Prouver les programmes :

Vérification et optimisation booléennes d'automates et circuits

Gérard Berry

Collège de France

Chaire Algorithmes, machines et langages

gerard.berry@college-de-france.fr

Cours 6, Paris, 1e avril 2015

Suivi du séminaire de Jean-Raymond Abrial

COLLÈGE DE FRANCE 1530

Agenda

1. Codages booléens d'ensembles
2. Codages d'automates et de circuits
3. Model-checking booléen
4. Des arbres de Shannon aux BDDs
5. Optimisation de circuits Esterel
6. Conclusion

Agenda

1. Codages booléens d'ensembles
2. Codages d'automates et de circuits
3. Model-checking booléen
4. Des arbres de Shannon aux BDDs
5. Optimisation de circuits Esterel
6. Conclusion

Codage booléen d'un ensemble

Notation : les booléens seront notés 0 ou ⊥ et 1 ou ⊤

Soit $E = \{e_0, e_1, \ldots, e_n\}$ un ensemble à coder non vide, et soit un ensemble codeur non vide $X = \{x_0, x_1, \ldots, x_k\}$ de variables x_i.

- Une fonction c de X dans {0,1} est appelée assignation booléenne ou code.
- Un code c : $\{x_0, x_1, \ldots, x_k\} \rightarrow \{0,1\}$ se note comme un n-uplet $(c(x_0), \ldots, c(x_k))$ ou un mot sur {0,1} dont la i^e lettre vaut $c(x_i)$.

  Ex. $x_0 \rightarrow 1$, $x_1 \rightarrow 0$, $x_2 \rightarrow 0$, $x_3 \rightarrow 1$, $x_4 \rightarrow 1$ se note (1,0,0,1,1) ou 10011
- Un codage (E,X,C) de E par X et C est défini par une fonction partielle surjective C des codes sur X dans E .

  (Chaque e doit être codé, il peut l'être par plusieurs codes, et il peut y avoir des codes inutilisés.)

Codage booléen d'un ensemble

- Codage binaire classique, avec n = 7 et k = 2 (k = binsize(n)-1)

  $000 \rightarrow e_0$, $001 \rightarrow e_1$, $010 \rightarrow e_2$, $011 \rightarrow e_3$,

  $100 \rightarrow e_4$, $101 \rightarrow e_5$, $110 \rightarrow e_6$, $111 \rightarrow e_7$

- Codage 2-adique, avec n = 7 et k = 2 (k = binsize(n)-1)

  idem, mais poids faibles d'abord

  $000 \rightarrow e_0$, $100 \rightarrow e_1$, $010 \rightarrow e_2$, $110 \rightarrow e_3$,

  $001 \rightarrow e_4$, $101 \rightarrow e_5$, $011 \rightarrow e_6$, $111 \rightarrow e_7$

- Codage one-hot, avec n = 5, k = 5 (k = n)

  $100000 \rightarrow e_0$, $010000 \rightarrow e_1$, $001000 \rightarrow e_2$,

  $000100 \rightarrow e_3$, $000010 \rightarrow e_4$, $000001 \rightarrow e_5$

  (C est partielle car les autres codes sont inutilisés)

Codage redondant : 2-adique mou

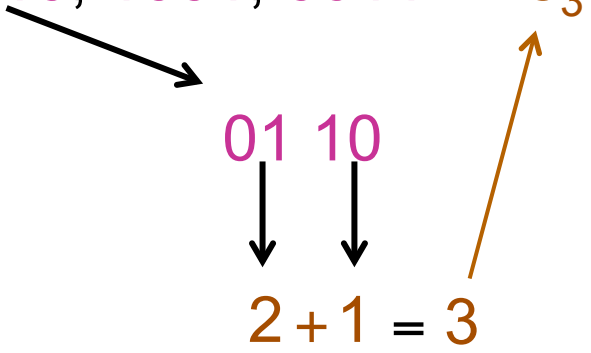
- pour n = 3 et k = 3 (k = 2× binsize(n)-1)

  0000 → e_0,

  1000, 0010 → e_1,

  0100, 1010, 0001 → e_2

  1100, 0110, 1001, 0011 → e_3



Codage de J. Vuillemin pour addition, multiplication et division rapides, cf cours du 9 avril 2013
http://www.college-de-france.fr/site/gerard-berry/course-2013-04-09-10h00.htm

Formules booléennes

- Une formule booléenne $P(x_0,x_1,\ldots,x_k)$ sur X est écrite avec x_0, $\ldots,x_k$ et $\top$, $\bot$, $\wedge$ (ou $\cdot$), $\vee$, $\neg$ (ou $\cdot$) $\overline{\ }$, $\Rightarrow$, $\Leftrightarrow$ (préféré à $=$).

- Si c est un code sur X, on note $P(c) = P(c(x_0),\ldots,c(x_k))$

- Un *minterm* est une conjonction ordonnée des x_i ou de leurs négations. On note $x_0\overline{x}_1\overline{x}_2x_3x_4$ pour $x_0\wedge\neg x_1\wedge\neg x_2\wedge x_3\wedge x_4$.
  Le minterm $m(c)$ ou c associé à un code c a des négations là où c a des 0. Par exemple, $c = x_0\overline{x}_1\overline{x}_2x_3x_4$ si $c = 10011$
  Ce minterm n'est vrai que pour le code c.

- Etant donné un codage (E, X, C), une formule $P(x_0,\ldots,x_k)$ est *compatible avec* C si $C(c) = C(c')$ implique $P(c) \Leftrightarrow P(c')$
  (i.e., P a la même valeur pour des codes redondants)

Codage de sous-ensembles

- Etant donné un codage (E, X, C) et une formule $P(x_0,\ldots,x_k)$ compatible avec C, le sous ensemble C(P) de E codé par P est celui qui contient les éléments codés par les codes vérifiant P :

  $C(P) = \{C(c) \mid C(c)$ est défini et $P(c)$ vrai$\}$

- Propriétés de bases du codage de sous-ensembles :
  - E est codé par $\top$, l'ensemble vide $\varnothing$ est codé par $\bot$
  - {e} est codé par la disjonction $\vee$ des minterms de ses codes, Cette formule pour e est notée e.
    p.ex. $e = x_0\bar{x}_1\bar{x}_2x_3x_4$ si $e = C(10011)$ et C est injectif
  - tout sous-ensemble P peut être codé par $\vee_i e_i$ pour $e_i \in P$
  - l'intersection est codée par $P \wedge Q$, l'union par $P \vee Q$
  - le complémentaire est codé par $\neg P$
    (attention, il est ici important que P soit compatible avec C)

Codages de relations et fonctions

- Toute relation R : E→E peut se représenter par une formule booléenne $R(x_0,\ldots,x_k,x'_0,\ldots,x'_k)$ telle que R(c,c') est vraie ssi e=C(c), e'=C(c') et R(e,e')

  preuve : construire un grand ∨ de conjonctions e∧e' pour tous les e et e' vérifiant R(e,e')

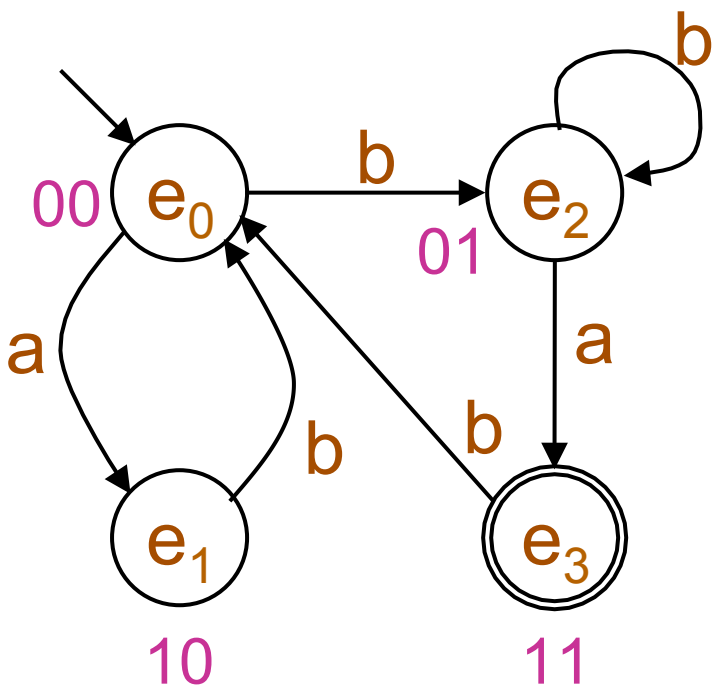
  (n.b.: montrer que ce ∨ ne peut être vrai que si un de ses sous-termes est vrai, donc que le tiers exclu n'est pas applicable)

- Toute fonction f : E→E peut se représenter par un vecteur de formules $F(x_0,\ldots,x_k) = (f_0(x_0,\ldots,x_k),\ldots,f_k(x_0,\ldots,x_k))$ telles que F(c) = c' si e' = f(e), e=C(c) et e'=C(c')

Agenda

1. Codages booléens d'ensembles
2. Codages d'automates et de circuits
3. Model-checking booléen
4. Des arbres de Shannon aux BDDs
5. Optimisation de circuits Esterel
6. Conclusion

Codage d'un automate déterministe



état initial : $\bar{x}_0\bar{x}_1$

état final : x_0x_1

relation de transition T_a : $e \xrightarrow{a} e'$

$$T_a = \bar{x}_0\bar{x}_1x'_0\bar{x}'_1 \vee \bar{x}_0x_1x'_0x'_1$$

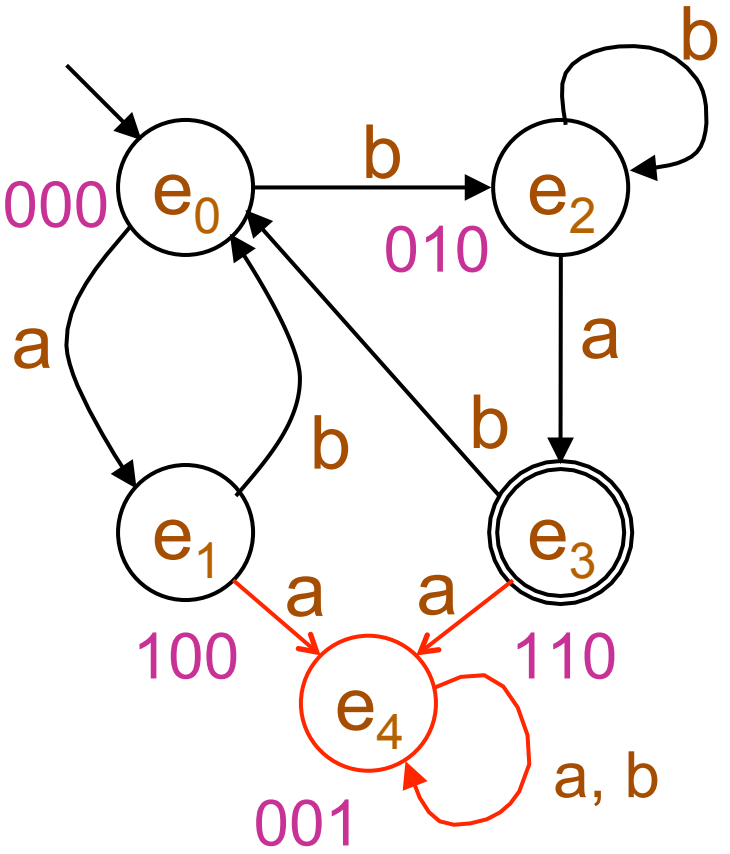
relation de transition T_b : $e \xrightarrow{b} e'$

$$T_b = \bar{x}_0\bar{x}_1\bar{x}'_0x'_1 \vee \bar{x}_0x_1\bar{x}'_0x'_1 \vee x_0\bar{x}_1\bar{x}'_0\bar{x}'_1 \vee x_0x_1\bar{x}'_0\bar{x}'_1$$

En codant a par a et b par b, relation de transition globale :

$$T(a,b,x_0,x_1,x'_0,x'_1) = (a \wedge T_a) \vee (b \wedge T_b)$$

Fonction de transition déterministe

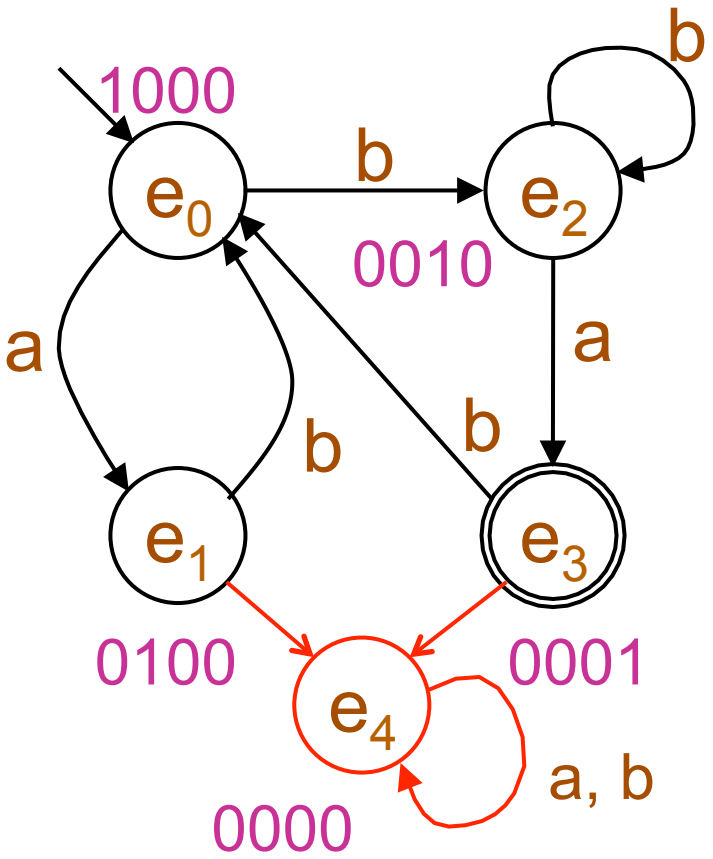


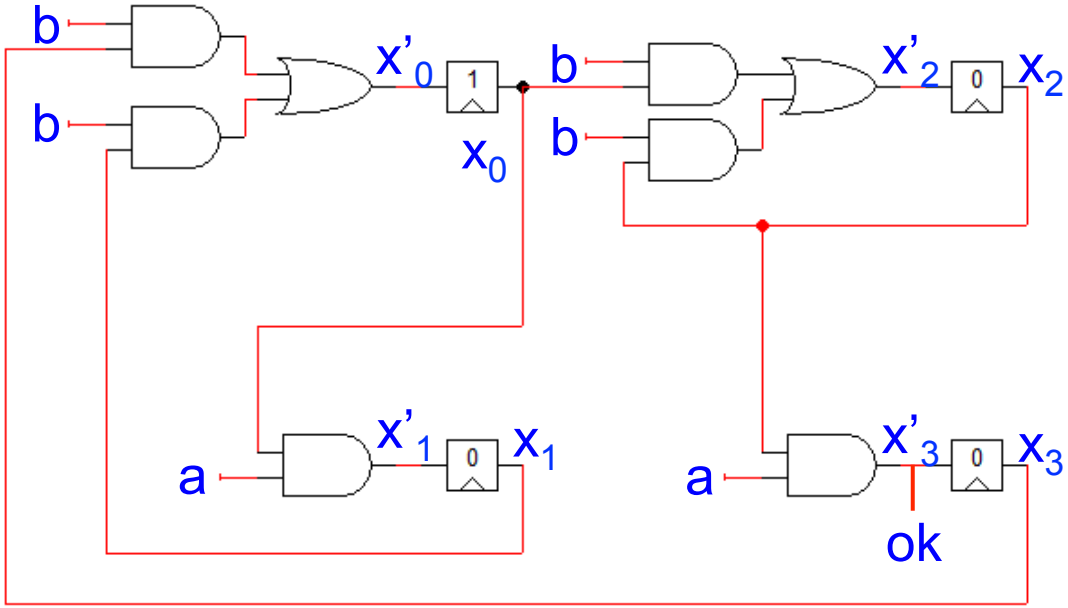
$e_0 = \bar{x}_0\bar{x}_1\bar{x}_2$ initial
$e_1 = x_0\bar{x}_1\bar{x}_2$
$e_2 = \bar{x}_0x_1\bar{x}_2$
$e_3 = x_0x_1\bar{x}_2$ final
$e_a = \bar{x}_0\bar{x}_1x_2$ erreur

Codage booléen de la fonction de transition :

$(x'_0,x'_1,x'_2) = F(x_0,x_1,x_2) = ((e_0 \wedge a) \vee (e_2 \wedge a),$
$(e_0 \wedge b) \vee (e_2 \wedge b) \vee (e_2 \wedge a),$
$(e_1 \wedge a)) \vee (e_3 \wedge a) \vee (e_4 \wedge a) \vee (e_4 \wedge b))$

Circuit one-hot+ de l'automate déterministe

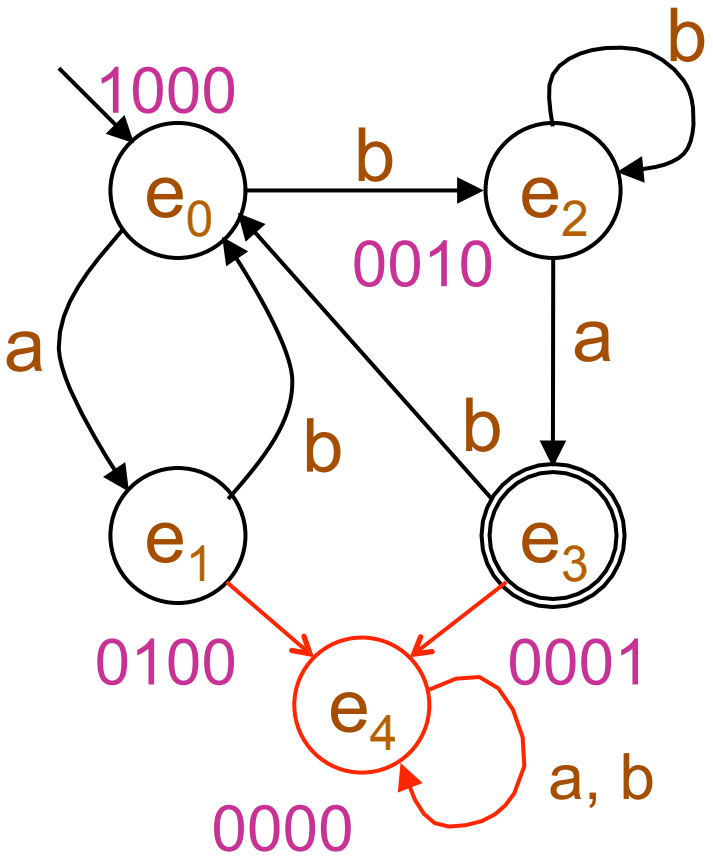


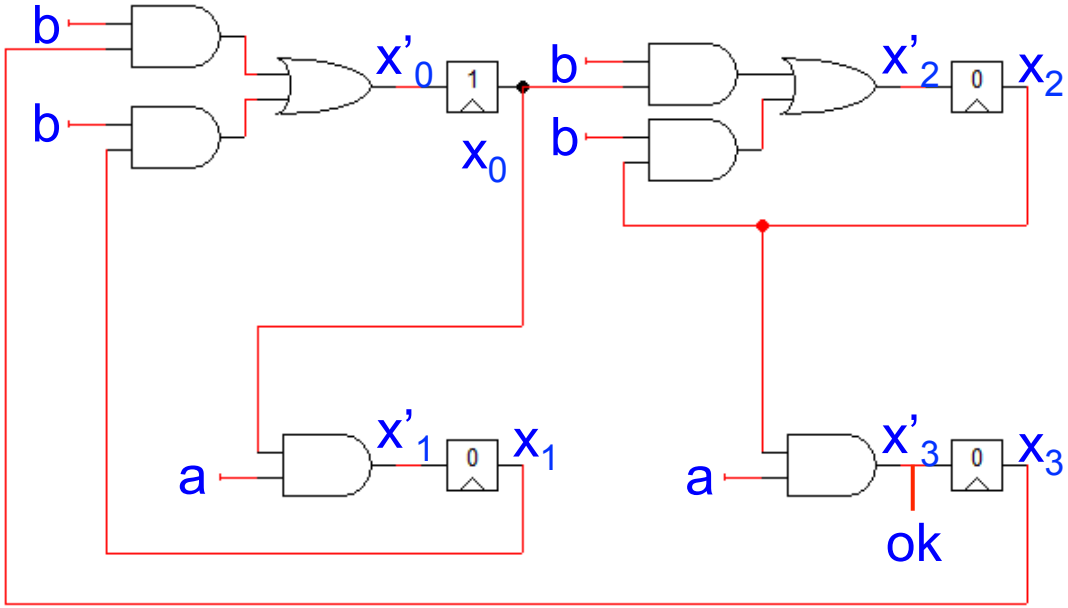


Relation de transition $T(a,b,x_0,x_1,x_2,x_3,x'_0,x'_1,x'_2,x'_3)$ =

$((e_1 \vee e_3) \wedge b \wedge x'_0) \vee (e_0 \wedge a \wedge x'_1) \vee ((e_0 \vee e_2) \wedge b \wedge x'_2) \vee (e_2 \wedge a \wedge x'_3)$

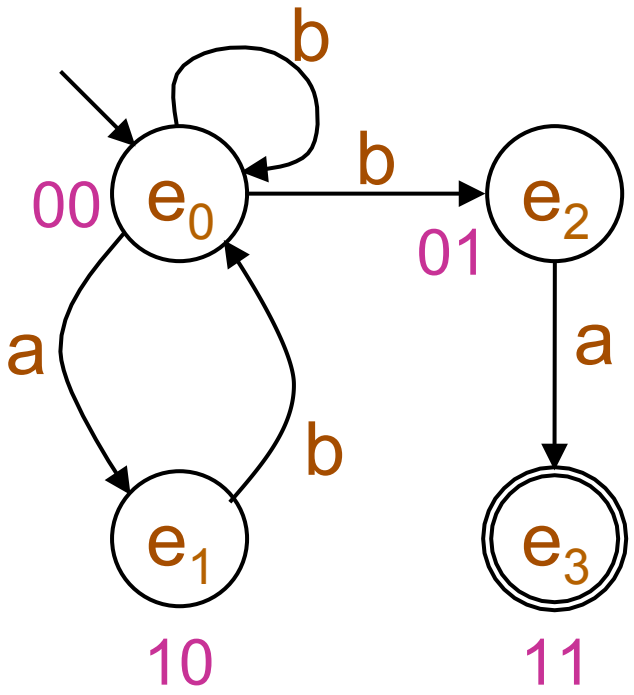
Circuit one-hot+ de l'automate déterministe





Fonction de transition $(x'_0,x'_1,x'_2,x'_3) = F(a,b,x_0,x_1,x_2,x_3) =$
$(((e_1 \vee e_3) \wedge b), (e_0 \wedge a), ((e_0 \vee e_2) \wedge b), (e_2 \wedge a))$

Automate non-déterministe



état initial : $e_0 = \bar{x}_0\bar{x}_1$

état final : $e_3 = x_0x_1$

relation de transition T_a : $e \xrightarrow{a} e'$

$T_a = (e_0 \wedge e'_1) \vee (e_2 \wedge e'_3)$

relation de transition T_b : $e \xrightarrow{b} e'$

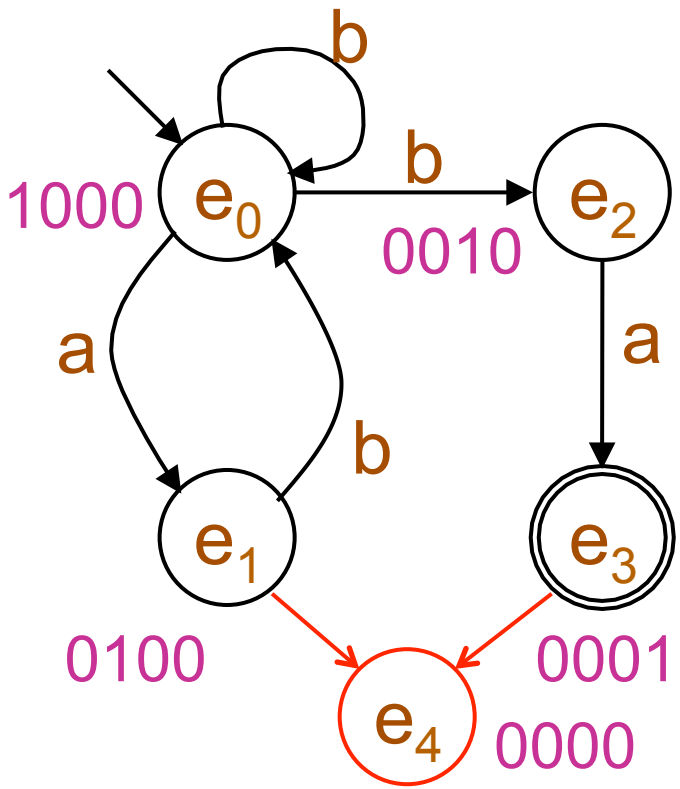
$T_b = (e_0 \wedge e'_0) \vee (e_0 \wedge e'_2) \vee (e_1 \wedge e'_0)$

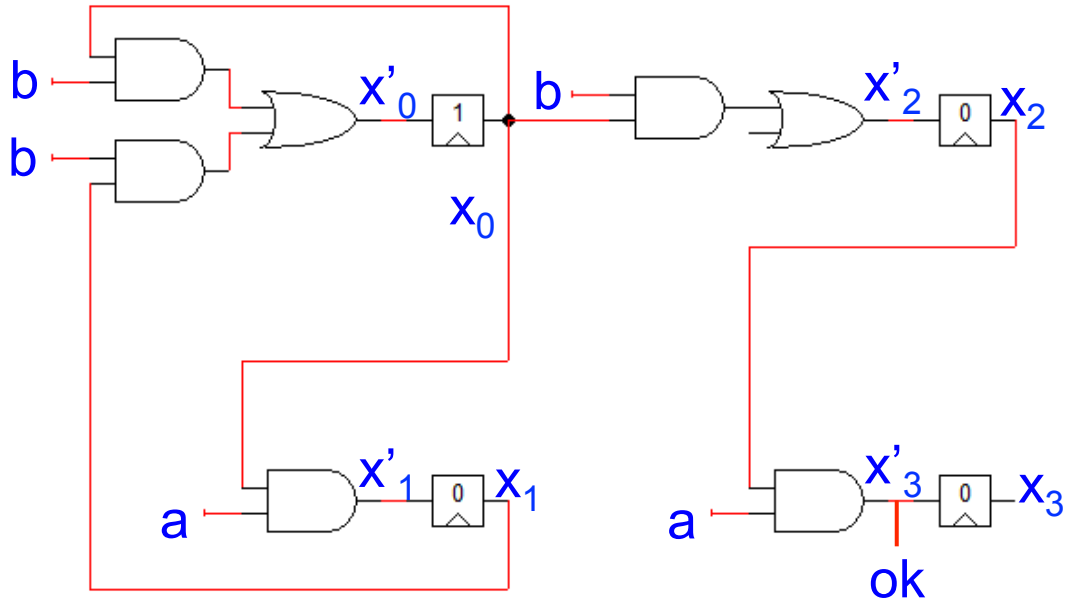
En codant a par a et b par b, relation de transition globale :

$T(a,b,x_0,x_1,x'_0,x'_1) = (a \wedge T_a) \vee (b \wedge T_b)$

Mais plus de fonction de transition !

Codage one-hot+ du non-déterministe





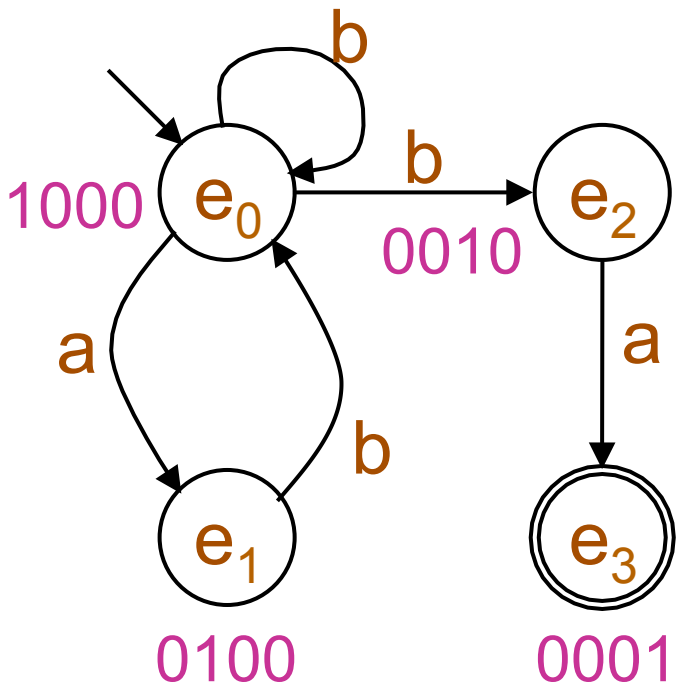
Relation de transition $T(a,b,x_0,x_1,x_2,x_3,x'_0,x'_1,x'_2,x'_3)$ =

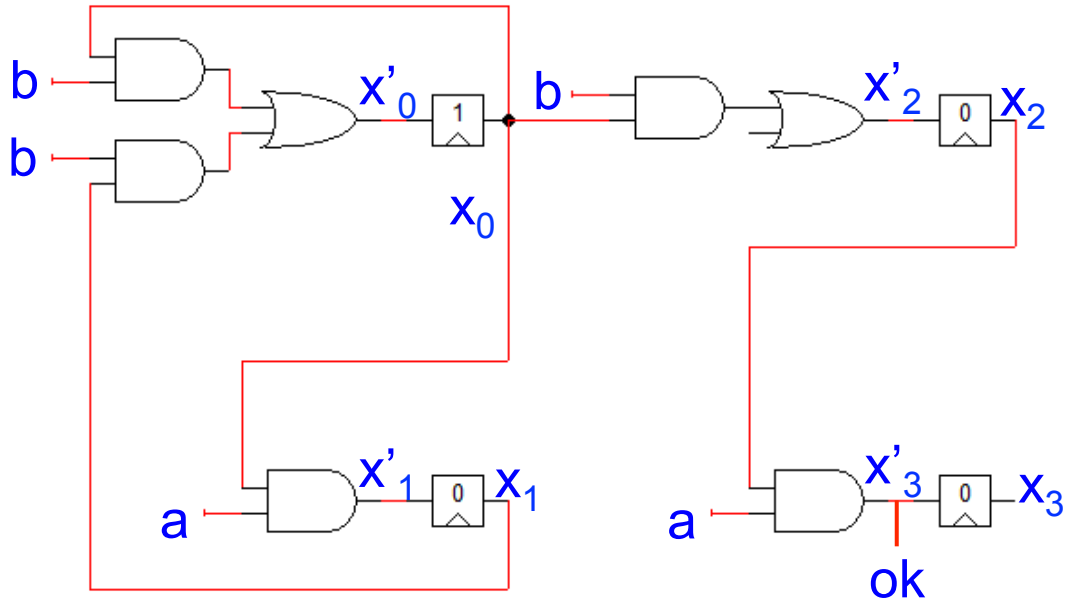
$a \wedge ((e_0 \wedge e'_1) \vee (e_2 \wedge e'_3))$

$\vee\ b \wedge ((e_0 \wedge e'_0) \vee (e_0 \wedge e'_2) \vee (e_1 \wedge e'_0))$

T n'est pas fonctionnelle sur e car $T(\bot,\top,e_0,e_0) \wedge T(\bot,\top,e_0,e_2)$

Le circuit du non-déterministe est déterministe !!





Fonction de transition $(x'_0,x'_1,x'_2,x'_3) = F(a,b,x_0,x_1,x_2,x_3) =$

$((e_0 \wedge b) \vee (e_1 \wedge b) , (e_0 \wedge a) , (e_0 \wedge b) , (e_2 \wedge a))$

De e_0 par b on passe à $e_0 \vee e_2$ puis par a à $e_1 \vee e_3$

→ le circuit peut être dans plusieurs états à la fois

→ déterminisation par *subset construction* électronique

Conséquence immédiate (et peu connue)

L'adjectif « non-déterministe » pour un automate est bien mal choisi. Transformé trivialement en circuit, tout automate « non déterministe » devient automatiquement déterministe, évitant l'explosion exponentielle de la déterminisation explicite

Le modèle de calcul « standard » de la théorie des automates est celui du mathématicien avec un seul crayon
Le parallélisme synchrone de l'électricité est bien meilleur !

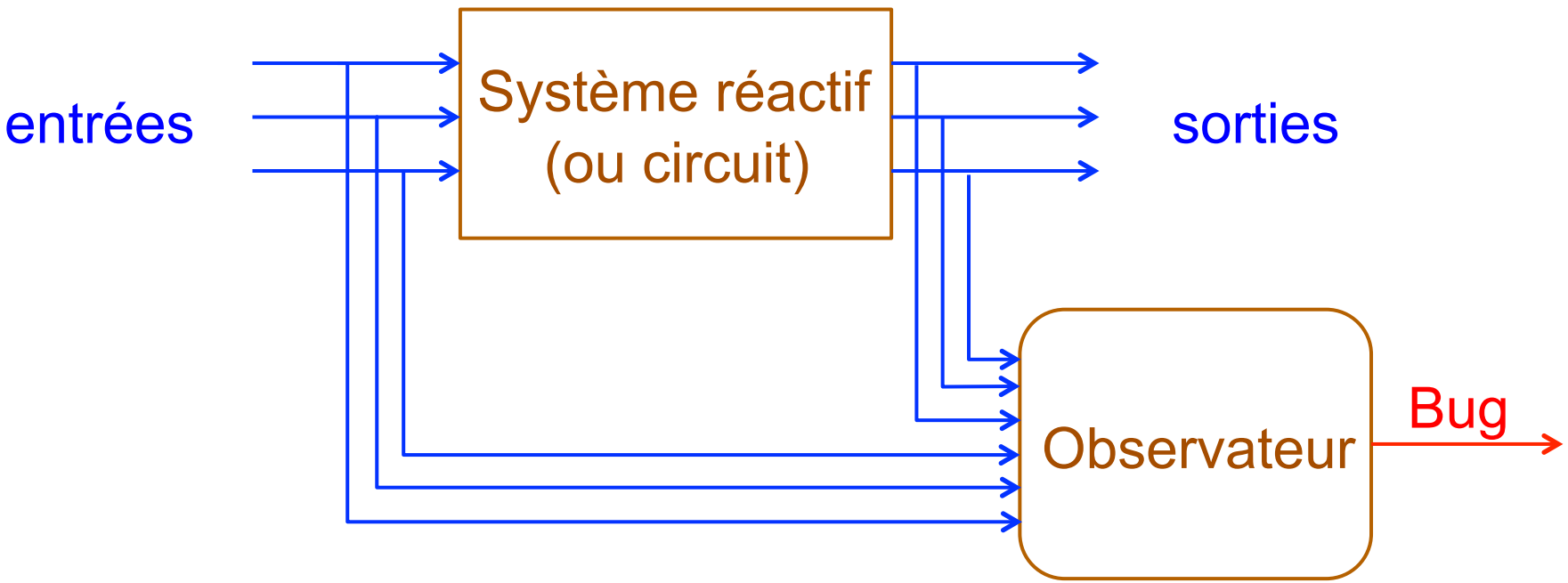
La plupart du temps, déterminiser les automates est exponentiellement inutile

Cf. cours du 16 décembre 2009
http://www.college-de-france.fr/site/gerard-berry/course-2009-12-16-10h00.htm

Agenda

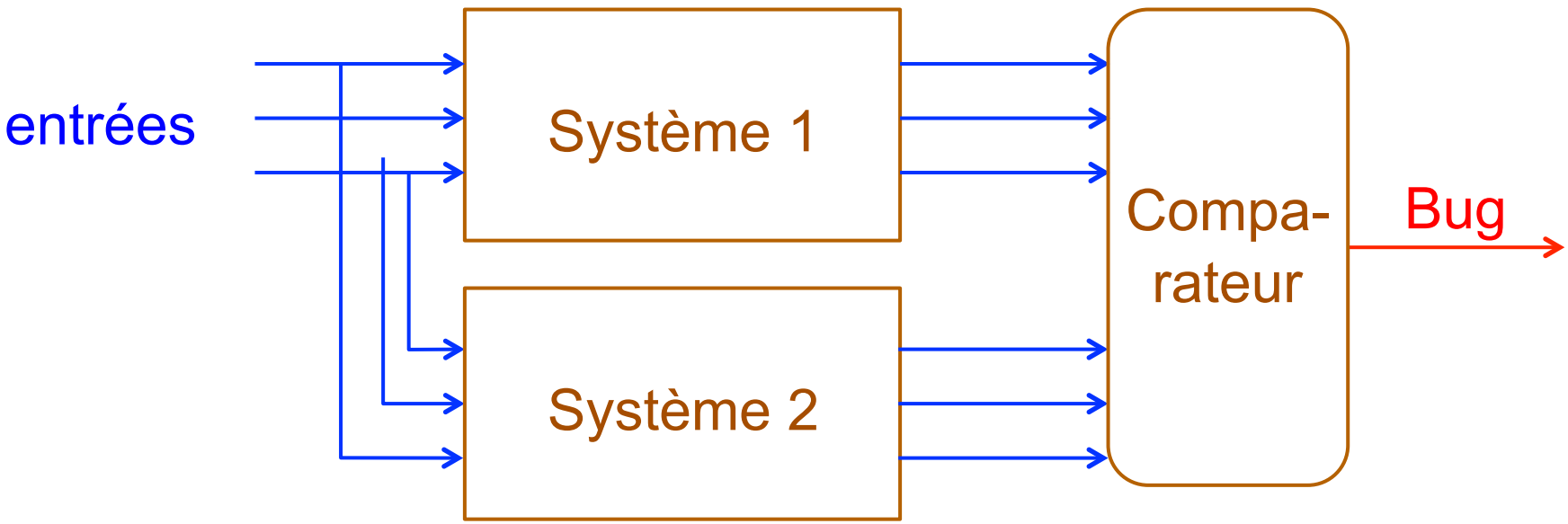
1. Codages booléens d'ensembles
2. Codages d'automates et de circuits
3. Model-checking booléen
4. Des arbres de Shannon aux BDDs
5. Optimisation de circuits Esterel
6. Conclusion

Vérification booléenne de sûreté



Formule booléenne exprimant que Bug n'est jamais émis :
pour tout état accessible R, O et toute entrée valide E
du couple système / observateur, le booléen Bug est faux.

Vérification booléenne d'équivalence



Formule booléenne exprimant que Bug n'est jamais émis :
pour tout état accessible R1, R2 et toute entrée valide E
du couple de systèmes, le booléen Bug est faux.

Préliminaires booléens

1. Les quantificateurs booléens sont définissables

   $\forall x.\ f(x,y) = f(\bot,y) \wedge f(\top,y)$

   $\exists x.\ f(x,y) = f(\bot,y) \vee f(\top,y)$

   Mais attention à l'explosion exponentielle !

   $\forall x,y.\ \exists z,t.\ \forall u.\ f(x,y,z,t,u) \quad \rightarrow 2^5 = 32$ termes !

2. Image d'un ensemble par une fonction

   Soit $f : E \rightarrow E$ codée par $(y_0,\ldots,y_n) = F(x_0,\ldots,x_n)$

   Soit $P \subseteq E$ codé par $P(x_0,\ldots,x_n)$

   L'image de P par f est $f(P) = \{\ y \in E \mid \exists x \in P.\ y = f(x)\ \}$

   Alors $f(P)$ peut être caractérisée par le prédicat suivant :

   $F(P)\ (y_0,\ldots,y_n) = \exists x_0,\ldots,x_n.\ P(x_0,\ldots,x_n) \wedge ((y_0,\ldots,y_n) \Leftrightarrow F(x_0,\ldots,x_n))$

   Mais attention à l'explosion exponentielle !

Préliminaires booléens

3. Image inverse d'un ensemble par une fonction

Soit $f : E \rightarrow E$ codée par $(y_0,\ldots,y_n) = F(x_0,\ldots,x_n)$

Soit $P \subseteq E$ codé par $P(x_0,\ldots,x_n)$

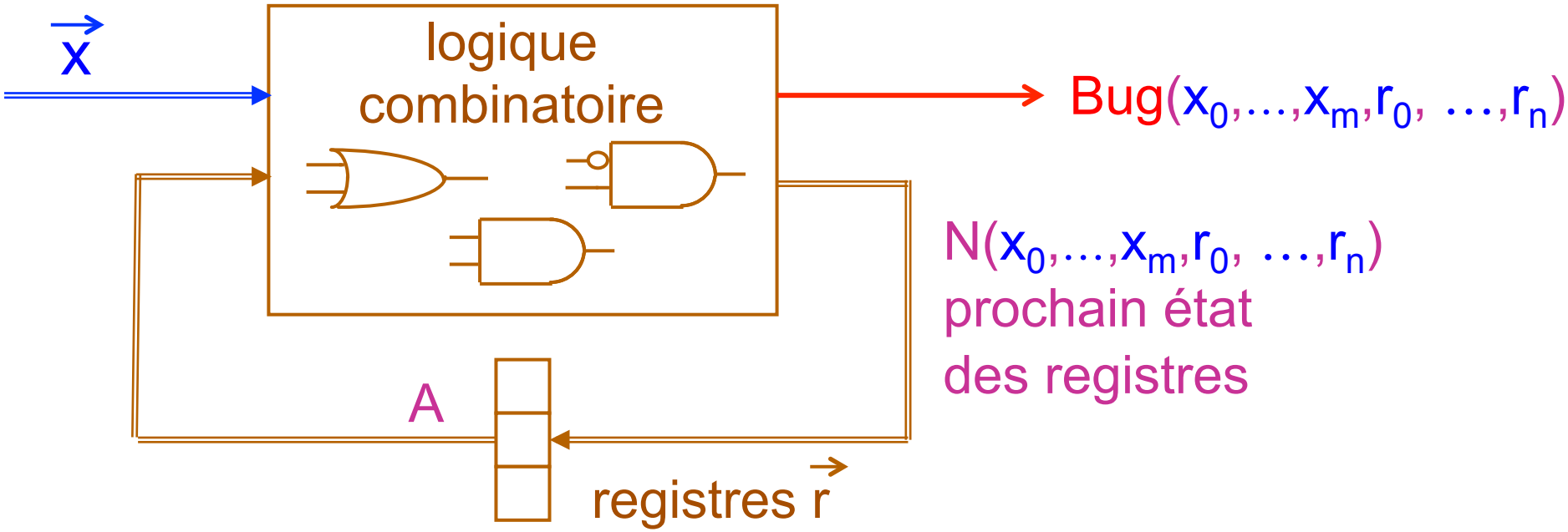
L'image inverse de P par f est $f^{-1}(P) = \{ x \in E \mid f(x) \in P \}$

Alors $f^{-1}(P)$ peut être caractérisée par le prédicat suivant :

$$F^{-1}(P)(x_0,\ldots,x_n) = \exists\, y_0,\ldots,y_n.\ ((y_0,\ldots,y_n) \Leftrightarrow F(x_0,\ldots,x_n))) \wedge P(y_0,\ldots,y_n)$$

Mais attention à l'explosion exponentielle !

Principe de la vérification booléenne



Prédicat de validité des entrées : $E(x_0,\ldots,x_m)$

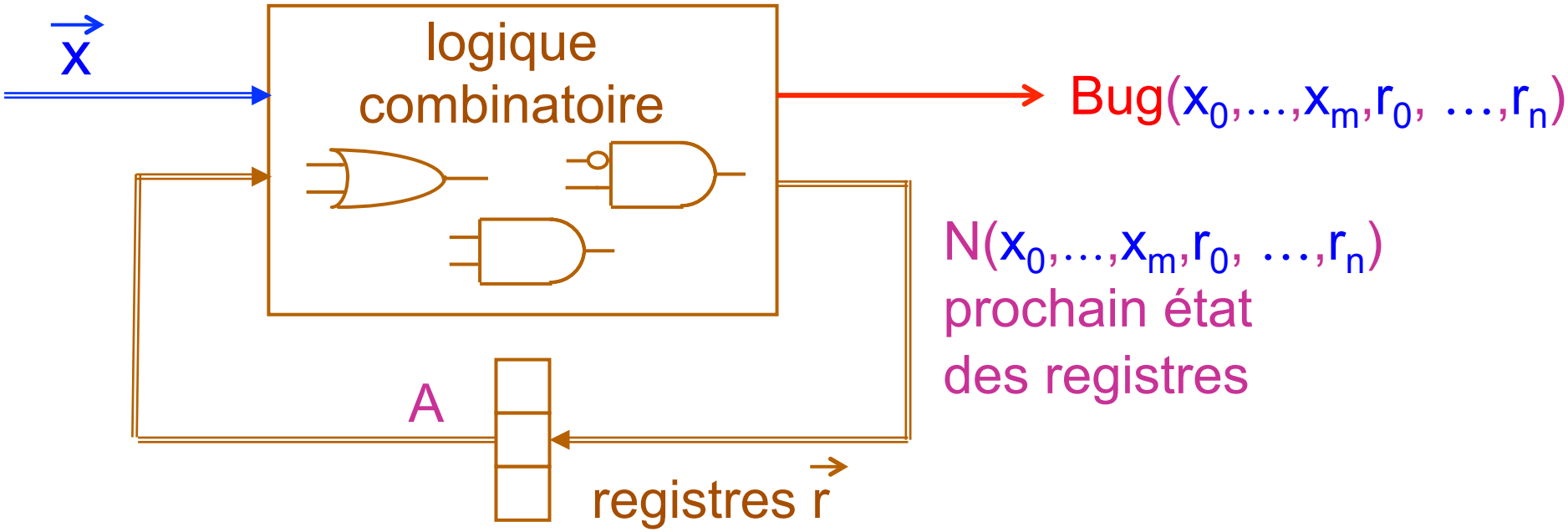
Prédicat d'accessibilité des états : $A(r_0,\ldots,r_n)$

- Formule à vérifier pour que Bug ne passe jamais à 1 :

$\neg(\exists x_0,\ldots,x_m,r_0,\ldots,r_n.\ E(x_0,\ldots,x_m) \wedge A(r_0,\ldots,r_n) \wedge \text{Bug}(x_0,\ldots,x_m,r_0,\ldots,r_n))$

Faire vraiment très attention à l'exponentielle !

Calcul en avant des états accessibles



Etat initial : A_0

Nouveaux états 1 : $A'_1 = N(E,A_0) \wedge \neg A_0$, $A_1 = A_0 \vee A'_1$

Nouveaux états n+1 : $A'_{n+1} = N(E,A'_n) \wedge \neg A_n$, $A_{n+1} = A_n \vee A'_{n+1}$

Test d'arrêt : $\neg A'_{n+1}$

Résultat : $A = A_n$

Construction en arrière de contre-exemple

- Formule à vérifier pour que Bug ne passe jamais à 1 :

  $\neg(\exists x_0,\ldots,x_m,r_0,\ldots,r_n.\ E(x_0,\ldots,x_m) \wedge A(r_0,\ldots,r_n) \wedge Bug(x_0,\ldots,x_m,r_0,\ldots,r_n))$

- Si la formule est fausse, $E \wedge A \wedge Bug$ est satisfiable
  - chercher le i minimal tel que $E \wedge A'_i \wedge Bug$ est satisfiable; le contre exemple aura la forme $(a, e_0;e_1;\ldots;e_i)$ où a est un vecteur d'état initial et les e_j sont des vecteurs d'entrée valides
  - choisir des vecteurs de valeurs e_i,a_i satisfaisant $E \wedge A'_i \wedge Bug$
  - si $i=0$, le contre exemple est (a_0,e_0)
  - sinon, calculer l'image inverse (Y,S) de a_i par N, puis choisir des vecteurs e_{i-1},a_{i-1} dans l'ensemble $(E \wedge Y, A'_{i-1} \wedge S)$, nécessairement non vide
  - Construire récursivement $(a_0, e_0;e_1;\ldots;e_{i-1})$ et retourner $(a_0, e_0;e_1;\ldots;e_{i-1};e_i)$

Sommaire des calculs booléens nécessaires

1. Construction des formules calculant les sorties et transitions d'états (compilation)
2. Constructions des formules de validité des entrées et de l'état initial (compilation)
3. Itération d'images directes pour les états accessibles
4. Vérification de la formule. Si elle est fausse, construire le contre-exemple à l'aide d'images inverses et de choix de valeurs rendant une formule vraie

Tous les calculs d'images directes ou inverses doivent être soigneusement optimisés : O. Coudert et J-C. Madre (ceci sera expliqué dans le cours 2015-2016)

Agenda

1. Codages booléens d'ensembles
2. Codages d'automates et de circuits
3. Model-checking booléen
4. Des arbres de Shannon aux BDDs
5. Optimisation de circuits Esterel
6. Conclusion

L'invention des BDDs

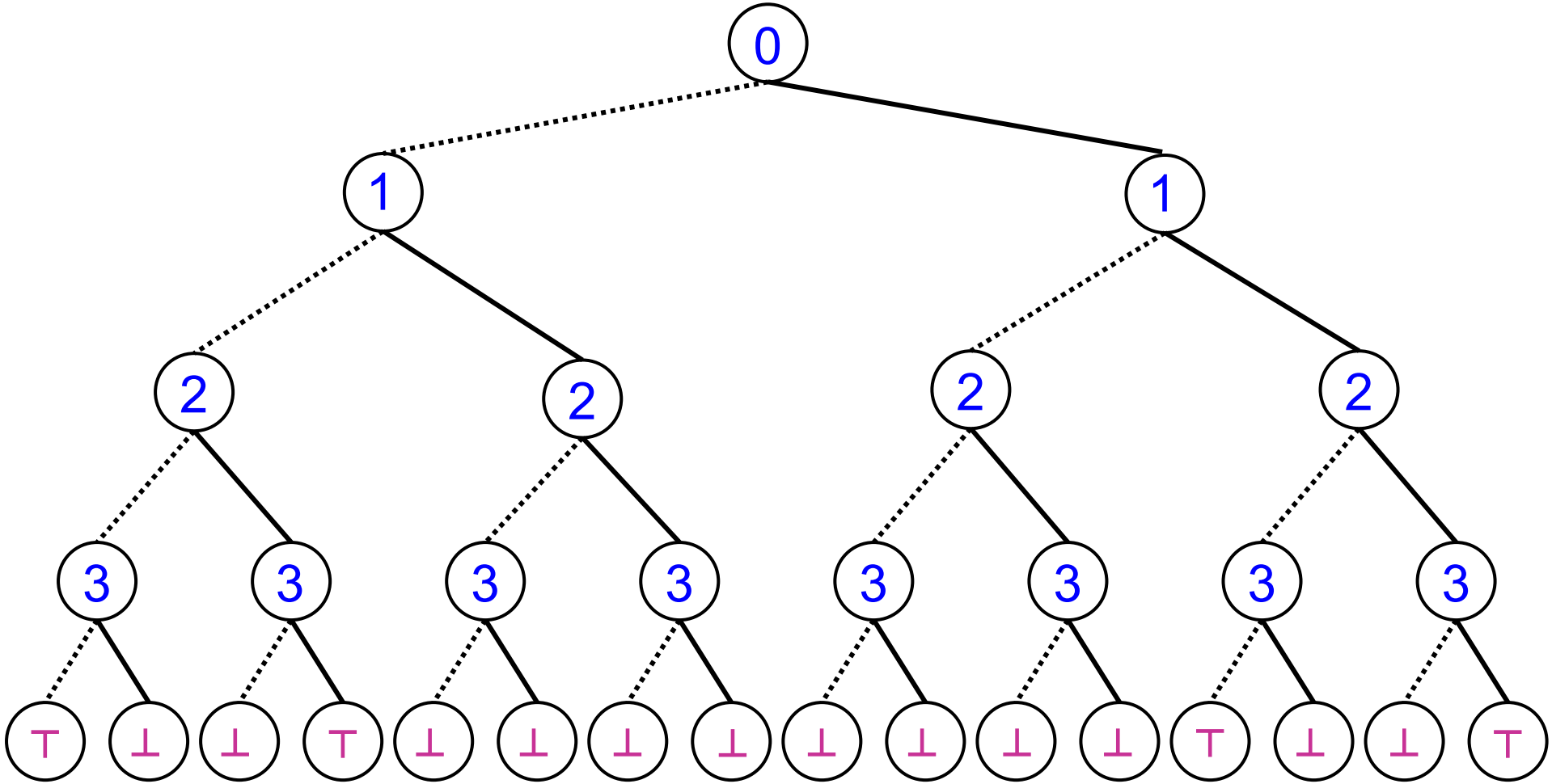
Tout cela paraissait impossible en pratique
jusqu'à la fin des années 1980...
...et l'invention des BDDs et leurs implémentations efficaces

- Premières idées : Shannon, Lee (1959), Akers (1978), Boute (1976)
- Vraie exploitation, structures de données : Bryant (1986), Billion et Billon & Madre (1988), Coudert & Madre (1990)
- Systèmes : TiGeR (Madre, Coudert, Touati) → Esterel
  CUDD (Somenzi)
  etc.

Référence moderne : D. Knuth vol. 4, voir bibliographie

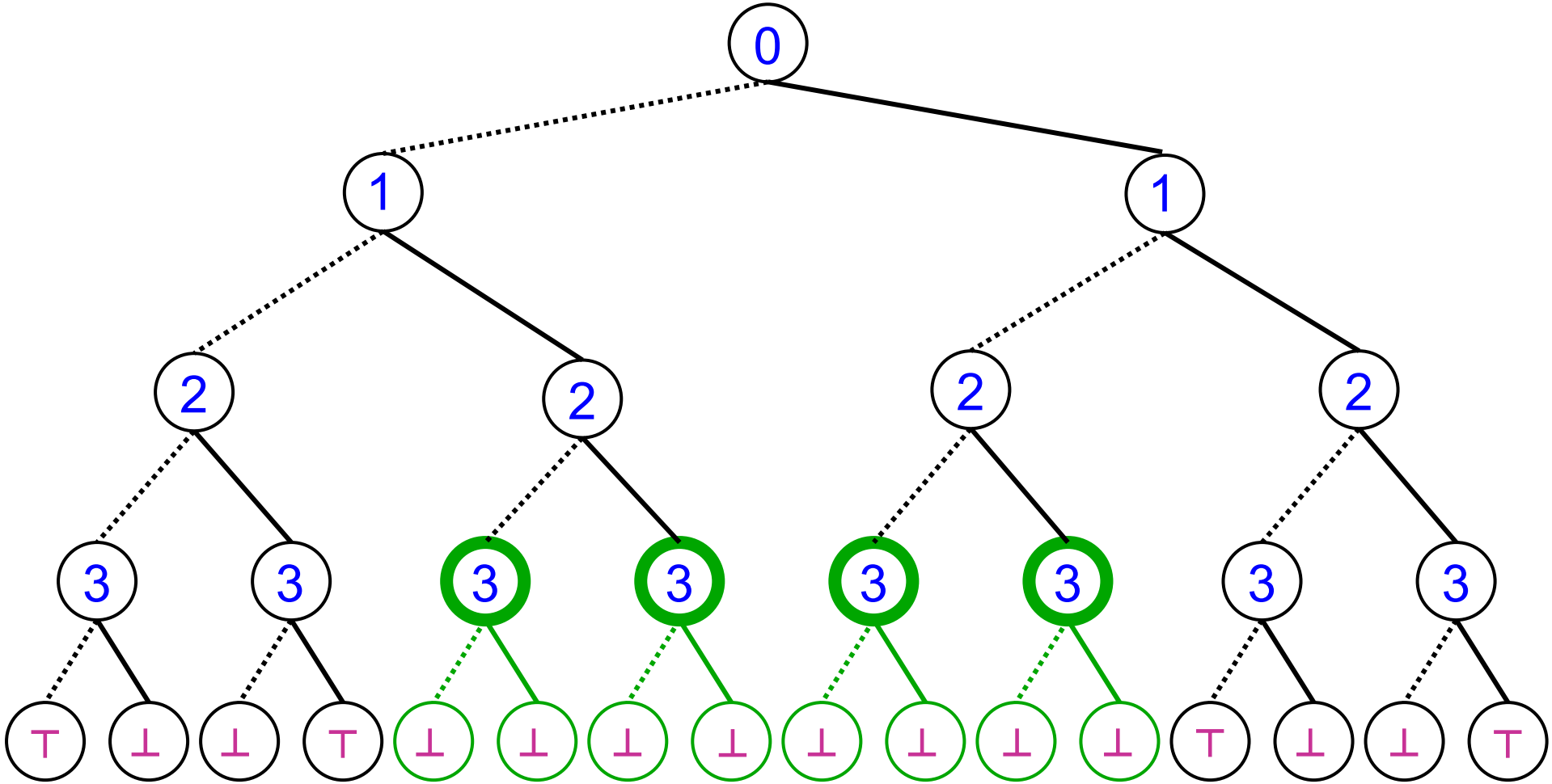
Codage d'une fonction par un arbre de Shannon

$$f(x_0,x_1,x_2,x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$$



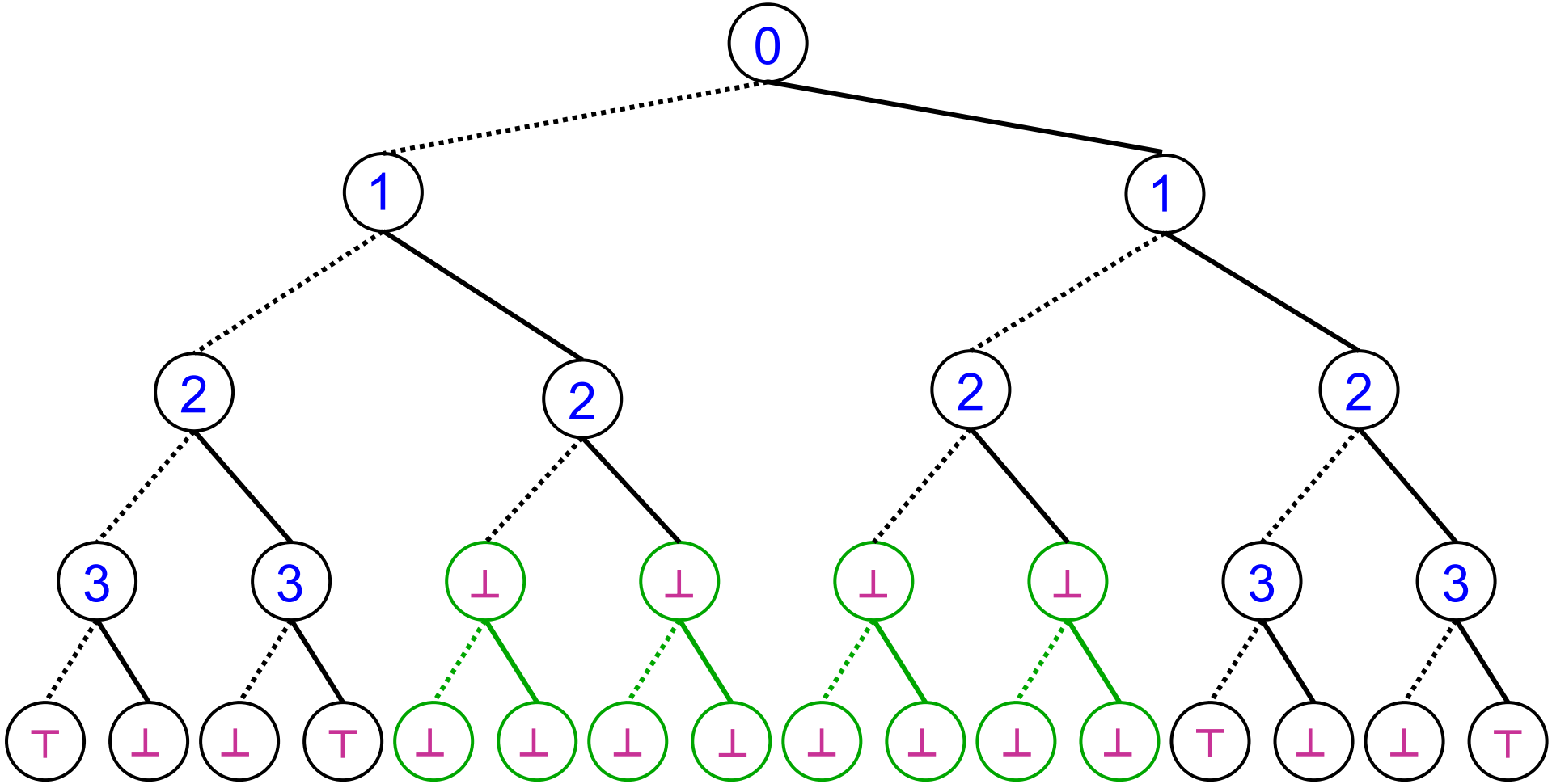
Suppression des nœuds redondants

$f(x_0,x_1,x_2,x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$



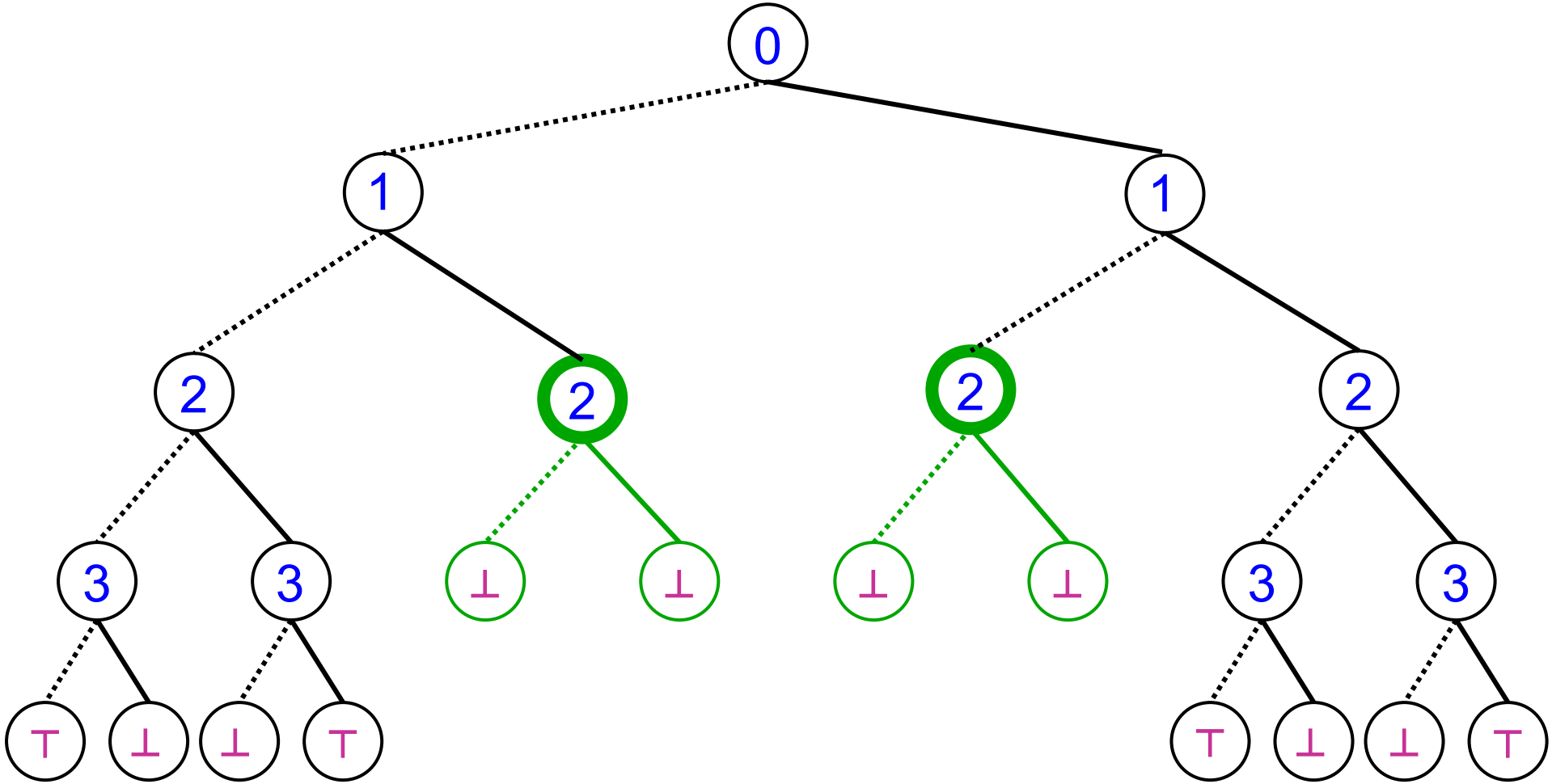
Suppression des nœuds redondants

$f(x_0,x_1,x_2,x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$



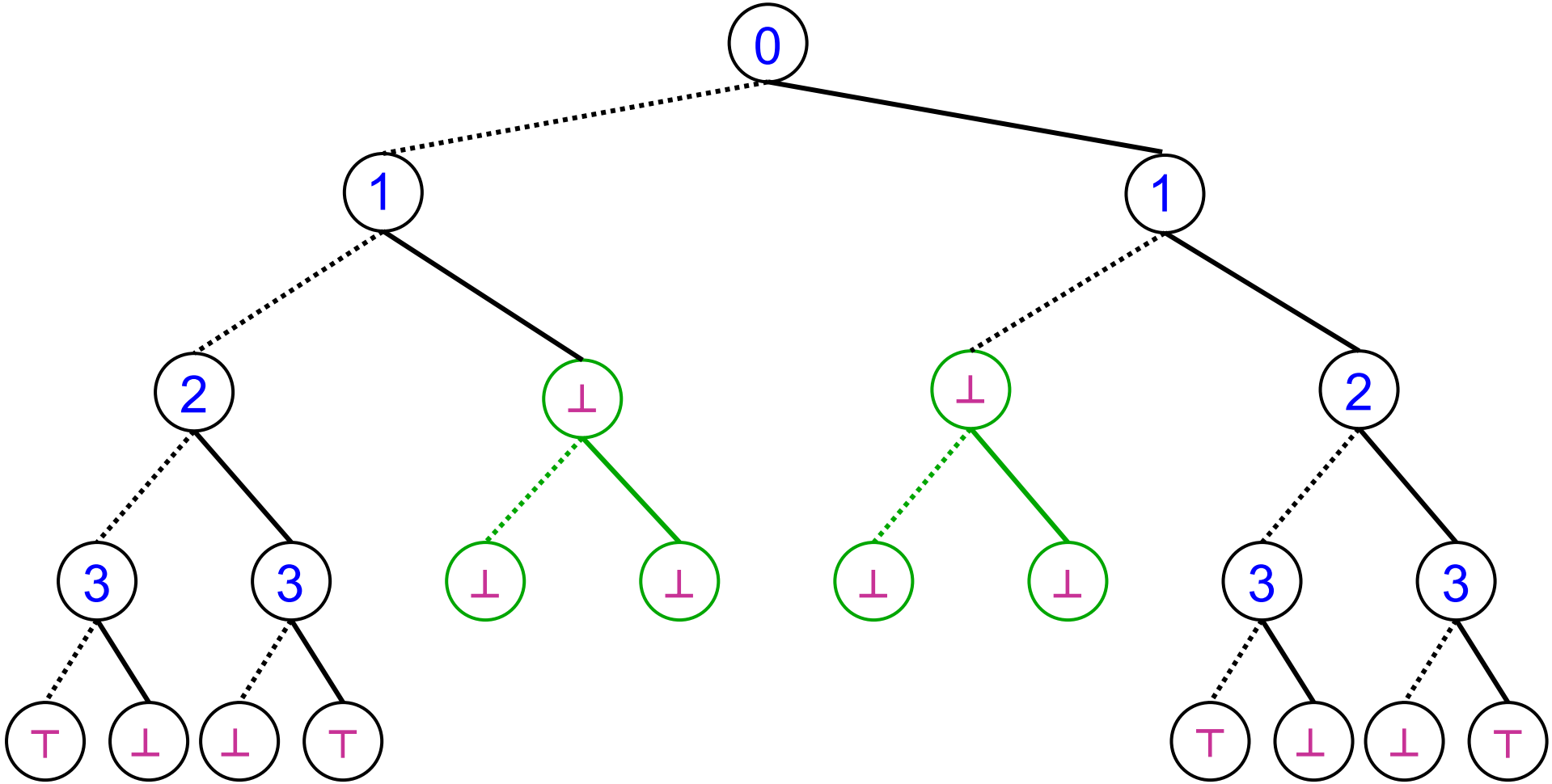
Suppression des nœuds redondants

$f(x_0,x_1,x_2,x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$



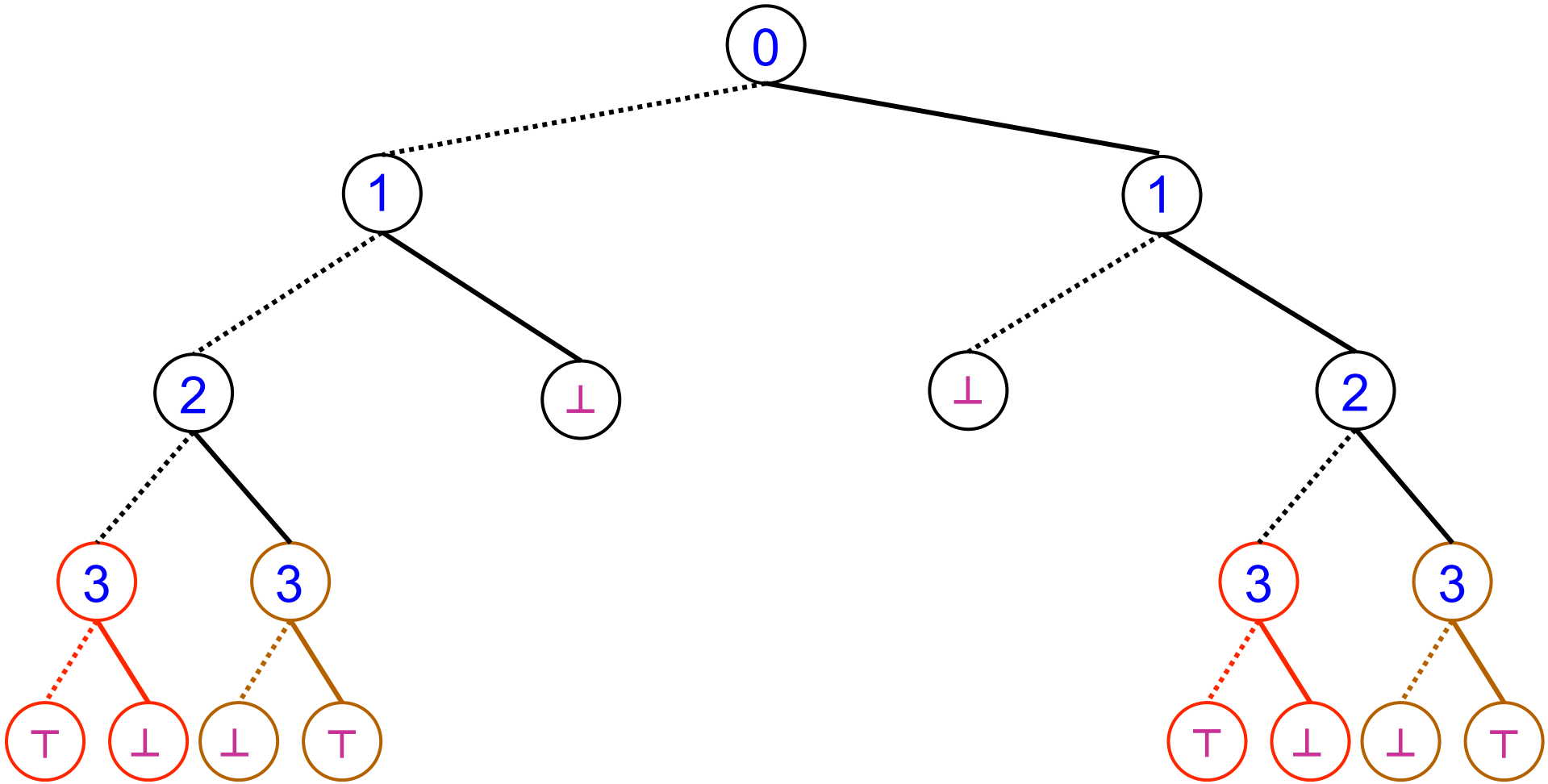
Suppression des nœuds redondants

$f(x_0,x_1,x_2,x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$



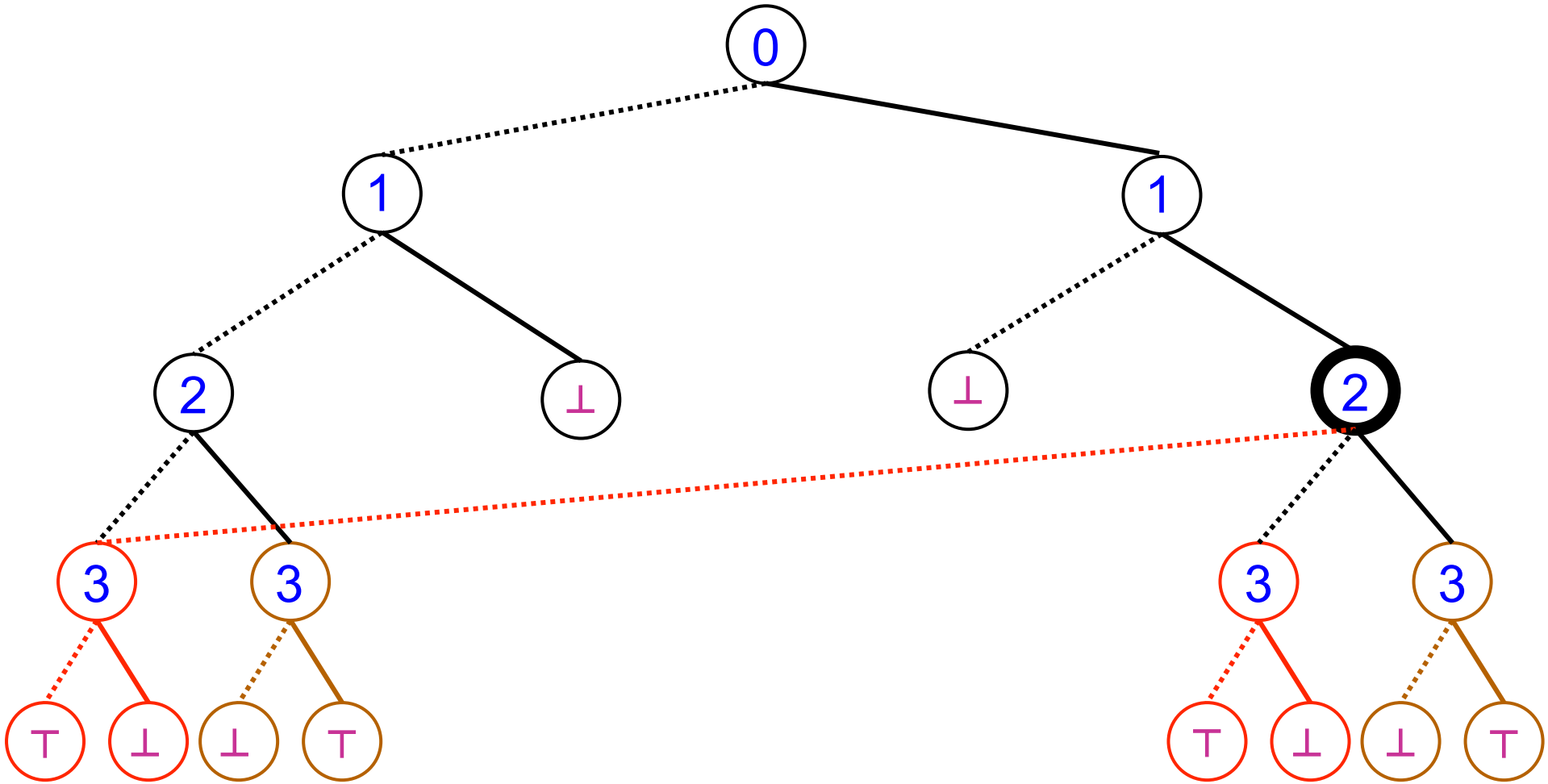
Partage des sous-arbres identiques

$f(x_0,x_1,x_2,x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$



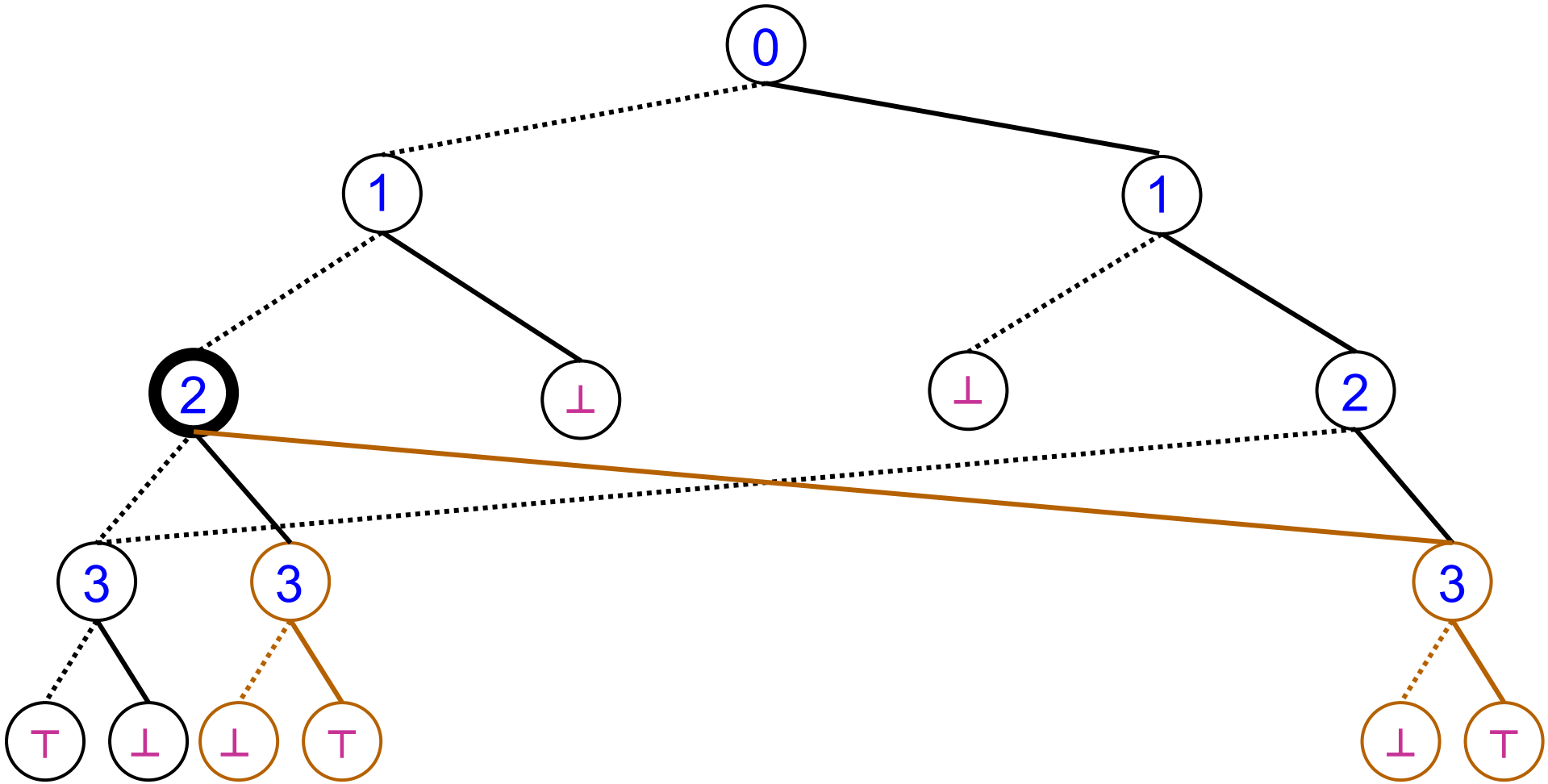
Partage des sous-arbres identiques

$f(x_0,x_1,x_2,x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$



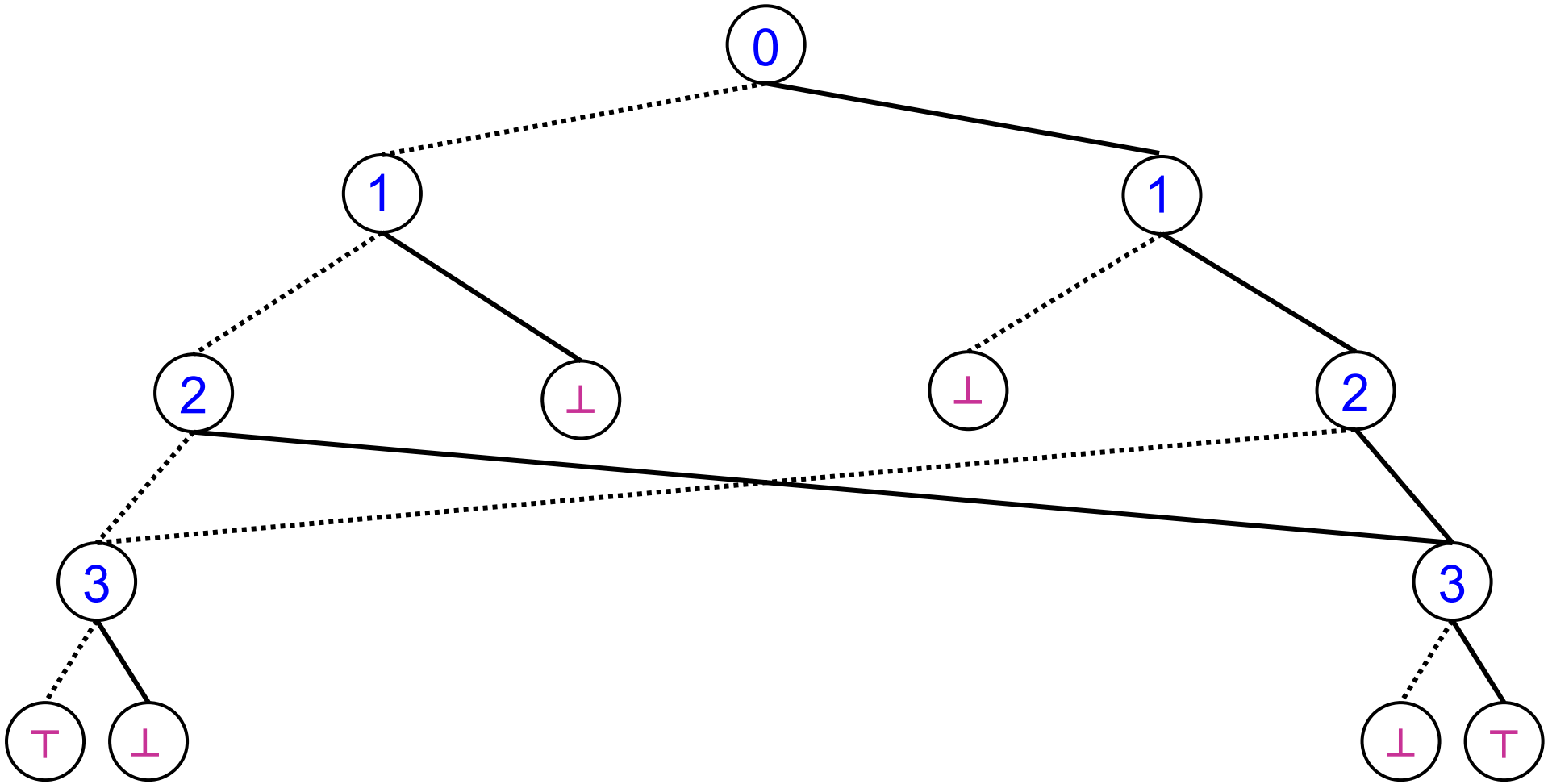
Partage des sous-arbres identiques

$f(x_0,x_1,x_2,x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$



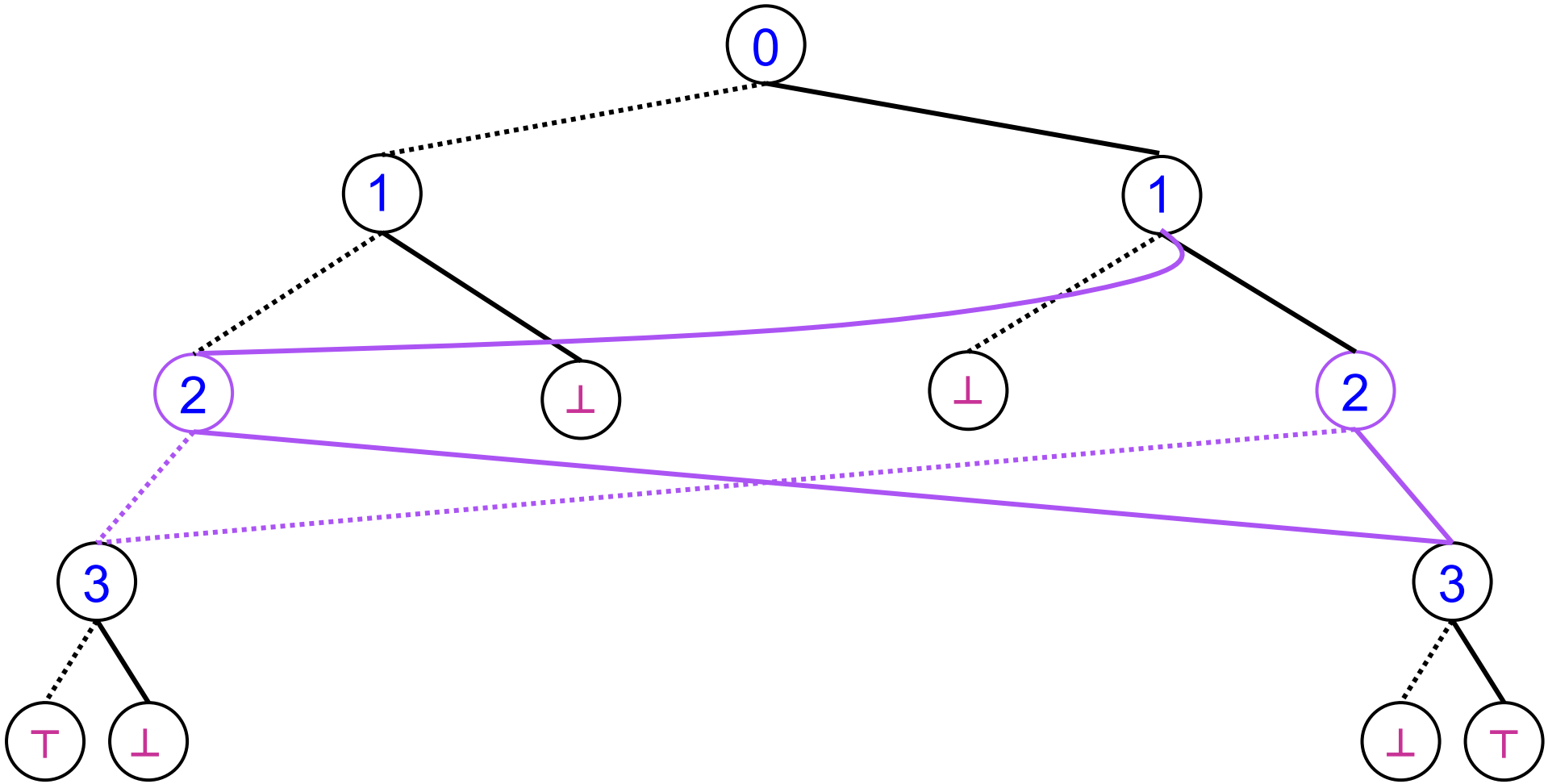
Partage des sous-arbres identiques

$f(x_0,x_1,x_2,x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$



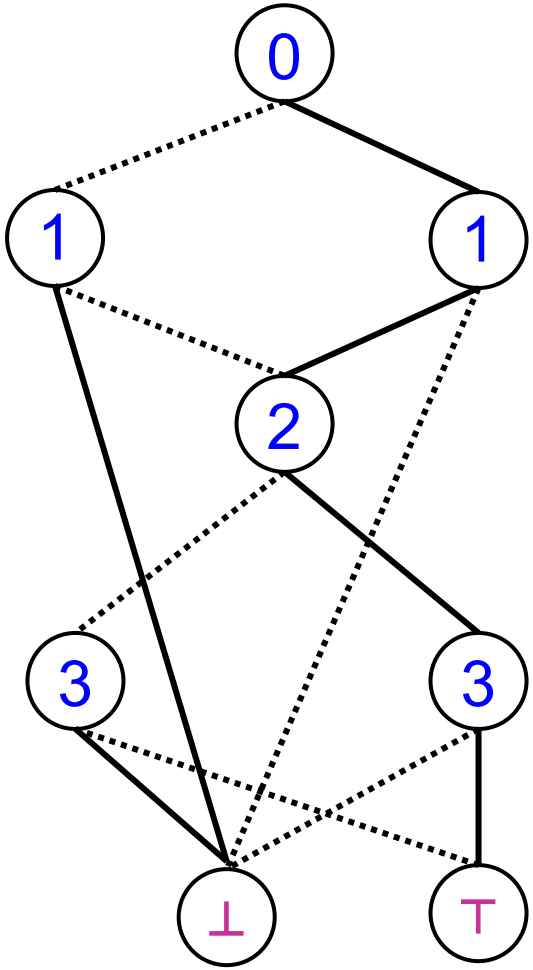
Partage des sous-arbres identiques

$f(x_0,x_1,x_2,x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$



Un peu de rangement

$f(x_0,x_1,x_2,x_3) = (x_0 \Leftrightarrow x_1) \wedge (x_2 \Leftrightarrow x_3)$



forme canonique
pour un ordre de variables

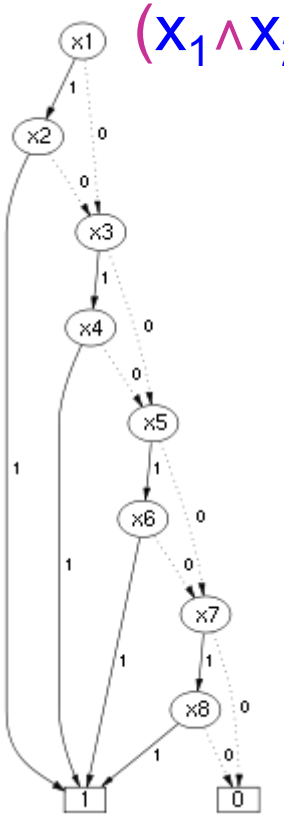
test d'égalité en temps 1
(comparaison de pointeurs)

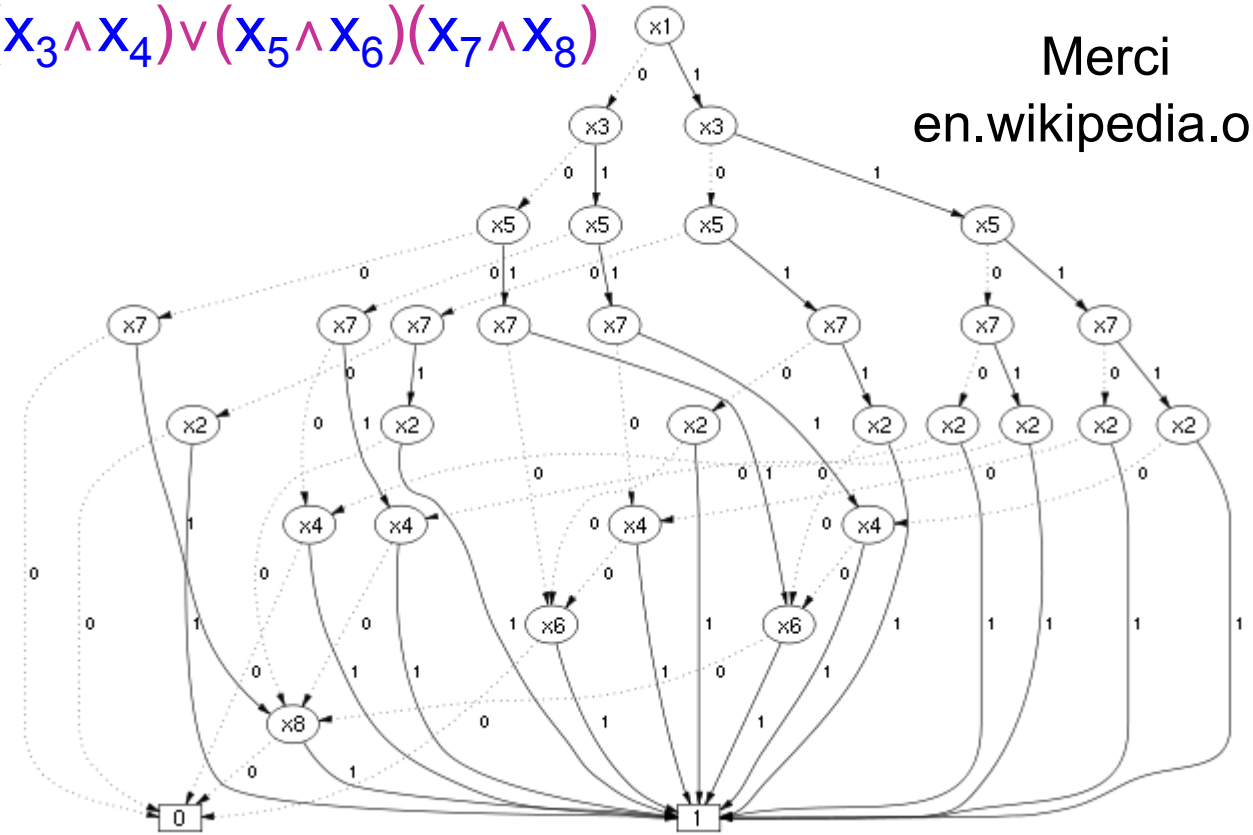
comptage des solutions trivial,
énumération possible
(les chemins vers ⊤)

opérations booléennes
rapides (polynomiales)

Mais forte dépendance à l'ordre des variables

$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee (x_5 \wedge x_6)(x_7 \wedge x_8)$





Merci
en.wikipedia.org

Trouver le bon ordre est NP-Complet →heuristiques

Agenda

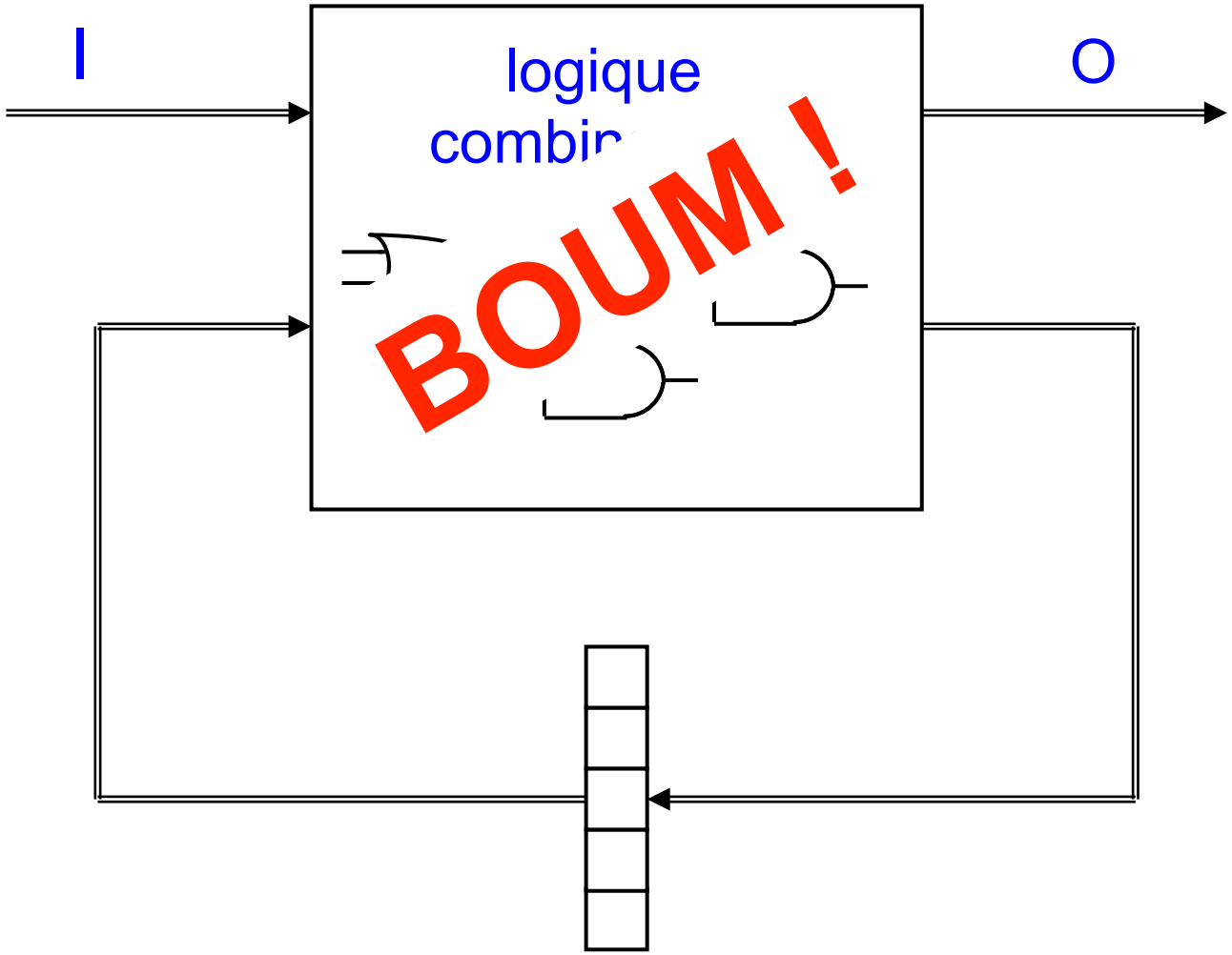
1. Codages booléens d'ensembles
2. Codages d'automates et de circuits
3. Model-checking booléen
4. Des arbres de Shannon aux BDDs
5. Optimisation de circuits Esterel
6. Conclusion

Codage des états d'un automate

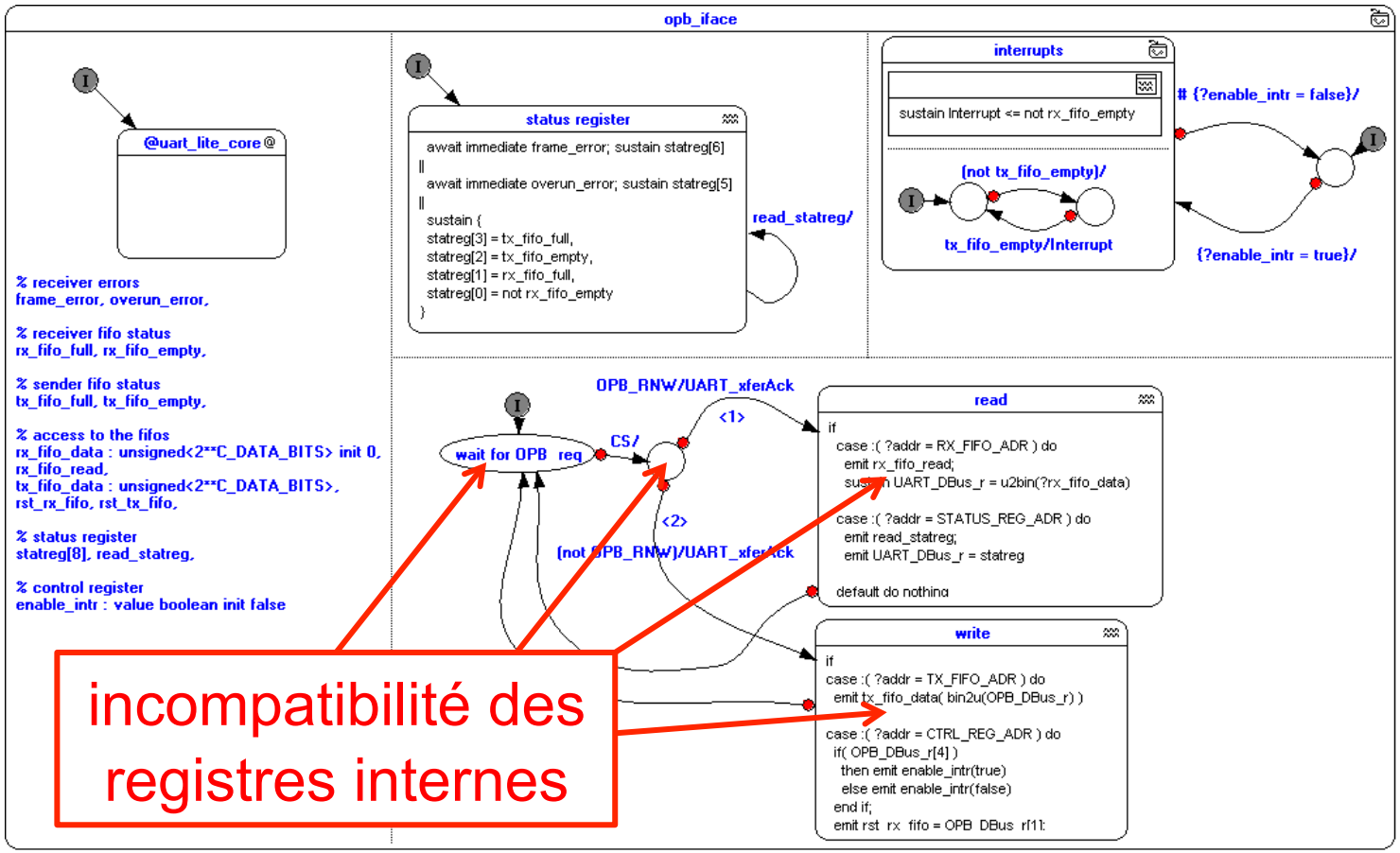
- One-hot, un registre par état : ne passe pas à l'échelle
- Dense (binaire), $\log_2(n)$ registres pour n états :
  ne passe pas à non plus à l'échelle, mais raison plus subtile
  1. codage dense = numérotation des états
  2. la fonction de transition peut exploser en portes logiques
  3. pour n états, !n numérotations possibles
  4. pas d'heuristique connue pour en choisir une bonne !
- Exemple : circuit à 15 états chez Intel (protocole mémoire)
  optimiseurs du commerce:
  - A : pas de réponse
  - B : en quelques heures, logique trop grosse
  - Esterel : immédiat, excellent compromis registres / logique

15 regs trop gros, 4 regs logique trop grosse, 6 regs ok!

Ne pas trop réduire les registres



Exploiter parallélisme, séquence et hiérarchie



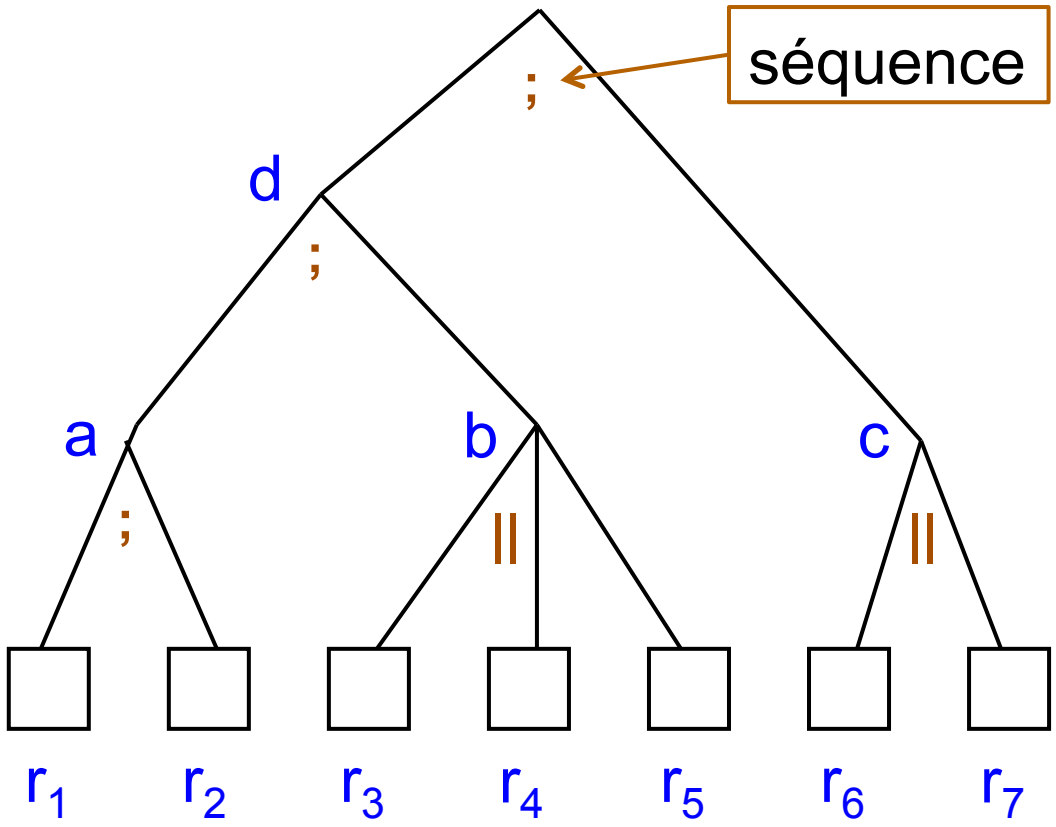
La qualité du langage source
est fondamentale pour l'optimisation
Esterel a toujours battu les designs manuels !

Sur-approximer les états accessibles

$x \# y = \neg(x \wedge y)$

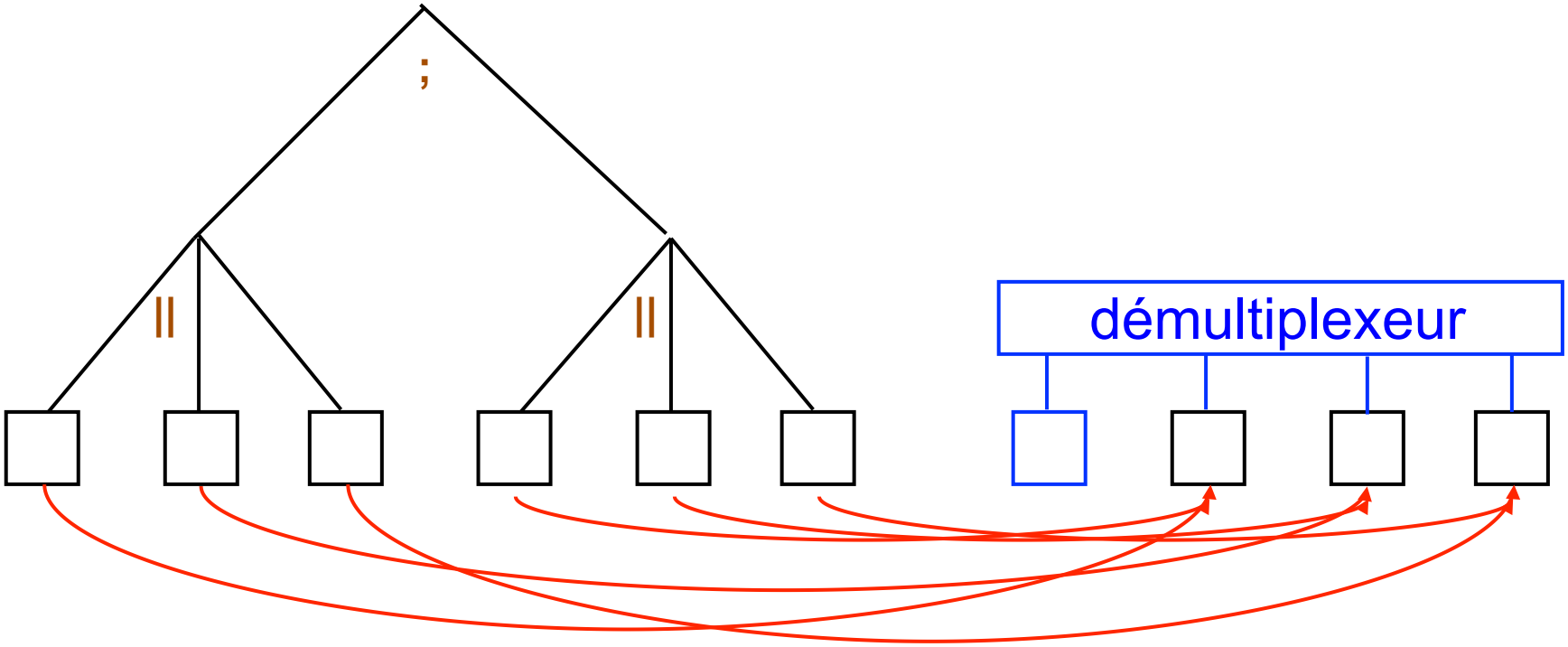
sur-approximation des états accessibles :

let $a = r_1 \vee r_2$
and $b = r_3 \vee r_4 \vee r_5$
and $c = r_6 \vee r_7$
and $d = a \vee b$ in
$(r1 \# r2)$
$\wedge\ (a \# b)$
$\wedge\ (d \# c)$



Peux suffire à prouver des propriétés
Simplifie toujours les calculs à très faible coût

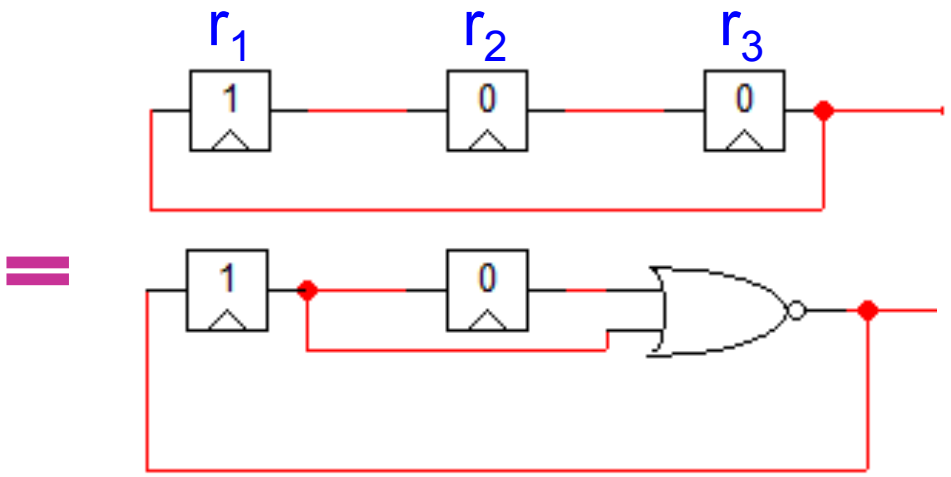
Multiplexage de registres incompatibles



efficacité non garantie...

Remplacement de registres par de la logique

Un registre est redondant si, dans tout état accessible, sa valeur est fonction des valeurs des autres registres

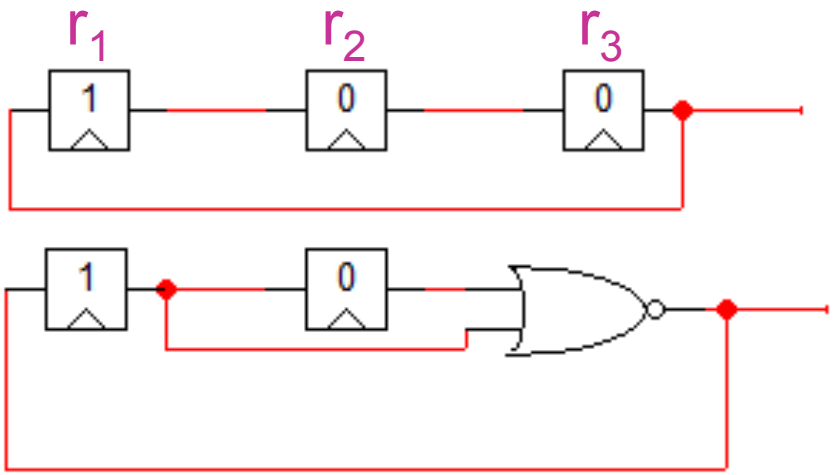


$r_3 \Leftrightarrow \neg(r_1 \vee r_2)$

Etonnamment facile à calculer :

- r_0 est fonction de $r_1,\ldots,r_n$ dans tout état accessible
  ssi $\neg(A(\bot,r_1,\ldots,r_n) \wedge A(\top, r_1,\ldots,r_n))$
- la fonction s'écrit alors $A(\top,r_1,\ldots,r_n)$
  ou $\neg\ A(\bot,r_1,\ldots,r_n)$, au choix

Remplacement de registres par de la logique



$A(r_1,r_2,r_3) = r_1\bar{r}_2\bar{r}_3 \vee \bar{r}_1 r_2 \bar{r}_3 \vee \bar{r}_1\bar{r}_2 r_3$

$A(\bot,r_2,r_3) = r_2\bar{r}_3 \vee \bar{r}_2 r_3$

$A(\top,r_2,r_3) = \bar{r}_2\bar{r}_3$

$A(\bot,r_2,r_3) \wedge A(\top,r_2,r_3) = \bot$

$f(r_2,r_3) = A(\top,r_2,r_3) = \bar{r}_2\bar{r}_3$

$f(r_2,r_3) = \neg A(\bot,r_2,r_3) = \neg(r_2\bar{r}_3 \vee \bar{r}_2 r_3)$

Optimisation conjointe registres / logique

- Circuit initial :

  WRISTWATCH pi= 8 po=92 nodes=462 latches=35
  lits(sop)= 990 lits(fac)= 990
  Total number of levels = 29

- Optimisation en vitesse :

  WRISTWATCH pi= 8 po=92 nodes= 97 latches=12
  lits(sop)= 406 lits(fac)= 366
  Total number of levels = 3

- Optimisation en surface :

  WRISTWATCH pi= 8 po=92 nodes= 98 latches=11
  lits(sop)= 195 lits(fac)= 195
  Total number of levels = 15

Conclusion

- Le calcul booléen fournit un moyen simple de coder beaucoup de questions en synthèse, vérification et optimisation de circuits et de programmes
- Les récentes avancées en calcul booléen (BDDs, SAT) permettent de calculer en (assez) grand, ce qu'on pensait impossible encore récemment
- Mais il n'y a pas de miracle, les calculs peuvent exploser...

Rendez-vous l'année prochaine pour comprendre comment tout cela est réalisé !
(plus le traitement par BDDs des circuits cycliques)

Références historiques

C. Y. Lee. *Representation of Switching Circuits by Binary-Decision Programs*.
Bell Systems Technical Journal, 38:985–999, 1959.

S. B. Akers. *Binary Decision Diagrams*.
IEEE Transactions on Computers, C-27(6):509–516, June 1978.

R. T. Boute. *The Binary Decision Machine as a programmable controller*.
EUROMICRO Newsletter, Vol. 1(2):16–22, January 1976.

R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation.
IEEE Transactions on Computers, C-35(8):677–691, 1986.

J-P. Billon. *Perfect Normal Forms for Discrete Functions.*
Bull Research Report 87019, 1987.

J-P. Billon, J-C. Madre. *Original Concepts of PRIAM, an Industrial Tool for Efficient Formal Verification of Combinational Circuits*
In *The Fusion of Hardware Design and Verification*, G,. Milne Editor, North-Holland 1988

K. S. Brace, R. L. Rudell and R. E. Bryant. "Efficient Implementation of a BDD Package".
In Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC 1990), pages 40–45. IEEE Computer Society Press, 1990.

R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams, ACM Computing Surveys, Vol. 24, No. 3 (September, 1992), pp. 293–318.

Références

O. Coudert, J. C. Madre. A Unified Framework for the Formal Verification of Sequential Circuits, in Proc. of *ICCAD'90*, Santa Clara CA, USA, Nov. 1990.

C. Berthet, O. Coudert, J. C. Madre. *New Ideas on Symbolic Manipulations of Finite State Machines*", in Proc. of *ICCD'90*, Cambridge MA, USA, Sept. 1990.

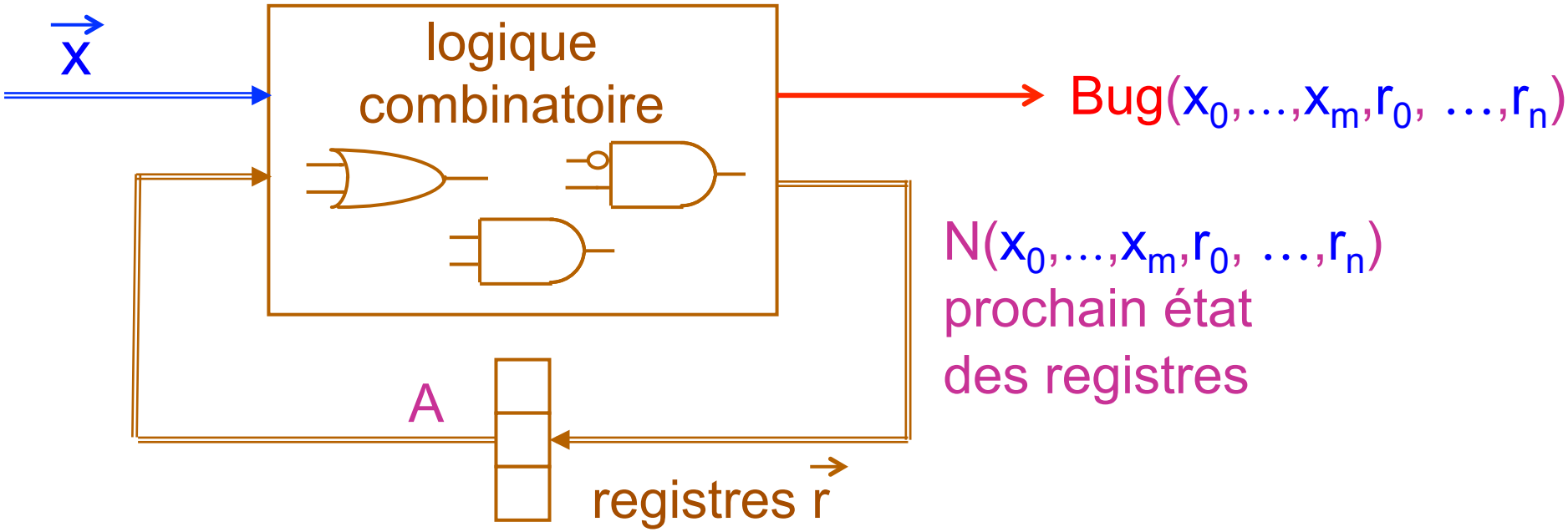
E. Sentovich, H. Toma, and G. Berry. *Efficient Latch Optimization Using Incompatible Sets*
International Digital Automation Conference DAC'97, Anaheim, USA, 1997.

H. Toma, E. Sentovich, and G. Berry.
Latch Optimization in Circuits Generated from High-Level Descriptions
Proc*. International Conf. on Computer-Aided Design* ICCAD'96.

La bible

D. Knuth. *The Art of Computer Programming, Vol. 4 : Combinatorial Algorithms*
Section 7.1.4 : Binary Decision Diagrams
Addison Wesley, 2014

Calcul en avant des états accessibles



Etat initial : A_0

Etats accessibles en 1 coup : $A_1 = F(E,A_0)$

Etats accessibles en n+1 coups : $A_{n+1} = F(E,A_n)$

Test d'arrêt : $A_{n+1} \Leftrightarrow A_n$

Résultat : $A = A_n = A_{n+1}$