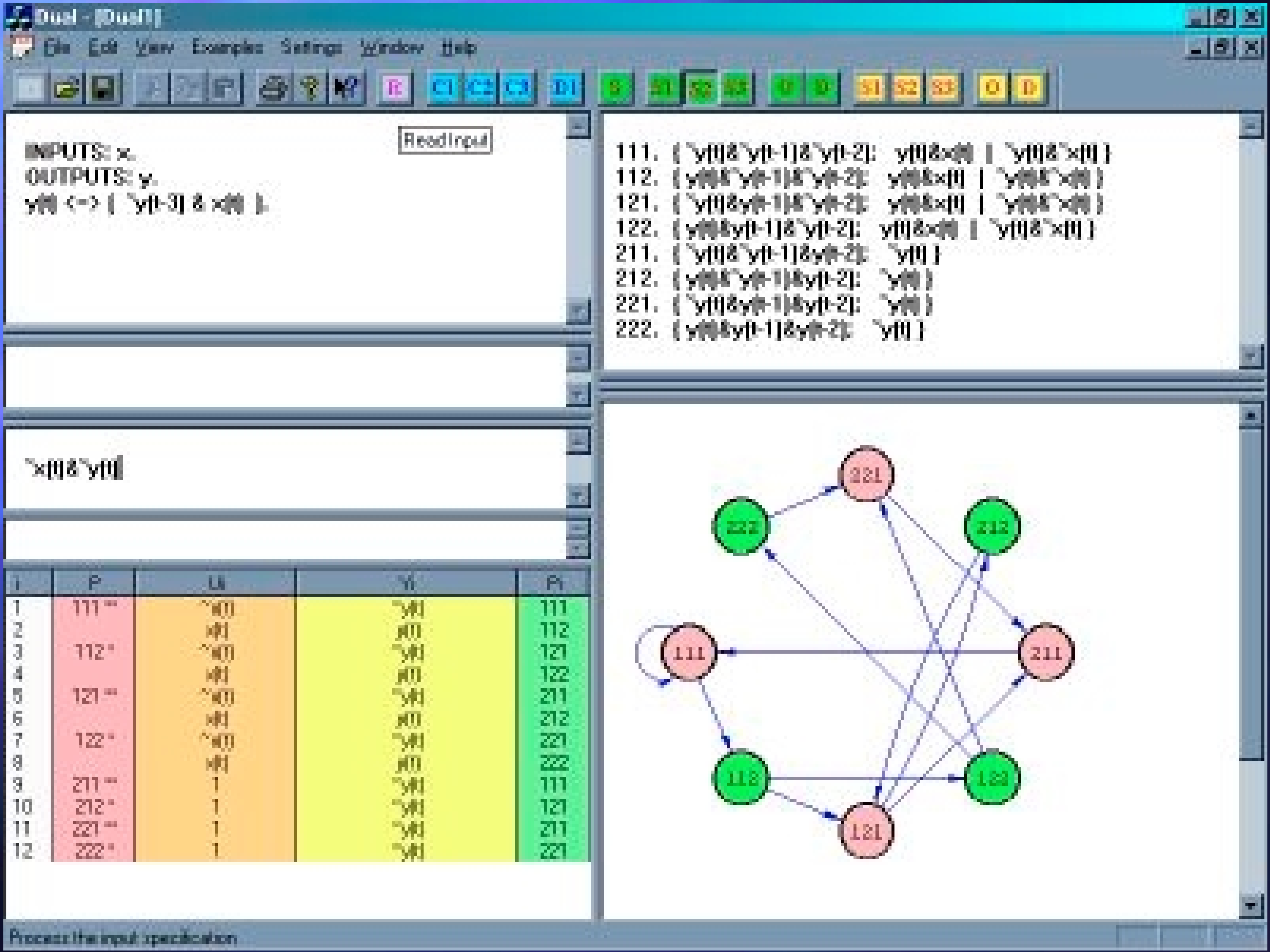
i	P	Ui	Yi	Pi
1	0 *	M	G	10
2	1 *	M	G & C	11
3		M	W & C	13
4	2 *	M	G	10
5		M	G & C	11
6		M	W & G	14
7	3	M	G & C	11
8		M	W & G & C	15
9	4 *	M	W & C	13

```
# Kiss Output
# for FSM "MWGC"
.i 1
.o 3
.p 20
.s 10
0 15 5 101
1 0 10 010
1 1 11 011
1 1 13 101
1 2 10 010
1 2 11 011
1 2 14 110
1 4 13 101
1 4 14 110
1 5 13 101
```

Sorry, no help system available

Practical Applications

- Provably-correct design of real-time (reactive) systems and protocols
- Synthesis of minimum state (non)deterministic FSMs
- Efficient translation of net-lists into minimum-state FSMs
- Conversion of regular expressions into FSMs
- FSM design for future technologies ("canonical machines")


Dual - [Dual1]
File Edit View Examples Settings Window Help
R C1 C2 C3 D1 S S1 S2 S3 U D S1 S2 S3 O D
ReadInput
222
221
212
111
211
112
122
121
Process the input specification

```
INPUTS: x.
OUTPUTS: y.
y[t] <=> ( ~y[t-3] & x[t] ).
```

```
~x[t]&~y[t]
```

```
111. { ~y[t]&~y[t-1]&~y[t-2]:   y[t]&x[t]  |  ~y[t]&~x[t] }
112. { y[t]&~y[t-1]&~y[t-2]:   y[t]&x[t]  |  ~y[t]&~x[t] }
121. { ~y[t]&y[t-1]&~y[t-2]:   y[t]&x[t]  |  ~y[t]&~x[t] }
122. { y[t]&y[t-1]&~y[t-2]:   y[t]&x[t]  |  ~y[t]&~x[t] }
211. { ~y[t]&~y[t-1]&y[t-2]:   ~y[t] }
212. { y[t]&~y[t-1]&y[t-2]:   ~y[t] }
221. { ~y[t]&y[t-1]&y[t-2]:   ~y[t] }
222. { y[t]&y[t-1]&y[t-2]:   ~y[t] }
```

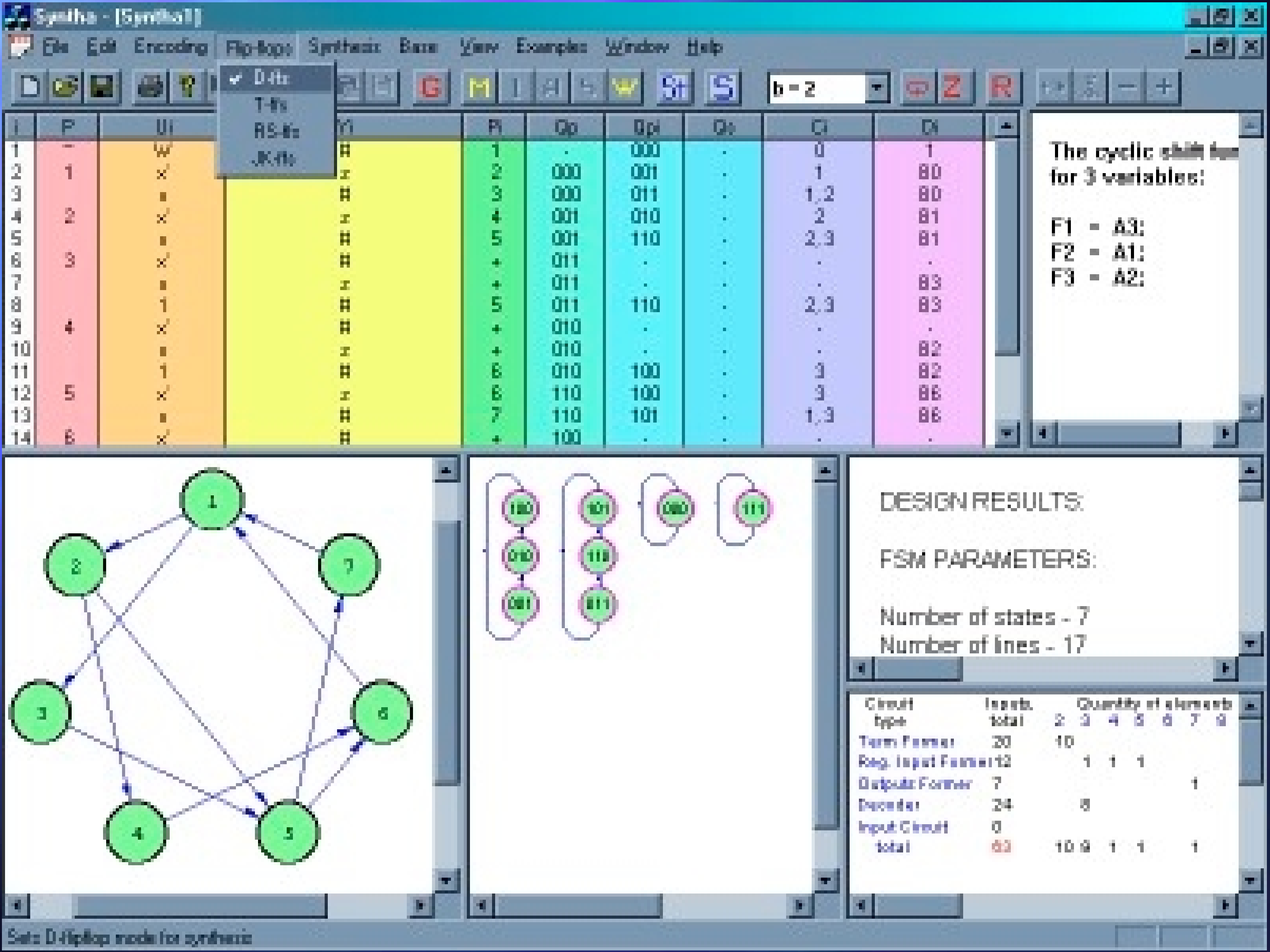
i	P	Ui	Yi	Pi
1	111 **	~x[t]	~y[t]	111
2		x[t]	y[t]	112
3	112 *	~x[t]	~y[t]	121
4		x[t]	y[t]	122
5	121 **	~x[t]	~y[t]	211
6		x[t]	y[t]	212
7	122 *	~x[t]	~y[t]	221
8		x[t]	y[t]	222
9	211 **	1	~y[t]	111
10	212 *	1	~y[t]	121
11	221 **	1	~y[t]	211
12	222 *	1	~y[t]	221

<u>Second Stage:</u>

Syntha and Trace: Optimization and Synthesis of State Machines

Syntha: Structural Synthesis of State Machines

- Table method for FSM specification
- Transparent logic synthesis algorithms
- Economic implementation of next-state logic
- Two-level boolean optimization (Espresso)
- VHDL output
- Synthesis for testability (forthcoming)
- Technology mapping (forthcoming)


Syntha - [Syntha1]
File Edit Encoding Flip-flops Synthesis Base View Examples Window Help
✓ D-ffs
T-ffs
RS-ffs
JK-ffs
b = 2

i	P	Ui	Yi	Pi	Qp	Qpi	Qo	Ci	Di
1	–	W	#	1	.	000	.	0	1
2	1	x'	z	2	000	001	.	1	80
3		s	#	3	000	011	.	1, 2	80
4	2	x'	z	4	001	010	.	2	81
5		s	#	5	001	110	.	2, 3	81
6	3	x'	#	+	011	.	.	.	.
7		s	z	+	011	.	.	.	83
8		1	#	5	011	110	.	2, 3	83
9	4	x'	#	+	010	.	.	.	.
10		s	z	+	010	.	.	.	82
11		1	#	6	010	100	.	3	82
12	5	x'	z	6	110	100	.	3	86
13		s	#	7	110	101	.	1, 3	86
14	6	x'	#	+	100	.	.	.	.

The cyclic shift fun
for 3 variables:

F1 = A3;
F2 = A1;
F3 = A2;

DESIGN RESULTS:

FSM PARAMETERS:

Number of states - 7
Number of lines - 17

Circuit type	Inputs total	Quantity of elements 2	3	4	5	6	7	8
Term Former	20	10						
Reg. Input Former	12		1	1	1			
Outputs Former	7						1	
Decoder	24		8					
Input Circuit	0							
total	63	10	9	1	1		1	

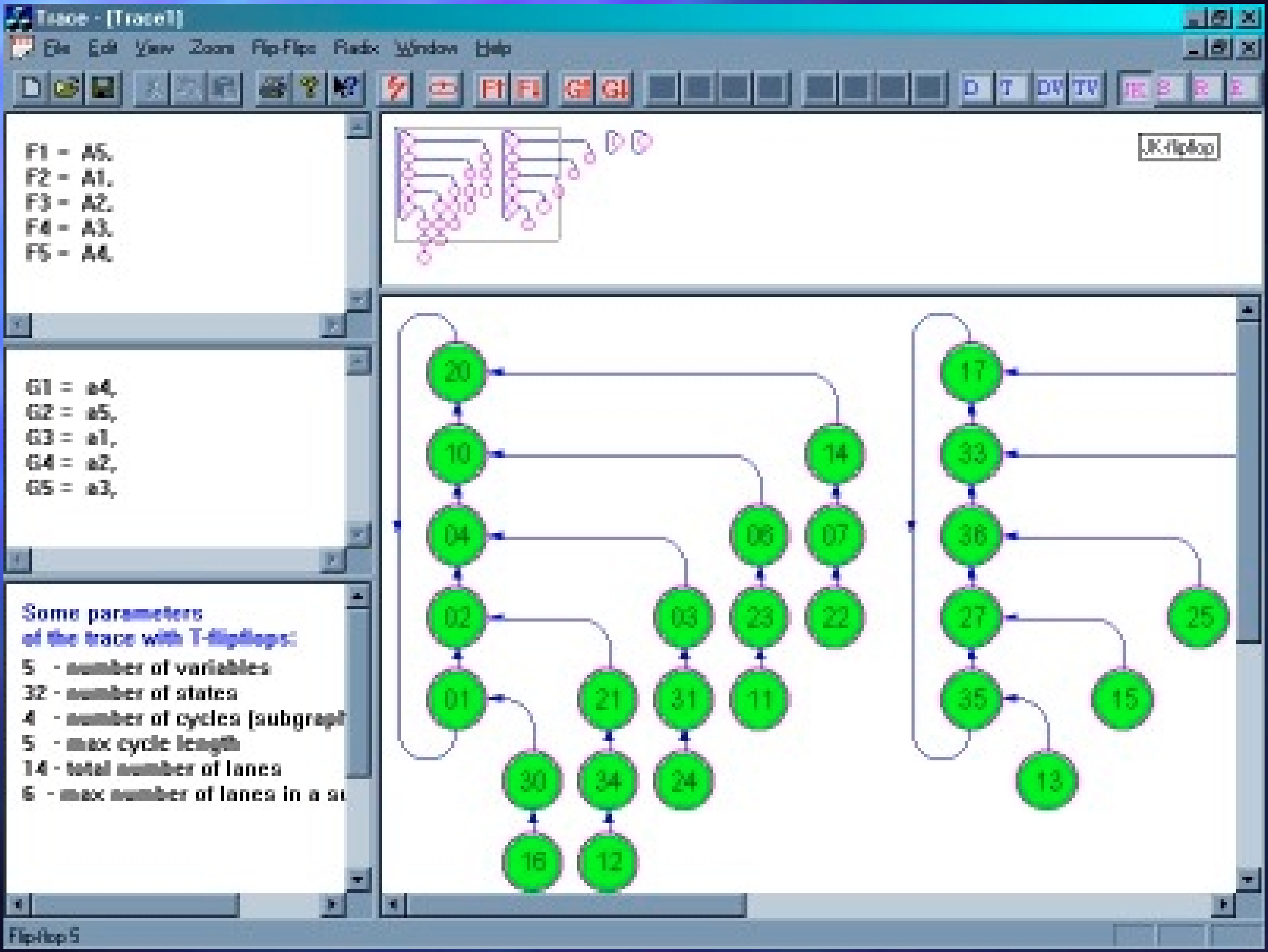
Sets D-flipflop mode for synthesis

Practical Applications

- Human-guided FSM specification and design of efficient controller
- Logic Synthesis of economical FSMs
- Building FSMs from universal blocks such as counters, shifters, EXOR modifiers
- Using various encodings and flip-flop types
- Adaptations for PLA, FPGA, ASIC
- Producing VHDL output

Trace: Optimization based on Autonomous State Machines

- New concept of “cyclic” boolean functions
- Finding “traces” of “cyclic” boolean functions for different types of flip-flops
- Properties of known and random functions
- Multi-zoom visualization using MFC



F1 = A5.
F2 = A1.
F3 = A2.
F4 = A3.
F5 = A4.

G1 = a4,
G2 = a5,
G3 = a1,
G4 = a2,
G5 = a3,

Some parameters of the trace with T-flipflops:

5 - number of variables
32 - number of states
4 - number of cycles (subgrapl
5 - max cycle length
14 - total number of lanes
6 - max number of lanes in a su

Practical Applications

- Research in boolean functions
- Synthesis of economical control units using embedded autonomous FSMs
- Design of FSMs using AND/EXOR circuits optimized for speed, area, and testability
- Case Study: Efficient reversible counters

Third Stage:

Optimization and Synthesis of Combinational Logic

MVGUD

Decomposition of Multi-Valued Multi-Output Functions and Relations

- Real-life data in ML and KDD are often relations
- Non-deterministic state machines are relations
- Functions can be decomposed to relations

Algorithms

- Decomposition strategy selection
- Variable partitioning (new algorithms)
- New Data Structure for weakly-specified functions and relations (LR-partitions)
- Column multiplicity (graph coloring)
- Decomposition evaluation using DFC
- Iterative processing of decompositions
- Verification of final results

Practical Applications

- Boolean and Multi-Valued Logic Synthesis
- Finite State Machine Decomposition
- Data Mining and Machine Learning
- Digital Image Processing
- Artificial Intelligence
- Evolutionary Algorithms

Conclusions and Work in Progress

- We define a new research area: Learning Hardware
- Realize knowledge-based combinational algorithms in reconfigurable FPGAs
- ***Generalization*** of Evolvable Hardware
- Use of the results from Machine Learning:
  - small learning error,
  - explanation,
  - no overfitting
- Use of the results from Logic Synthesis/EDA tools:
  - speed
  - and high quality of resulting circuits

Conclusions and Work in Progress

- Efficient Hardware for Combinational Problems
- Applications in Data Mining, Design Automation and Robotics
- New FPGAs from Xilinx and Altera will allow to design truly practical systems

Further Reading

- Lots of different learning techniques
  - Use dependant on application
- Introductory text
  - Artificial Intelligence: A Modern Approach by Russell and Norvig
- Research Journals
  - Machine Learning, Artificial Intelligence, Fuzzy Computing, Neural Networks, Adaptive Computing

References

- 1. Robot Learning, *by Jonalthan H. Connell and Sridhar Mahadevan*
- 2.Marek A. Perkowski, Anatoly Chebotarev, Alan Mishchenko," *Evolvable Hardware or Learning Hardware ? Induction of State Machines from Temporal Logic Constraints".*
- 3. *Marek M. Perkowski, Stanislaw Grygiel, Qihong Chen, and Dave Mattson* "Constructive Induction Machines for Data Mining".

Sources

Robert Burbidge, Computer Science, UCL.

Primal, 14th July 1999.

Steven R. Hutsell

Fall 1999

Portland State University

Marek Perkowski, Anatoly Chebotarev+,

Alan Mishchenko

Portland State University,

Department of Electrical and Computer Engineering,

Portland, OR,

+ Glushkov Institute of Cybernetics, Kiev, Ukraine

July 21, 1999, NASA/DOD Workshop

Summary

- Distinctions between learning and knowledge based methods are not clear-cut
- Representation is vital
  - (and equivalent to solving the problem!)
- Extensions should consider adaptive techniques
- No free lunch?