6

REALIZATION OF BOOLEAN FUNCTIONS USING A PULSE CODED NEURON

Marc de Savigny and Robert W. Newcomb

Electrical Engineering Department
University of Maryland, College Park MD, 20742

ABSTRACT

It is proven here that all of the sixteen Boolean functions of two variables can be realized by a pulse coded neuron, thus guaranteeing that present day digital computers could be implemented using pulse-coded neural networks. The result uses an adjustable pulse coded neuron that is realized with a neural-type cell in each of the two input branches and one in the output branch. These neural type cells allow for the adjustment of pulse repetition frequencies to obtain the appropriate coding for each function by the adjustment of internal weights. Suitable values for the weights are given in Table 2. The strictly greater than function is used to explain the weight choices along with SPICE simulation results for the exclusive Or.

INTRODUCTION

One of the frequent questions posed of researchers in other areas is "what are the capabilities of pulse-coded neural networks ?" In particular "are they as good as present day digital computers?" Since the logic of present day digital computers is primarily based upon Boolean functions, as one step in starting to answer these questions is to answer the question "can pulse-coded neural

networks realize all of the Boolean functions?" Here we show that indeed they can by presenting a pulse-coded neuron that realizes all the Boolean functions of two variables. Since we wish to consider digital logic functions [Habib 89], we concentrate on digital data coded on analog signals analogous to biological neurons processing analog data. And although we only consider two inputs to the neuron, extensions to more inputs are readily accomplished.

However, the basic motivation for a pulse coded neuron is the work of neurobiologists who have found that biological neurons pulse code the information they process [Ganong 85, p. 94]. Since it may be desired to have a neuron that mimics the pulse coded processing of the brain [Hartline 89], it has appeared reasonable to try to match a model neuron as closely as possible to what we know by way of a description of pulse handling in biological neurons [Cole 89, Murray 88]. As for engineering importance, it further appears that pulse coding can achieve a better noise-immunity when used for present day artificial neural network computation [Meador 91, Tomberg 90]. This could be important in some environments, or when the accuracy or reliability of a neural network is capital.

The main contributions of this chapter include: the presentation of a pulse coded neuron; showing how to obtain the sixteen Boolean functions of two variables with the same neuron via weights; and simulation of the neuron. As such this paper extends the results reported in [Savigny 92].

DESCRIPTION OF NEURON

Our neuron, shown in Figure 1, uses pulse coding of the information [Murray 89] to implement Boolean functions. We are interested here in implementing Boolean functions which requires an effective coding of the two logical levels "0" & "1". We will code by a neural-type pulse at repetition rate r_1 when the information is a logic "1" and repetition rate r_0, chosen to be zero, that is, a DC constant, when the information is a logic "0". In Figure 1 the inputs In_1 & In_2 can be these pulse coded signals, or, if needed, TTL-compatible logic levels. Next, we give a short description of how the neuron of Figure 1 works.

General principle

There are two inputs In_1 & In_2 with the signal on each branch being routed through three components: a pulse inverter, a Neural Type Cell (NTC) [El-Leithy 88], and a lowpass filter. The pulse inverter, which is controlled by the weight w_{is}, i=1,2, allows us to invert the logic level of the input. Not all of the 16 Boolean functions could be realized without this inversion. The NTC is our pulse generating element coding the logic level passed from the pulse

inverter. Finally, the lowpass filter (LPF) will let pass through the fundamental and possibly the second harmonic of the pulse repetition frequency of the NTC pulses. The choice is made through a weight w_{if}.

A key element of the neuron is the multiplier that takes the product of the filtered signals x_i, i=1,2, yielding $y=k\cdot x_1\cdot x_2$. Depending on which harmonics are present in the x_i's, y has certain peaks in its spectrum. We select some of these peaks by a bandpass filter (BPF) which gives the signal z. The weight w_{3f} sets the center frequency of the passband of this filter.

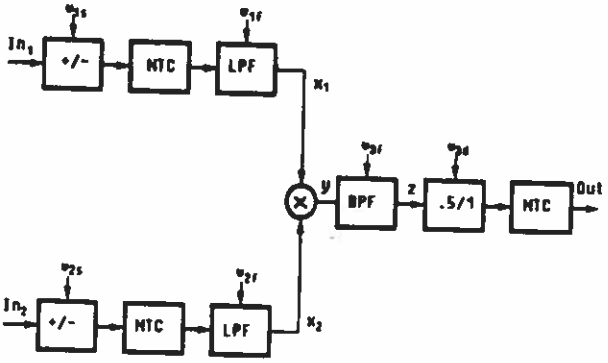

Figure 1. Block diagram of the pulse-coded neuron.

Unfortunately, we may end up with signals that have a repetition rate double that of an input pulse's repetition rate. For Boolean functions, this is not acceptable if we want to cascade our neuron, in which case, the output of a neuron is then the input of a second one, and if a change in the output repetition rate occurs, every stage would need to be optimized for a different rate. Therefore, we design a frequency divider to divide by two (.5/1 in Figure 1). The weight w_{3d} on the divider of Figure 1 permits selection of the presence of frequency division or not.

Lastly, we need an output NTC to regenerate the shape of the pulses, and therefore guarantee that the output signal of the neuron will be standardized over the different processing elements in a network regardless of what Boolean function each individual neuron processes.

In the following parts we study how to code information on the NTC pulses and show how we can use the spectrum of the internal signals x_i's, y, and z to get the Boolean functions.

Choice of Repetition Rates

The repetition rate r_1 representing a "1" is chosen in the higher range of

the frequencies that the NTC can generate in order to further increase the difference between "0" and "1". Because the neural-type pulses are not sinusoidal, the pulse signal for a logic "1" is seen in the frequency domain as made of harmonics denoted $n \cdot f_1$, where n is an integer. We have $r_1=f_1$. If we had chosen r_0 different from 0, it would have been harder to design efficient filters, since r_1 and r_0 would be given by the same NTC which has a somewhat restricted range. In other words, r_1 would have been closer to r_0 and we would have need for much better filters. Besides, signals at 0 frequency are relatively easy to use.

Role of the frequency in separating cases

The signal generated by the NTC is far from being sinusoidal [Tsay 91, p. 2]. In fact, the fundamental of r_1 is very strong if the NTC's input voltage, V_{in} is close to $V_{in.low}$ or $V_{in.high}$ which are the limits of the range of the values of V_{in} for which the NTC generates pulses. [Savigny 90, p. 38] For r_1 that is chosen in the upper region of the repetition rates generated by the NTC, the second harmonic contains up to 40% of the energy of the fundamental.

Since others have much less energy, we consider only the first two harmonics of the neural-type signals. Therefore, X_i, the Fourier transform of the signal on branch i just before the multiplier, has peaks at $\pm f_1$, $\pm 2 \cdot f_1$, in which case, Y, the signal after the multiplier, has peaks at $\pm f1$, $\pm 2 \cdot f1$, $\pm 3 \cdot f1$, and $\pm 4 \cdot f1$. This is because, if, with $\delta(\cdot)$, the Dirac function, and $\Im(\cdot)$, the Fourier transform,for i=1,2,

$$\Im(x_i(t)) = X_i(f) = A_i + \frac{B_i}{2} \cdot \left[\delta(f - f_1) + \delta(f + f_1)\right] + \frac{C_i}{2} \cdot \left[\delta(f - 2f_1) + \delta(f + 2f_1)\right] \quad (1)$$

where A_i, B_i, C_i are strictly positive real constant unless otherwise specified. Then, with * denoting the convolution operator,

$$Y(f) = X_1(f) * X_1(f) = \Im\left[x_1(t) \cdot x_2(t)\right] = \frac{2A_1A_2 + B_1B_2 + C_1C_2}{2} \cdot \delta(f) + \frac{A_1B_2 + A_2B_1 + B_1C_2 + B_2C_1}{2} \cdot \left[\delta(f - f_1) + \delta(f + f_1)\right] \quad (2)$$

$$+\frac{2A_1C_2+2A_2C_1+B_1B_2}{4}\cdot\left[\delta(f-2f_1)+\delta(f+2f_1)\right]$$

$$+\frac{B_1C_2+B_2C_1}{2}\cdot\left[\delta(f-3f_1)+\delta(f+3f_1)\right]$$

$$+\frac{C_1C_2}{4}\cdot\left[\delta(f-4f_1)+\delta(f+4f_1)\right] \quad .$$

We summarize this in Table 1 recognizing that a logic "0" is represented by a DC constant, and a "1" by a signal having harmonics at f_1 and $2\cdot f_1$. The table gives the peaks in Y for the four logic combinations of the inputs. By choosing between these peaks, we create the different functions.

	X1	logic "0"	logic "1"	
X_2		0	f_1	$2f_2$
logic"0"	f_2	0	f_1	$2f_2$
logic"1"	f_1	f_1	$0,2f_1$	$f_1,3f_1$
	$2f_1$	$2f_1$	$f_1,3f_1$	$0,4f_1$

Table 1. Peaks in the spectrum Y of the product signal for different pulse coded logic inputs. A logic "0" at the i-th input of the neuron is transformed into a constant X_i, on the other hand, a logic "1" at the i-th input is transformed into pulses at repetition rate r_1, the upper harmonics are lowpass filtered out and the resulting signal X_i contains the first, or the first and second harmonics of r_1.

Role of the filters

Changing the cutoff frequency of the lowpass filters will suppress some frequencies in x_1 or x_2 and suppress certain harmonics of r_1 in y. This can be seen in Table 1. Let us study an example: suppose we filter out the harmonic $2\cdot f_1$ of the pulses from the first branch (C_1=0 in 2). Practically, this corresponds to disregarding the column of Table 1 labeled $2\cdot f_1$. The combination "01" (B_1=C_1=0 in 2) on the inputs can be differentiated from "10" (C_1=B_2=C_2=0 in 2) - where the left bit is the logic state of the first branch, similarly for the second - since peaks at $\pm 2\cdot f_1$ are present in y when the inputs are "01", but are not present in Y for "10".

The pulse inverter

It turns out that not all the Boolean functions of two variables can be

realized with filters and a multiplier. To achieve this we add the pulse inverter on the input of the neuron [Savigny 91, p. 46]. Its purpose is to take the incoming signal on its input and invert its logic level. Therefore, if a logic "0" is present on the input of the pulse inverter, the pulse inverter transforms it into a logic "1" and vice versa. That is, if the pulse inverter is selected the pulse repetition rate at the input to the pulse inverter is changed to the "inverse" pulse repetition rate at its output.

We will see in the next section that some functions (out of the sixteen, exactly 6, including and, or, xor, 0) do not need to invert their inputs, while others do (for example, nor, nand). This has the consequence that we need the capability to enable or disable the pulse inverters depending on what function we want to implement.

THE DIFFERENT BOOLEAN FUNCTIONS

We discussed in a previous section that changing the filters and enabling or disabling the pulse inverters and frequency divider in the neuron of Figure 1 results in changing the processing of information in the neuron. We show in this section that all of the sixteen Boolean functions of two variables can be achieved by such changes.

We will call "weight" these quantities that are changed to obtain different functions. This is motivated by the terminology being used for neural networks [Dayhoff 90], where changing the weights modifies the processing. Table 2 lists the weights that are needed to obtain the different Boolean functions.

In the left column of Table 2, we use the notation Bh to label the Boolean function: B refers to "Boolean," and the number h in hexadecimal is a reminder of the output; just translate the hexadecimal number in binary and you have the output of the function. For example, for the four combinations of inputs listed at the top of Table 2, xor has the outputs 0 1 1 0, which is 6 in hexadecimal, so it is labeled B6. Also in the table, the sign weight w_{is}, i=1,2 takes on the value "-" if the pulse inverter is enabled, "+" if it is not. Similarly, the divider weight w_{3d} is "Y" when the frequency divider is selected, "N" when it is not required. The weights on the filters are labeled w_{if}, i=1,2,3, and are the cutoff frequencies for which we abbreviate the cutoff frequency by n, when an actual cutoff frequency of $n \cdot f_1$ is necessary. In the output columns, we denote zero for a pulsed signal with zero frequency representing a logic "0," and f_1 for pulses at the frequency which denotes a logic "1." In the next section, we show an example to illustrate the choice of weights to be taken from Table 2.

SIMULATION RESULTS

In order to check the validity of the theory, we first present the function

add the pulse inverter
urpose is to take the
herefore, if a logic "0"
erter transforms it into
r is selected the pulse
nged to the "inverse"

ns (out of the sixteen,
ert their inputs, while
ience that we need the
ding on what function

he filters and enabling
the neuron of Figure 1
e neuron. We show in
two variables can be

ged to obtain different
ed for neural networks
ie processing. Table 2
oolean functions.
h to label the Boolean
adecimal is a reminder
inary and you have the
iations of inputs listed
s 6 in hexadecimal, so
=1,2 takes on the value
irly, the divider weight
vhen it is not required.
, and are the cutoff
y by n, when an actual
ins, we denote zero for
"0," and f_1 for pulses
t section, we show an
om Table 2.

st present the function

B2 as an example of how the weights of Table 2 apply. Then we implement the pulse coded neuron using SPICE for the xor (B6) function.

	Inputs			
In_1	0	0	1	1
In_2	0	1	0	1

	logic output				Name	w_{1s}	w_{1f}	w_{2s}	w_{2f}	w_{3f}	w_{3d}	Output			
B0	0	0	0	0	0	+	1	+	1	>4	N	0	0	0	0
B1	0	0	0	1	and	+	1	+	1	2	Y	0	0	0	f_1
B2	0	0	1	0	sgt	+	1	-	1	2	Y	0	0	f_1	0
B3	0	0	1	1	In_1	+	1	+	0	1	N	0	0	f_1	f_1
B4	0	1	0	0	sst	-	1	+	1	2	Y	0	f_1	0	0
B5	0	1	0	1	In_2	+	0	+	1	1	N	0	f_1	0	f_1
B6	0	1	1	0	xor	+	1	+	1	1	N	0	f_1	f_1	0
B7	0	1	1	1	or	+	1	+	2	1	N	0	f_1	f_1	f_1
B8	1	0	0	0	nor	-	1	-	1	2	Y	f_1	0	0	0
B9	1	0	0	1	nxor	+	1	-	1	1	N	f_1	0	0	f_1
BA	1	0	1	0	nIn_2	+	0	-	1	1	N	f_1	0	f_1	0
BB	1	0	1	1	goe	+	1	-	2	1	N	f_1	0	f_1	f_1
BC	1	1	0	0	nIn_1	-	1	+	0	1	N	f_1	f_1	0	0
BD	1	1	0	1	soe	-	1	+	2	1	N	f_1	f_1	0	f_1
BE	1	1	1	0	nand	-	1	-	2	1	N	f_1	f_1	f_1	0
BF	1	1	1	1	1	In_1	2	In_2	2	1	N	f_1	f_1	f_1	f_1

Table 2. Weights of the pulse-coded neuron in order to obtain the 16 Boolean functions of 2 inputs variables In_1 & In_2. w_{is}, i=1,2 is "-" if the pulse inverter is selected, "+" otherwise. $f_c=w_{if}f_m$ i=1,2 is the cutoff frequency of the lowpass filters with fm between f_1 and $2 \cdot f_1$, $f_c=w_{3f}f_1$ is the cutoff frequency of the bandpass filter. Finally, w_{3d}="Y" when frequency division is necessary, "N" else.

Verification of the weights for "sgt"

The function B2 is also called "sgt," for "strictly greater than." From Table 2, we get the values of the weights for the neuron of Figure 1. We have four cases for the input vector (In_1,In_2), namely (0,0), (0,1), (1,0), and (1,1). According to the w_{1s} & w_{2s} entries of Table 2, we invert the second branch only, which swaps the bits of that branch. Then, the NTCs transform a logic "0" to 0V, and a logic "1" to pulses at repetition rate r_1. Both input branch NTCs have pulses with harmonics at $n \cdot f_1$, where n is a positive integer. According to $w_{1f}=1$ & $w_{2f}=1$ a lowpass operation is performed on each input branch cutting off all high harmonics. After the filters, we are left with signals having most of their energy at the frequency f_1 for the logic "1," and we still have 0 for the logic "0". The next step is to perform a modulation by taking the product of the two input branch signals. The resulting signal, y, has energy around frequencies which are the sum and difference of the frequencies of the signals on each branch. From the w_{3f} column, in which $w_{3f}=2$, we take the bandpass around $2 \cdot f_1$ which creates a signal with energy at $2 \cdot f_1$ only when the input to the network is (1,0). A division of the frequency by two (w_{3d}="Y") reduces the frequencies present in the output signal to f_1 or 0 (recall that the logic "0" which is represented by a DC constant is not affected by the frequency division). The logic equivalent of the obtained signal is "0" when the input is (0,0), (0,1), or (1,1) and "1" results from the input (1,0). We therefore have the desired output for the strictly greater than Boolean function B2.

We have chosen to illustrate Table 2 by this function because B2 requires all the elements of the neuron, so if the proof is understood for this function, it should be straightforward to write the proofs for the 15 other functions. Indeed, given an understanding of how to prove the result for B2, it is easy to prove all the results of Table 2; consequently, except for the comments on "xor" which follow, the details of proving the validity of the other entries is omitted, though they can be found in detail in [Savigny 91].

Xor

The point of this section is to realize an exclusive or (xor) using the neuron of Figure 1 with the weights of Table 2 for B6. An MOS circuit of the neuron using MOSIS parameters was simulated via PSPICE.

Figure 2 gives the signals at the outputs of the NTCs on each branch of the neuron and the output. We check that the output (lower curve) is a logic "1" (i.e. pulses, possibly after a delay) if and only if exactly one input is "1"; note that a "0" or "1" in In_i is accurately represented by the pulses displayed in the two upper curves.

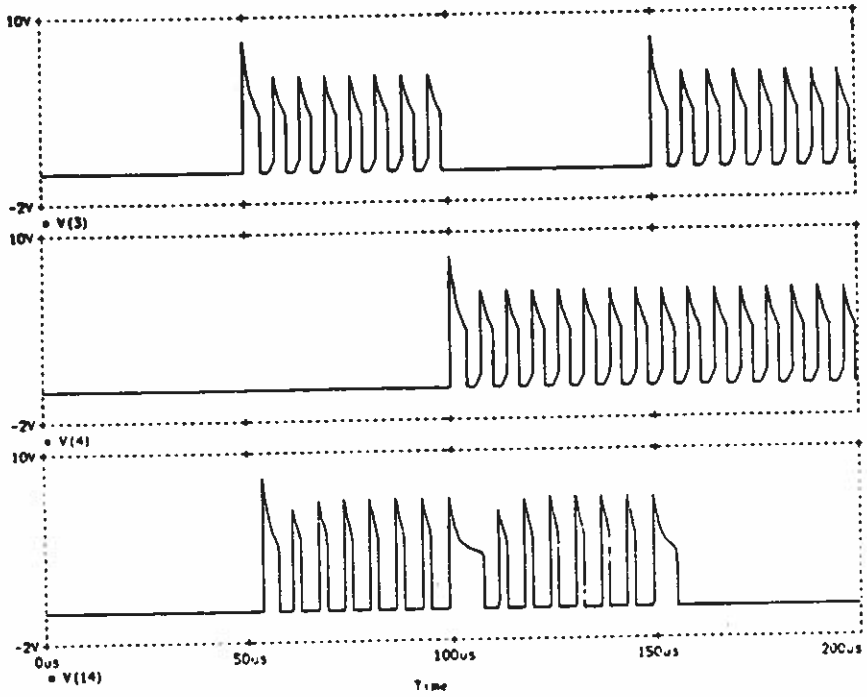

Figure 2. SPICE analysis results for the xor (B6) function.

CONCLUSIONS

We have presented a neuron realizing the sixteen Boolean functions of two variables, noting that the same structure can implement any of these functions, if well-chosen weights are used. Simulation results are given to verify that the neuron works as claimed.

Using de Morgan's work, we know that any binary valued logic function, no matter how complicated, can be realized using the basic two-input functions discussed in this paper. This leads us to the end result that any digital computer can be realized by a pulse coded neural network. Since our pulse coded neuron might be interfaced with TTL-compatible computers, the neuron is designed to allow for TTL input signals, as well as for input pulse coded signals. If it is desired that the output should also be TTL-compatible, a simple device exists [Savigny 91] to convert the pulses to TTL voltage levels. In other words, we can replace logic gates by this neuron, as well as use it throughout a pulse coded computer.

At this point, we are researching a way to reduce the size of the electronic circuit realizing the neuron: The one presented in this paper gives an existence

proof that neurons can implement logic functions, but to get a reasonable size network, its neurons must have the smallest size possible. Thus, it would be best if the filters, frequency dividers, and pulse inverters could be dispensed with.

Finally we wish to acknowledge discussions with many of our colleagues and students as well as the research support of the AFOSR portion of ONR Grant No. N00014-90-J-1114.

REFERENCES

[Cole 89] Clint Cole, Angus Wu, & Jack Meador, "A CMOS impulse neural network," Proceedings of the Colorado Microelectronics Conference, Colorado Springs, Colorado, March 1989, pp. 678-686.[Dayhoff 90] Judith E. Dayhoff, "Neural networks architectures: an introduction," Van Nostrand Reinhold, New York, 1990.

[El-Leithy 88] Nevine El-Leithy & Robert W. Newcomb, "Hysteresis in Neural-TypeCircuits," Proceedings of the 1988 IEEE International Symposium on Circuits and Systems, Vol 2, June 1988, Helsinki, Finland, pp. 993-996.

[Ganong 85] William F. Ganong, "Review of medical physiology," Lange Medical Publications, 12th edition, Los Altos, CA, 1985.

[Geiger 90] Randall L. Geiger, Phillip E. Allen, & Noel R. Strader, "VLSIdesign techniques for analog and digital circuits," McGraw Hill Publishing Co, New York, 1990.

[Gray 84] Paul R. Gray & Robert G. Meyer, "Analysis and design of analog integrated circuits," John Wiley & Sons, New York, 2nd edition, 1984.

[Habib 89] Mahmoud K. Habib & H. Akel, "A Digital Neuron-Type Processor andIts VLSI Design," IEEE Transactions on Circuits and Systems, Vol 36, No 5, May 1989, pp. 739-746.

[Hartline 89] Daniel K. Hartline, "Simulation of restricted neural networks with reprogrammable neurons," IEEE Transactions in Circuits and Systems, Vol 36, No 5, May 1989, pp. 653-660.

[Hopfield 87] John J. Hopfield, "Artificial Neural Networks," IEEE Circuits andDevices Magazine, Vol 4, No 5, September 1988, pp. 3-10.

[Lippmann 91] Richard P. Lippmann, "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, Vol 4, No 2, April 1987, pp. 4-22.

[Meador 91] Jack L. Meador, Angus Wu, Clint Cole, Novat Nintunze, & Pichet Chintrakulchai, "Programmable Impulse Neural Circuits," IEEE Transactions on Neural Networks, Vol 2, No 1, January 1991, pp. 101-109.

[Murray 88] Alan F. Murray & Antony V. W. Smith, "Asynchronous VLSI Neural Networks Using Pulse-Stream Arithmetic," IEEE Journal of Solid-State Circuits, Vol 23, No 3, June 1988, pp. 688-697.

[Murray 89] Alan F. Murray, "Pulse Arithmetic in VLSI Neural Networks," IEEE Micro, December 1989, pp. 64-p74.

[Savigny 90] Marc de Savigny, Guy Moon, Nevine El-Leithy, M. Zaghloul, Robert W. Newcomb, "Hysteresis Turn-On-Off Voltages for a Neural-Type Cell," Proceedings of the 33rd Midwest Symposium on Circuits and Systems, Calgary, Canada, August 1990, pp. 37-40.

[Savigny 91] Marc de Savigny, "Pulse coded neuron realizing Digital functions," Master of Science Thesis, University of Maryland, 1991.

[Savigny 92] Marc de Savigny and R. W. Newcomb, "Realization of Boolean Runctions Using a Pulse Coded Neuron," Proceedings of the 1992 IEEE International Symposium on Circuits and Systems, San Diego, CA, May 1992, pp. 2228-2231.

[Tomberg 90] Jouni E. Tomberg & Kimmo K. K. Kaski, "Pulse-Density Modulation Technique in VLSI Implementations of Neural Network Algorithms," IEEE Journal of Solid-State Circuits, Vol 25, No 5, October 1990, pp. 1277-1286.

[Tsay 91] S. W. Tsay, Marc de Savigny, Nevine El-Leithy, and Robert W. Newcomb, "An All MOS Neural-Type Cell," Proceedings of the 34th Midwest Symposium on Circuits and Systems, Monterey, California, May 1991, to be published.

:RIES
CIENCE

SILICON IMPLEMENTATION OF PULSE CODED NEURAL NETWORKS

edited by

Mona E. Zaghloul
The George Washington University

Jack L. Meador
Washington State University

Robert W. Newcomb
University of Maryland

KLUWER ACADEMIC PUBLISHERS
Boston / Dordrecht / London