Model checking simbolic. Diagrame de decizie binare

22 octombrie 2004

Model checking cu reprezentare explicită a spaţiului stărilor

Algoritmii de până acum: consideră individual fiecare stare
$\Rightarrow$ dimensiunea spaţiului stărilor limitează aplicabilitatea (dacă o stare are n biţi, rezultă direct câte putem reprezenta în memorie)
– tipic, limitat la câteva milioane de stări

Dacă spaţiul stărilor atinse e mic faţă de spaţiul potenţial complet, se poate încerca o codificare a stărilor pe număr mai mic de biţi (*bitstate hashing*, metodă folosită în SPIN). Metoda e însă doar o *aproximaţie*: dacă se ajunge la o stare cu cod deja întâlnit, explorarea se opreşte (deşi poate starea e diferită).
$\Rightarrow$ o parte din spaţiul stărilor poate rămâne neexplorată

Explorarea cu stări şi mulţimi

Calculul stărilor care pot fi atinse din stările iniţiale
(**EF** *true*)
– prin traversare a grafului pornind de la stările iniţiale
– R: mulţimea stărilor explorate; F: *frontiera* stărilor atinse

Cu *stări individuale*

$R = \emptyset$; $F = S_0$
while $(F \neq \emptyset)$
  choose $s \in F$;
  $F \leftarrow F \setminus \{s\}$; $R \leftarrow R \cup \{s\}$
  forall s' with $s \rightarrow s'$
    if $s' \notin F \cup R$
      $F \leftarrow F \cup \{s'\}$

Cu *mulţimi de stări*

$R = \emptyset$; $F = S_0$
while $(F \not\subseteq R)$
  $R \leftarrow R \cup F$
  $F = \{s' \in S | \exists s \in F \,.\, s \rightarrow s'\}$
  /* eventual $F = F \setminus R$
   * cu test $F \neq \emptyset$ */

⇒Algoritmul se exprimă mult mai simplu dacă se poate calcula *într-o singură operaţie* mulţimea *succesorilor* unei mulţimi de stări
⇒mulţimea R a stărilor explorate creşte la fiecare iteraţie, dar e finită

Model checking simbolic

- O nouă abordare: explorarea mulţimilor de stări
  – ideea: o mulţime poate fi uneori reprezentată (printr-o formulă) într-un spaţiu mai mic decât explicit pentru fiecare stare în parte
  – reprezentare eficientă pt. mulţimi de stări, relaţie de tranziţie [McMillan'92]
  – cu ajutorul diagramelor de decizie binare (BDD) [Bryant'86]
- idee cheie: Operarea cu *mulţimi* de stări
  – folosită şi în cazul în care mulţimea stărilor este infinită (sisteme în timp continuu, sisteme hibride)
- idee cheie: prelucrarea iterativă până când nu se mai produc modificări
  ⇒noţiunea de *punct fix*

Reprezentări de punct fix

Def: $x \in D$ este *punct fix* pentru $f : D \to D$ dacă $f(x) = x$. Def: O *latice* e o mulţime parţial ordonată în care orice submulţime finită are un cel mai mic majorant şi un cel mai mare minorant
Ex: mulţimea părţilor $\mathcal{P}(S)$ a lui S, cu relaţia de incluziune $\subseteq$

– Lucrăm cu funcţii $\tau : \mathcal{P}(S) \to \mathcal{P}(S)$ peste *laticea* $\mathcal{P}(S)$

– Privim $S' \subseteq S$ ca un *predicat* peste S: $S'(s) = true \Leftrightarrow s \in S'$
în particular: $\emptyset = false$, $S = true$
$\Rightarrow \tau : \mathcal{P}(S) \to \mathcal{P}(S)$ este o *transformare de predicate*
Def:

- τ este *monoton* dacă $P \subseteq Q \Rightarrow \tau(P) \subseteq \tau(Q)$
- τ este *continuu la uniune* dacă pentru orice şir $P_1 \subseteq P_2 \subseteq \ldots$ avem $\tau(\cup_i P_i) = \cup_i \tau(P_i)$
- τ este *continuu la intersecţie* dacă pentru orice şir $P_1 \supseteq P_2 \supseteq \ldots$ avem $\tau(\cap_i P_i) = \cap_i \tau(P_i)$

Teoreme de punct fix

O transformare de predicate monotonă pe $\mathcal{P}(S)$ are întotdeauna
– un punct fix minimal, notat $\mu Z.\tau(Z)$
– și un punct fix maximal, notat $\nu Z.\tau(Z)$ [Tarski]

Dacă S e finită și τ e monotonă, atunci τ e continuă la uniune și la intersecție.

τ monotonă $\Rightarrow \tau^i(\mathit{False}) \subseteq \tau^{i+1}(\mathit{False})$ și $\tau^i(\mathit{True}) \supseteq \tau^{i+1}(\mathit{True})$

Dacă τ e monotonă și S e finită, există $i, j \geq 0$ astfel ca
$\forall k \geq i$, $\tau^k(\mathit{False}) = \tau^i(\mathit{False})$ și $\forall k \geq j$, $\tau^k(\mathit{True}) = \tau^j(\mathit{True})$

Dacă τ e monotonă și S e finită, există $i, j \geq 0$ astfel ca
$\mu Z.\tau(Z) = \tau^i(\mathit{False})$ și $\nu Z.\tau(Z) = \tau^j(\mathit{True})$

Calculul punctului fix minimal/maximal

function $Lfp(\tau : Trans) : Pred$

$Q := False;$
$Q' := \tau(Q);$
while $(Q' \neq Q)$ **do**
$Q := Q';$
$Q' := \tau(Q);$
return $Q;$

function $Gfp(\tau : Trans) : Pred$

$Q := True;$
$Q' := \tau(Q);$
while $(Q' \neq Q)$ **do**
$Q := Q';$
$Q' := \tau(Q);$
return $Q;$

Relaţii de punct fix pentru CTL

Identificăm formula CTL f cu mulţimea de stări $\{s \mid M, s \models f\}$

- $\mathbf{AF}\, f = \mu Z \,.\, f \vee \mathbf{AX}\, Z$ $\qquad$ $\mathbf{EF}\, f = \mu Z \,.\, f \vee \mathbf{EX}\, Z$
- $\mathbf{AG}\, f = \nu Z \,.\, f \wedge \mathbf{AX}\, Z$ $\qquad$ $\mathbf{EG}\, f = \nu Z \,.\, f \wedge \mathbf{EX}\, Z$
- $\mathbf{A}\,[f_1\, \mathbf{U}\, f_2] = \mu Z \,.\, f_2 \vee (f_1 \wedge \mathbf{AX}\, Z)$
- $\mathbf{E}\,[f_1\, \mathbf{U}\, f_2] = \mu Z \,.\, f_2 \vee (f_1 \wedge \mathbf{EX}\, Z)$
- $\mathbf{A}\,[f_1\, \mathbf{R}\, f_2] = \nu Z \,.\, f_2 \wedge (f_1 \vee \mathbf{AX}\, Z)$
- $\mathbf{E}\,[f_1\, \mathbf{R}\, f_2] = \nu Z \,.\, f_2 \wedge (f_1 \vee \mathbf{EX}\, Z)$

punct fix minimal: proprietăţi de evoluţie: **F**

punct fix maximal: proprietăţi de siguranţă (invarianţi): **G**

Algoritmul de model checking simbolic

Prin descompunere după structura formulei.

Check(f) returnează $\{s \in S \mid M, s \models f\}$

Check(p) $= \{s \in S \mid p \in L(s)\}$ propoziţii atomice

Check($\neg f$) $= S \setminus$ *Check*(f) complement

Check($f \wedge g$) $=$ *Check*(f) $\cap$ *Check*(g) intersecţie

Check($\mathbf{EX}\, f$) $=$ *CheckEX*(*Check*(f))

CheckEX($f(\bar{v})$) $= \exists \bar{v}' \,.\, [f(\bar{v}') \wedge R(\bar{v}, \bar{v}')]$ produs relaţional

Check($\mathbf{E}\,[f\,\mathbf{U}\,g]$) $=$ *CheckEU*(*Check*(f), *Check*(g))

folosind $\mathbf{E}\,[f_1\,\mathbf{U}\,f_2] = \mu Z \,.\, f_2 \vee (f_1 \wedge \mathbf{EX}\, Z)$ şi funcţia *Lfp*

Check($\mathbf{EG}\, f$) $=$ *CheckEG*(*Check*(f))

folosind $\mathbf{EG}\, f = \nu Z \,.\, f \wedge \mathbf{EX}\, Z$ şi funcţia *Gfp*

Aceste operaţii de bază se pot implementa cu BDD-uri

Diagrame de decizie binare

Binary Decision Diagrams (BDDs)

- o reprezentare compactă şi canonică a funcţiilor boolene
- cu algoritmi eficienţi de manipulare

  [R. Bryant, "Graph-based algorithms for boolean function manipulation", *IEEE Transactions on Computers*, 1986]
- impact deosebit asupra verificării formale:

  ACM Kanellakis Award for Theory & Practice, 1998
  - Randal E. Bryant: BDDs ('86)
  - Edmund M. Clarke, E. Allen Emerson: model checking ('81)
  - Ken McMillan: symbolic model checking ('92)

Funcţii boolene. Reprezentări

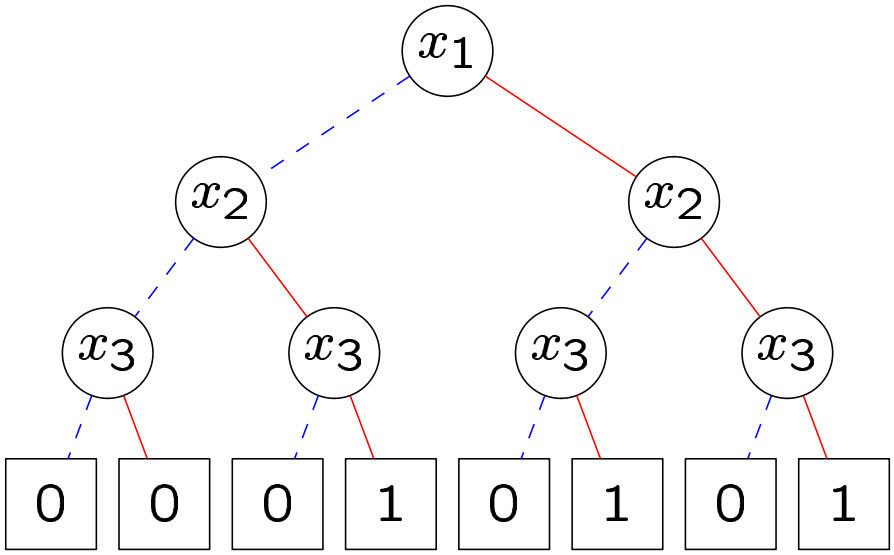
Obiectul de lucru: funcţii Boolene $f : B^n \to B$

- reprezentări uzuale: tabele de adevăr, diagrame Karnaugh, sumă canonică de mintermi – dimensiune exponentială
- sumă redusă de produse, factorizări, etc.
  – exponenţiale pentru anumite funcţii comune (ex. paritate)
- operaţii elementare pot rezulta în creştere exponenţială (ex. complementarea)
- reprezentări necanonice $\Rightarrow$ e dificil de testat:
  – echivalenţa (după transformări în proiectarea circuitelor)
  – satisfiabilitatea: $\exists x_1, \cdots x_n \,.\, f(x_1, \cdots, x_n) = 1$?

$$\forall x \,.\, f_1(x) = f_2(x) \equiv \neg \exists x \,.\, f_1(x) \oplus f_2(x) = 1$$

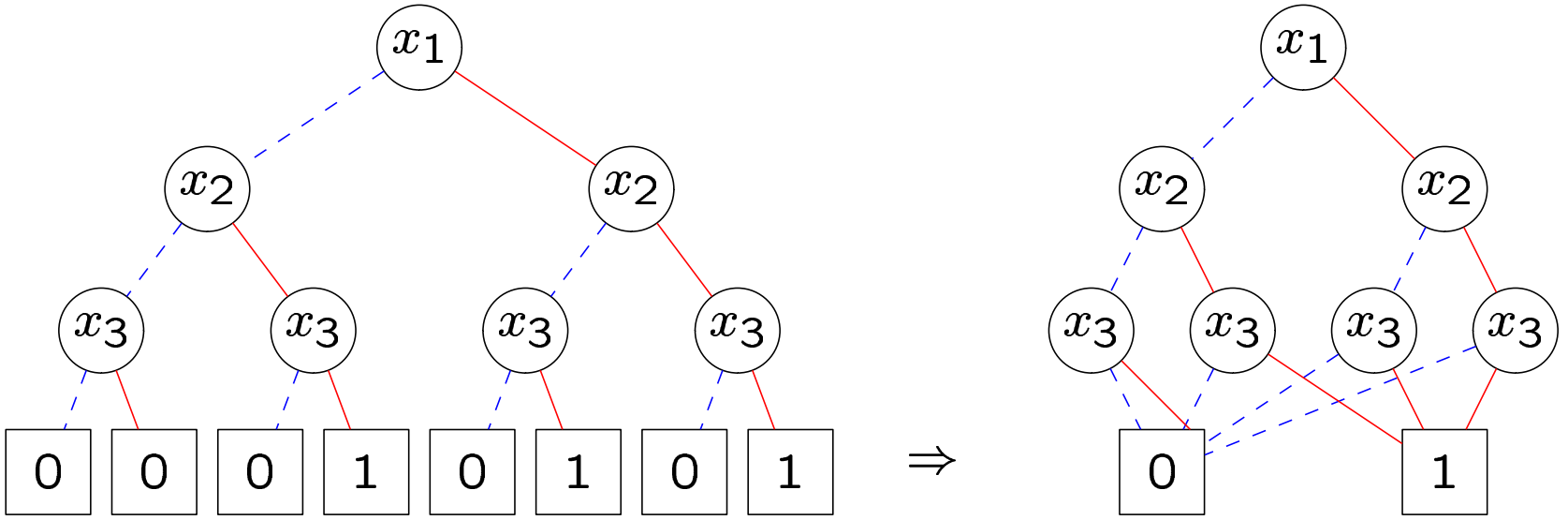
Arbori de decizie binari

- noduri terminale: valoarea funcţiei (0 sau 1)
- noduri neterminale: variabile
- ramuri: $low(v)$ (stânga) / $high(v)$ (dreapta): atribuire 0/1 pt. variabila din nod

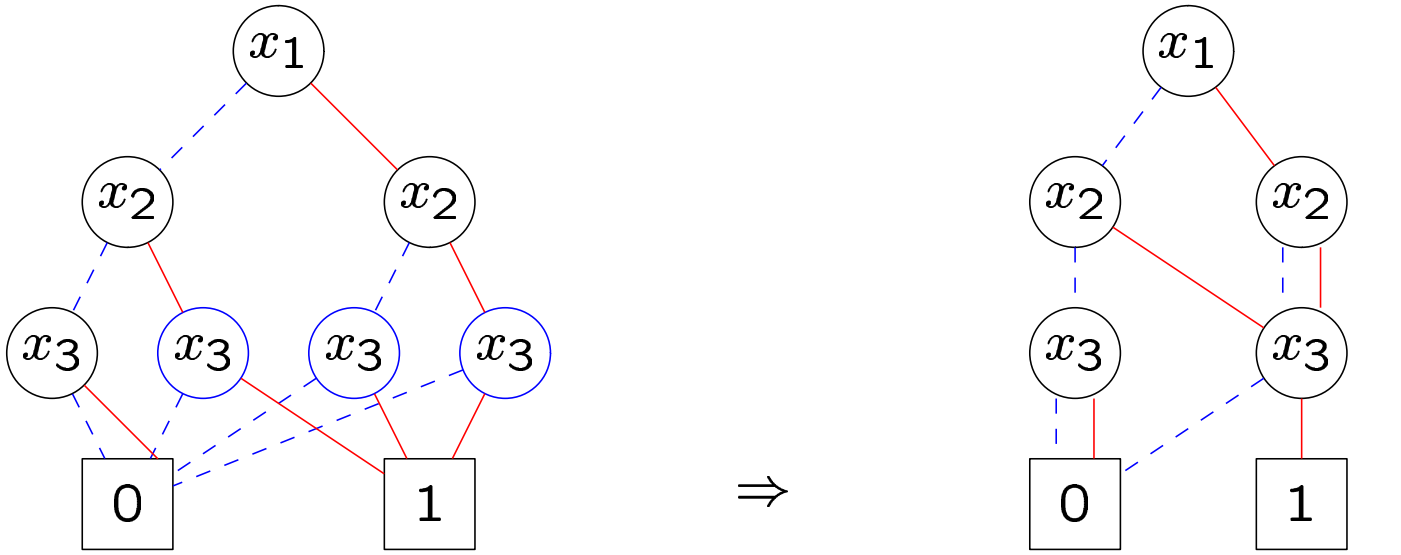


BDD-uri: obţinute prin 3 reguli de reducere

Reducerea nr. 1: Comasarea nodurilor terminale

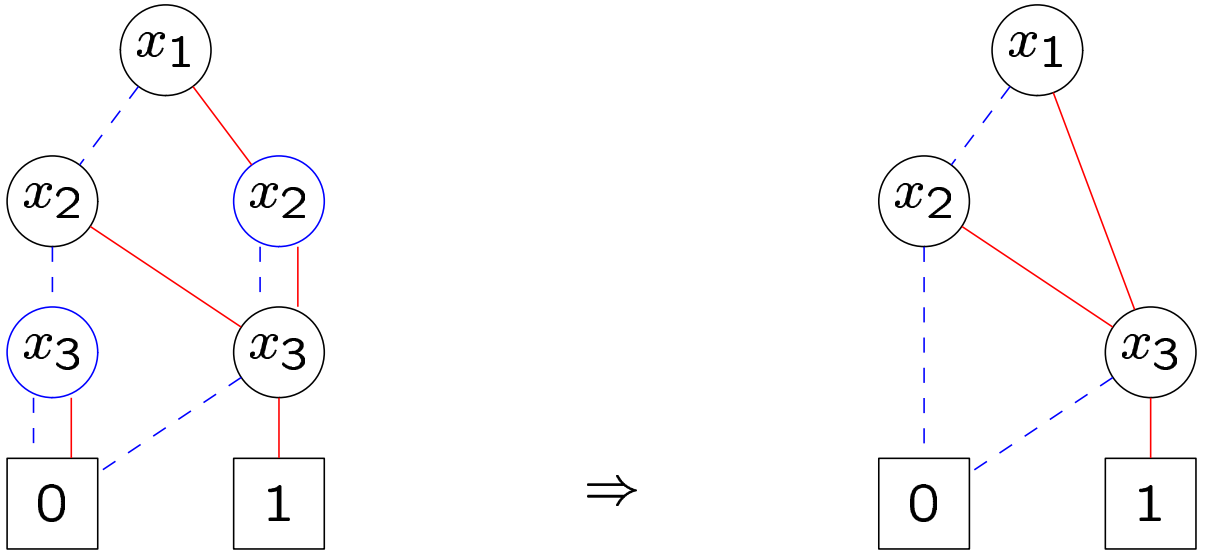


Reducerea nr. 2: Comasarea nodurilor izomorfe



$f(n_1) = f(n_2) \Rightarrow$ comasează n_1 și n_2

Reducerea nr. 3: Eliminarea testelor inutile



$low(n) = high(n) \Rightarrow$ elimină testarea lui n

Proprietăţi esenţiale

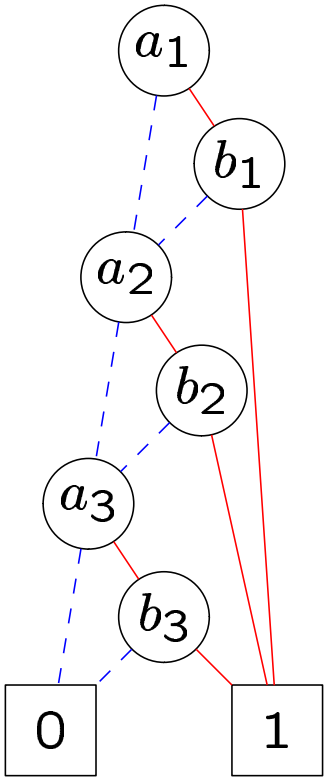
Cele 3 reguli se pot aplica indiferent de ordonarea variabilelor.
Pentru a defini o BDD *ordonată* (Ordered BDD = OBDD)
se impune o condiţie esenţială:
Pe toate căile de la vârf la nodurile terminale, variabilele apar
în aceeaşi ordine (= există o ordonare globală a variabilelor).

Teoremă: Pentru orice funcţie, reprezentarea ca BDD *ordonată*,
redusă cf. regulilor 1-3 e unică pâna la un izomorfism.
$\Rightarrow$ reprezentare *canonică*
$\Rightarrow$ testare de echivalenţă sau satisfiabilitate în $O(1)$

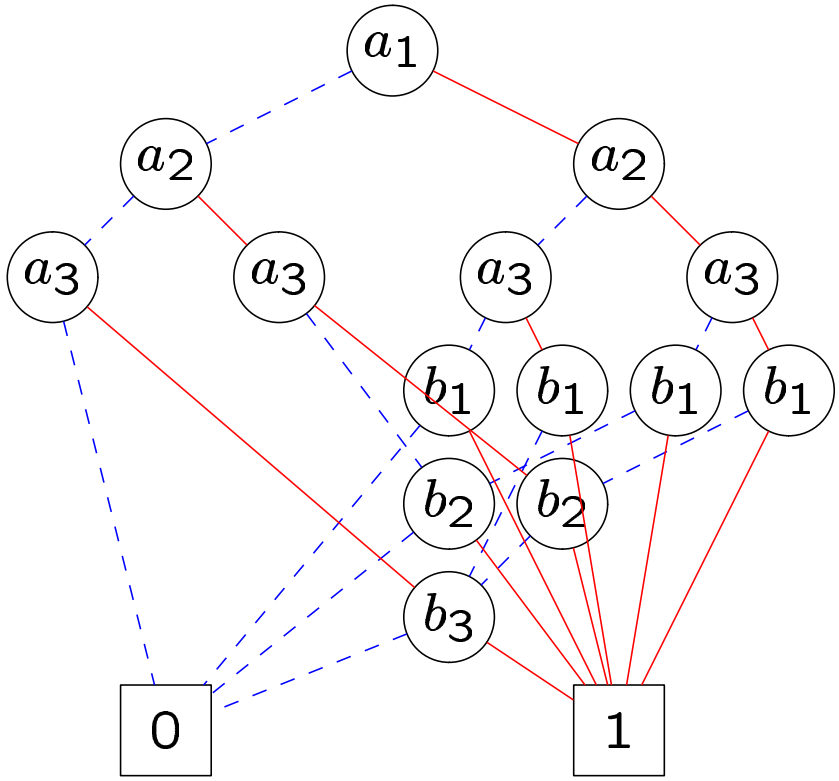
Obs:
Un subgraf cu rădăcina într-un nod de BDD este tot o BDD.

Efectul ordonării variabilelor

Funcţia: $(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$



Creştere liniară: $2(n+1)$



Creştere exponenţială: 2^{n+1}

Algoritmi cu BDD-uri: Apply

function *Apply*(f, g : *OBDD*, op : *Operator*) : *OBDD*
if *is_leaf*(f) $\wedge$ *is_leaf*(g) **return** $op(f, g)$;
elsif (f, g, op, h) in apply_cache **return** h;
else
  x := *topvar*(f) /* variabila din vârful lui f */
  y := *topvar*(g)
  if (*ord*(x) = *ord*(y)) /* x = y = aceeași variabilă */
    h := *find_bdd*(x, *Apply*($f\mid_{x=0}, g\mid_{x=0}, op$), *Apply*($f\mid_{x=1}, g\mid_{x=1}, op$))
    /* *find_bdd* creează un nou BDD dacă nu există deja */
  elsif (*ord*(x) < *ord*(y)) /* x inaintea lui y în ordine */
    h := *find_bdd*(x, *Apply*($f\mid_{x=0}, g, op$), *Apply*($f\mid_{x=1}, g, op$))
  else h := *find_bdd*(y, *Apply*($f, g\mid_{y=0}, op$), *Apply*($f, g\mid_{y=1}, op$))
  insert (f, g, op, h) in apply_cache
  return h

Algoritmi cu BDD-uri: Produs relational

function *Relprod*(f, g : *OBDD*, E : *varset*) : *OBDD*
if $f = \mathit{false} \vee g = \mathit{false}$ **return** *false*
elsif $f = \mathit{true} \wedge g = \mathit{true}$ **return** *true*
elsif (f, g, E, h) in relprod_cache **return** h
else
  $x :=$ *topvar*(f) /* variabila din vârful lui f */
  $y :=$ *topvar*(g)
  $z :=$ *topmost*(x, y) /* prima în ordinea variabilelor */
  $h_0 :=$ *RelProd*($f\,|_{z=0}, g\,|_{z=0}, E$)
  $h_1 :=$ *RelProd*($f\,|_{z=1}, g\,|_{z=1}, E$)
  if $z \in E$ $h :=$ *bdd_or*(h_0, h_1) /* $\exists z \,.\, h = h_0 \vee h_1$ */
  else $h :=$ *bdd_if_then_else*(z, h_1, h_0)
  insert (f, g, E, h) in relprod_cache
  return h

Complexitatea algoritmilor

- Reducere (la forma canonică) $O(|G| \cdot \log |G|)$
- Apply ($f_1 \langle op \rangle f_2$) $O(|G_1| \cdot |G_2|)$
- Restrict ($f\ |_{x_i=b}$) $O(|G| \cdot \log |G|)$
- Compose ($f_1\ |_{x_i=f_2}$) $O(|G_1|^2 \cdot |G_2|)$
- Satisfy-one (un $\bar{x}$ cu $f(\bar{x}) = 1$) $O(n)$
- Satisfy-count ($|\{\bar{x} \mid f(\bar{x}) = 1\}|$) $O(|G|)$

Factorii logaritmici pot fi eliminaţi
(prin algoritmi mai sofisticaţi sau hashing).

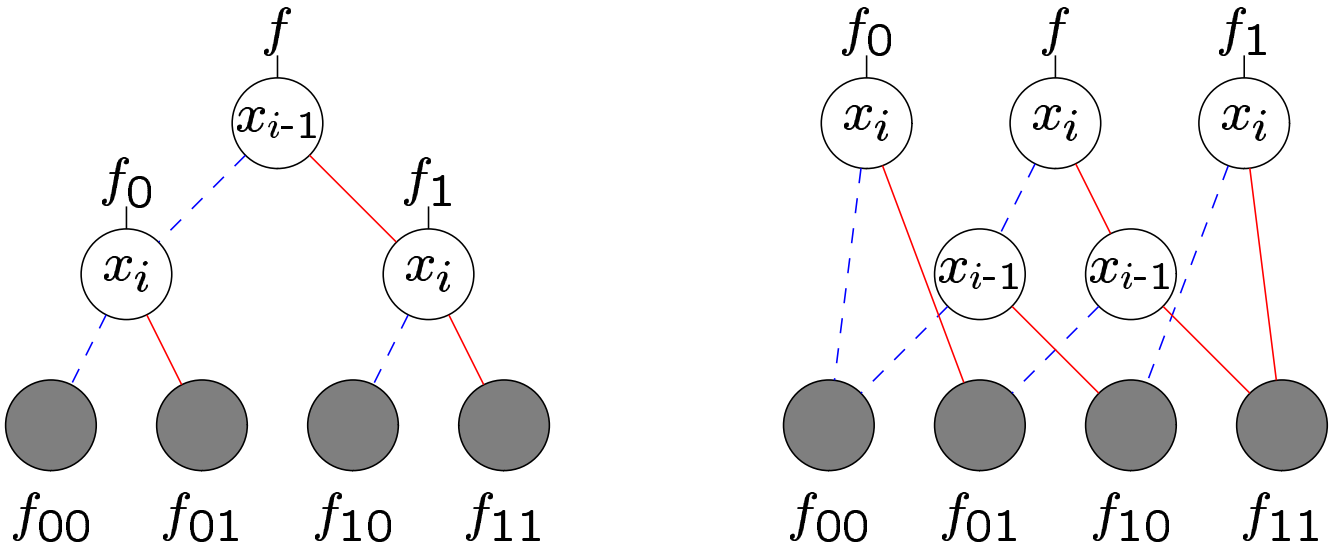
Produsul relaţional poate avea complexitate exponenţială.

Implementare

- Există implementări mature de biblioteci (pachete) de BDD-uri
- Într-o aplicaţie tipică, multe BDD-uri au subgrafuri comune
  $\Rightarrow$ pointeri într-un unic graf multi-rădăcină
- Gestiunea memoriei: cu contor de referinţă şi garbage-collection
- Un vast număr de optimizări şi euristici:
  – metode de dispunere şi traversare pt. caching avantajos
  – calcul paralel şi distribuit, etc.

Reordonarea dinamică a variabilelor

- Ordonarea variabilelor - efect critic asupra dimensiunii
- Există funcţii cu reprezentări exponenţiale indiferent de ordonare (ex. bitul mijlociul al unui multiplicator [Bryant'91])
- BDD-urile evoluează în timpul execuţiei aplicaţiei
  $\Rightarrow$ f. importantă reordonarea *dinamică*
  – efectuată transparent pentru algoritmii de verificare
  – reordonarea nivelelor adiacente nu modifică pointerii externi



Variante de BDD-uri

- Relaxarea condiţiei de ordonare: FreeBDDs
  – variabilele apar în orice ordine, fiecare doar o dată
  – $\forall$ atribuire la $\bar{v}$, aceeaşi ordine în toate funcţiile
  – reprezentări bune pt. unele funcţii cu OBDD-uri exponenţiale
- Alegerea relaţiei de decompoziţie funcţională
  pentru OBDD: descompunerea Boole-Shannon:

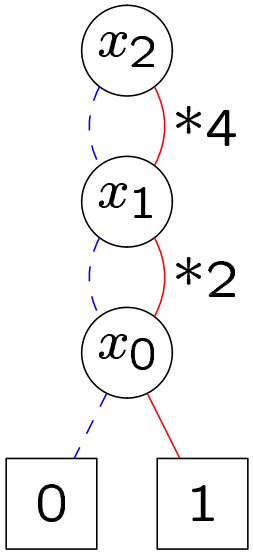
$$f = \bar{x} \wedge f_{\bar{x}} \vee x \wedge f_x$$

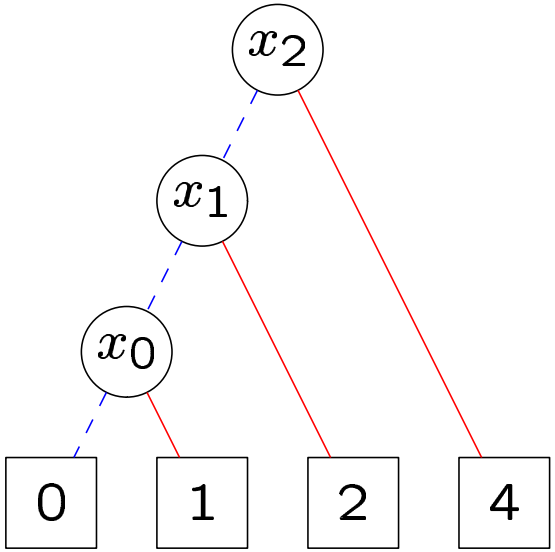
  $f_{\bar{x}} = f\,|_{x=0}$: cofactorul negativ
  $f_x = f\,|_{x=1}$: cofactorul pozitiv

  Alte relaţii de decompoziţie:
  – $f = f_{\bar{x}} \oplus x \wedge f_{\delta x}$ descompunere Reed-Muller
  – $f = f_x \oplus \bar{x} \wedge f_{\delta x}$ descompunere Davio pozitivă
  unde $f_{\delta x} = f_x \oplus f_{\bar{x}}$
- Multiterminal BDDs: extensie cu noduri terminale arbitrare (întregi, etc.)

Diagrame pentru operaţii aritmetice

Exemplu: codificare numerică pe 3 biţi: $f = x_0 + 2 * x_1 + 4 * x_2$





Edge-Valued BDD (EVBDD)
ponderi multiplicative pe muchii

Binary Moment Diagram (BMD)
$f = f_{\bar{x}} + x \cdot (f_x - f_{\bar{x}})$

Hybrid Decision Diagrams = BDD + BMD + MTBDD

Word-level Verification

Cazul tipic: demonstrarea corespondenţei dintre:
– o descriere matematică (numerică) F_{num} (ex. operatia de înmulţire)
– un circuit f_{bit} cu operanzi în reprezentare binară

Fie $Num : B^n \to \mathbf{Z}$ (sau $\mathbf{R}$) funcţia care descrie semnificaţia numerică a unui număr reprezentar binar.

Trebuie verificat:

$$Num(f_{bit}(\vec{x_1}, \vec{x_2})) = F_{num}(Num(\vec{x_1}), Num(\vec{x_2}))$$

$\Rightarrow$ reprezentare eficientă pentru funcţii pe biţi cât şi numerice
$\Rightarrow$ se utilizează EVBDD, BMD, HDD, etc.

Aplicaţii

Principala aplicaţie: in CAD şi verificare formală.

În general: pentru reprezentarea compactă a datelor cu o anumită regularitate/structură comună, dificil de exprimat analitic.

- teoria codurilor
- structuri mari de date, indexare
- biologie computaţională

  Exemplu: sistem de filtrare publish-subscribe în timp real

  – flux mare de mesaje; sute de mii de criterii de filtrare

  – criteriile de filtrare reprezentate ca structură de BDD-uri

  (propoziţii atomice: atribut $\langle$*relaţie*$\rangle$ valoare)

  – mesajele noi sunt filtrate prin algoritmi de evaluare a BDD-urilor

  – subcriteriile comune din BDD: evaluate o singură dată

Model checking simbolic: Istoric

Ken McMillan (CMU, 1987): ideea de a reprezenta sisteme în mod simbolic cu BDD-uri.

$\Rightarrow$ [Burch, Clarke, Dill, McMillan, Hwang: *Symbolic model checking:* 10^{20} *states and beyond*, 1990]

$\Rightarrow$ verificatorul SMV; teza de doctorat: *Symbolic Model Checking: An Approach to the State Explosion Problem* (CMU, 1992)

Independent:

– Coudert, Berthet, Madre [1989, 1990]

– Pixley [Motorola, 1990]

Model checking simbolic cu BDD-uri

Reprezentare: codificare binară pentru stări şi propozitii atomice
$\Rightarrow$ BDD-uri pentru mulţimi de stări, relaţie de tranziţie

$\mathit{Check}(p) = \{s \in S \mid p \in L(s)\}$	$\mathit{bdd\_if\_then\_else}(p, 1, 0)$
$\mathit{Check}(\neg f) = S \setminus \mathit{Check}(f)$	$\mathit{bdd\_not}$
$\mathit{Check}(f \wedge g) = \mathit{Check}(f) \cap \mathit{Check}(g)$	$\mathit{bdd\_and}$
$\mathit{Check}(\mathbf{EX}\, f) = \mathit{CheckEX}(\mathit{Check}(f))$	
$\mathit{CheckEX}(f(\bar{v})) = \exists \bar{v}' \,.\, [f(\bar{v}') \wedge R(\bar{v}, \bar{v}')]$	$\mathit{RelProd}(f, R, \bar{v}')$
$\mathit{Check}(\mathbf{E}\,[f\, \mathbf{U}\, g]) = \mathit{CheckEU}(\mathit{Check}(f), \mathit{Check}(g))$	
$\mathbf{E}\,[f_1\, \mathbf{U}\, f_2] = \mu Z \,.\, f_2 \vee (f_1 \wedge \mathbf{EX}\, Z)$	algoritmul *Lfp*
$\mathit{Check}(\mathbf{EG}\, f) = \mathit{CheckEG}(\mathit{Check}(f))$	
$\mathbf{EG}\, f = \nu Z \,.\, f \wedge \mathbf{EX}\, Z$	algoritmul *Gfp*

Partiționarea relației de tranziție

O relație de tranziție monolitică poate deveni foarte mare

principala dificultate: produsul relațional

- partiționare *disjunctivă* (sisteme asincrone)

$$R(\bar{v}, \bar{v}') = R_1(\bar{v}, \bar{v}') \vee \cdots \vee R_n(\bar{v}, \bar{v}')$$

prin distributivitate: $\exists \bar{v}'[f(\bar{v}') \wedge R(\bar{v}, \bar{v}')] =$

$$= \exists \bar{v}'[f(\bar{v}') \wedge R_1(\bar{v}, \bar{v}')] \vee \cdots \vee \exists \bar{v}'[f(\bar{v}') \wedge R_n(\bar{v}, \bar{v}')]$$

- partiționare conjunctivă (sisteme sincrone)

$\exists$ nu distribuie față de $\wedge$, dar se poate exploata localitatea

(dacă R_i nu depind de toate variabilele din $\bar{v}'$):

$$R(\bar{v}, \bar{v}') = R_1(\bar{v}, v_1') \wedge \cdots \wedge R_n(\bar{v}, v_n')$$

$\exists \bar{v}'[f(\bar{v}') \wedge R(\bar{v}, \bar{v}')] =$

$$= \exists v_n'[\cdots \exists v_1'[f(\bar{v}') \wedge R_0(\bar{v}, v_1'] \wedge R_1(\bar{v}, v_1')] \cdots \wedge R_n(\bar{v}, v_n')]$$

(conjuncție și cuantificare succesivă pentru fiecare componentă)

Model checking simbolic cu fairness

Restricţia de fairness: $F = \{P_1, P_2, \cdots, P_n\}$, cu $P_i \subseteq S$

$\mathbf{EG}\, f$ e adevărată în mulţimea maximală Z cu proprietăţile:
– toate stările din Z satisfac f
– $\forall P_k \in F, s \in Z$ există un drum din s într-o stare din $Z \cap P_k$
(trecând doar prin stări care satisfac f)

$\Rightarrow$ exprimare ca punct fix, poate fi calculată simbolic:

$$\mathbf{EG}_{\mathit{fair}} f = \nu Z \,.\, f \wedge \bigwedge_{i=1}^{n} \mathbf{EX}\, \mathbf{E}\, [f\, \mathbf{U}\, (Z \wedge P_k)]$$

La fel cu cele definite pentru reprezentarea explicită:

$$\mathbf{EX}_{\mathit{fair}}\, f = \mathbf{EX}\, (f \wedge \mathit{fair})$$
$$\mathbf{EU}_{\mathit{fair}}\, (f, g) = \mathbf{EU}\, (f, g \wedge \mathit{fair})$$

Generarea de contraexemple

Principalele avantaje ale tehnicii de model checking:
– metodă complet automată
– generează contraexemple care identifică erorile

- formule existenţiale (**E**) : produce o traiectorie "martor" (*witness*) pentru care formula este adevărată

- formule universale (**A**): produce un contraexemplu

- contraexemplul pentru o formulă universală este traiectoria martor pentru negaţia ei (formula existenţială dală)

Traiectorie pentru **EF** f

– punct fix minimal: $\mathbf{EF}\, f = \mu Z \,.\, f \vee \mathbf{EX}\, Z$
– se obţin şi se reţin aproximări succesive $f = Q_0 \subseteq Q_1 \subseteq \ldots \subseteq Q_k$
– Q_k: mulţimea stărilor din care se poate atinge f în cel mult k paşi
– se găseşte o intersecţie $Q_k \cap S_0 \neq \emptyset$
(o primă secvenţă de traversare: înapoi, simbolică)
– se alege $s_k \in S_0 \cap Q_k$
– se calculează mulţimea $Succ(s_k)$ a succesorilor lui s_k
– ea trebuie să aibă o intersecţie cu Q_{k-1} (din s_k se atinge f în cel mult k paşi, deci există un succesor din care se ajunge în $k-1$ paşi)
– se alege $s_{k-1} \in Succ(s_k) \cap Q_{k-1}$, etc. până la $Q_0 = f$
(a doua traversare, înainte, prin stări individuale)
– s-a găsit o traiectorie $s_k \to \ldots \to s_0$ care atinge f

Traiectorie pentru **EG** f cu fairness

– calcul prin iteraţie (*Gfp*): $\mathbf{EG}\, f = \nu Z.f \wedge \bigwedge_{i=1}^{n} \mathbf{EX}\, \mathbf{E}\,[f\, \mathbf{U}\, (Z \wedge P_k)]$ (*)
– iteraţii interioare (*Lfp*) pentru $\mathbf{E}\,[f\, \mathbf{U}\, (Z \wedge P_k)]$, $\forall P_k$
– în ultima iteraţie exterioară se reţine şirul de aproximări
$Q_0^{P_k} \subseteq Q_1^{P_k} \subseteq \ldots \subseteq Q_i^{P_k} \subseteq \ldots$ pentru care $\mathbf{EG}\, f \wedge P_k$ e atins în i paşi
– se alege o stare iniţială $s_0 \models \mathbf{EG}\, f$
– din mulţimea de succesori $Succ(s_0)$ (vezi **EX** în (*)) se caută s_1 care atinge un P_k în număr minim de paşi: $\min i\, .\, Succ(s_0) \cap Q_i^{P_k} \neq \emptyset$
(euristică *greedy* pentru găsirea unei traiectorii scurte)
– se găseşte traiectoria: i paşi din $s_1 \Rightarrow i-1$ paşi din $Succ(s_1)$
$\Rightarrow$ alegem $s_2 \in Succ(s_1) \cap Q_{i-1}$ ... $\Rightarrow$ se atinge $s_{i+1} \in \mathbf{EG}\, f \cap P_k$
– eliminăm P_k, continuăm din s_{i+1} până vizităm şi ultimul P_j (în s')
– dacă $s' \models \mathbf{EX}\, \mathbf{E}\,[f\, \mathbf{U}\, s_1]$, închidem ciclul spre $s_1 \Rightarrow$ avem martorul
– dacă nu, s' e în altă SCC decât s_1. Repetăm procedeul din s'.
– graful SCC-urilor e aciclic $\Rightarrow$ procedeul e finit.