Experiments in BDD Applications to Algebra of Galois Fields *

Ivan Radivojević

Department of Electrical and Computer Engineering
University of California, Santa Barbara, U.S.A.

* ECE594K: *VLSI Testing and Testable Design*, Spring 1995 (Prof. K.-T. Cheng). Implemented using *Homebrew* C++ generic object library (Prof. F. Brewer's CAD research group)

BINARY DECISION DIAGRAMS

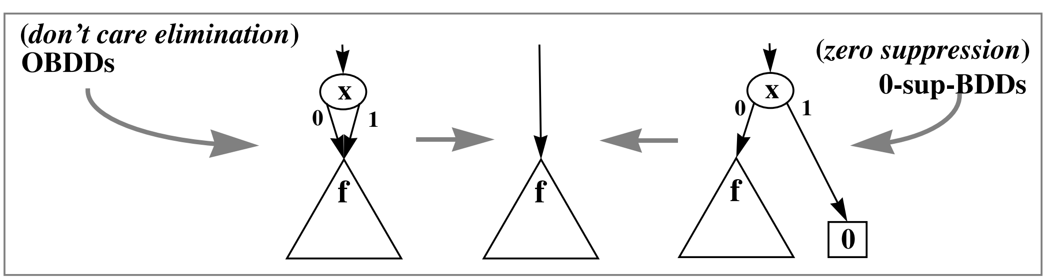
- ***OBDD*** **(Ordered Binary Decision Diagrams [**Bryant86**])**

  - *canonical* and *efficient* way to represent and manipulate Boolean functions

- ***0-sup-BDD*** **(Zero-Suppressed Binary Decision Diagrams [**Minato93**])**

  - BDDs for implicit set representation in combinatorial problems

- **Different reduction rules:**



- **Effectiveness of *0-sup-BDDs* demonstrated in scheduling applications [**Radivojevic95**]**

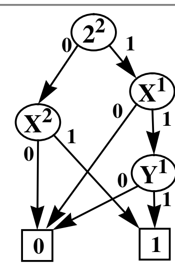
POLYNOMIAL REPRESENTATION

- **Decomposition of powers and coefficients** [Minato95a][Minato95b]**:** *

$$X^k = X^{(k_1 + k_2 + \ldots + k_m)} = X^{k_1} X^{k_2} \ldots X^{k_m}$$

$$C = 2^{c_1} + 2^{c_2} + \ldots + 2^{c_m}$$

Example: **P(x,y) =x^2+4xy**



- **Canonical form:**

  e.g. **same BDD for** $(X^2 + 4XY)$ **,** $(X(X(1 + XY) + Y(4 - X^2)))$

* To represent signed integer coefficients $(-2)^k$ encoding is used: *e.g.* (-12) = $(-2)^5+(-2)^4+(-2)^2$

POTENTIAL APPLICATIONS

- **Conventional algebra on polynomials with integer coefficients** [Minato95a][Minato95b]**:**
  - **Formal design verification**
  - **Exact computation of signal probabilities**

- **We are interested in algebra of *Galois Fields*:** *
  - **Design for testability**
  - **Coding theory**

- ***Every finite field is isomorphic to an algebra of polynomials***

- ***All finite fields of the same order are isomorphic***

* Very readable introduction to this topic can be found in [Stone73][Blahut83].

GALOIS FIELD ALGEBRA

- **Finite field $GF(2^n)$ represented by polynomials with coefficients from $GF(2)$**

$$P(x) = a_{(n-1)} x^{(n-1)} + \dots + a_1 x + a_0$$

and polynomial algebra modulo prime polynomial P(x) of order *n*. *

Example: **$GF(2^3)$, primitive polynomial: $P(x) = x^3+x+1$**

$A(x) = (x^2+x+1)$, $B(x) = x$
$A(x) + B(x) = x^2+1$
$A(x)B(x) = (x^3+x^2+x) \bmod P(x) = x^2+1$

primitive element: $\alpha = \alpha^1 = x$
$\alpha^2 = x^2$, $\alpha^3=(x+1)$, $\alpha^4=(x^2+x)$, $\alpha^5=(x^2+x+1)$, $\alpha^6=(x^2+1)$, $\alpha^7=1$

* A primitive polynomial P(x) over GF(2) is a prime polynomial over GF(2) with the property that in the extension field constructed modulo P(x), the field element represented by *x* is primitive.

EXPERIMENTS

- **Representation of random bit-vectors with different bit probabilities**

- **Computation of powers of primitive polynomials**

- **Generation of all non-zero elements of $GF(2^n)$ by computing powers of a primitive element**

- **(*i*) Computation of pseudorandom sequences $P_m(x)$ generated by primitive polynomials $P_n(x)$, where $m=2^n-2$**

  (*ii*) Computation of signatures ($P_m(x) \bmod P_n(x)$)

Representation of random bit-vectors

vector length	***compression*** [a] [nodes / vector_length]		
	p = 0.1	*p = 0.5*	*p = 0.9*
256	0.144	0.277	0.188
512	0.121	0.256	0.158
1,024	0.124	0.229	0.137
2,048	0.104	0.208	0.117
4,096	0.092	0.180	0.105
8,192	0.083	0.155	0.093
16,384	0.076	0.140	0.084
32,768	0.070	0.132	0.076
65,536	0.065	0.127	0.069
131,072	0.060	0.122	0.063
262,144	0.056	0.110	0.058
524,288	0.052	0.098	0.054
1,048,576	0.048	0.095	0.050

a. *p* -- bit probability

POWER($P_n(x)$, k) for n=[1 ... 36], k=[1 ... 1024]

$P_n(x)$	***max #nodes*** [a]	***max #terms*** [b]	***compression*** [nodes / (1024n + 1)]	***CPU time*** [c] [s]
1: 0	10	1,024	0.010	0.2
2: 1 0	29	1,365	0.014	0.7
3: 1 0	62	1,175	0.020	1.4
4: 1 0	119	2,185	0.029	1.5
5: 2 0	216	2,641	0.042	3.7
6: 1 0	377	3,119	0.061	4.5
7: 1 0	634	3,609	0.088	5.4
8: 6 5 1 0	1,019	4,111	0.124	21.8
9: 4 0	1,532	4,611	0.166	8.9
10: 3 0	2,045	5,121	0.200	11.8
20: 3 0	2,871	9,899	0.140	17.0
30: 16 15 1 0	4,280	15,389	0.139	327.8
36: 11 0	4,475	17,257	0.121	29.2
average [d]:			0.125	76.6

a. largest BDD encountered
b. largest polynomial encountered (non-zero terms)
c. cumulative CPU time to compute all 1024 powers of $P_n(x)$
d. computed for all 36 primitive polynomials listed in [Abramovici90], p.440

Powers of higher-order primitive polynomials [a]

$P_n(x)$	max #nodes [b]	max #terms [c]	compression [nodes / (1024n + 1)]	CPU time [d] [s]
40: 5 4 3 0	5,093	20,239	0.124	272.6
47: 5 0	6,152	21,233	0.128	34.7
48: 7 5 4 2 1 0	6,724	24,271	0.137	960.1
59: 6 5 4 3 1 0	8,128	29,911	0.135	907.6
60: 1 0	7,241	22,669	0.118	21.6
63: 1 0	6,062	20,853	0.094	19.9
64: 4 3 1 0	8,272	31,183	0.126	354.7
70: 5 3 1 0	8,611	34,387	0.120	507.7
80: 7 5 3 2 1 0	9,202	40,303	0.112	1156.4
90: 5 3 2 0	10,575	45,079	0.115	756.6
100: 8 7 2 0	12,516	48,693	0.122	635.8
average:			0.121	511.6

a. primitive polynomials listed in [Stone73], p.337
b. largest BDD encountered
c. largest polynomial encountered (non-zero terms)
d. cumulative CPU time to compute all 1024 powers of $P_n(x)$

Generation of all non-zero elements of GF(2^n) by computing (2^n-1) powers of a primitive element [a]

$P_n(x)$ [b]	max #nodes [c]	CPU time [s]
8: 6 5 1 0	5	0.1
9: 4 0	7	0.2
10: 3 0	9	0.5
11: 2 0	9	1.1
12: 7 4 3 0	9	2.1
13: 4 3 1 0	9	4.4
14: 12 11 1 0	9	9.6
15: 1 0	9	20.7
16: 5 3 2 0	9	38.1
17: 3 0	11	91.3
18: 7 0	13	181.6

a. primitive element: P(x)=x
b. primitive polynomial of order *n*
c. largest BDD encountered

Pseudorandom sequence generation and signature computation

$P_n(x)$	*pseudorandom sequence* [a]			*signature* [b]
	#nodes	***compression*** [nodes / (2^n - 1)]	***CPU time*** [s]	***CPU time*** [s]
8: 6 5 1 0	75	0.294	0.3	0.6
9: 4 0	132	0.258	0.5	0.8
10: 3 0	245	0.239	1.1	2.4
11: 2 0	425	0.201	2.5	6.7
12: 7 4 3 0	738	0.180	6.7	17.4
13: 4 3 1 0	1,273	0.155	16.9	47.4
14: 12 11 1 0	2,296	0.140	39.4	123.5
15: 1 0	4,331	0.132	83.2	273.4
16: 5 3 2 0	8,426	0.129	212.8	709.7

a. long division $(x^m+1)/P_n(x)$, $m=2^n-1$
b. remainder computed by long division

SUMMARY

- **Addition (bit-wise XOR), Hamming distance -- relatively cheap**

- **Compression is rather low, except for some very sparse or special polynomials (*e.g.* $(\text{Primitive}_n(x))^k$, for small values of *n*)**

- **CPU times likely to be improved by more careful implementation of BDD package (*e.g.* multiplication rather slow)**

- **Is it possible to develop more efficient division/remainder algorithms that exploit the BDD representation of polynomials?**

REFERENCES:

- **[Abramovici90]** M. Abramovici, M.A. Breuer, and A.D. Friedman, *"Digital Systems Testing and Testable Design"*, IEEE Press, 1990, revised printing.
- **[Blahut83]** R.E. Blahut, *"Theory and Practice of Error Control Codes"*, Addison Wesley Publishing Company, 1983.
- **[Bryant86]** R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Trans. Computers*, vol. c-35, no.8, pp. 677-691, Aug. 1986.
- **[Minato93]** S.-I. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems", *Proc. 30th ACM/IEEE Design Automation Conference,* pp. 272-277, 1993.
- **[Minato95a]** S.-I. Minato, "Implicit Manipulations of Polynomials Using Zero-Suppressed BDDs", *Proc. European Design and Test Conference*, pp. 449-454, 1995.
- **[Minato95b]** S.-I. Minato, "BDD-Based Manipulation of Polynomials and Its Applications", *Proc. Intl. Workshop on Logic Synthesis*, pp. 5.31-5.43, 1995.
- **[Radivojevic95]** I. Radivojević and F. Brewer, "On Applicability of Symbolic Techniques to Larger Scheduling Problems", *Proc. European Design and Test Conference*, pp. 48-53, 1995.
- **[Stone73]** H.S. Stone, *"Discrete Mathematical Structures and Their Applications"*, Science Research Associates, 1973.