Génération Automatique de Vecteurs de Test (ATPG)

Vérification, Emulation et Simulation

Ahcène Bounceur

Génération de vecteurs de test

- Vecteur de Test :
  - Vecteur d'entrées au Circuit Sous Test (CUT) qui garantit la détection d'une faute (si elle existe) en observant les sorties primaires du CUT.
- Génération Automatique de Vecteurs de Test
  - Le processus qui permet de déterminer les stimuli à appliquer en entrée d'un circuit pour mettre en évidence un défaut potentiel.
- Faute détectée
  - Une faute pour laquelle un vecteur de test valide est généré.
- Faute indétectable
  - Une faute pour laquelle il n'existe aucun vecteur de test qui permet de la détecter.
- Faute redondante
  - Une faute pour laquelle il n'existe aucun vecteur de test (à cause d'une redondance logique dans le circuit $\Rightarrow$ Matériel non nécessaire).

Génération de vecteurs de test (2)

- Couverture de fautes FC

$$F = \frac{\text{Nombre de fautes détectées}}{\text{Nombre total de fautes}}$$

- Efficacité de fautes FE

$$F = \frac{\text{Nombre de fautes détectées} + \text{Nombre de fautes redondantes}}{\text{Nombre total de fautes}}$$

Génération de vecteurs de test (3)

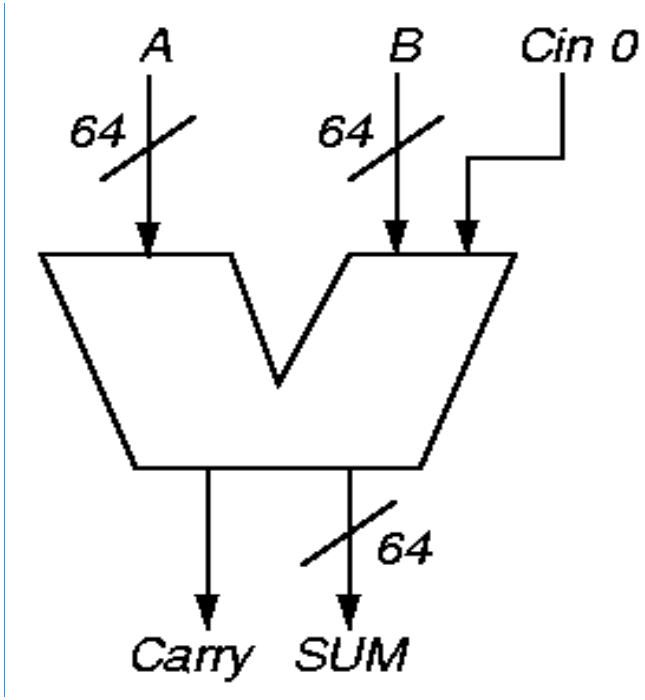
- Contrôlabilité
  - Une métrique qui permet de mesurer la difficulté d'imposer une valeur à un nœud (du CUT)
- Observabilité
  - Une métrique qui permet de mesurer la difficulté de propager une valeur d'un nœud à la sortie
- Testabilité
  - Une métrique qui permet de mesurer la difficulté de générer un vecteur de test pour une ligne spécifique du circuit

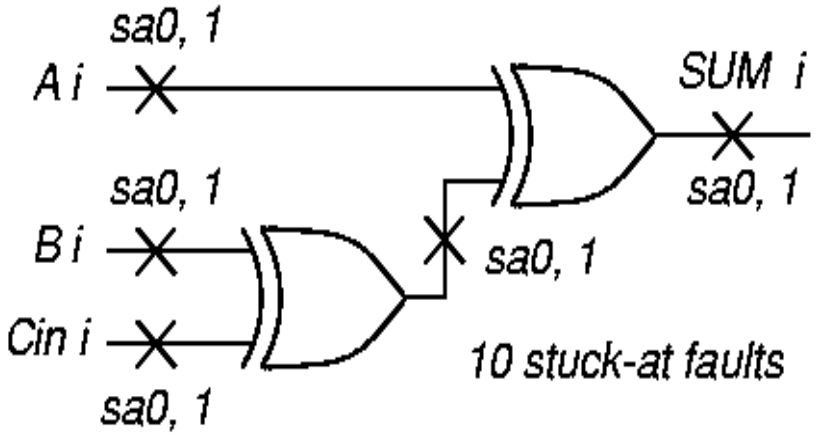
Génération de vecteurs de test (4)

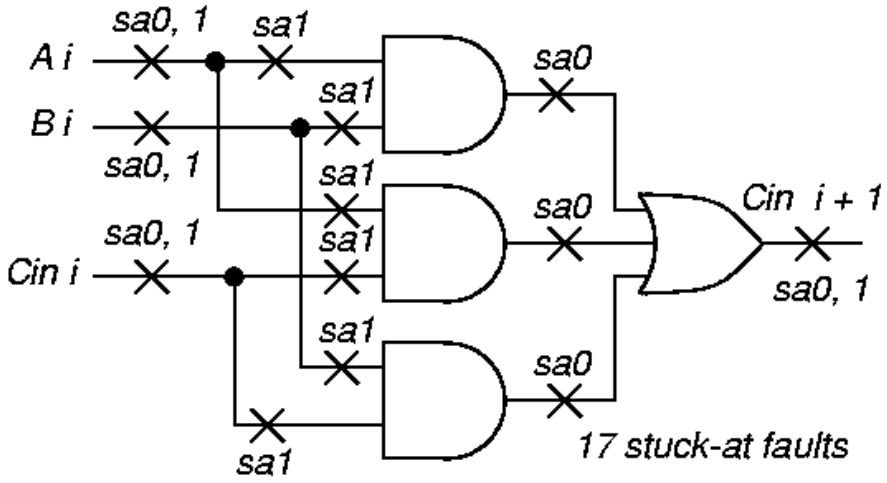
- Sensibilisation (Activation)
  - Le processus qui met le circuit dans un état permettant de causer une erreur au point de la faute.
- Propagation
  - Le processus qui met le circuit dans un état permettant d'observer une faute dans l'une de ces sorties primaires.
- Justification
  - Processus de combinaison de plusieurs entrées permettant de forcer une valeur dans un nœud interne spécifique du CUT

Fonctionnel vs. Structurel ATPG

Additionneur avec 129 entrées et 65 sorties







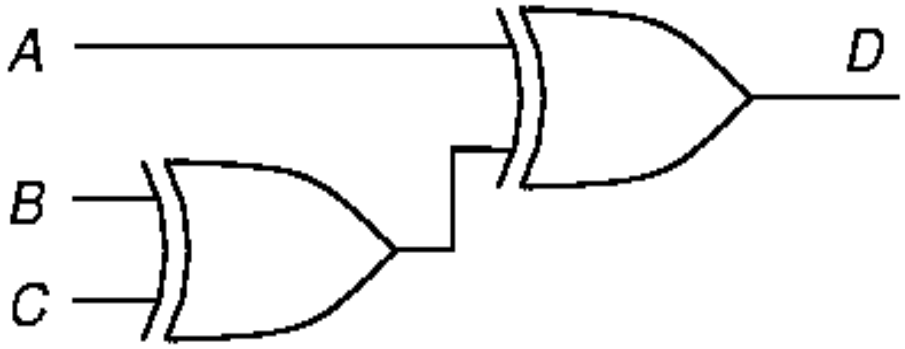
Fonctionnel vs. Structurel ATPG

- **ATPG Functionnel** – Générer un ensemble complet de vecteurs de test pour les entrées/sorties du circuit
  - 129 entrées et 65 sorties :
  - 2^{129} = **680 564 733 841 876 926 926 749 214 863 536 422 912** vecteurs de test
  - En utilisant un ATE de 1 GHz, on a besoin de $2{,}15 \times 10^{22}$ ans

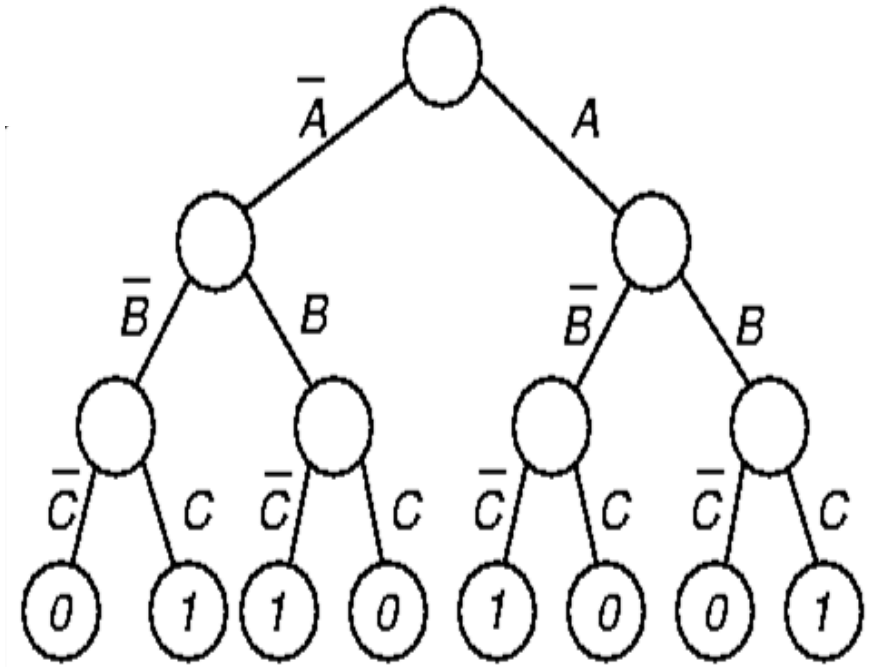
  => Pratiquement **impossible**

- **ATPG Structurel** :
  - Pas d'additionneurs redondants, **64** bit par partie
  - Chacune avec **27** fautes (équivalence de fautes pris en compte)
  - Au plus **64 x 27 = 1728** fautes (donc, au plus 1728 vecteurs)
  - Ceci prend **0.000001728** s sur un ATE de 1 GHz

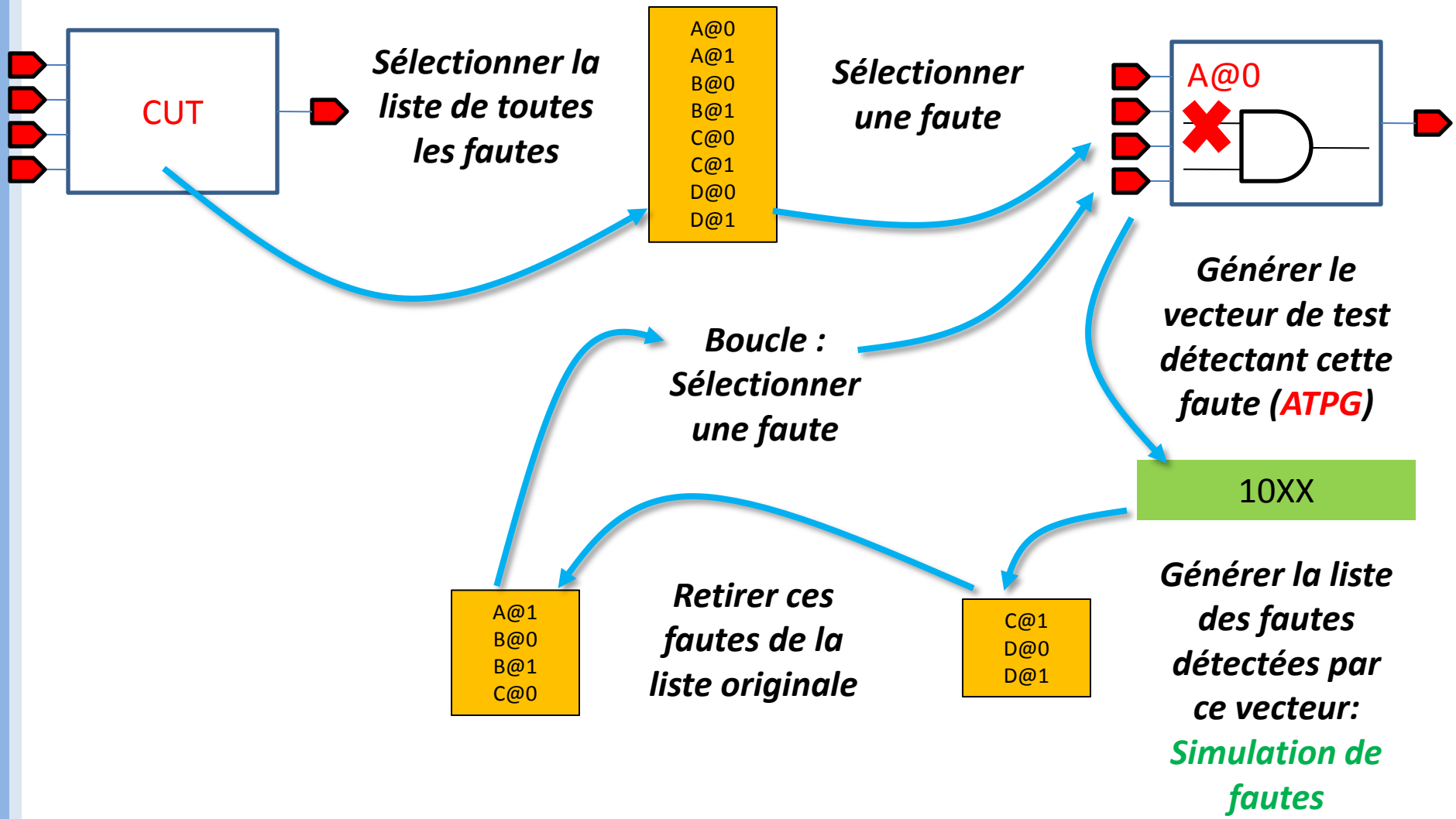
Décision binaire



- L'arbre binaire correspondant
  - Donne toutes les entrées possibles
  - Feuille : représente la bonne sortie
  - L'ATPG cherche implicitement l'arbre pour trouver le vecteur de test
  - Pire cas : Parcours complet de l'arbre
    - Exponentiel !!!



Processus de génération



Types de génération

- Génération manuelle
  - Les vecteurs de test sont décrits manuellement
- Génération exhaustive
  - Utilisation de tous les vecteurs de test possibles
- Génération pseudo-aléatoire
  - Les vecteurs de test sont choisis aléatoirement
- Génération automatique (déterministe)
  - Pour chaque faute non détectée on génère un vecteur de test qui la détecte

Génération manuelle

- Utilisation des vecteur de test fonctionnels
  - Utilisation d'un simulateur de fautes pour déterminer la couverture de fautes
  - Localisation des fautes non détectées
  - Ajout d'autres vecteurs de test jusqu'à atteindre la couverture de fautes désirée
- Avantage :
  - Ne nécessite pas d'algorithme de génération
- Inconvénient :
  - Très difficile d'atteindre un taux de couverture élevé

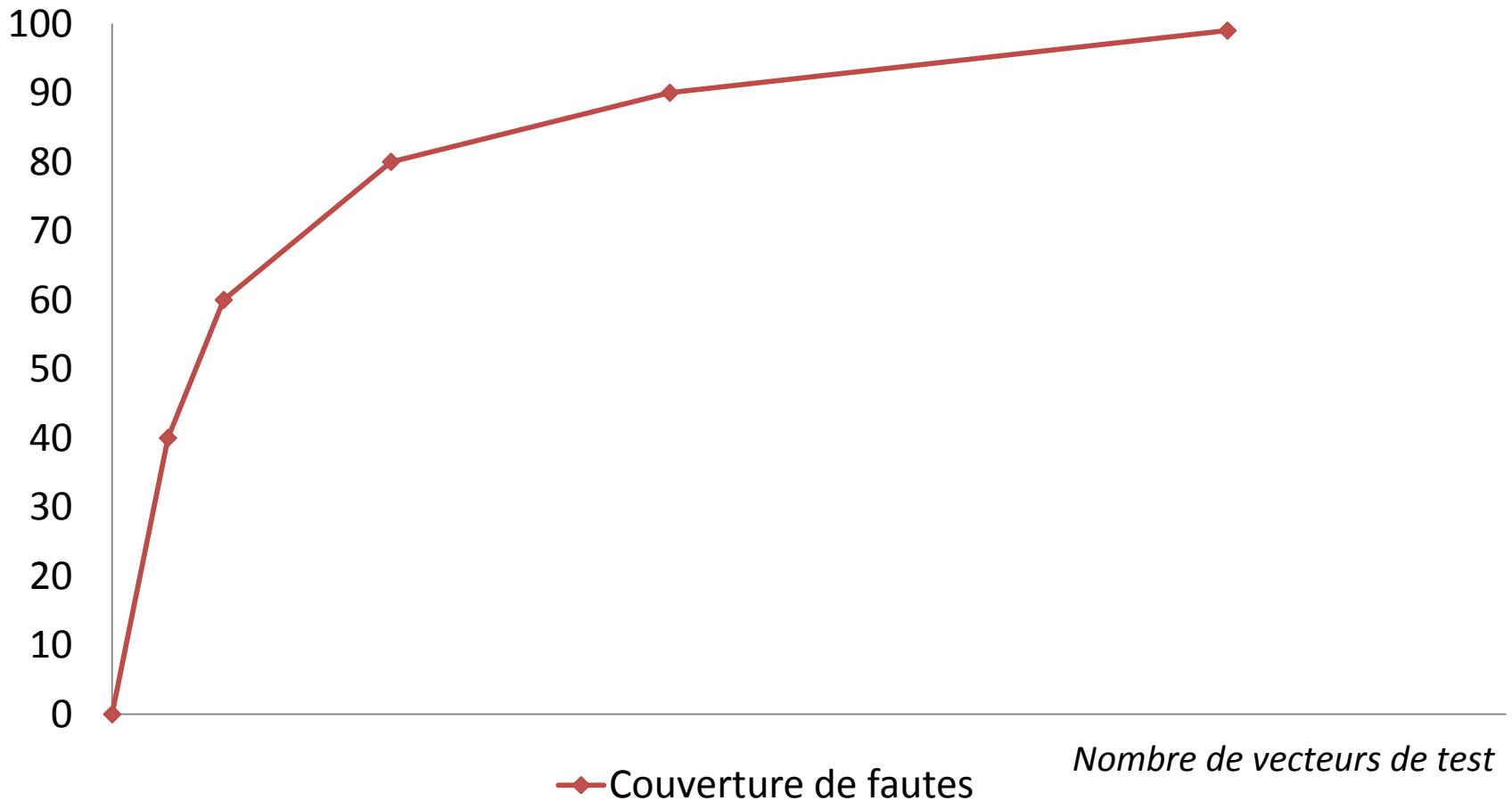
Génération exhaustive

- Choix de tous les vecteurs d'entrées possibles
- Avantages :
  - Pas de simulation de fautes (pas de modèle de fautes)
  - Détecte toute les fautes détectables de tous les modèles de fautes
- Inconvénients :
  - Test très coûteux (grand # de vecteurs)
  - Inutilisable pour les circuit avec un grand # d'entrées
- Amélioration :
  - Techniques de partitionnement du circuit pour réduire le nombre de vecteurs
  - Génération localement exhaustive

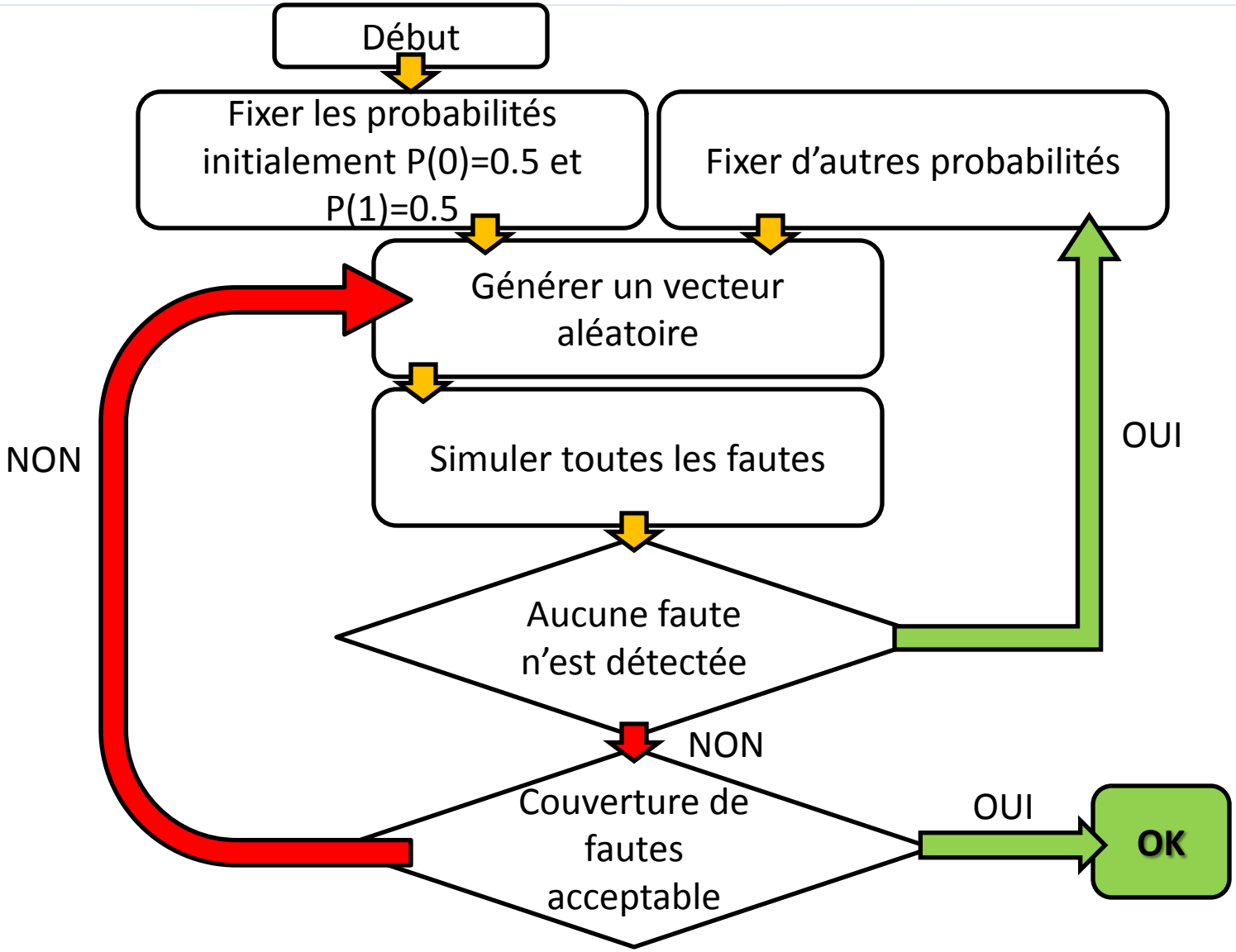
Génération pseudo-aléatoire

- Choix aléatoire des vecteurs d'entrées
  - Utilisation d'un simulateur de fautes pour déterminer la couverture de fautes
  - Ajout d'autres vecteurs de test jusqu'à atteindre la couverture de fautes désirée
- Avantages :
  - Détecte facilement les fautes faciles
  - Ne nécessite pas d'algorithme de génération
- Inconvénient :
  - Ne détecte pas les fautes difficiles

Génération pseudo-aléatoire (2)



Algorithme de génération aléatoire



Génération Automatique

- ATPG : Automatic Test Pattern Generation
- Choix déterministe des vecteurs de test
  - Pour chaque faute non détectée on génère un vecteur de test qui la détecte
  - Calcul du taux de couverture avec un simulateur de fautes
- Avantages :
  - Un taux de couverture maximum avec un coût minimum
- Inconvénients :
  - Complexité des algorithmes
    - Temps CPU
    - Mémoire utilisée

Détection d'une faute

- Trouver un vecteur de test T qui détecte la faute F revient à :
  - trouver un vecteur de test T qui :
    - Active la faute F à partir des entrées primaires (PIs)
    - 2- Propage la faute F jusqu'aux sorties primaires (POs)
- Exemple : F = S@0
  - Le vecteur qui détecte la faute S@0 doit :
    - Imposer la valeur 1 au signal S
    - Propager la faute X=1 jusqu'aux sorties primaires (POs)
      - Le changement de la valeur du signal S de 1 vers 0 doit être visible sur au moins une des sorties primaires POs du circuit

Exemple : Activation d'une faute

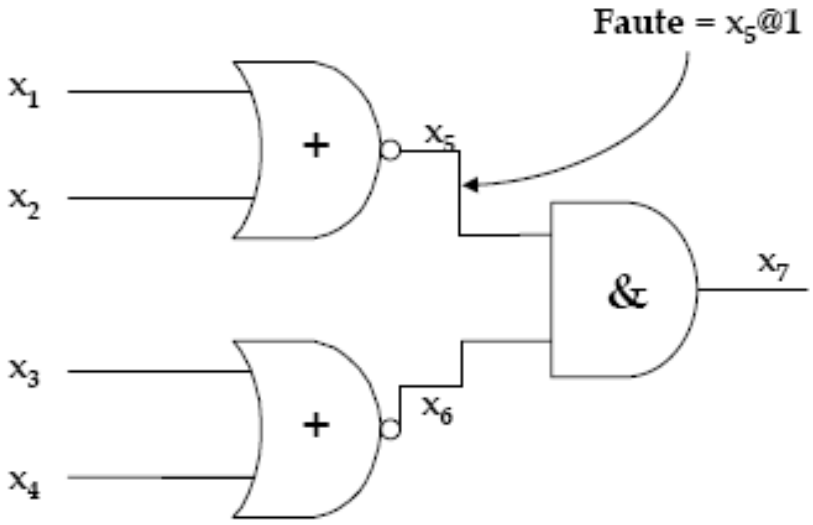
- Définition :
  - C'est le processus qui permet de ramener le circuit dans l'état ou la faute produit une erreur dans le circuit. Pour le modèle des collages, c'est ramener la valeur opposée du collage sur le signal en question
- Exemple : F = x5@1

Activation : (x5 = 0)

=> (x1x2 = 11)

ou (x1x2 = 01)

ou (x1x2 = 10)



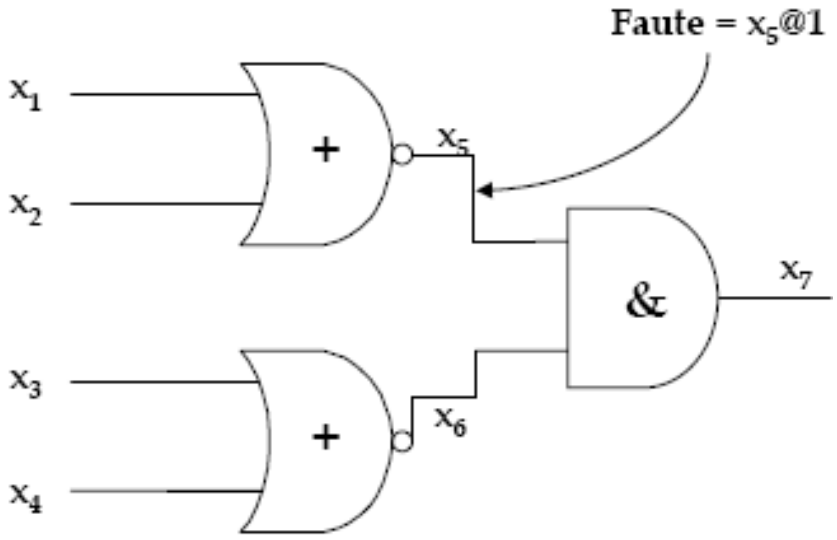
Exemple : Propagation d'une faute

- Définition :
  - C'est le processus qui permet de propager la faute à travers toutes les portes du circuit jusqu'aux sorties primaires
- Exemple : F = x5@1

  Faute = x5@1 (x5 = 0)

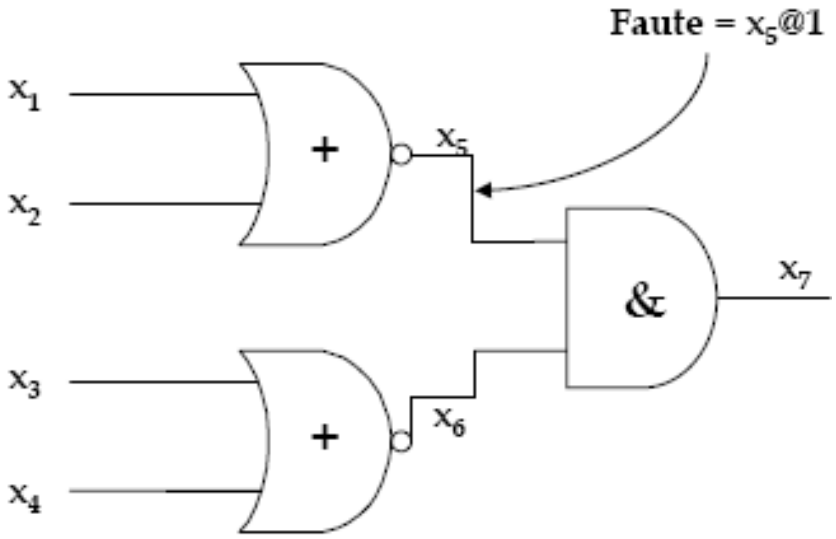
  Propagation => (x6 = 1)

  => (x3x4 = 00)



Exemple : Détection d'une fautes

- Faute = x5@1
- Activation => (x5 = 0)
  => (x1x2 = 11)
  ou (x1x2 = 01)
  ou (x1x2 = 10)
- Propagation => (x6 = 1)
  => (x3x4 = 00)
- Détection =>
  x1x2x3x4 = 1100
  ou x1x2x3x4 = 0100
  ou x1x2x3x4 = 1000



Génération Automatique : méthode

- Il existe plusieurs types de génération automatiques de vecteurs de test. Les algorithmes les plus utilisés sont basés sur la notion de SENSIBILISATION DE CHEMIN.
- Utilisation de la notation D
  - Pour un signal S, on a :
    - **S=D ⇔ S=1 dans le bon circuit et S=0 dans le circuit fautif**
    - **S=D* ⇔ S=0 dans le bon circuit et S=1 dans le circuit fautif**

D= D barre*

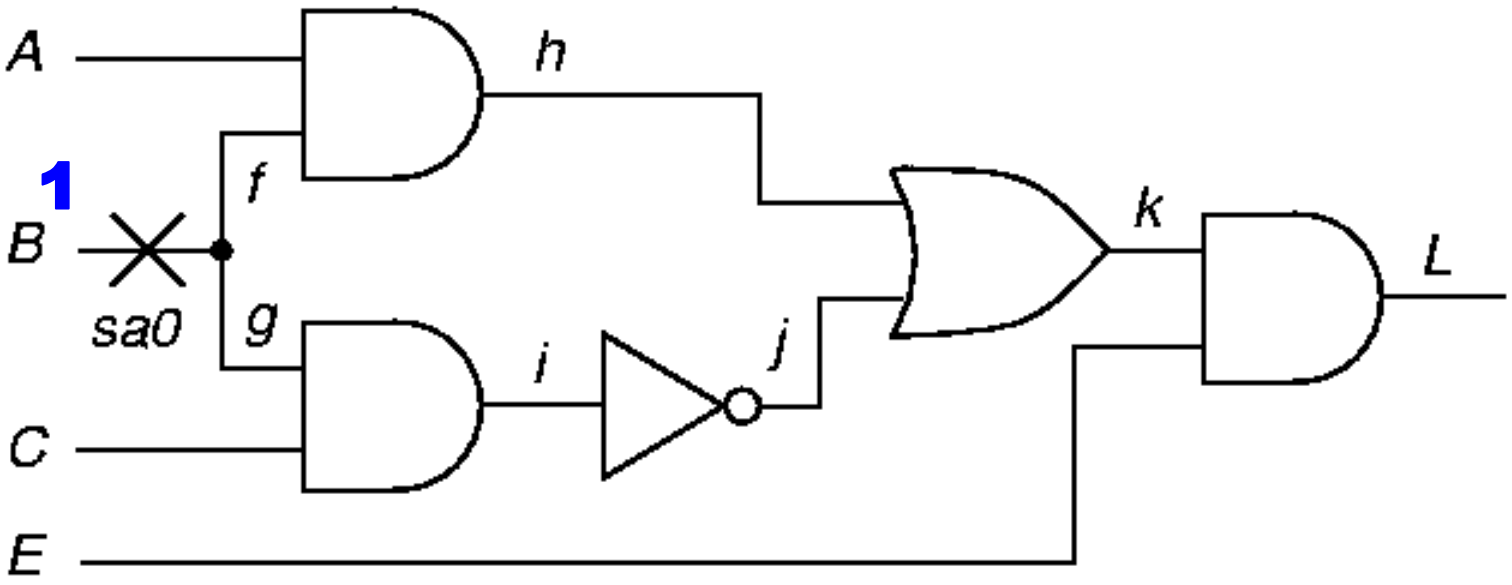
Méthode de sensibilisation de chemin

- Cette méthode est la plus préférée des méthodes
- Son approche est composée de trois étapes :

1. **Activation de la faute**
   - La ligne où il y a la faute (SAF) est activée en forçant sa valeur à l'opposite de la valeur de la faute
   - On l'appelle aussi : Sensibilisation de fautes ou excitation de la faute
2. **Propagation de la faute**
   - Il faut propager la faute à travers une ou plusieurs chemin vers la sortie
   - On l'appelle aussi : Sensibilisation de chemin
3. **Justification de ligne**
   - Justifier la ligne activée en fixant les entrées du circuit

Méthode de sensibilisation de chemin

- Exemple:
  - **Activation de faute** : mettre B à 1

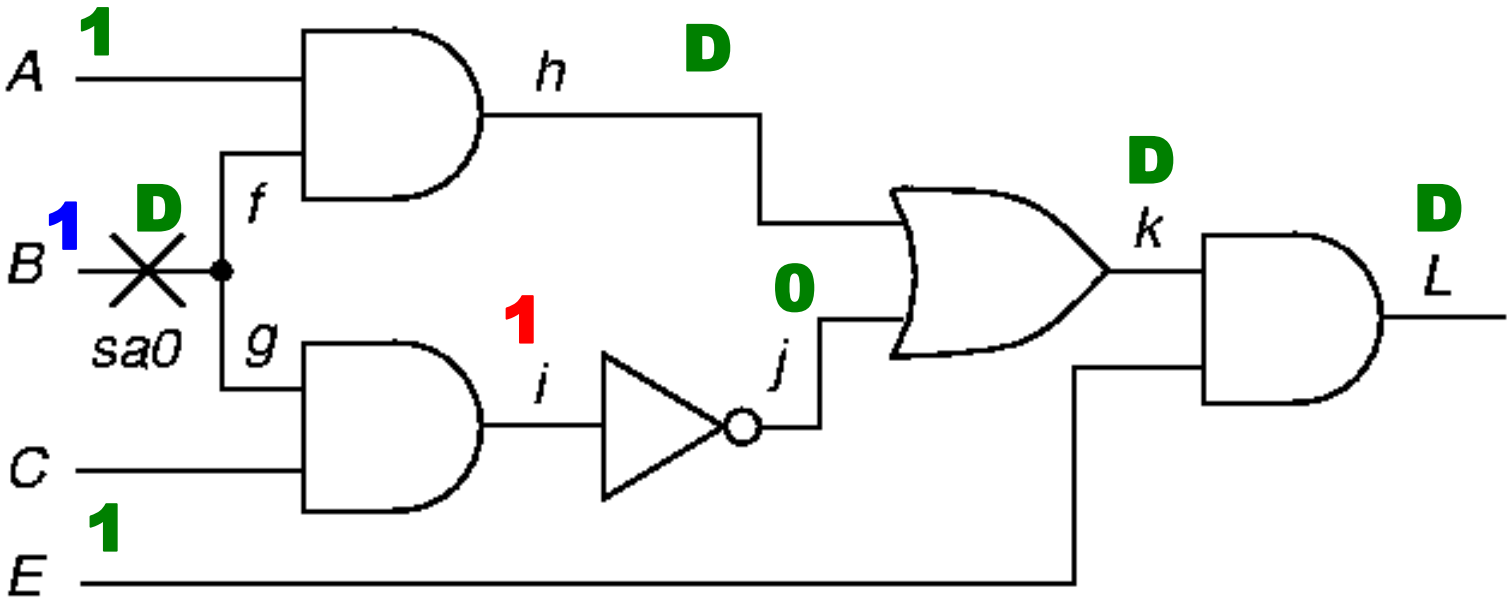


Méthode de sensibilisation de chemin

- Exemple:
  - **Activation de faute** : mettre B à 1
  - **Propagation de faute** : essayer le chemin $f - h - k - L$
  - **Justification de ligne :** il n'existe pas de moyen justifiant 1 à i

⇒**Conflit**

⇒Retour



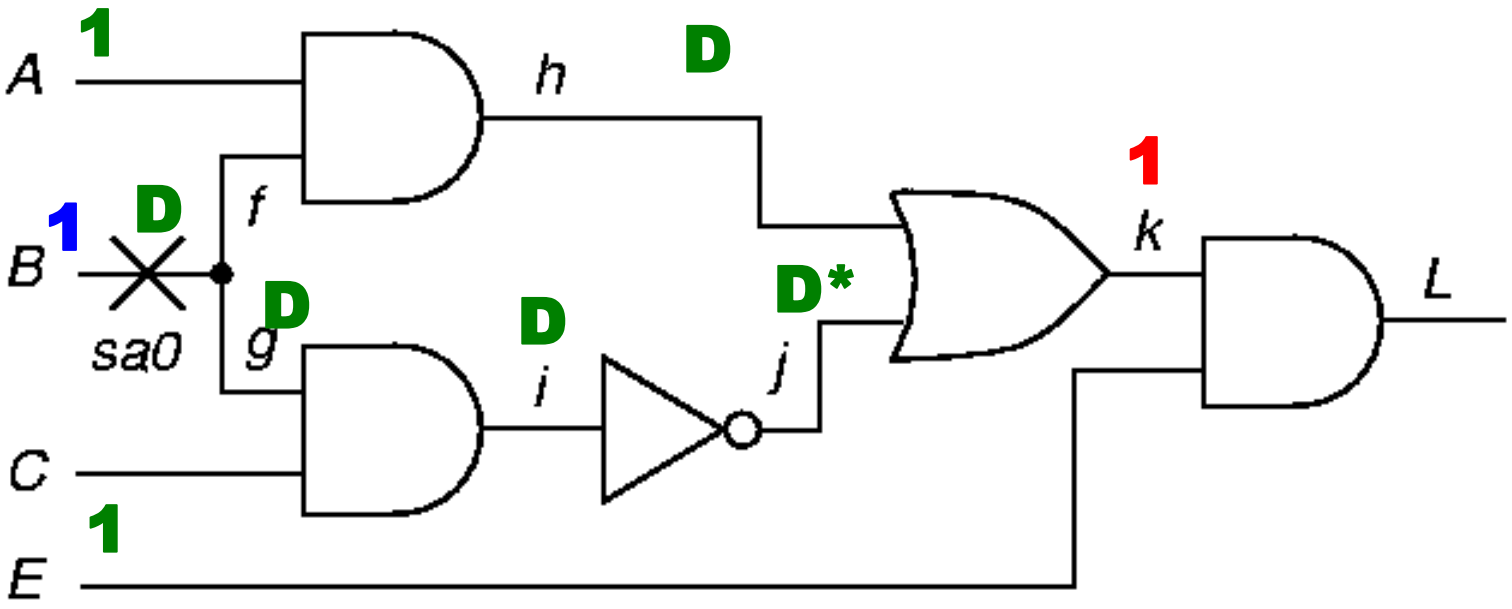
Méthode de sensibilisation de chemin

- Exemple:

D= D barre*

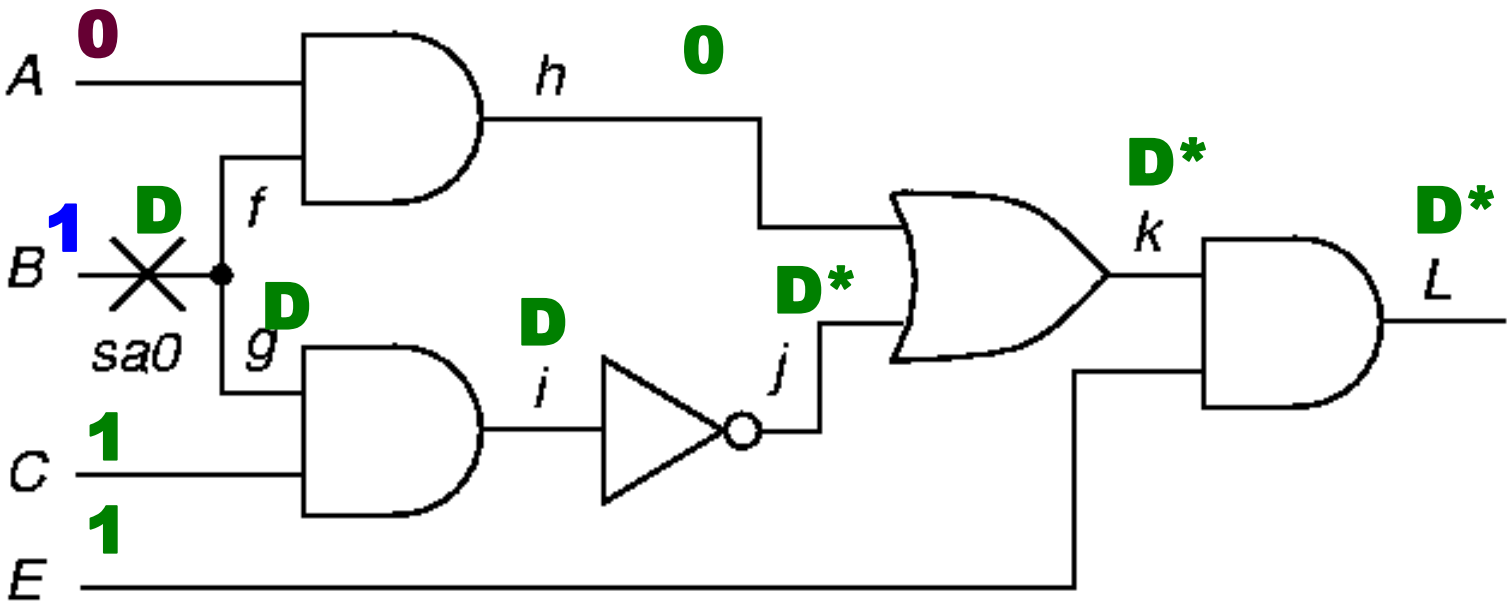
  - **Activation de faute** : mettre B à 1
  - **Propagation de faute** :
    - Essayer simultanément les chemins

      f – h – k – L et *g – i – j – k – L*
    - Blocage à *k* à cause de la *frontière-D* (*D* or *D**) : **D disparait**



Méthode de sensibilisation de chemin

- Exemple:
  - **Activation de fautes** : mettre B à 1
  - **Propagation de faute** : essayer le chemin $g - i - j - k - L$
  - **Justification de ligne** : mettre A à 0



Complexité d'un algorithme ATPG

- Ibarra and Sahni analysis [1975] – ***NP-Complet***
  - Il n'existe pas d'algorithmes polynomiaux pour le temps de calcul, présumé exponentiel
- Pire cas :
  - *no_pi* entrées, 2^{no_pi} combinaisons possibles
  - *no_ff* bascules, 4^{no_ff} états initiaux de bascules
  - Complexité : $O(n \times 2^{no\_pi} \times 4^{no\_ff})$

Algorithmes de génération ATPG

- Il existe plusieurs algorithmes de génération automatique des vecteurs de test (ATPG)
  - Algorithme D
  - Algorithme PODEM
  - Algorithme FAN
  - Autres algorithmes

Algorithme D

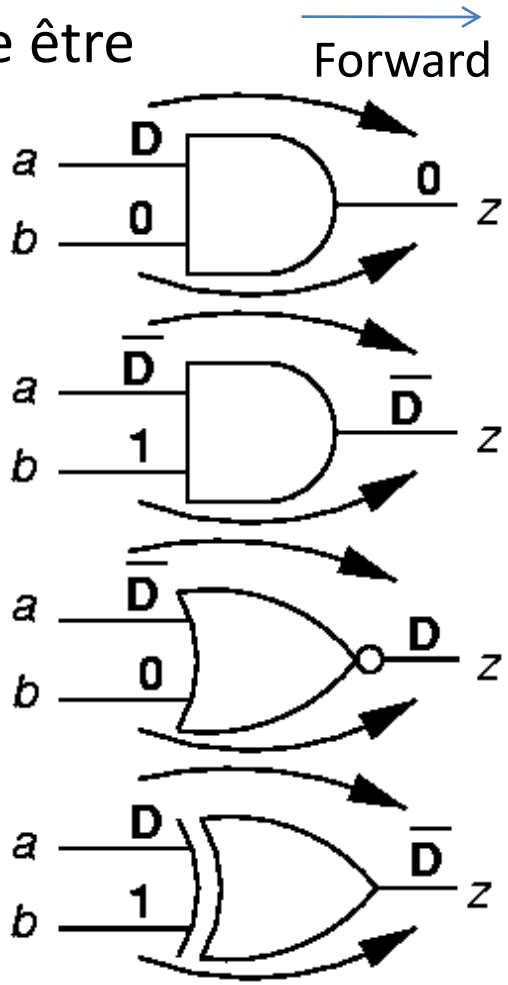
- Premier algorithme complet (test tous les cas possibles), développé par IBM en 1966 (Roth). Il a été prouvé que l'algorithme D génère toujours un test qui détecte la faute s'il en existe un.

Algorithme D : Définitions

Implication en-avant (forward)

- Résulte des entrées d'une porte logique qui sont significativement fixées telles que la sortie puisse être déterminée de manière ***unique***.
- Exemple:
  - Table de l'implication en-avant d'une Porte AND :

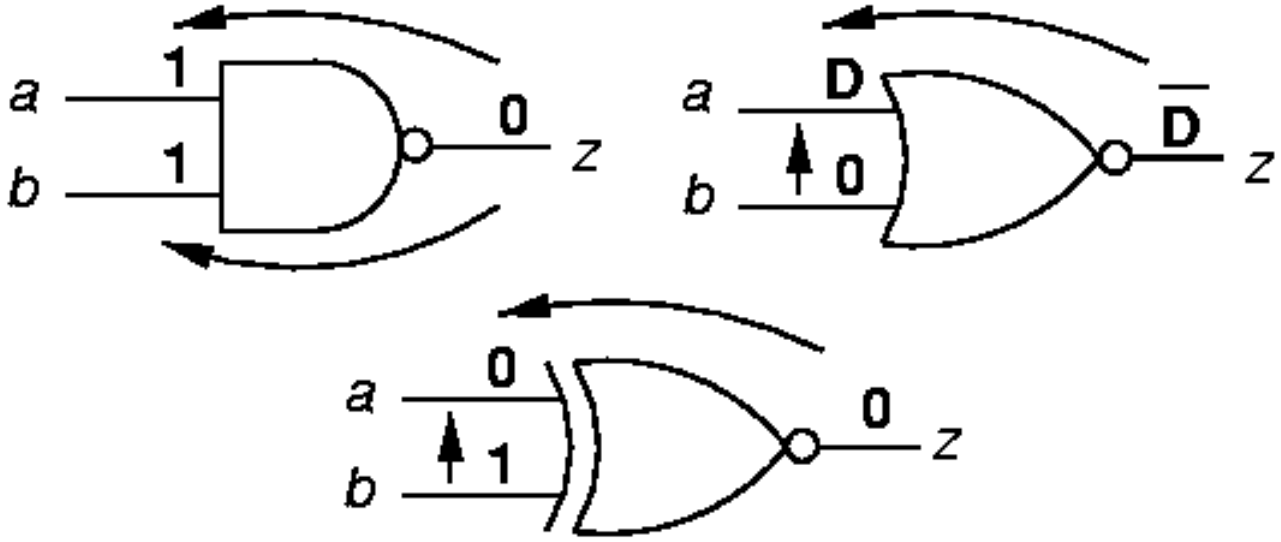
a \ b	0	1	X	D	$\overline{D}$
0	0	0	0	0	0
1	0	1	X	D	$\overline{D}$
X	0	X	X	X	X
D	0	D	X	D	0
$\overline{D}$	0	$\overline{D}$	X	0	$\overline{D}$



Algorithme D : Définitions

Implication en-arrière (backward)

- **Détermination unique de toutes les entrées d'une porte quand la sortie et certaines entrées sont données.**
- Exemple



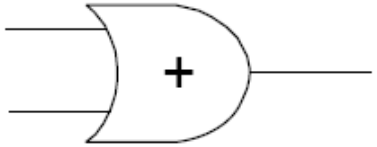
Algorithme D : Notation D

Valeur du signal pour circuit correct C.C. (V)	Valeur du signal pour circuit fautif C. F. (V_f)	Notation D
0	0	0
0	1	D'
1	0	D
1	1	1

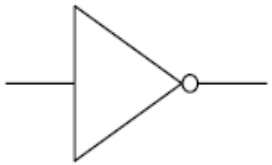
$D' \Leftrightarrow \overline{D}$

Algorithme D : Table de vérité

AND	0	1	D	D'
0	0	0	0	0
1	0	1	D	D'
D	0	D	D	0
D'	0	D'	0	D'



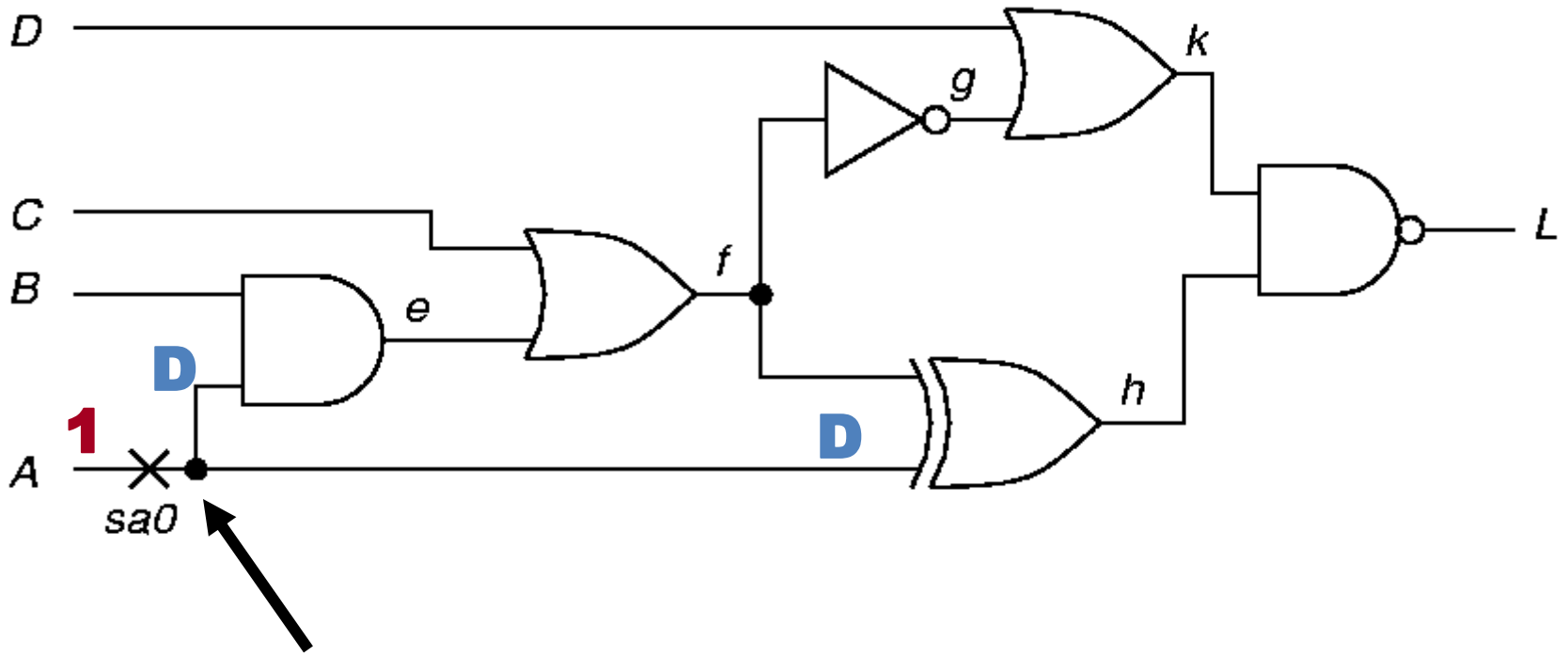
OR	0	1	D	D'
0	0	1	D	D'
1	1	1	1	1
D	D	1	D	1
D'	D'	1	1	D'



NOT	0	1	D
	1	0	D'

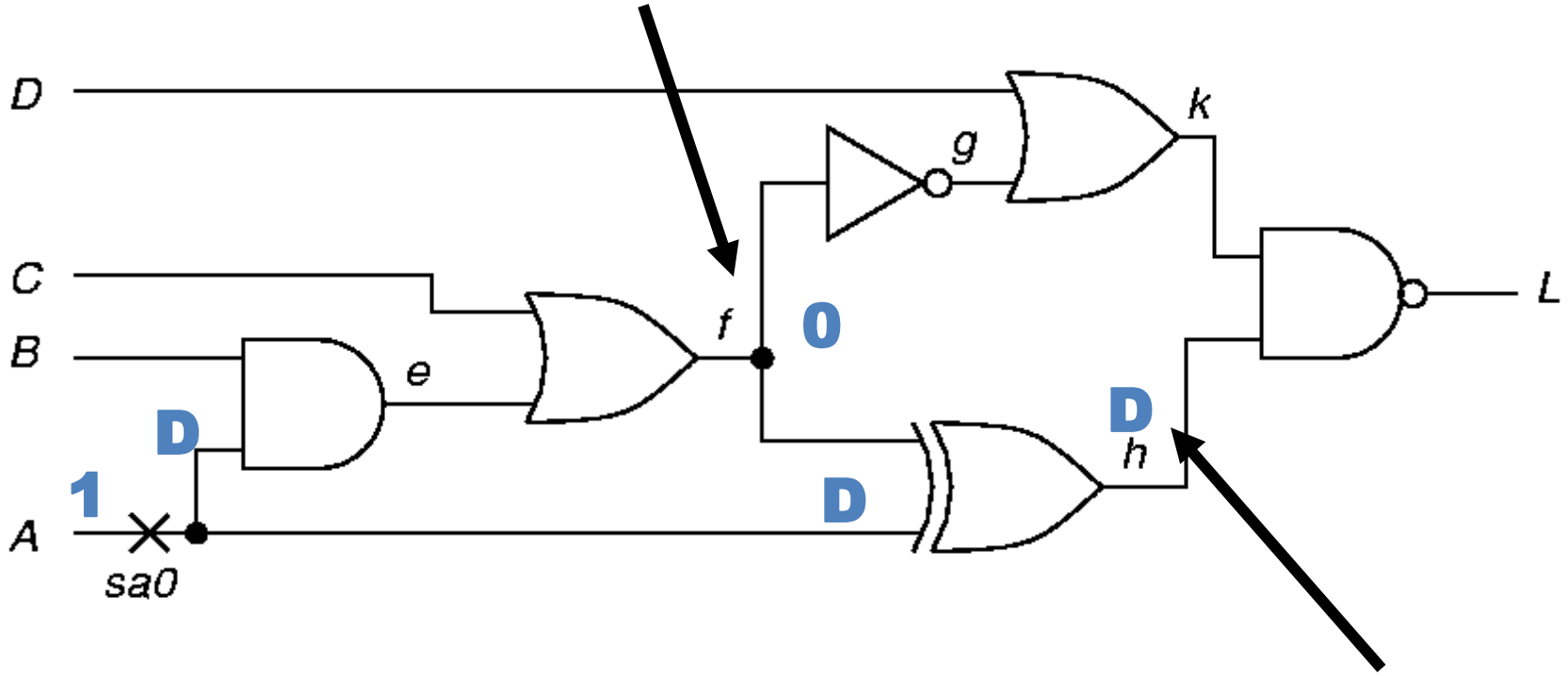
D-Algorithm : Exemple *(1)*

- Exemple : Faute *A*@0
- **Etape 1 :** Sélectionner le D-cube de la faute. Poser ***A=1***



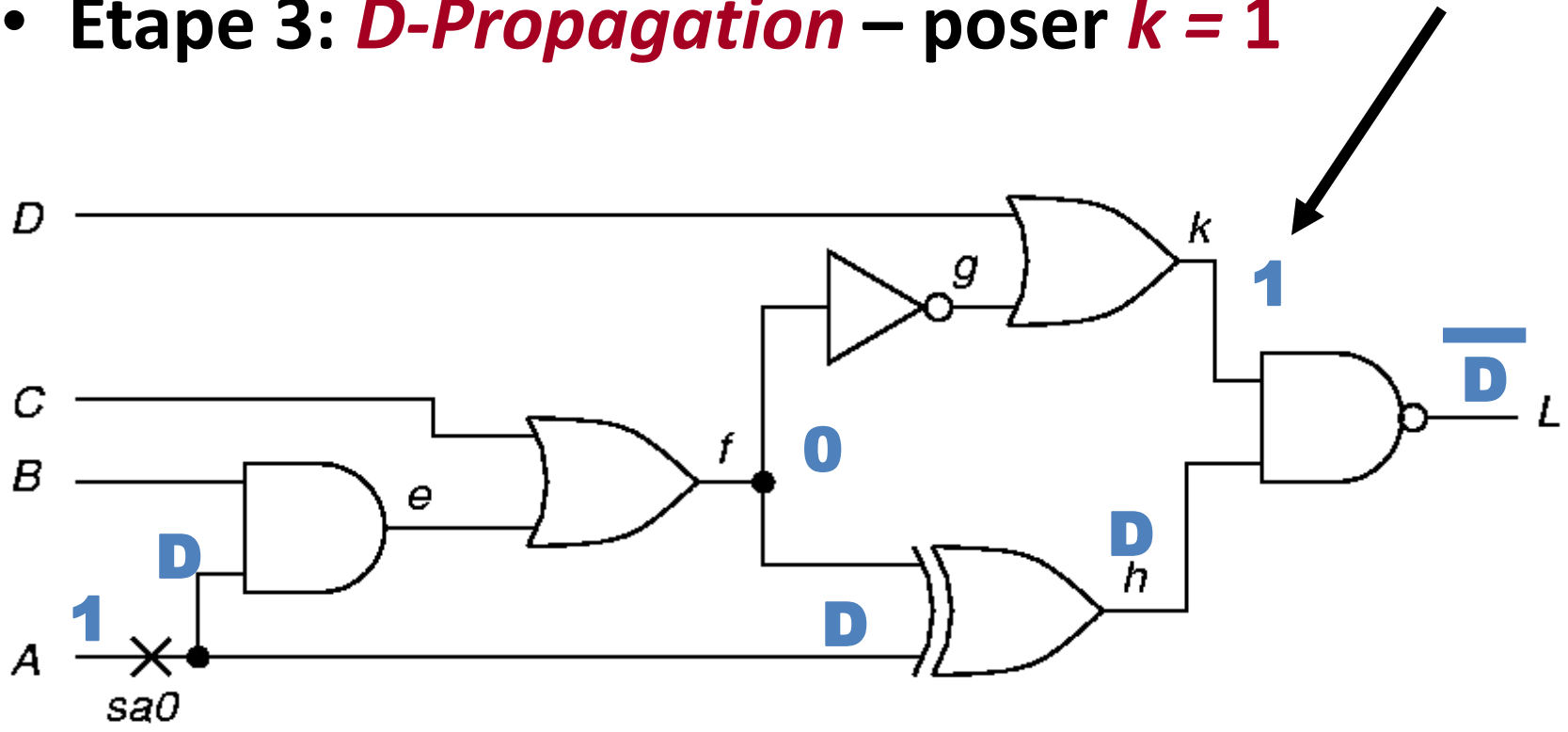
D-Algorithm : Exemple *(2)*

- **Étape 2 : *D-Propagation* – poser *f* = 0**



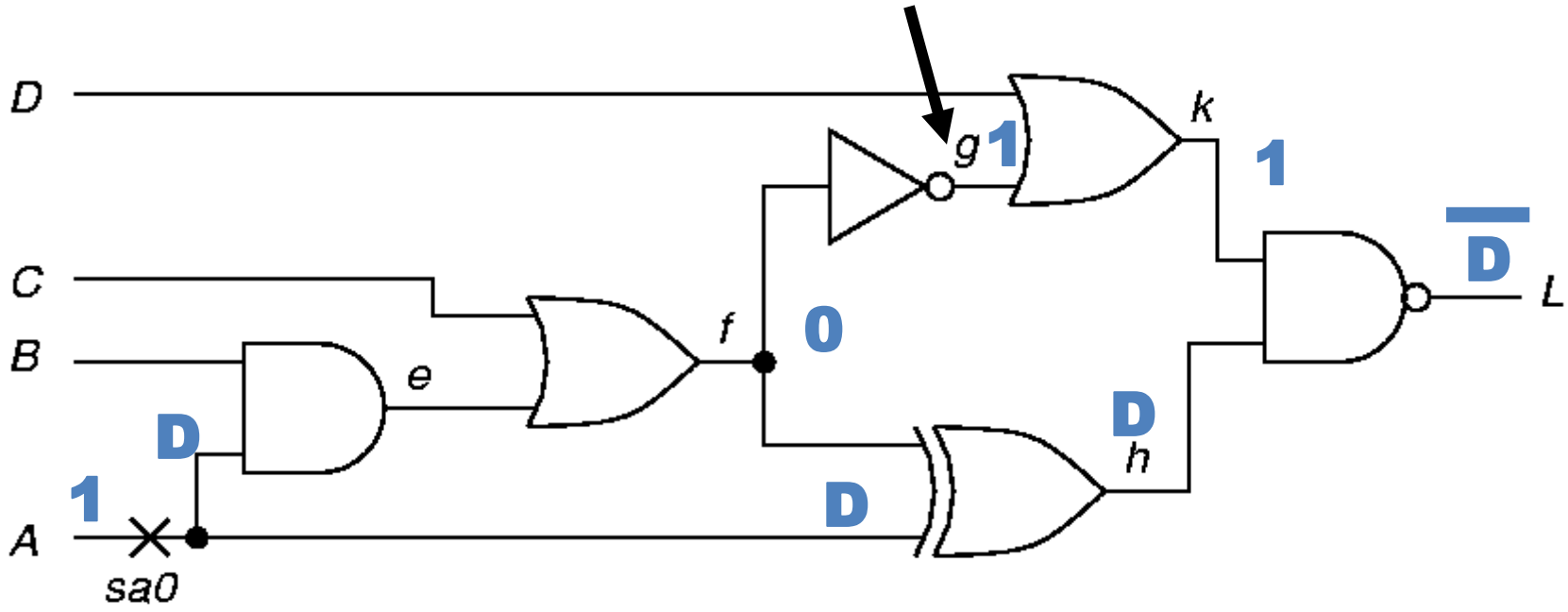
D-Algorithm : Exemple *(3)*

- **Étape 3: *D-Propagation* – poser *k* = 1**



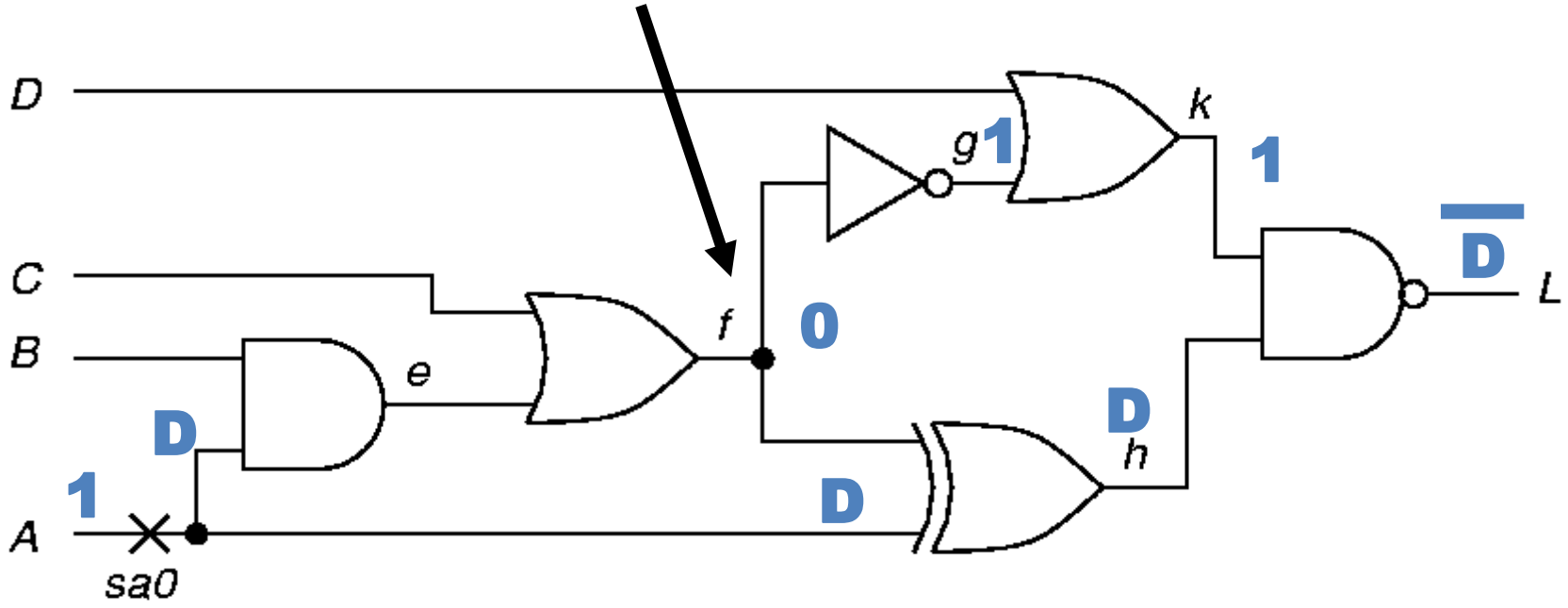
D-Algorithm : Exemple *(4)*

- **Étape 4: *Retour* – poser *g* = 1**



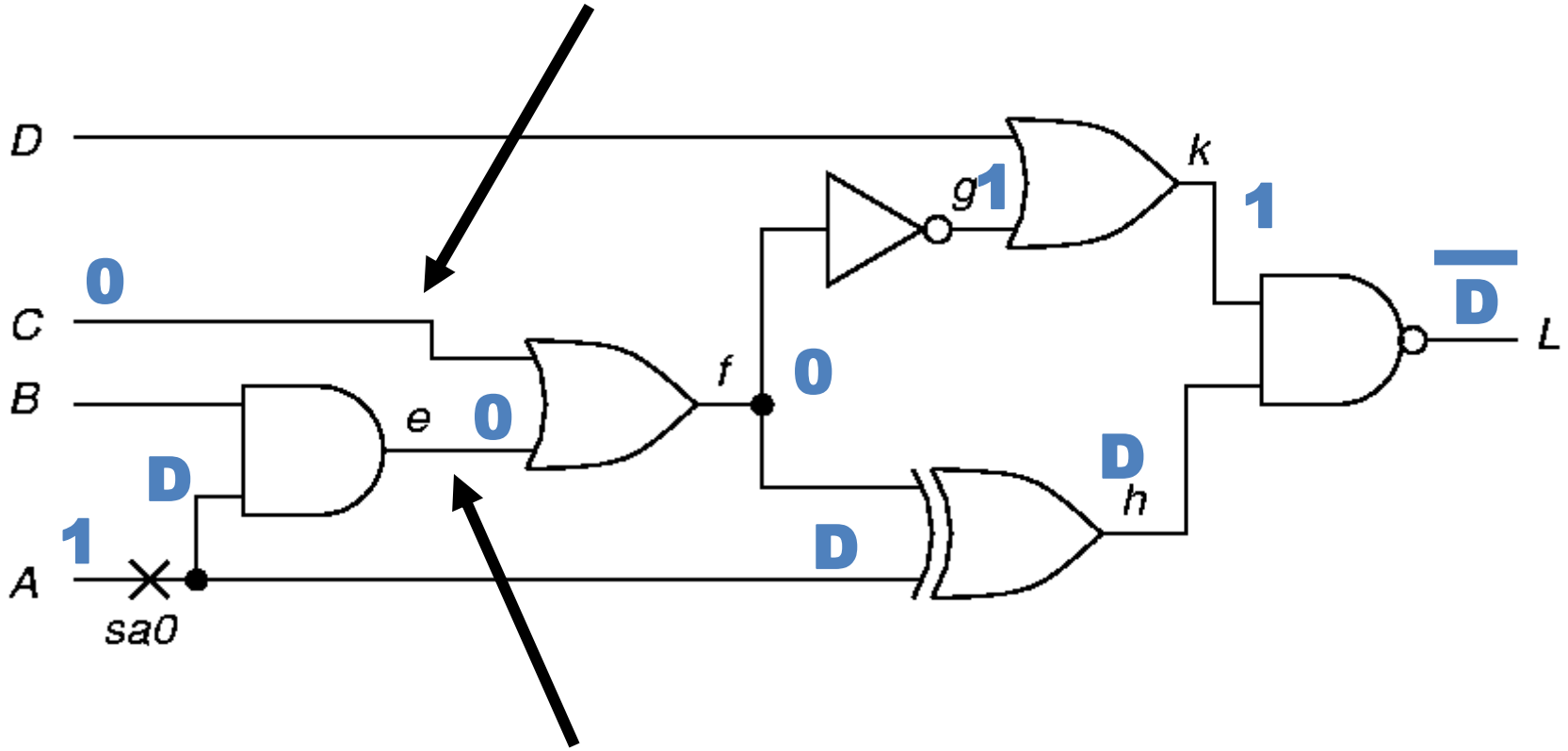
D-Algorithm : Exemple *(5)*

- **Étape 5: *Retour* – *f* = 0 déjà fixé**



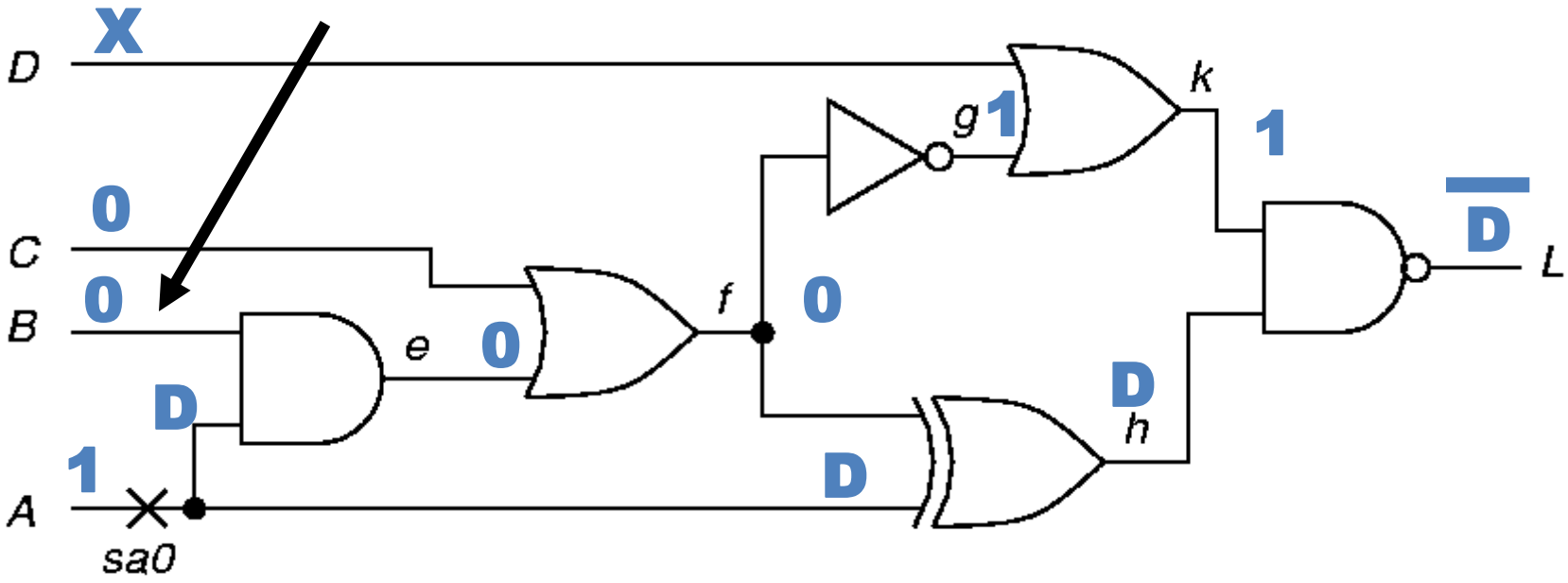
D-Algorithm : Exemple *(6)*

- **Étape 6: *Retour* – mettre *c*=0, mettre *e*=0**



D-Algorithm : Exemple *(7)*

- **Étape 7 : *retour* – poser *B = 0***
- ***Pas de ligne à justifier : Vecteur de test trouvé***



- ***Cube de test*** : *A, B, C, D, e, f, g, h, k, L*
- *TV = D00X001D1D**

Algorithme D

- Inconvénients :
  - Nombre très élevé de justifications créant des conflits
    - Choix aléatoire des combinaisons des entrées permettant de justifier la valeur souhaitée
    - Choix aléatoire de la porte qui va propager la faute jusqu'au sorties primaires
  - Complexité exponentielle
- Amélioration :
  - Algorithme PODEM
  - Algorithme FAN
  - Autres algorithmes

Algorithme PODEM (1)

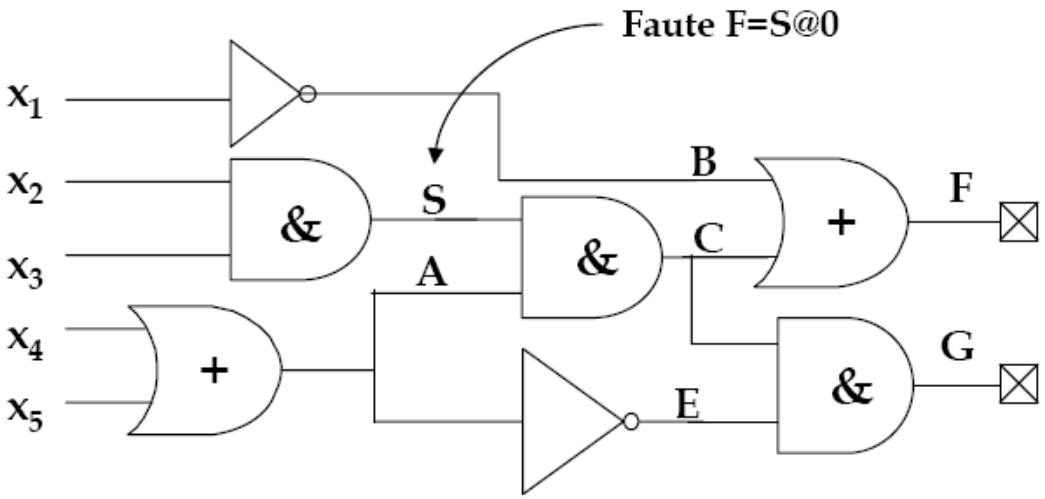
- L'algorithme PODEM "Path Oriented DEcision Making" a été développé pour remédier au problème de justifications multiples (avec conflits) dans l'algorithme D
- Plusieurs améliorations
  - 1 - Dans les phases de génération, la priorité est donnée à la phase de sensibilisation jusqu'au entrées primaires
  - 2 - L'étape d'implication n'est utilisé que pour les entrées primaires, si la valeur imposée crée un conflit on essaye la valeur opposée
  - 3 - Pour la propagation de la valeur de test (D ou D*) vers les sorties primaires PODEM utilise un seul niveau logique à la fois
  - 4 - Pour le choix de la porte de propagation, PODEM choisit la porte la plus proche des sorties primaires

Algorithme PODEM (2)

- L'algorithme PODEM se décompose en trois phases:
  - Détermination de l'objectif
    - Deux types d'objectifs :
      - 1 - Initialement, l'objectif est de ramener la valeur de test (D ou D*) à la sortie de la porte fautive
      - 2 - Propager la valeur de test de l'entrée à la sortie de la porte
  - Remontée (Backtrace) jusqu'aux entrées primaires qui satisfassent l'objectif défini
  - Simulation (implication) de la valeur de l'entrée assignée
    - S'il existe toujours un chemin possible, on continue avec un autre objectif
    - Sinon, on simule la valeur opposée de l'entrée

Algorithme PODEM : exemple

- Exemple : Faute = S@0
  - Obj 1 : S=1
    - Bac 1 : x2=1
    - Sim : S=X ; F=X ; G=X
    - Bac 2 : x3=1
    - Sim : S=1 ; F=X ; G=X
  - Obj 2 : C=D/D*
    - Bac 1 : A=1
    - Bac 2 : x4=1
    - Sim : C=D ; F=X ; G=0
  - Obj 3 : F=D/D*
    - Bac 1 : B=0
    - Bac 2 : x1=1
    - Sim : F=D ; G=0

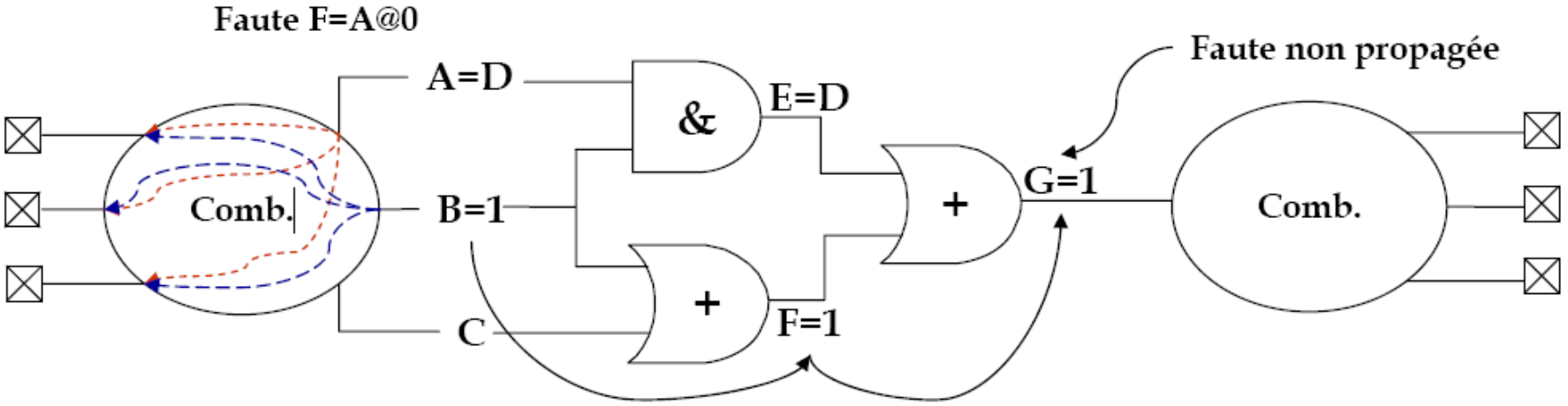


Algorithme PODEM

- Avantage :
  - Moins de backtrace que l’algorithme D
  - Utilisation d'heuristiques pour réduire le nombre de backtraces
- Inconvénients :
  - Nombre élevé de backtraces créant des conflits
  - Complexité exponentielle
- Amélioration :
  - Algorithme FAN

Algorithme FAN

- L'algorithme FAN est une amélioration des algorithmes D et PODEM
  - Les algorithmes D et PODEM ont un inconvénient majeur qui est le nombre élevé de backtraces dû au problème de reconvergence dans les circuits
- Exemple de reconvergence :



Algorithme FAN

- FAN utilise la topologie du circuit pour réduire le nombre de backtraces
  - Identification des noeuds qui sont des sources de reconvergence (checkpoint)
  - Identification des portes de reconvergence
  - Dans la phase de génération :
    - On effectue une simulation (implication) à chaque fois qu'on assigne un signal qui est source de reconvergence (chekpoint)
    - Pour l'étape de justification, on justifie en premier les signaux qui sont des sorties des portes de reconvergence

Questions ?

Algorithme FAN