Algoritma Dasar *Ordered Binary Decission Diagram* (BDD)

WENDHI YUNIARTO

Jurusan Teknik Elektronika Politeknik Negeri Pontianak,
Jalan Ahmad Yani Pontianak 78124

Abstrak: Tulisan ini memperlihatkan algoritma dasar dari OBDD untuk memverifikasi fungsi Boolean. Dengan menggunakan paket KBDD (yang dikembangkan oleh Karl Berry dari Carnegie Melon University), program kbdd berjalan di sistem UNIX (Linux). Algoritma yang digunakan mempunyai kompleksitas waktu yang sebanding terhadap ukuran graph di dalam operasinya, dan efisien sepanjang graph tidak berkembang lebih besar. Eksperimen memperlihatkan hasil dari penggunaan algoritma untuk permasalahan pada verifikasi perancangan logic.

Kata kunci: *OBDD, UNIX, Algoritma, Graph, Logic*

Kompleksitas sistem komputer, telekomunikasi, dan perangkat keras secara umum terus meningkat. Bahkan beberapa sistem meningkat secara eksponensial. Perangkat keras (*hardware*), seperti rangkaian digital dalam bentuk Integrated Circuit (IC), tidak lagi berisi beberapa gerbang logika (*logic gates*) saja, melainkan sudah mencapai jutaan *gates*. Prosesor sudah mencapai lebar data 128-bit. Di sisi perangkat lunak (*software*) kompleksitas dinyatakan dalam jumlah "*lines of codes*" (LOC). Sistem operasi Windows 2000 memiliki jumlah LOC lebih dari 30 juta. Bagaimana menguji sistem (hardware / software) yang demikian kompleks?

Ad hoc exhaustive testing, yaitu mencoba seluruh kombinasi input, tidak dapat dilakukan untuk sistem yang demikian kompleks. Pengujian sistem digital dengan 128-bit input membutuhkan 2^{128} buah kombinasi. Seluruh kombinasi ini tidak dapat dicobakan dengan metode testing konvensional dan teknologi komputer yang ada saat ini karena membutuhkan waktu yang sangat lama sekali (jutaan tahun).

Salah satu cara melakukan verifikasi pada perancangan logic adalah dengan metode OBDD. Paket KBDD (yang dikembangkan oleh Karl Berry dari Carnegie Melon University) berjalan di sistem UNIX (Linux). Untuk itu, penguasaan penggunaan UNIX (secara minimal, seperti login, logout, mengedit, dan menjalankan program) merupakan syarat untuk melakukan verifikasi.

MASUK KE SISTEM UNIX

Langkah pertama adalah masuk ke sistem UNIX dengan menggunakan user id dan password.

```
Login:
Password: *********
unix%
```

Setelah masuk ke sistem UNIX, diberikan perintah "kbdd" untuk menjalankan

program kbdd. Jika prompt "KBDD" tidak nampak berarti program kbdd tidak berada di dalam PATH. Untuk itu harus dipastikan terlebih dahulu dimana kbdd berada.

```
unix% kbdd
KBDD:
```

MENCOBA BDD

Program kbdd dapat dijalankan secara interaktif dengan mengetikkan perintah satu persatu dan mengamati hasilnya, atau dengan mekanisme batch. Dalam mekanisme batch, perintah dimasukkan ke dalam berkas kemudian dijalankan kbdd dengan standar input (stdin) diarahkan ke nama berkas tersebut.

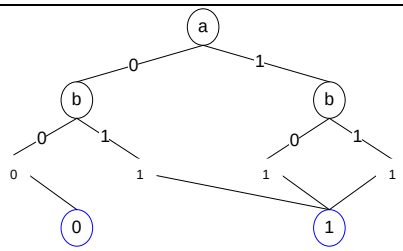
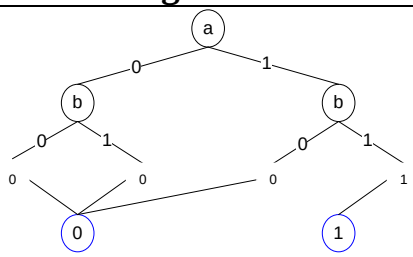
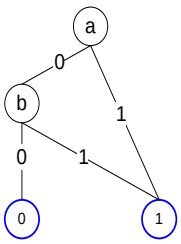
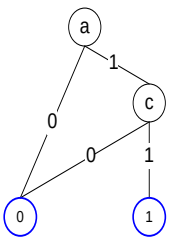
```
Unix% kbdd
bool a b c
eval f (a+b)
bdd f
eval g (a&c)
bdd g
```

Fungsi *f* dan *g* pada tampilan akan diekplorasi. Apakah gambar graph dari f dan g itu sesuai dengan teori?
eksplorasi perintah-perintah kbdd seperti:

```
sop f
satisfy f
sop g
```

Analisa tree diagram

Order : f = a < b < c

a. f = a + b	**b. g = a & c**
	
	
Dari hasil percobaan didapat : KBDD: bool a b c KBDD: eval f (a+b)	Dari hasil percobaan didapat : KBDD: eval g (a&c) KBDD: bdd g

```
KBDD: bdd f
(a:134647956
  [1]
  (b:134647860))
KBDD: sop f
 a
+ b
KBDD: sat f
Variables: a b
01
10
11
KBDD:
```

```
(a:134648068
  (c:134647892)
  [0])
KBDD: sop g
 a & c
KBDD: sat g
Variables: a c
11
KBDD:
```

Analisa f = a + b : Dari gambar dapat dilihat bahwa apabila a = 1, maka f = 1, tetapi jika a = 0 maka f = (Jika b = 1 maka, f = 1, jika b = 0 maka f = 0). Pada eksekusi tidak ditampilkan karena dianggap proses ”default”. Extended sop merupakan ”sum of product” f = a + c. Extended sat memperlihatkan hasil semua variabel yang ada, yaitu kombinasi input yang keluarannya bernilai 1. Pada percobaan f = a + b, menghasilkan kombinasi input yang keluarannya adalah (01,10,11). Jadi hasil eksekusi dengan graph pada fungsi f sesuai dengan teori.

Analisa f = a & b : Dari gambar dapat dilihat bahwa apabila a = 0 maka g = 0, tetapi jika a = 1 maka nilai g = (jika c = 1 maka g = 1, jika c = 0 maka g = 0). Extended menghasilkan *sum of product* g = a & b. Extended sat hasilnya adalah (11), Jadi hasil eksekusi dengan graph pada fungsi g sesuai dengan teori.

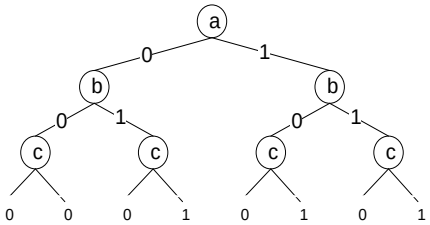
Equivalency, isomorphism

Pada percobaan Equivalency, isomorphism ini, yaitu mencoba beberapa fungsi/perintah lain dari kbdd. Perintah “verify” digunakan untuk menguji apakah dua buah fungsi merupakan equivalen, atau dengan kata lain terdapat isomorphism pada kedua graph tersebut. Perintah-perintah dari kbdd diketikkan dalam sebuah berkas (file), kemudian dijalankan kbdd dengan mengarahkan standard input (stdin) ke berkas tersebut.

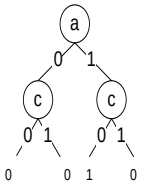
```
bool a b c
eval f1 ((a+b)&c)
eval f2 (a&!c)
eval f f1+f2
eval g (a+(b&c))
verify f g
quit
```

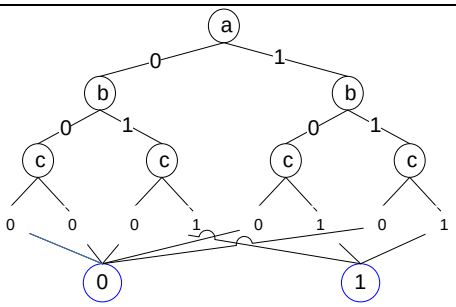
Analisa tree diagram

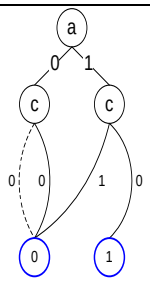
f1 = ((a + b) & c)

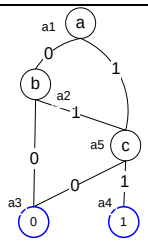


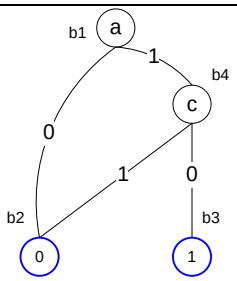
f2 = a &! c



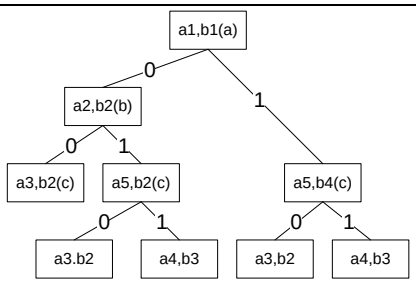


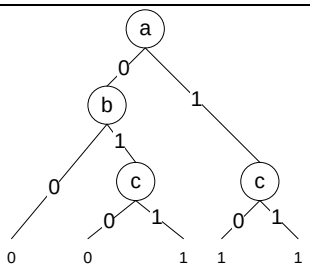


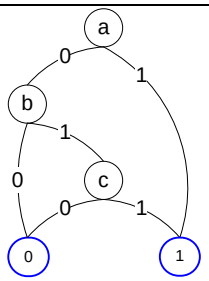




Didapat **f = f1 + f2** :







Untuk Fungsi **g = a + (b & c)**, hasilnya sama dengan f = ((a + b) & c)
Dari hasil percobaan didapat :

```
KBDD: bool a b c
KBDD: eval f1 ((a+b)&c)
```

```
KBDD: eval f2 (a&!c)
KBDD: eval f f1+f2
KBDD: eval g (a+(b&c))
KBDD: ver f g
1 verify ok     → untuk fungsi f dan g adalah sama
KBDD:
KBDD:
```

Efek dari Ordering

Melihat efek dari ordering pada sebuah fungsi. Fungsi yang akan diuji memiliki persamaan :

f = ((a1&b1) + (a2&b2) + (a3&b3) + (a4&b4))

Percobaan menggunakan ordering a1<a2<a3<a4<b1<b2<b3<b4 didapat :

```
KBDD: bool a1 a2 a3 a4 b1 b2 b3 b4
KBDD: eval f ((a1&b1) + (a2&b2) + (a3&b3) + (a4&b4))
KBDD: bdd f
(a1:134649220
  (a2:134649012
    (a3:134648820
      (a4:134648692
        (b1:134648660
          [1]
          (b2:134648628
            [1]
            (b3:134648276
              [1]
              (b4:134648052))))
        (b1:134648340
          [1]
          (b2:134648148
            [1]
            (b3:134648020))))
      (a4:134648788
        (b1:134648756
          [1]
          (b2:134648724
            [1]
            [b4:134648052]))
        (b1:134648084
          [1]
          (b2:134647988))))
    (a3:134648980
      (a4:134648884
        (b1:134648852
          [1]
          [b3:134648276])
        (b1:134648260
          [1]
          [b3:134648020]))
```

```
      (a4:134648948
        (b1:134648916
          [1]
          [b4:134648052])
        (b1:134647956))))
  (a2:134649188
    (a3:134648676
      (a4:134649044
        [b2:134648628]
        [b2:134648148])
      (a4:134649076
        [b2:134648724]
        [b2:134647988]))
    (a3:134649156
      (a4:134649124
        [b3:134648276]
        [b3:134648020])
      (a4:134648596
        [b4:134648052]
        [0])))))
KBDD: quit
```

Percobaan menggunakan ordering a1<b1<a2<b2<a3<b3<a4<b4 didapat :

```
KBDD: bool a1 b1 a2 b2 a3 b3 a4 b4
KBDD: eval f ((a1&b1) + (a2&b2) + (a3&b3) + (a4&b4))
KBDD: bdd f
(a1:134648580
  (b1:134648548
    [1]
    (a2:134648516
      (b2:134648484
        [1]
        (a3:134648452
          (b3:134648228
            [1]
            (a4:134648404
              (b4:134648052)
              [0]))
          [a4:134648404]))
      [a3:134648452]))
  [a2:134648516])
KBDD: quit
```

Analisa

Pada percobaan 1 terlihat adanya perbedaan ordering dari percobaan 2 (menggunakan persamaan yang sama), dimana pada percobaan 1 orderingnya berjauhan (a1<a2<a3<a4<b1<b2<b3<b4) sehingga hasil eksekusi BDD menjadi lebih panjang Pada percobaan 2 lebih pendek karena orderingnya berdekatan (a1<b1<a2<b2<a3<b3<a4<b4) sehingga proses eksekusi lebih cepat.

SIMPULAN

Dari Eksperimen BDD dapat dibuktikan bahwa hasil eksekusi dengan graph pada fungsi *f* dan *g* sesuai dengan teori. Pada efek dari order apabila di letakkan berjauhan, terlihat hasil eksekusi akan lebih panjang (lebih lama) dari pada order yang letaknya berdekatan. Hal ini membuktikan bahwa terjadi efek waktu eksekusi apabila nilai order diletakkan sembarangan (tidak berdekatan). OBDD adalah perangkat bantu yang sangat bermanfaat di dalam melakukan pembuktian terhadap suatu verifikasi design logic tanpa melakukan simulasi dan implementasi dari desain tersebut. OBDD (Ordered Binary Decission Diagram) merupakan alat bantu dan representasi dari sistem dalam bentuk logic tingkat tinggi yang didukung oleh *theorem proving tools.*

DAFTAR PUSTAKA

C.Y. Lee. (1959). *Representation of Switching Circuits by Binary-Decission Programs.* Bell System Technical Journal. Vol. 38 July, pp. 985-999.

F.J. Hill and G.R. Peterson. (1974). *Introduction to Switching Theory and Logical Design*. New York: Wiley.

Budi Rahardjo. (2005). *Manual OBDD*. Laboratorium Reduced Ordered Binary Decision Diagram, Teknik Elektro. Institut Teknologi Bandung.

S.B.Akers. (1978). *Binary Decission Diagrams***.** IEEE Transactions on computers. Vol. C-27 No.6, June pp. 509-516.