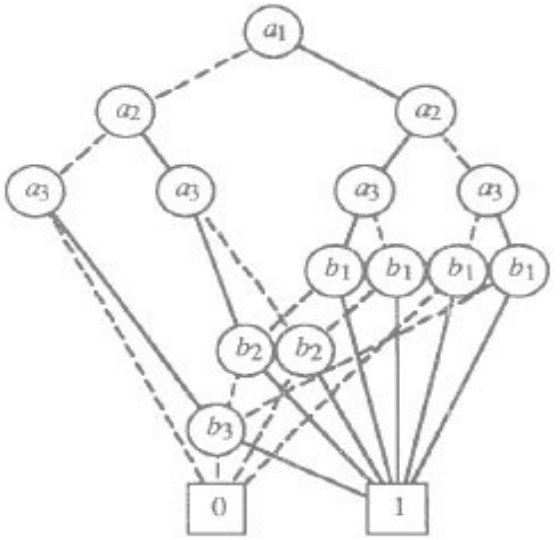
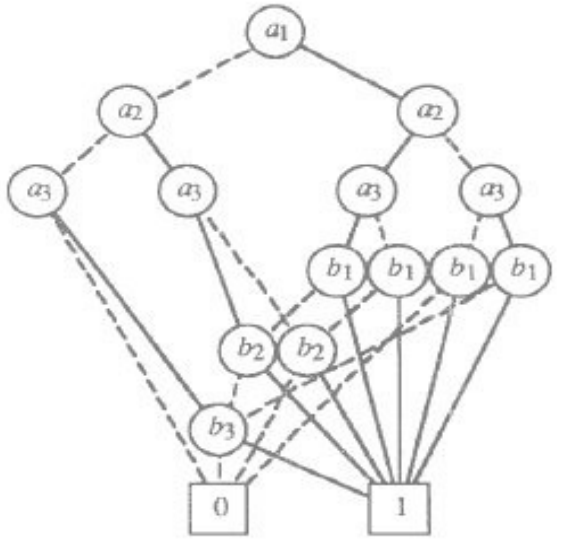
Plan prezentacji



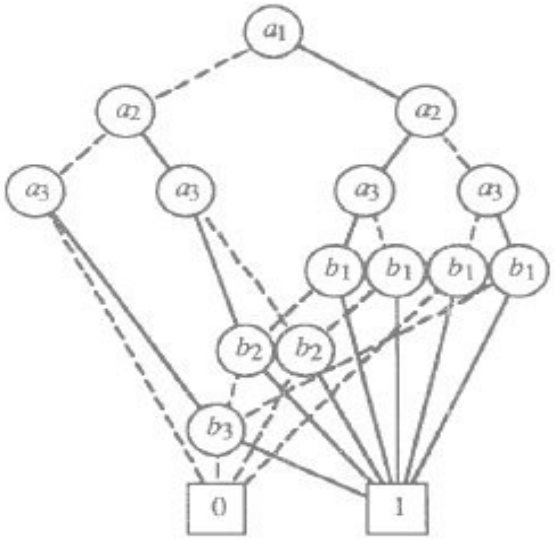
1. Model-Checking
2. BDD a OBDD
3. Analiza złożoności OBDD
4. Operacje na OBDD
5. Zastosowania
6. Możliwe modyfikacje

Model-Checking



model + formuła (teoria) + metodologia = weryfikacja

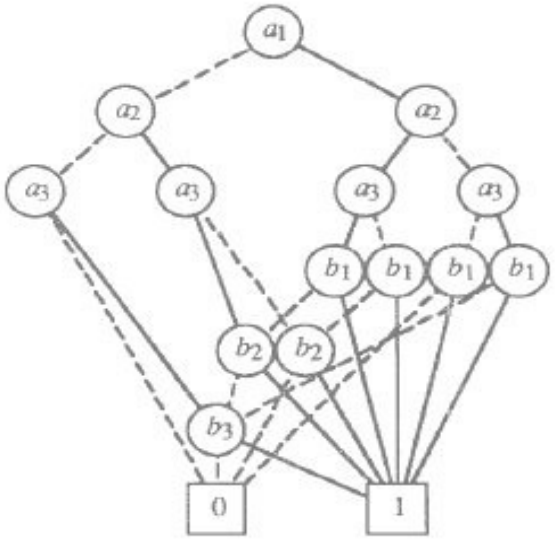
Model-Checking



dowodzenie poprawności programów

- trudne, bo program jest czymś wirtualnym;
- możemy napisać cokolwiek, przyjmować jakąś semantykę tych napisów;
- program jest kompilowany jakimś kompilatorem i uruchamiany na jakiejś maszynie

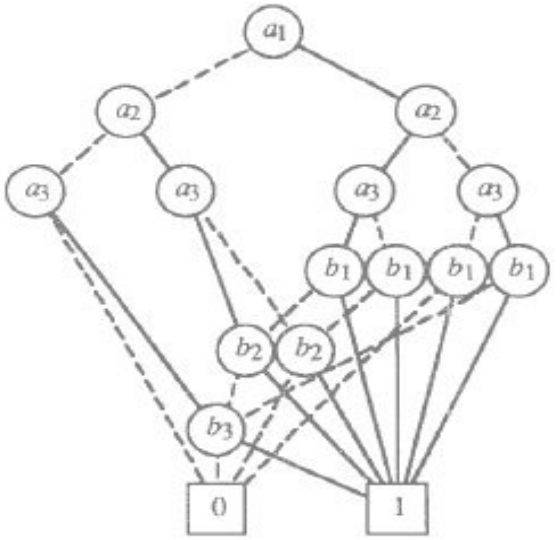
Model-Checking



weryfikacja sprzętu

- mamy prawdziwy system;
- możemy na podstawie obserwacji skonstruować model, który będzie odpowiadał systemowi;
- często nie jest wymagany duży stopień odpowiedniości;
- możemy przyjmować różne modele

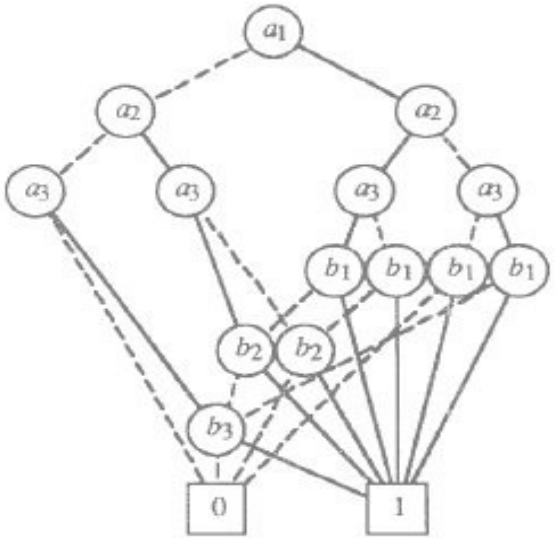
Model-Checking



CTL (Computational Tree Logic)

- logika temporalna, stosowana do weryfikacji programów współbieżnych;
- pozwala specyfikować własności bezpieczeństwa i żywotności

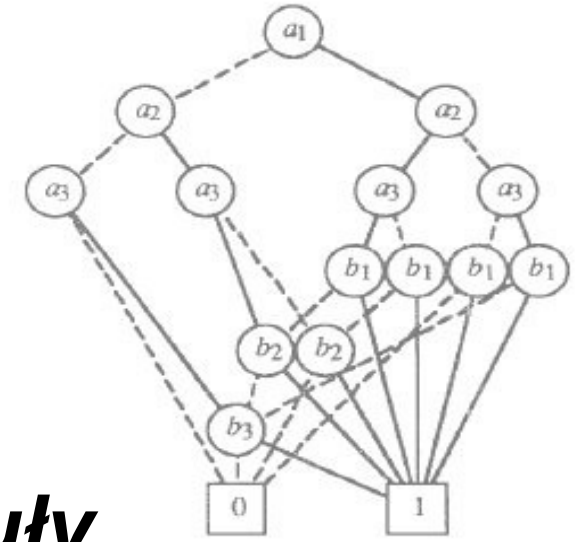
Model-Checking



CTL – składnia

- operatory logiczne, zmienne, stałe
- operatory: A, E, F, G, U, X
- operatory punktów stałych

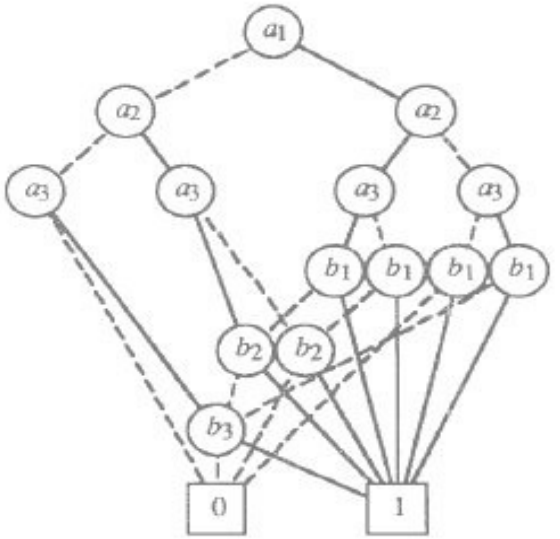
Model-Checking



CTL – przykładowe formuły

- AF g = A (true U g)
- EF g = E (true U g)
- AG f = ~E (true U ~f)
- EG f = ~A (true U ~f)
- EF p = μY.(p or EXY)
- EG p = *ν*Y.(p and EXY)
- E(q U p) = μY.(p or (q and EXY))

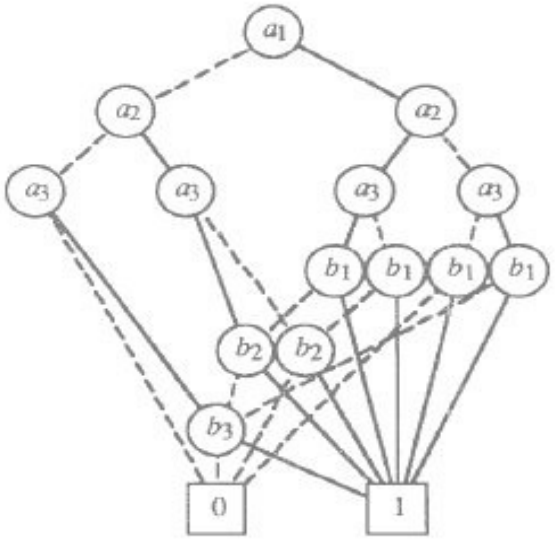
Model-Checking



Algorytm znajdowania punktów stałych

μY.*τ*[Y] {lub *ν*Y.*τ*[Y]}:
let Y = false; {or Y = true}
do
    let Y' = Y, Y = *τ*[Y]
until Y' = Y;
return Y

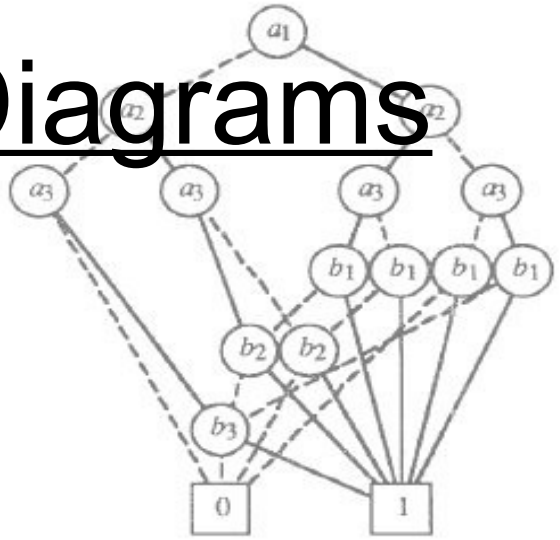
Model-Checking



Po co nam BDD?

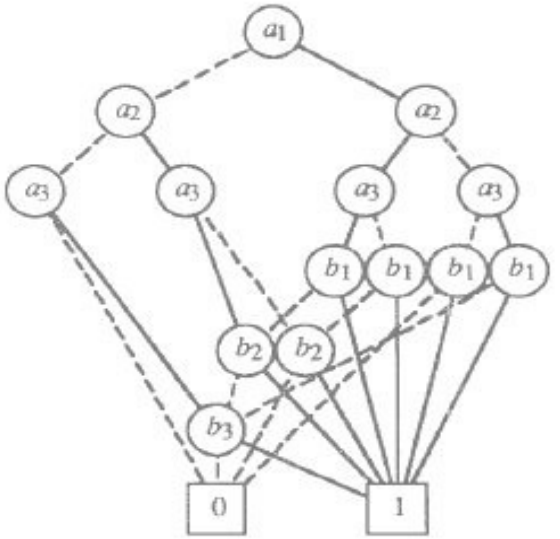
- możemy sprawdzać, czy formuła jest spełniona w danym stanie poprzez operacje na formułach boolowskich;
- potrzebujemy efektywnej metody manipulacji takimi formułami

Ordered Binary-Decision Diagrams

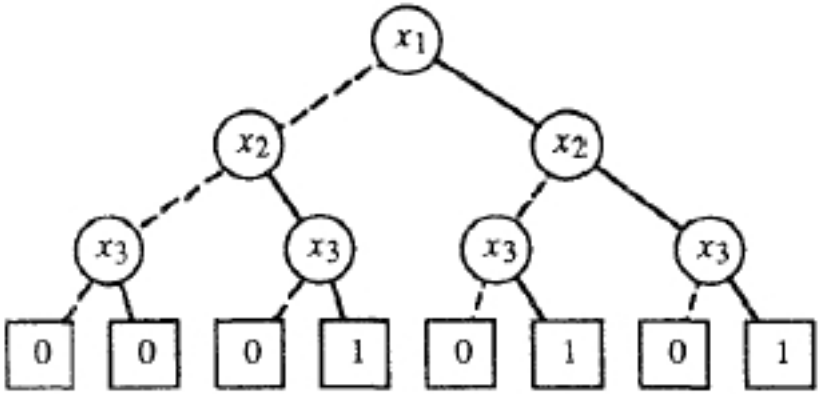


- BDD służą do reprezentacji funkcji boolowskich w postaci DAGów;
- OBDD stanowią rozszerzenie idei BDD, reprezentacja kanoniczna;
- OBDD są praktyczne, ponieważ ich rozmiar dla wielu przydatnych funkcji jest niewielki;
- stosowane głównie w: projektowaniu układów cyfrowych, weryfikacji i testowaniu;
- także przy projektowaniu systemów współbieżnych, w logice matematycznej oraz sztucznej inteligencji

OBDD

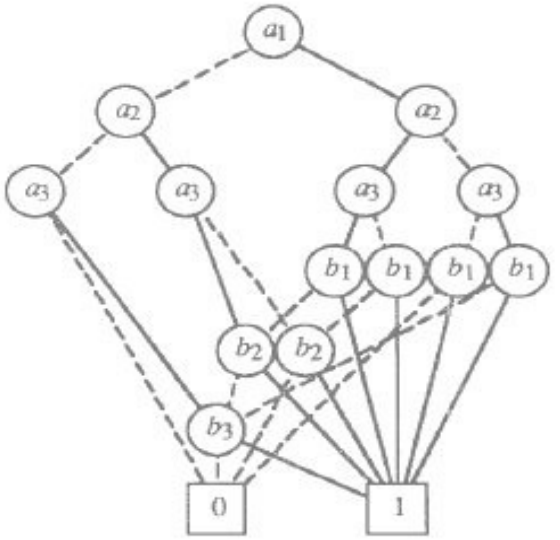


x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



przykład BDD – tabela wartości funkcji *f* oraz drzewo decyzyjne

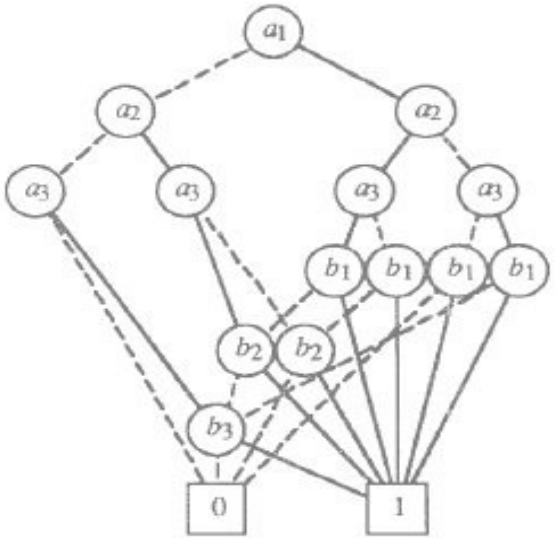
OBDD



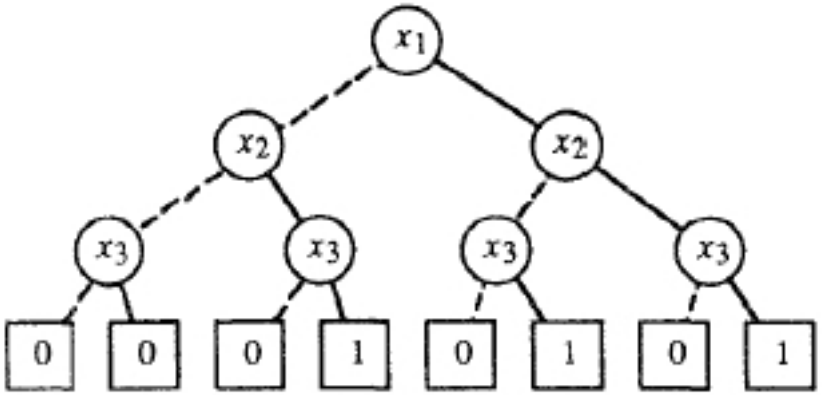
Wprowadzone restrykcje:

- wprowadzenie porządku zupełnego na zmiennych;
- usunięcie dublujących się liści;
- usunięcie dublujących się wierzchołków;
- usunięcie zbędnych wierzchołków

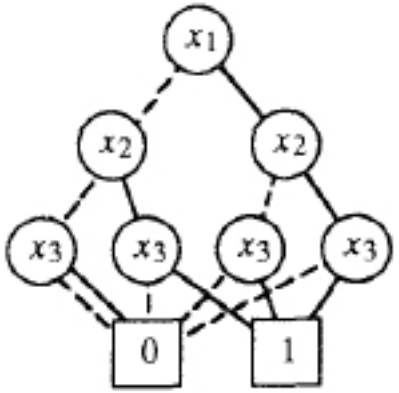
OBDD

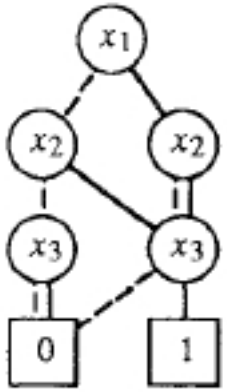


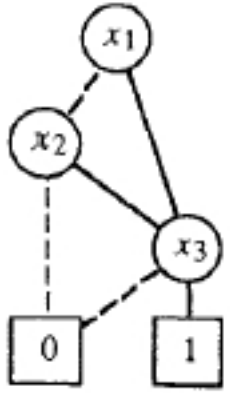
BDD – dla przypomnienia



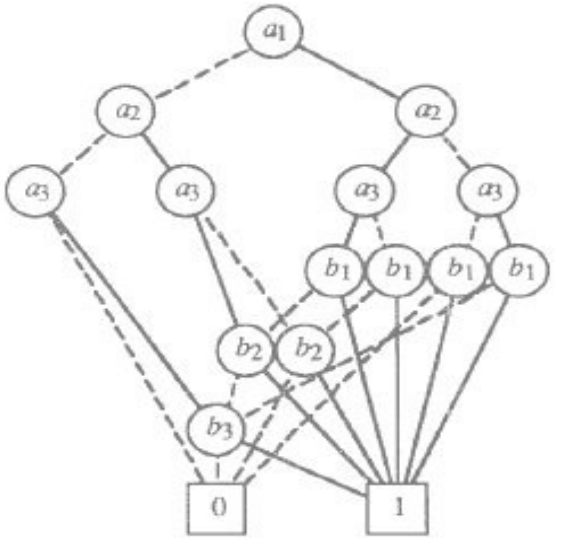
wprowadzamy kolejne restrykcje:







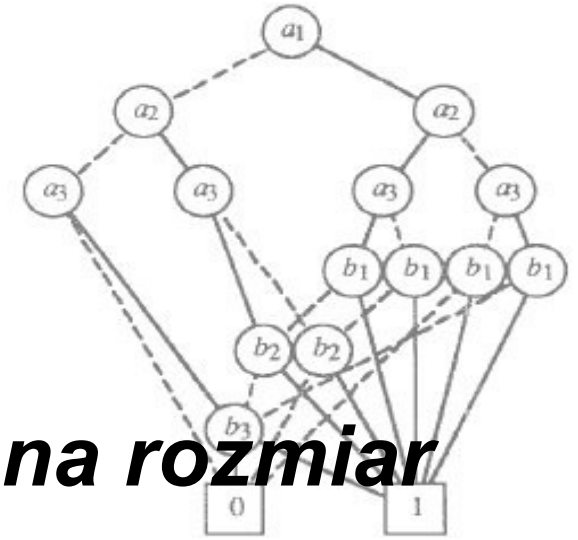
OBDD



Ważne cechy:

- taka reprezentacja jest kanoniczna;
- porządek zmiennych ma wpływ na rozmiar

OBDD



Wpływ porządku na zmiennych na rozmiar OBDD:

Niech

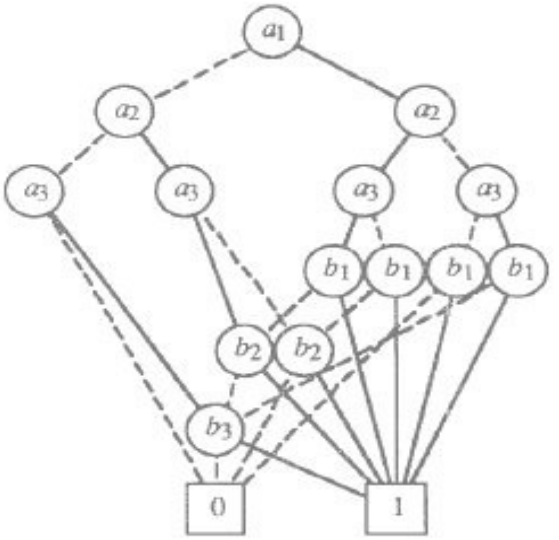
$$f(a_1, a_2, .., a_n, b_1, b_2, .., b_n) = (a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee .. \vee (a_n \wedge b_n)$$

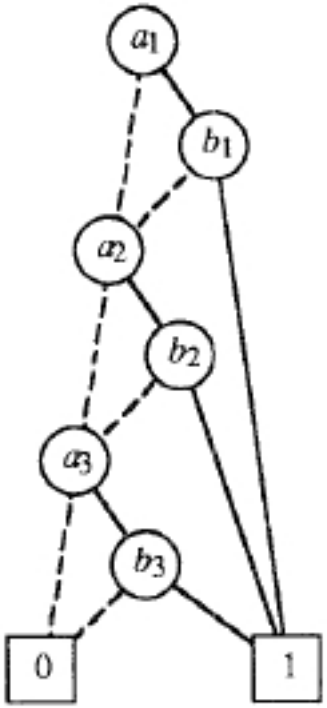
Określmy dwa porządki:

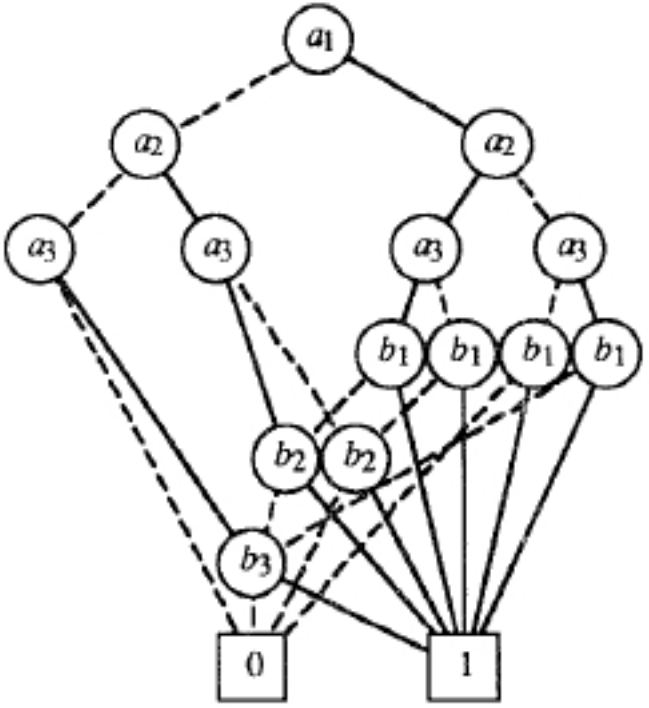
$$a_1 < b_1 < a_2 < b_2 < .. < a_n < b_n$$

$$a_1 < a_2 < .. < a_n < b_1 < b_2 < .. < b_n$$

OBDD

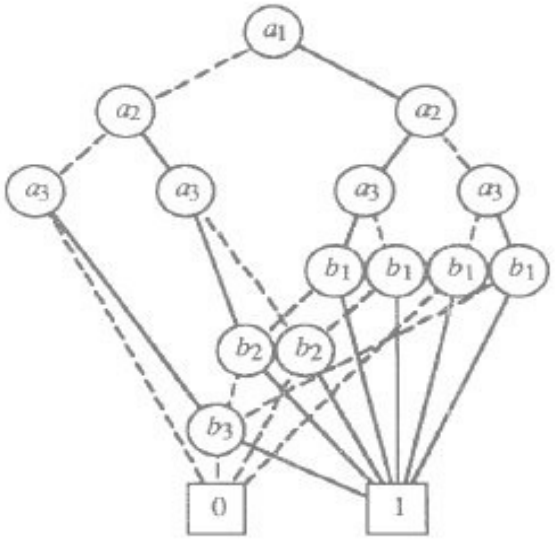






porównanie rozmiarów OBDD dla tej samej funkcji i różnych porządków

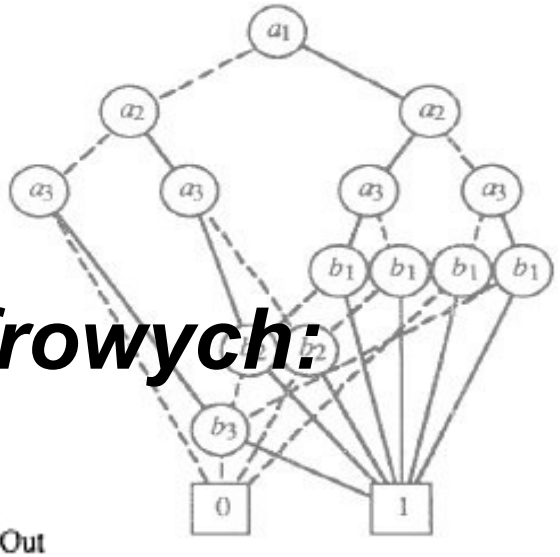
OBDD



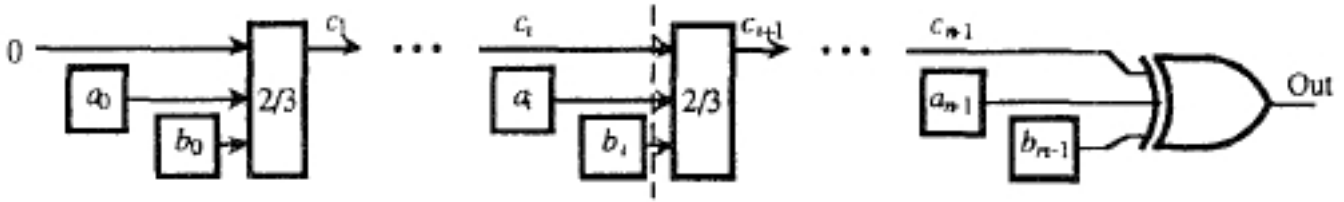
Analiza złożoności:

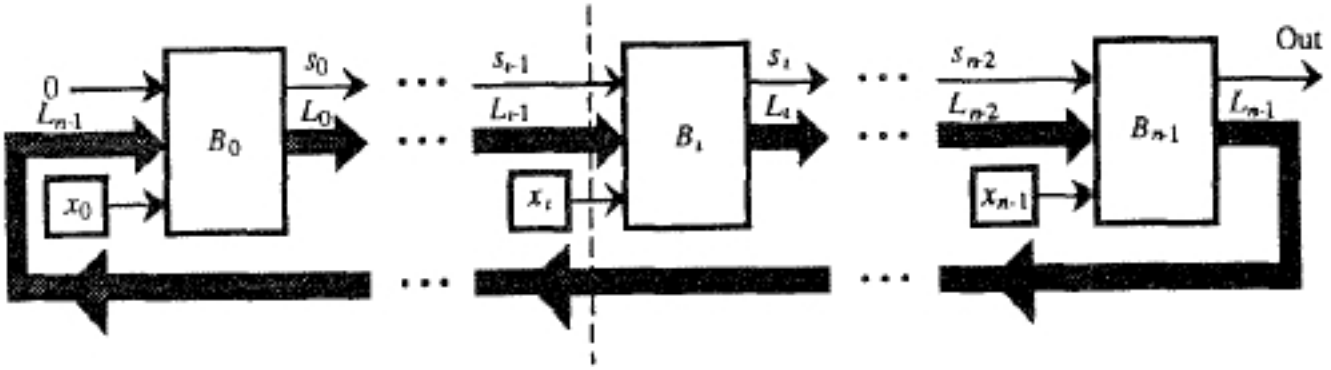
Function Class	Complexity	
	Best	Worst
Symmetric	linear	quadratic
Integer Addition (any bit)	linear	exponential
Integer Multiplication (middle bits)	exponential	exponential

OBDD



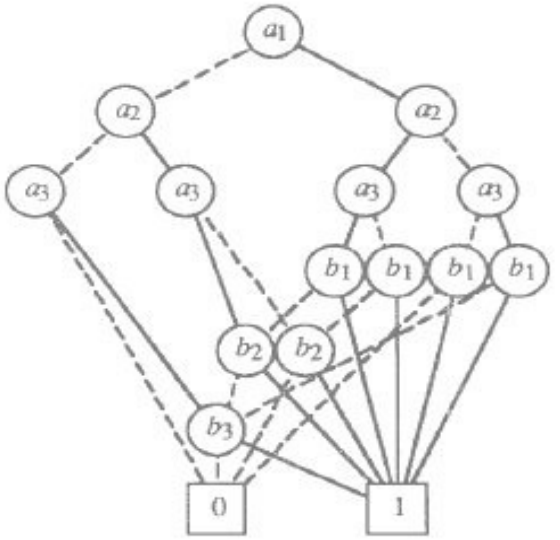
Analiza złożoności układów cyfrowych:





Ograniczenie na rozmiar OBDD: $n\left[2^b \cdot n^{b-1}\right]^{w_f \cdot 2^{w_r}}$

OBDD

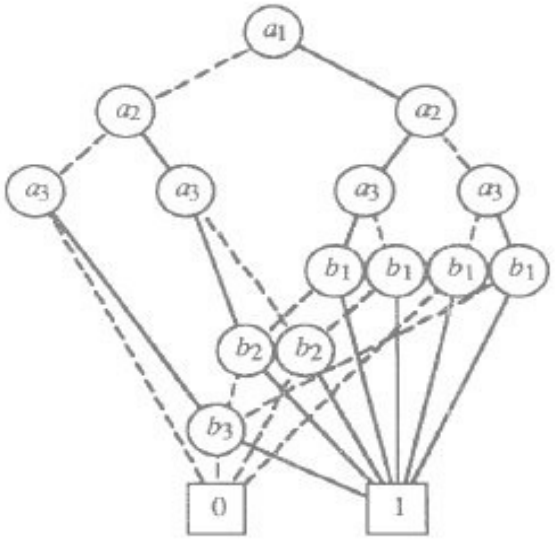


Dozwolone operacje:

- OR, AND, XOR, NOT, np.: $(f \Leftrightarrow g) \vee d$
- podstawienie (obcięcie), np.: $f\downarrow_{x \leftarrow g} = (\neg g \wedge f\downarrow_{x \leftarrow 0}) \vee (g \wedge f\downarrow_{x \leftarrow 1})$
- kwantyfikatory, np.: $\exists x\ f = f\downarrow_{x \leftarrow 0} \vee f\downarrow_{x \leftarrow 1}$

$$\forall x\ f = f\downarrow_{x \leftarrow 0} \wedge f\downarrow_{x \leftarrow 1}$$

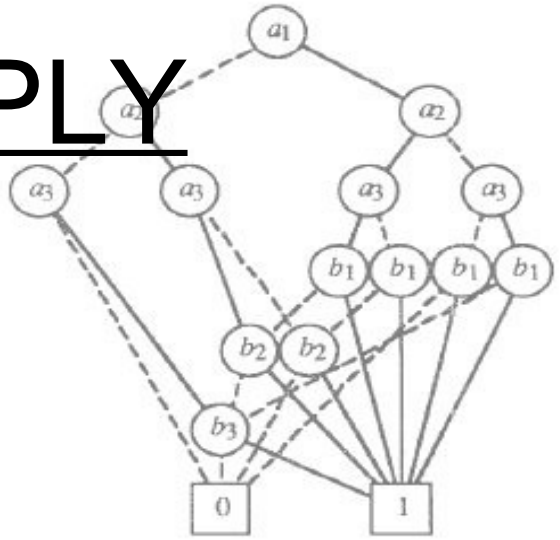
OBDD



Budowa i operacje na OBDD:

- algorytmy grafowe, zachowujące porządek
- algorytm APPLY
- algorytm RESTRICT
- inne operacje implementowane przy użyciu powyższych algorytmów

OBDD – algorytm APPLY



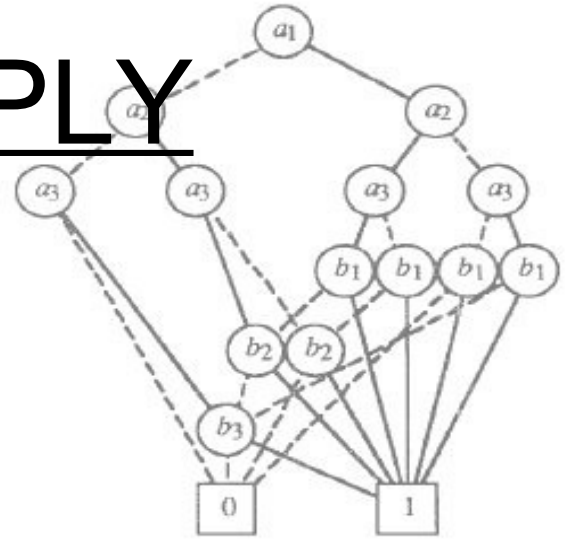
$$f(a,b,c,d)=(a\vee b)\wedge c\vee d$$
$$g(a,b,c,d)=(a\wedge\neg c)\vee d$$

Szukamy $f\vee g$

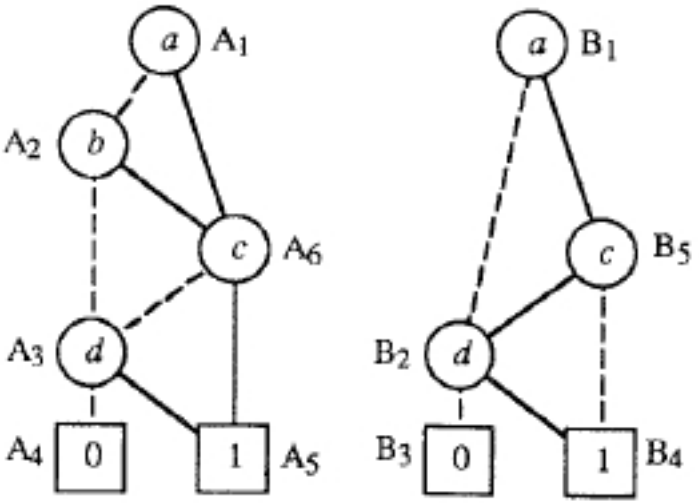
Używamy rozwinięcia:

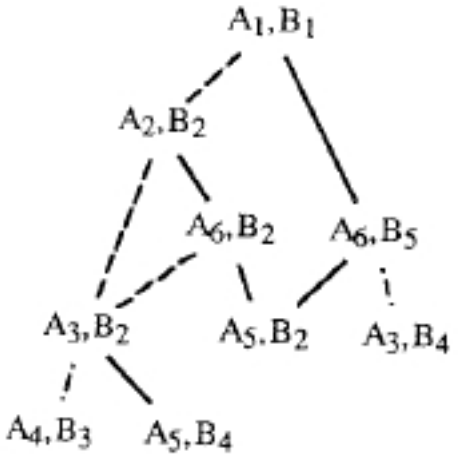
$$f<op>g=(\neg x\wedge(f\downarrow_{x\leftarrow 0}<op>g\downarrow_{x\leftarrow 0}))\vee(x\wedge(f\downarrow_{x\leftarrow 1}<op>g\downarrow_{x\leftarrow 1}))$$

OBDD – algorytm APPLY

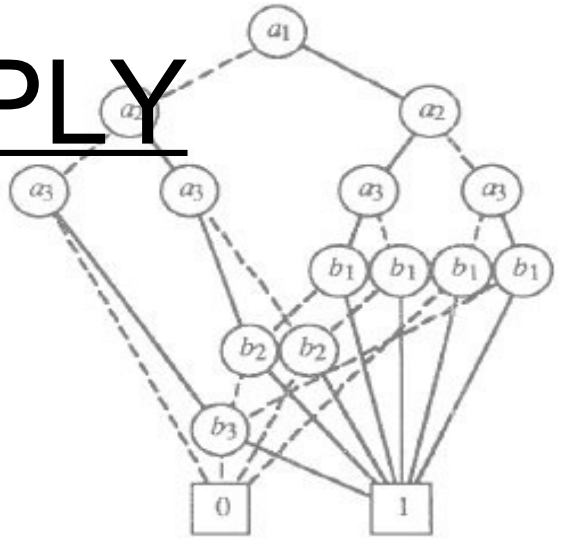


$$f(a,b,c,d)=(a\vee b)\wedge c\vee d$$
$$g(a,b,c,d)=(a\wedge\neg c)\vee d$$

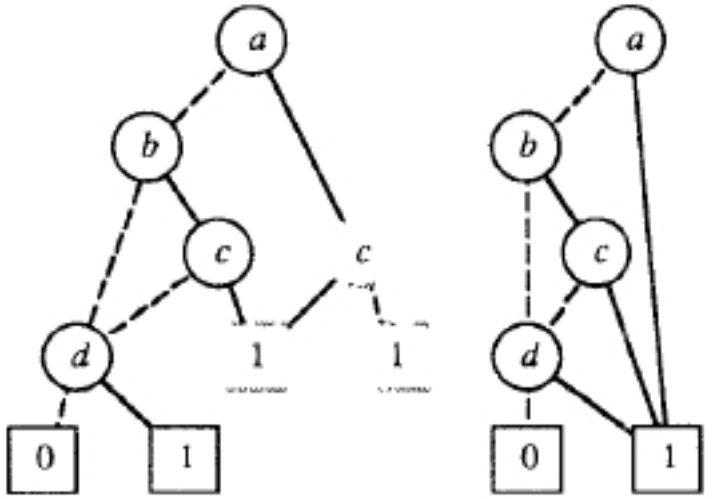




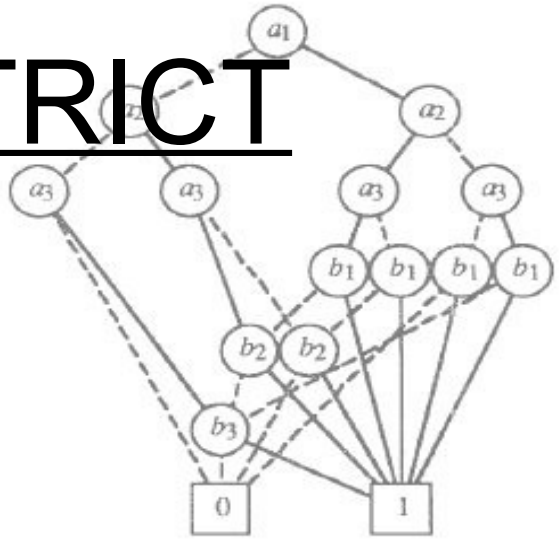
OBDD – algorytm APPLY



$$f(a,b,c,d)=(a\vee b)\wedge c\vee d$$
$$g(a,b,c,d)=(a\wedge\neg c)\vee d$$
$$(f\vee g)(a,b,c,d)=a\vee(b\wedge c)\vee d$$

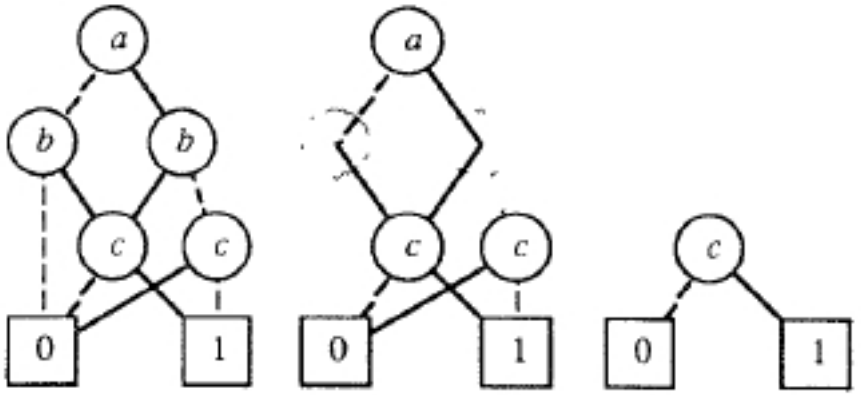


OBDD – algorytm RESTRICT

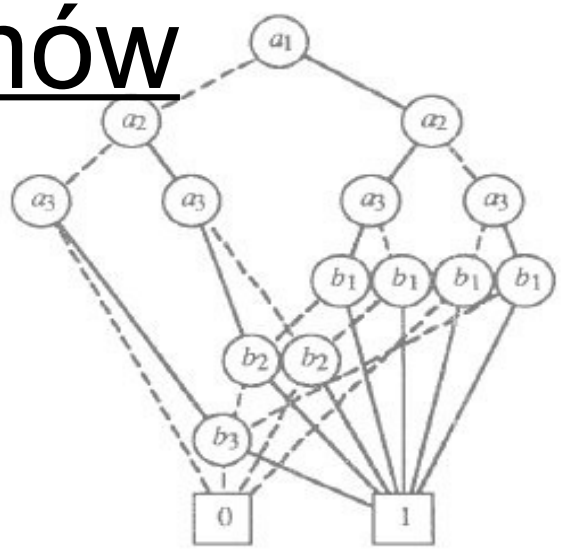


$$f(a,b,c)=(b\wedge c)\vee(a\wedge\neg b\wedge\neg c)$$

Szukamy $f\downarrow_{b\leftarrow 1}$



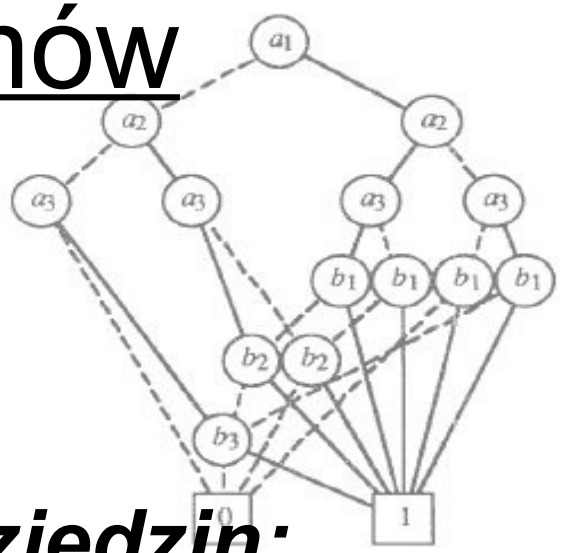
Reprezentacja systemów matematycznych



Przykładowe dziedziny:

Class	Typical Operations	Typical Tests
Logic	$\wedge, \vee, \neg, \forall, \exists$	satisfiability, implication
Finite domains	*domain dependent*	equivalence
Functions	application, composition	equivalence
Sets	$\cup, \cap, -$	subset
Relations	composition, closure	symmetry, transitivity

Reprezentacja systemów matematycznych



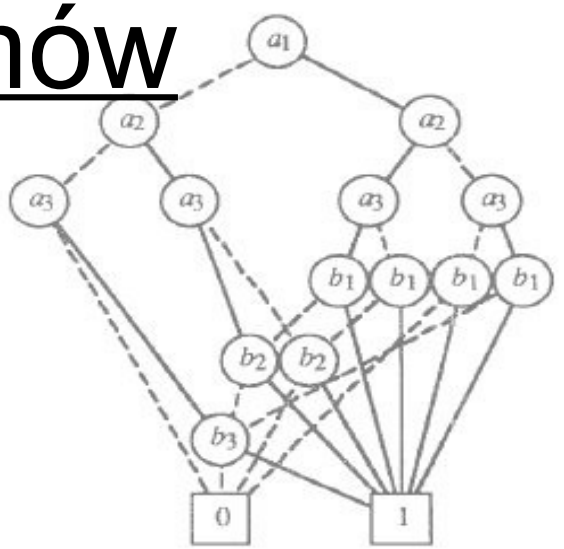
Reprezentacja skończonych dziedzin:

$$|A| = N$$
$$n = [\![\log(N)]\!]$$
$$\sigma : A \to \{0, 1\}^n$$

Reprezentacja odwzorowania $f : A \to A$

$$\vec{f}$$
$$f_i : \{0, 1\}^n \to \{0, 1\}$$
$$f_i(\sigma(a)) = \sigma_i(f(a))$$

Reprezentacja systemów matematycznych



Przykład:

t	0	1	X
0	0	0	0
1	0	1	X
X	0	X	X

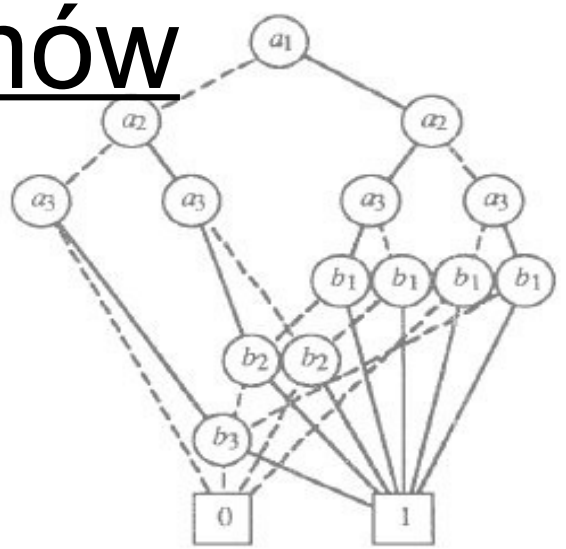
$+_t$	0	1	X
0	0	1	X
1	1	1	1
X	X	1	X

a	$\overline{a}^t$
0	1
1	0
X	X

$$\sigma(0)=[0,1],\sigma(1)=[1,0],\sigma(X)=[1,1]$$

$$[a_1,a_0]\vee_t[b_1,b_0]=[a_1\wedge b_1,a_0\vee b_0]$$
$$[a_1,a_0]\wedge_t[b_1,b_0]=[a_1\vee b_1,a_0\wedge b_0]$$
$$\neg_t[a_1,a_0]=[a_0,a_1]$$

Reprezentacja systemów matematycznych



Reprezentacja zbiorów:

$$S \subseteq A$$

Reprezentowany jako: $\chi_S(\vec{x}) = \vee_{a \in S} \wedge_{1 \leqslant t \leqslant n} x_t \Leftrightarrow \sigma_t(a)$

Operacje:

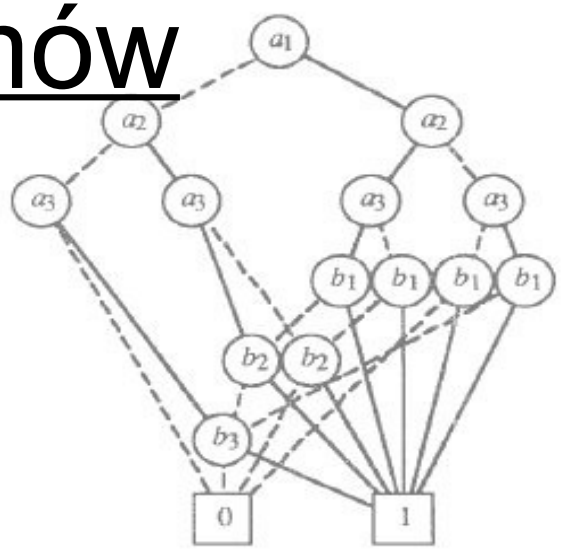
$$\chi_{\varnothing} = 0$$

$$\chi_{S \cup T} = \chi_S \vee \chi_T$$

$$\chi_{S \cap T} = \chi_S \wedge \chi_T$$

$$\chi_{S \setminus T} = \chi_S \wedge \neg \chi_T$$

Reprezentacja systemów matematycznych



Reprezentacja relacji:

$$R \subseteq A \times A$$

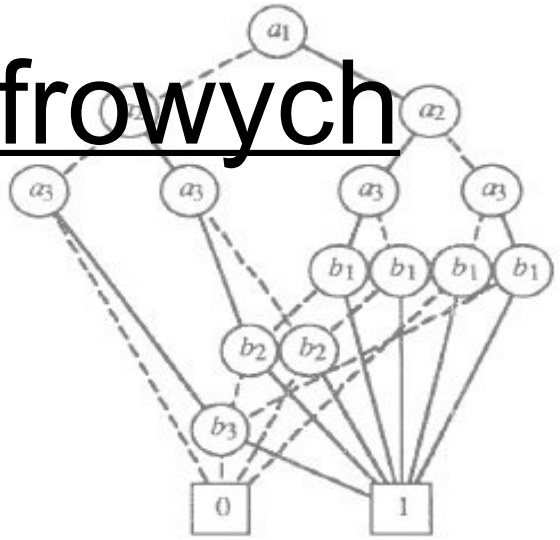
Reprezentowana jako: $\chi_R(\vec{x}, \vec{y}) = \vee_{a \in A} \vee_{\substack{b \in A \\ a R b}} [\wedge_{1 \leqslant i \leqslant n} x_i \Leftrightarrow \sigma_i(a)] \wedge [\wedge_{1 \leqslant i \leqslant n} y_i \Leftrightarrow \sigma_i(b)]$

Złożenie: $\chi_{R \circ S} = \exists \vec{z} [\chi_R(\vec{x}, \vec{z}) \wedge \chi_S(\vec{z}, \vec{y})]$

Znajdowanie punktów stałych: $R_0 = Id$

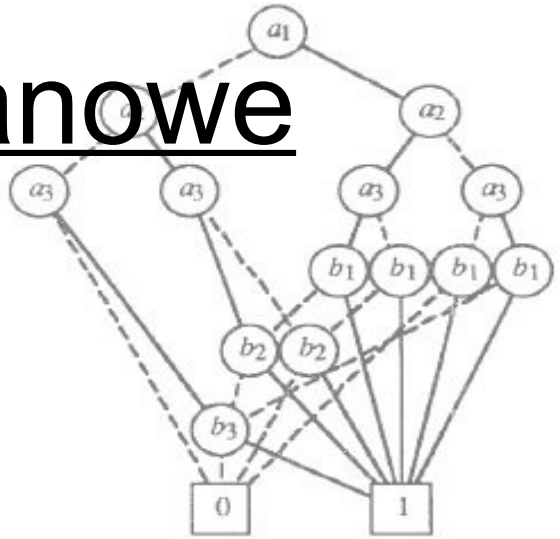
$$R_{t+1} = Id \cup R \circ R_t$$

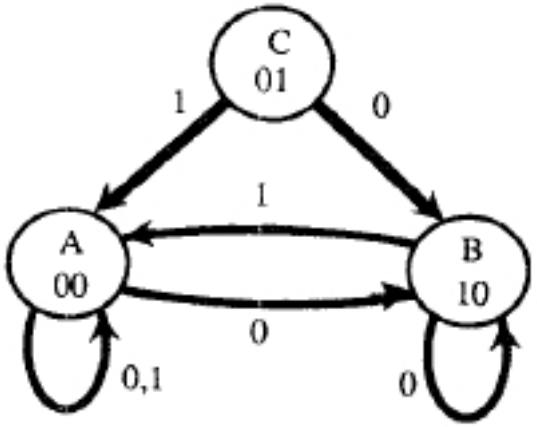
Projektowanie układów cyfrowych

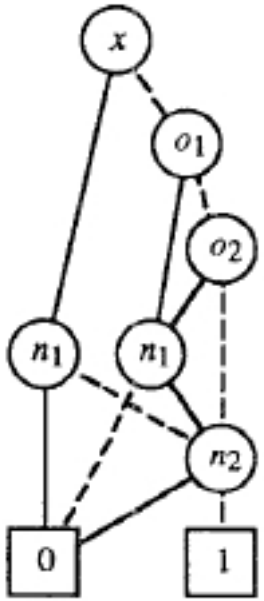


1. **Weryfikacja**
2. **Wykrywanie błędów projektowych**
3. **Badanie reakcji na zaburzenie wartości sygnałów**
4. **Analiza probabilistyczna**

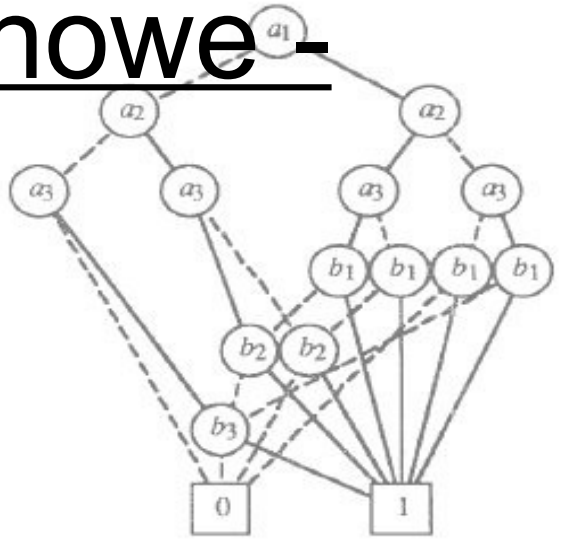
Systemy skończenie-stanowe



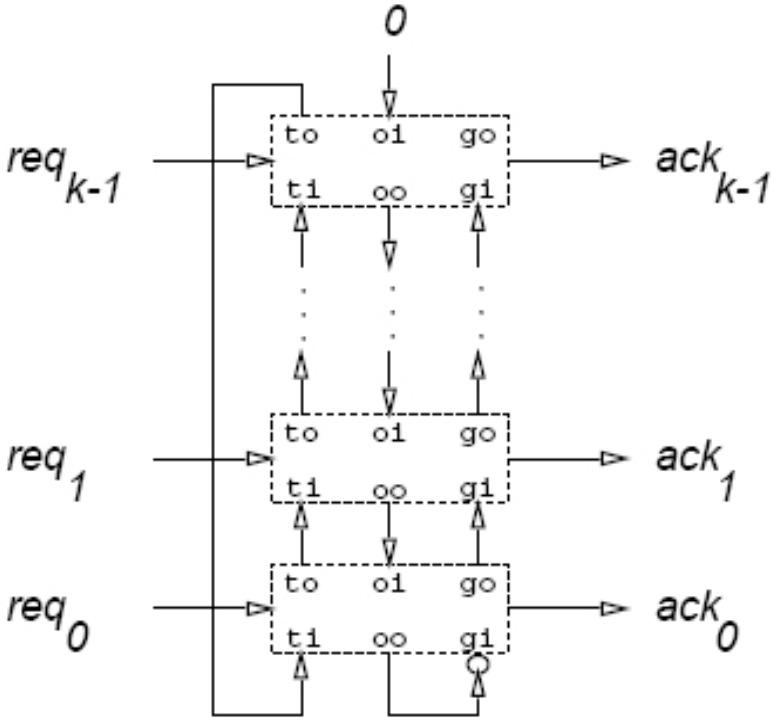




Systemy skończenie-stanowe- przykład



- synchroniczny system skończenie-stanowv
- k klientów, jeden sędzia

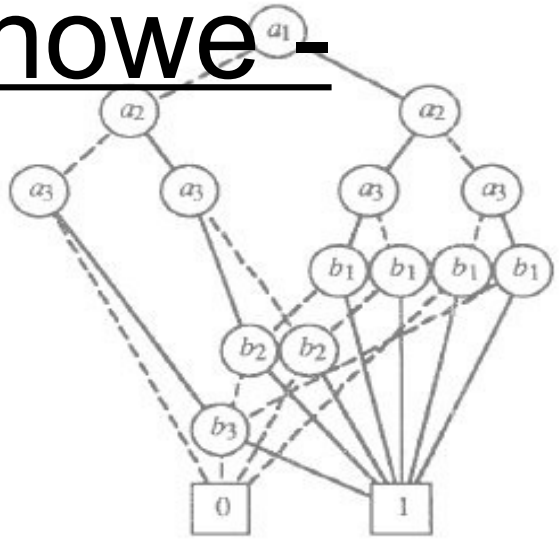


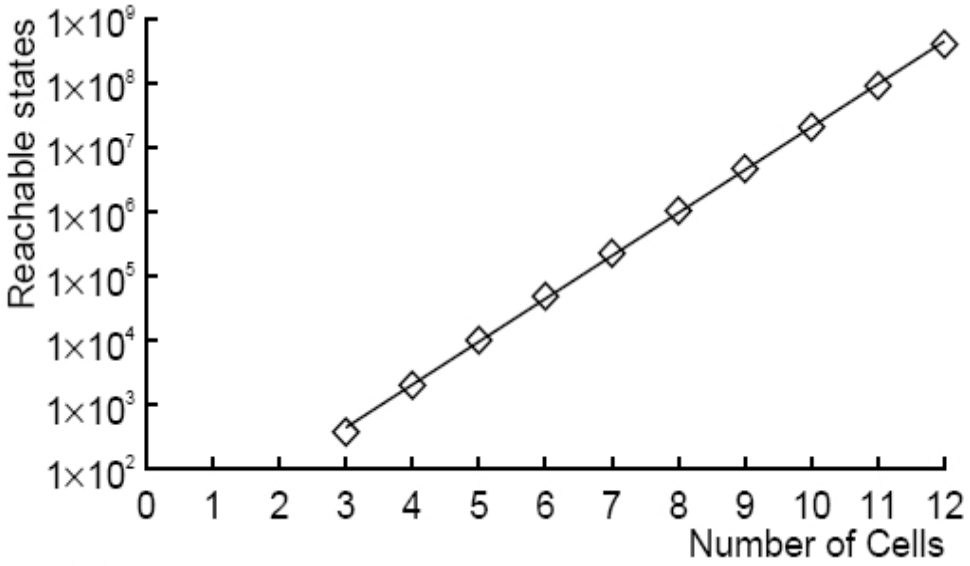
- sprawdzamy formuły: $\wedge_{i \neq j} AG \neg (ack_i \wedge ack_j)$

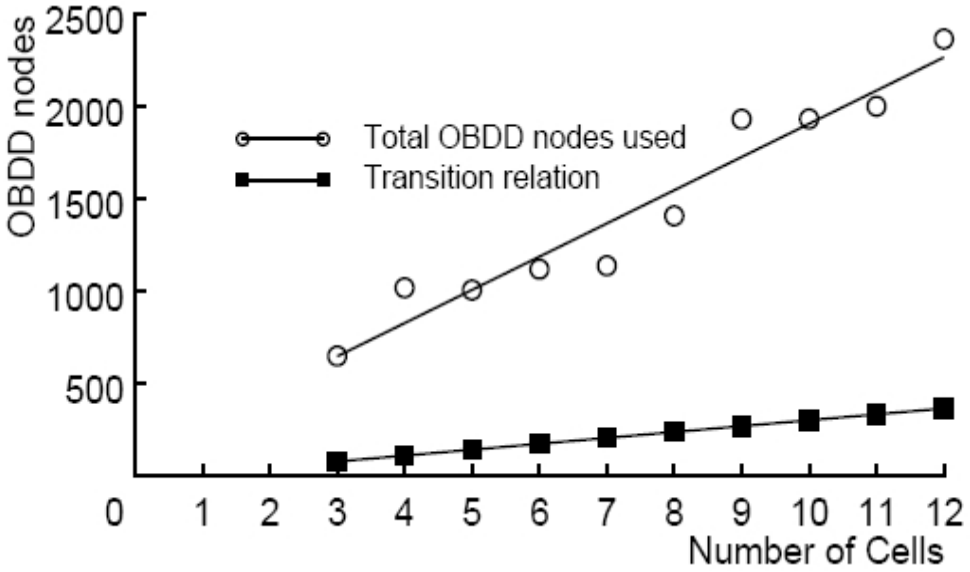
$$\wedge_i AGAF(req_i \Rightarrow ack_i)$$

$$\wedge_i AG(ack_i \Rightarrow req_i)$$

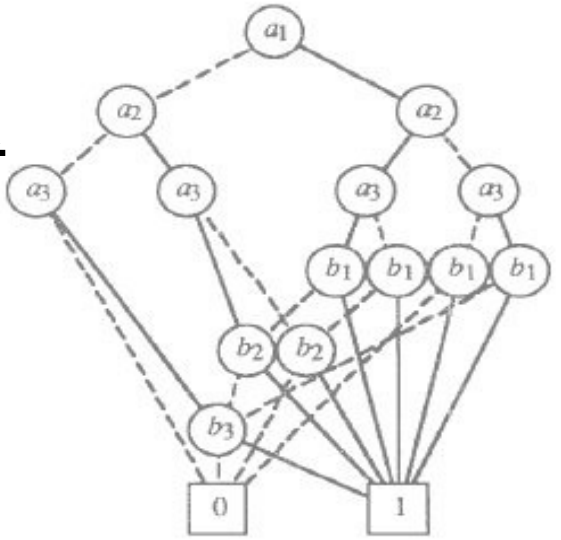
Systemy skończenie-stanowe - przykład





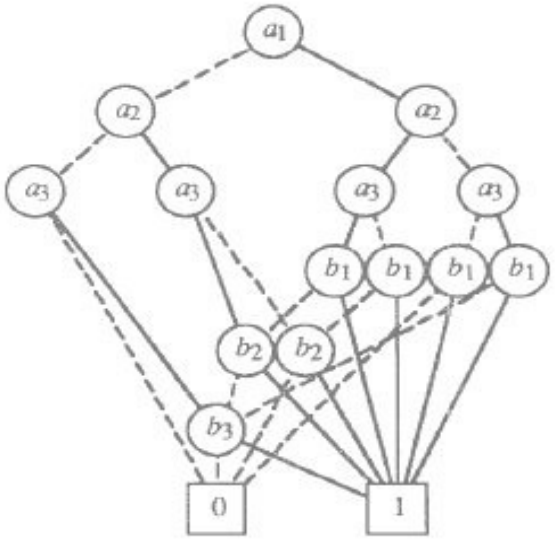


Inne zastosowania



- technika stało-punktowa stosowana w logice matematycznej i językach formalnych;
- OBDD w sztucznej inteligencji, jako baza wiedzy

Propozycje zmian



- bardziej skomplikowane testy w wierzchołkach;
- złagodzenie wymogów stawianych porządkowi na zmiennych;
- opóźnione („lazy”) wyliczanie grafu