Validation d'algorithmes

Chapitre VI

Validation & Vérification : Tests en boîte noire

Raphaël Charrondière raphael.charrondiere@inria.fr

Concept

Un test en "boîte noire" ne connaît pas la structure interne du système. D'où son nom !

Concept

Un test en "boîte noire" ne connaît pas la structure interne du système. D'où son nom !

- Les résultats des tests correspondent-ils à la spécification ?
- Taille du jeu de test : éviter l'explosion du nombre de cas à tester ?

Edsgar W. Dijkstra

Testing can reveal the presence of errors but never their absence

Albert Einstein

No amount of experimentation can ever prove me right; a single experiment can prove me wrong.

```
double distance(double[2] P1, double[2] P2);
```

- Si l'on veut tester tout les cas, on a $(2^{64})^4$ cas, soit environ 10^7 cas. Ce qui sur un processeur à 1GHz mettrait 10^{61} ans.

Solution : Partitionner l'espace d'entrée en un nombre fini de classes disjointes. On ne teste que la fonction sur un représentant de chaque classe. Il faut espérer que le programme se comporte correctement sur l'ensemble de la classe !

```
double distance(double[2] P1, double[2] P2){
  double d = sqrt((P1[0]-P2[0])*(P1[0]-P2[0])
                  +(P1[1]-P2[1])*(P1[1]-P2[1]));
  if(P1[0]==1.234)return -d;
  return d;
}
```

Partitionnement

- L'entrée est un réel, dans un intervalle particulier ou non
  - ⇝ Découpe en intervalles autour de valeurs particulières
- L'entrée est un ensemble
  - ⇝ L'ensemble vide est un bon cas particulier, un ensemble ayant trop de valeurs aussi !
- L'entrée est dans un ensemble fini (par exemple c'est un booléen)
  - ⇝ On peut tout tester !
- L'entrée contient plusieurs variables
  - ⇝ On combine les classes précédentes, c-à-d produit cartésien (Attention à l'explosion du nombre de classes)
  - ⇝ On trouve des cas limites pour faire les tests.

Il faut surtout penser aux cas limites ! Les erreurs y sont plus fréquentes.

- Valeur absolue (`int`$\mapsto$`int`) 3 classes particulières : <0, $=0$, >0
- Max d'un ensemble : 0 élément, 1 élément, 2 éléments, [3-10]éléments, [10-∞]éléments.
- Fonction `EstUneApprox`:{MéthodeA, MéthodeB, MéthodeC}$\mapsto$`bool` : pas besoin de classes, on peut tout tester !
- `Alignés`:(Pt a,Pt b,Pt c)$\mapsto$`bool`
  Considérer les trois points séparément est une aberration, cependant on peut aisément créer plusieurs classes "cas particuliers" :
  - $a = b = c$
  - $a = b$ et $a \neq c$, puis les classes symétriques
  - Les points sont alignés, mais créons trois classes selon que a,b ou c soit au milieu
  - Les points ne sont pas alignés, créons deux classes selon que a,b,c soient dans le sens horaire ou anti-horaire

```
double distance(double[2] P1, double[2] P2);
```

On dresse une liste des cas intéressants :

(a1i)	P1[0]<P2[0]	(b1i)	P1[0]<0	(c1i)	P2[0]<0
(a2i)	P1[1]<P2[1]	(b2i)	P1[1]<0	(c2i)	P2[0]<0
(a1e)	P1[0]=P2[0]	(b1e)	P1[0]=0	(c1e)	P2[0]=0
(a2e)	P1[1]=P2[1]	(b2e)	P1[1]=0	(c2e)	P2[0]=0
(a1s)	P1[0]>P2[0]	(b1s)	P1[0]>0	(c1s)	P2[0]>0
(a2s)	P1[1]>P2[1]	(b2s)	P1[1]>0	(c2s)	P2[0]>0

Il y a trop de cas ! Si l'on fait le produit des classes engendrées par a1, a2, b1, b2, c1 et c2, on se retrouve avec $3^6 = 719$ classes (certes, certaines sont vides, par ex. avoir a1e et b1i et c1s est impossible)

- Il s'agit de réduire le nombre de classes.

Imaginons que l'on ait un produit cartésien d'ensemble de classe.
On peut voir la chose comme un ensemble de caractéristiques
$c_i : \mathcal{E} \mapsto \mathcal{S}_i$ fonctions de l'espace d'entrée dans un ensemble fini
(par exemple une propriété est vrai ou faux, ou une expression est strictement négative, nulle ou strictement positive)
L'approche "pairwise" (par paires) :
$\forall i, j \ \forall s_i \in \mathcal{S}_i \ \forall s_j \in \mathcal{S}_j \ \exists e \in \mathcal{DT} \ c_i(e) = s_i \wedge c_j(e) = s_j$ où $\mathcal{DT}$ désigne les données de test. « Pour toute paire de caractéristiques, pour toute valeur de ces caractéristiques, il existe une donnée de test qui les satisfait. »

Rendu d'une page web :

Os	Navigateur	Sécurité[1]	Résolution
Linux	Firefox	Sans	Petite
Mac	Chrome	Modérée	Moyenne
Windows	Safari	Haute	Grande

Solution par paires : 9 d.t. au lieu de 81

Linux	Firefox	Sans	Petite	1	1	1	1
Linux	Chrome	Haute	Grande	1	2	3	3
Linux	Safari	Modérée	Moyenne	1	3	2	2
Mac	Safari	Haute	Moyenne	2	1	3	2
Mac	Chrome	Modérée	Petite	2	2	2	1
Mac	Safari	Sans	Grande	2	3	1	3
Windows	Firefox	Modérée	Grande	3	1	2	3
Windows	Chrome	Sans	Moyenne	3	2	1	2
Windows	Safari	Haute	Petite	3	3	3	1

[1] Activation de javascript, des cookies...

- Diminution drastique du nombre de tests nécessaires.
- En pratique la majorité des fautes est détectée
- Les bugs demandant une combinaison de 3 caractéristiques ne sont pas forcément détectés (à moins d'avoir de la chance et que les DT contiennent la bonne combinaison)
- Généralisation de paire à triplet, à t-uplet

Attention trouver l'ensemble minimal de tests pour couvrir tout les t-uplets est NP-complet (mais en pratique, l'explosion combinatoire nous contraint à borner t !)

Lorsque l'espace de sortie est fini, et que les classes sont un produit cartésien de caractéristiques, on peut présenter les besoins de tests sous forme d'une table de décisions.
On appelle alors les caractéristiques « causes » et la sortie « effets » (il peut y avoir plusieurs variables de sorties).

Exemple : assurance pour un prêt

Variante	1	2	3	4	5	6	7	8
Note	A	A	A	A	B	B	B	B
Montant du prêt ($\times$100 000)	< 1	$[1, 3[$	$[3, 6[$	$\geqslant 6$	< 1	$[1, 3[$	$[3, 6[$	$\geqslant 6$
Assurance obligatoire	Non	Oui	Oui	Oui	Non	Oui	Oui	Oui
Taux assurance (en ‰)		1	3	5		2.5	5	7.5

Si un ensemble de colonnes de la table sont identiques sauf pour une cause, et que toutes les variantes de cette cause sont présentes, alors on peut fusionner les colonnes.

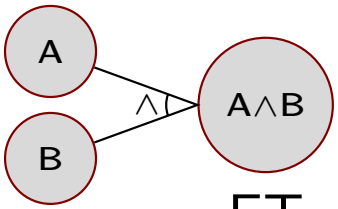
Variante	1	2	3	4	6	7	8
Note	*	A	A	A	B	B	B
Montant du prêt ($\times$100 000)	< 1	$[1, 3[$	$[3, 6[$	$\geqslant 6$	$[1, 3[$	$[3, 6[$	$\geqslant 6$
Assurance obligatoire	Non	Oui	Oui	Oui	Oui	Oui	Oui
Taux assurance (en ‰)		1	3	5	2.5	5	7.5

On peut alors faire un test par colonne de la table simplifiée.

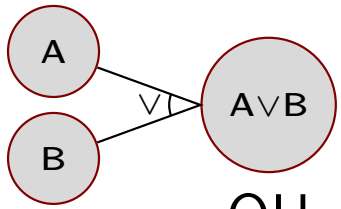
Le graphe Causes-effets est une autre façon pour générer une table de décision compacte. Les causes sont cependant binaires.
Nœuds du graphe:

- Causes : ce sont les entrées possibles
- Effets : ce sont les sorties
- Des nœuds intermédiaires peuvent représenter des combinaisons de causes, de manière générale, chaque nœud est une formule logique.

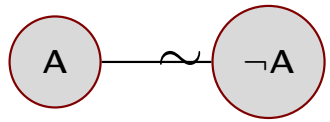
Liens du graphe:



ET

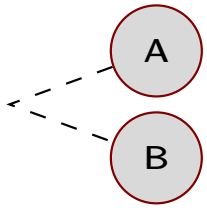


OU



NON

Exclusion mutuelle



Etc.

Module : `CheckEntrée`
Entrée : 2 caractères données par l'utilisateur.
Comportement :

- Si le premier caractère n'est ni 'A' ni 'B', le message X est imprimé
- Si le second caractère n'est pas un chiffre, le message Y est imprimé
- Sinon une instruction est transmise à un driver.

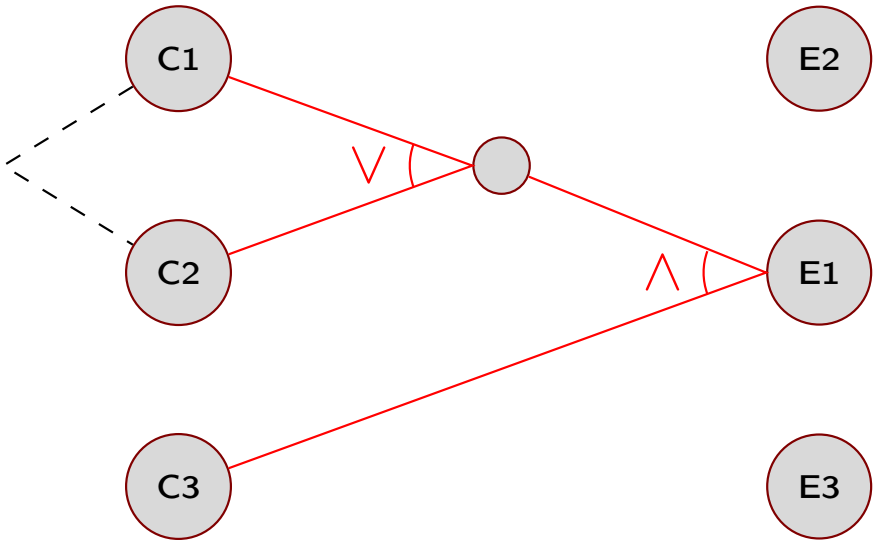
- **Choix des causes :**
  - C1 : le 1er caractère est 'A'
  - C2 : le 1er caractère est 'B'
  - C3 : le 2nd caractère est un chiffre
- **Liste des effets** :
  - E1 : Une instruction est transmise
  - E2 : Le message X est imprimé
  - E3 : Le message Y est imprimé

Graphe Causes-effets : exemple

C1 : le 1er caractère est 'A'
C2 : le 1er caractère est 'B'
C3 : le 2nd caractère est un chiffre
E1 : Une instruction est transmise
E2 : Le message X est imprimé
E3 : Le message Y est imprimé

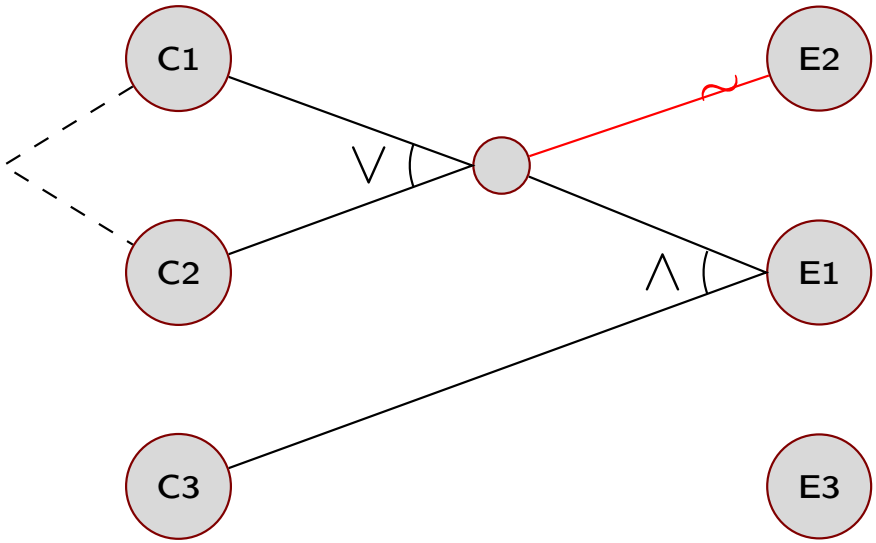
On ne peut avoir C1 et C2 en même temps, car A et B sont des caractères distincts.

Graphe Causes-effets : exemple



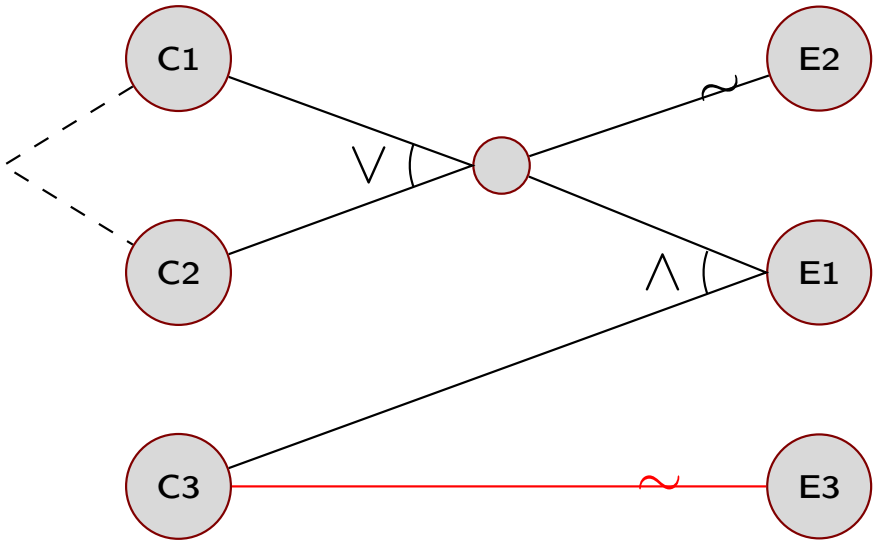
C1 : le 1er caractère est 'A'
C2 : le 1er caractère est 'B'
C3 : le 2nd caractère est un chiffre
E1 : Une instruction est transmise
E2 : Le message X est imprimé
E3 : Le message Y est imprimé

E1 est vrai (une instruction est transmise), quand le premier caractère est A ou B et le deuxième caractère est un chiffre
(C1∨ C2)∧ C3



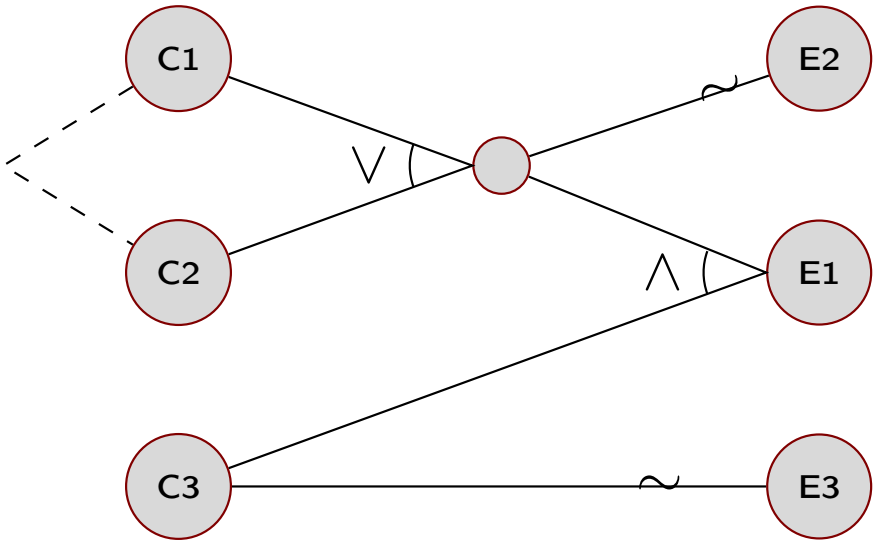
C1 : le 1er caractère est 'A'
C2 : le 1er caractère est 'B'
C3 : le 2nd caractère est un chiffre
E1 : Une instruction est transmise
E2 : Le message X est imprimé
E3 : Le message Y est imprimé

On imprime le message X (E2), si le premier caractère n'est ni A, ni B



C1 : le 1er caractère est 'A'
C2 : le 1er caractère est 'B'
C3 : le 2nd caractère est un chiffre
E1 : Une instruction est transmise
E2 : Le message X est imprimé
E3 : Le message Y est imprimé

On imprime le message Y (E3), si le second caractère n'est pas un chiffre

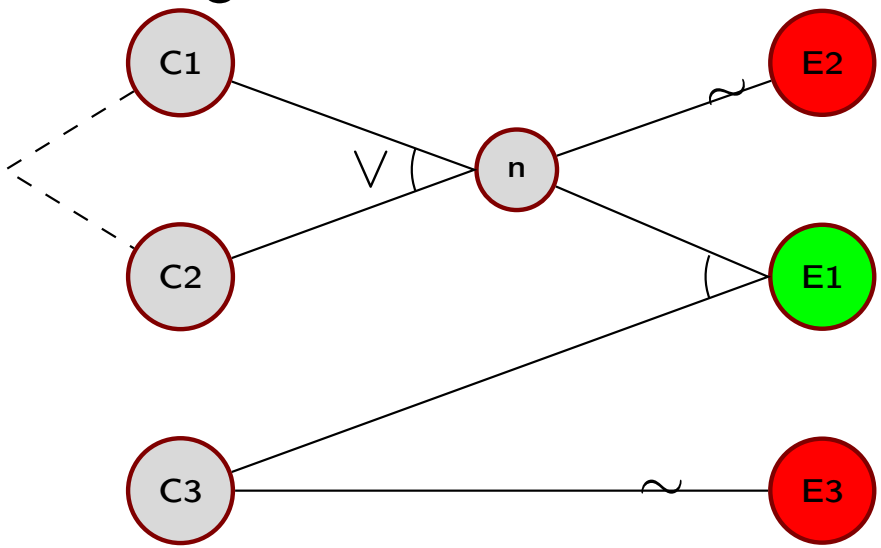


C1 : le 1er caractère est 'A'
C2 : le 1er caractère est 'B'
C3 : le 2nd caractère est un chiffre
E1 : Une instruction est transmise
E2 : Le message X est imprimé
E3 : Le message Y est imprimé

Pour un effet voulu il va falloir se concentrer sur un effet et remonter jusqu'à la source. Cela nous donnera un jeu de causes précis pour activer un effet.

- Utilisons deux couleurs : vert pour vrai, rouge pour faux.
- On commence par colorier les nœuds effets. On peut ne pas colorier un nœud pour ignorer sa valeur.
- On gère les nœuds un à un, une seule fois, et seulement quand ils sont coloriés.
- Si un nœud doit être colorié des deux couleurs, on termine sans solution.
- Pour un lien "NON" on va colorier son prédécesseur de la couleur complémentaire
- Pour un lien "ET" colorié vert, on colorie ses deux prédécesseurs en vert.
- Pour un lien "ET" colorié rouge,
  - si l'un des deux nœuds prédécesseurs est vert on colore l'autre en rouge.
  - si les deux sont verts, on termine sans solution.
  - si les deux sont rouges, on peut passer au nœud suivant
  - sinon on scinde l'exécution : on colore l'un des deux en rouges et on continue, et quand ce sera fini, on reprendra en coloriant l'autre en rouge.
- Quand il n'y a plus de nœud à traiter, on vérifie que deux causes exclues mutuellement ne soient pas colorées, sinon, on a trouvé une combinaison que l'on peut ajouter dans la table de décision.

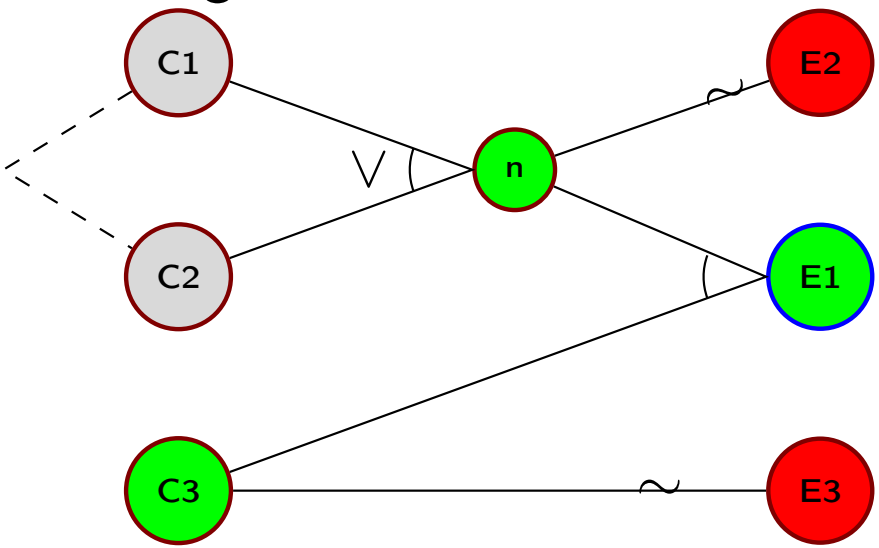
Effet : seul E1 est activé "l'instruction est transmise, pas de message"



Nœuds activés : { }
Solutions trouvées :

`Initialisation, coloriage des effets.`

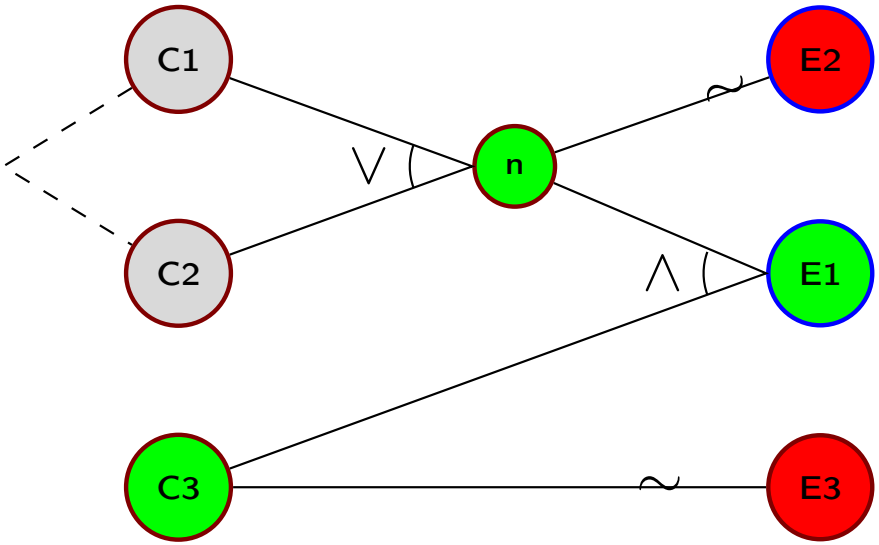
Effet : seul E1 est activé "l'instruction est transmise, pas de message"



Nœuds activés : { E1 }
Solutions trouvées :

On active E1, on colorie les deux prédécesseurs.

Effet : seul E1 est activé "l'instruction est transmise, pas de message"

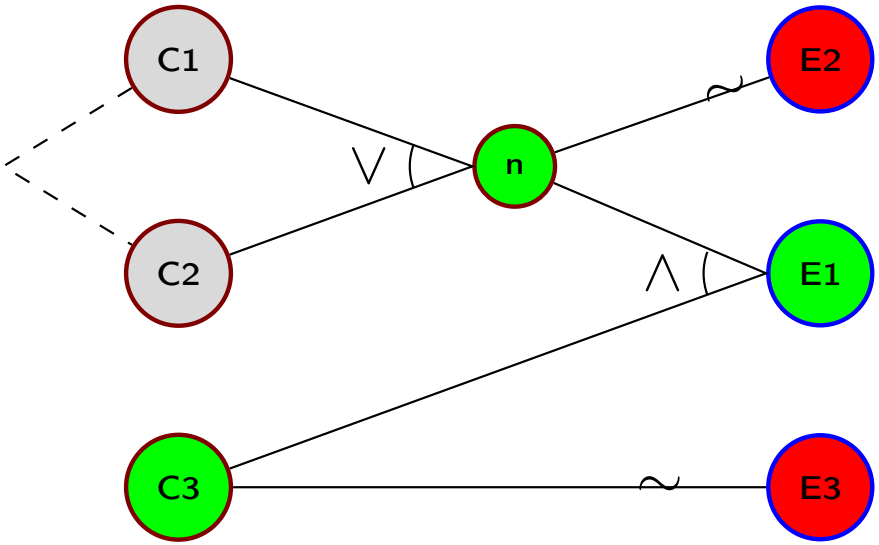


Nœuds activés : { E1,E2 }
Solutions trouvées :

On active E2, le prédécesseur a déjà la bonne couleur.

Remarque : Si on avait désiré l'effet avec le message X, alors l'algorithme aurait terminé sans solution

Effet : seul E1 est activé "l'instruction est transmise, pas de message"

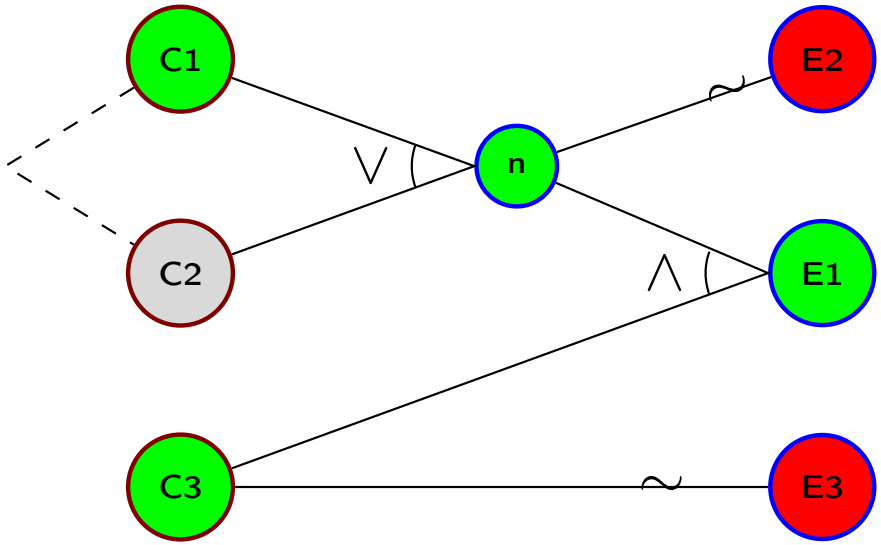


Nœuds activés : { E1,E2,E3 }
Solutions trouvées :

On active E3, le prédécesseur a déjà la bonne couleur.

Remarque : Si on avait désiré l'effet avec le message Y, alors l'algorithme aurait terminé sans solution

Effet : seul E1 est activé "l'instruction est transmise, pas de message"

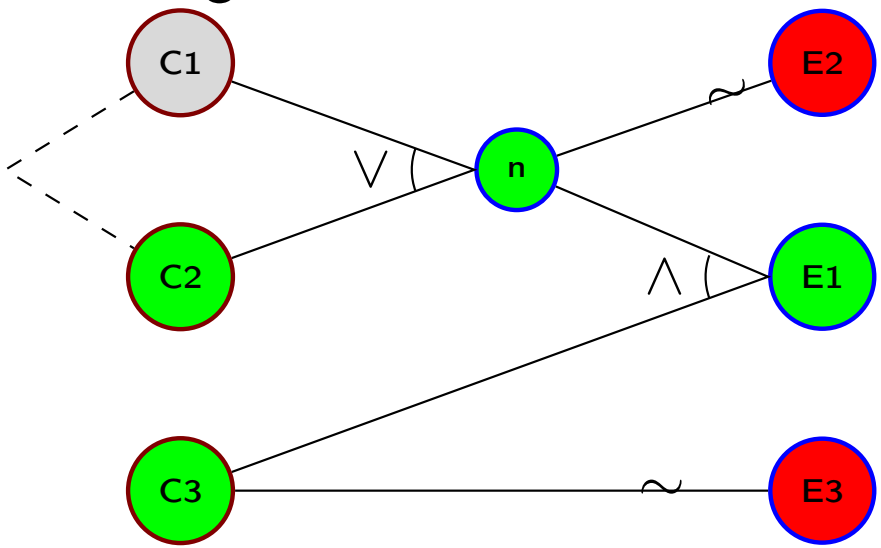


État alternatif en mémoire

Nœuds activés : { E1,E2,E3,n }
Solutions trouvées :
➪ C1 et C3 sont vraies

On active n, l'exécution est scindé en deux , C1 est colorié, l'algorithme est fini.

Effet : seul E1 est activé "l'instruction est transmise, pas de message"



Nœuds activés : { E1,E2,E3,n }

Solutions trouvées :

- C1 et C3 sont vraies
- C1 et C2 sont vraies

L'état alternatif est repris, C2 est colorié, l'algorithme est fini.