Validation d'algorithmes

Chapitre III

Validation & Vérification : Tests en boîte blanche

Raphaël Charrondière raphael.charrondiere@inria.fr

Concept

Les tests en "boite blanche" regardent la structure interne du programme

Concept

Les tests en "boite blanche" regardent la structure interne du programme

Pour un ensemble de tests, on regarde

- La couverture du graphe de flot de contrôle
  - ⇝ Toutes les instructions, conditions sont-elles couvertes ?

  etc.
- La couverture du flot de données
  - ⇝ Toutes les définitions ont-elles une utilisation etc.
- Fautes
  - ⇝ Les tests sont-ils sensibles aux mutations.

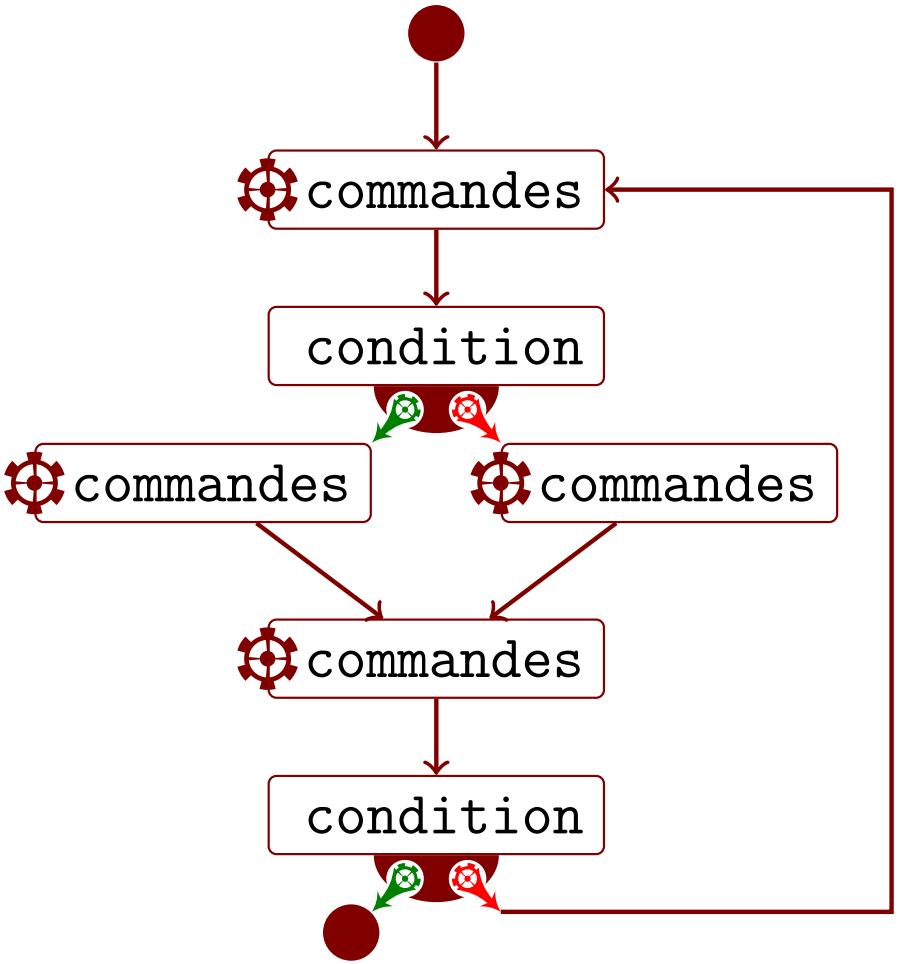
Edsgar W. Dijkstra

Testing can reveal the presence of errors but never their absence

Albert Einstein

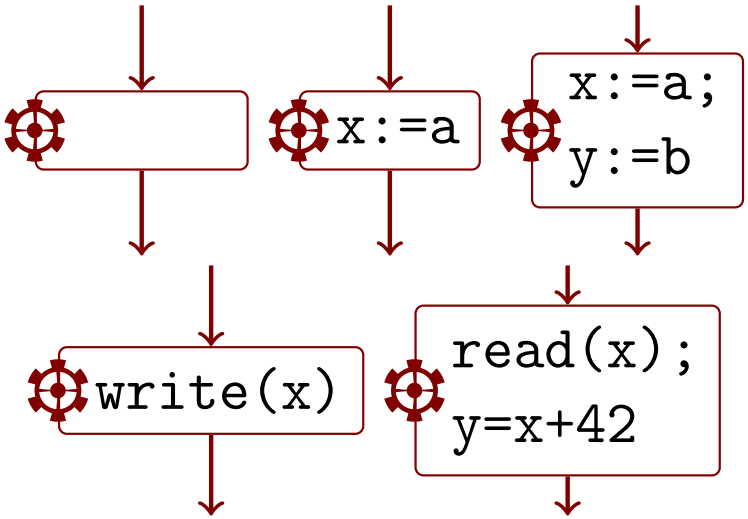
No amount of experimentation can ever prove me right; a single experiment can prove me wrong.

Graphe de flot de contrôle



- Graphe orienté
- Un unique nœud d'entrée
- Un unique nœud de sortie
- Des nœuds de commandes
  - Un seul arc sortant
- Des nœuds de conditions
  - Exactement deux arcs sortants

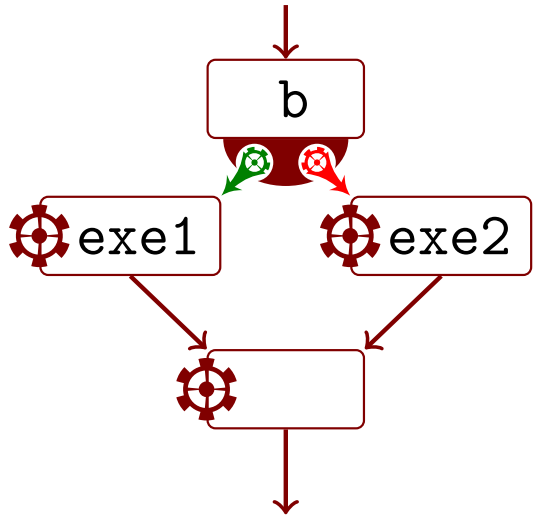
Nœuds de commandes



Tout ce qui est lu est exécuté. Le nœud peut être vide.

Nœuds de condition

if `b` **then** `exe1` **else** `exe2` **fi**

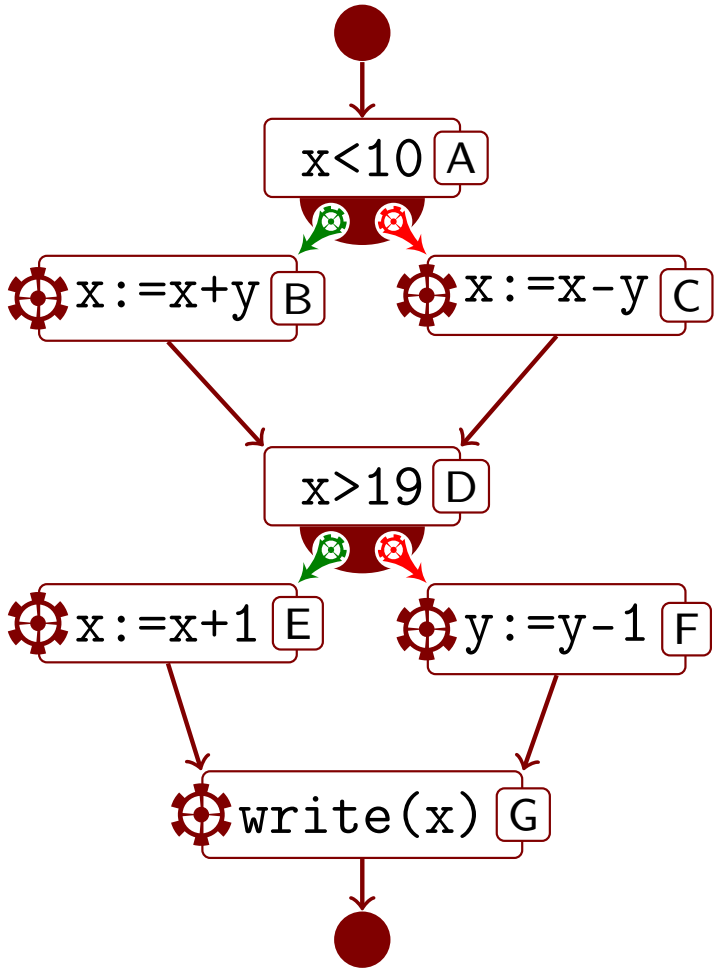


Petite question : traduire
while `b` **do** `exe1` **od**

Code

```
if  x<10  then  x:=x+y
          else  x:=x-y
fi;
if  x>19  then  x:=x+1
          else  y:=y-1
fi;
write(x)
```

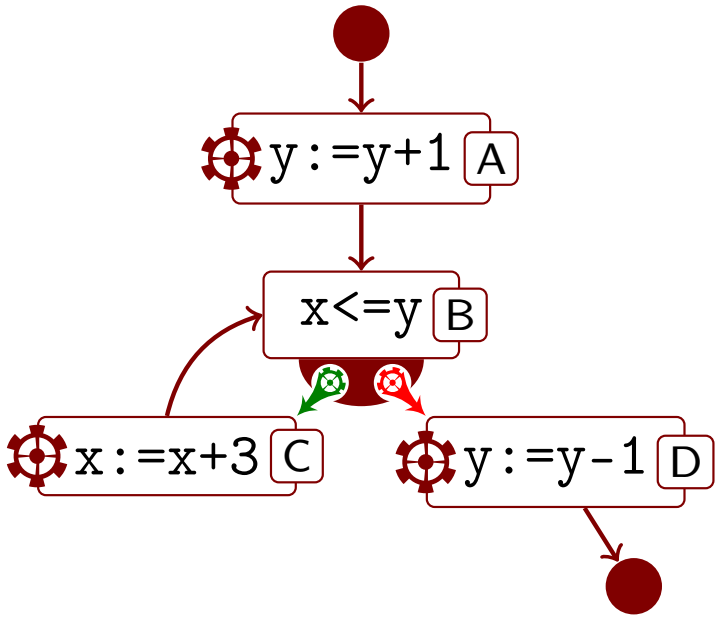
Graphe de flot de contrôle



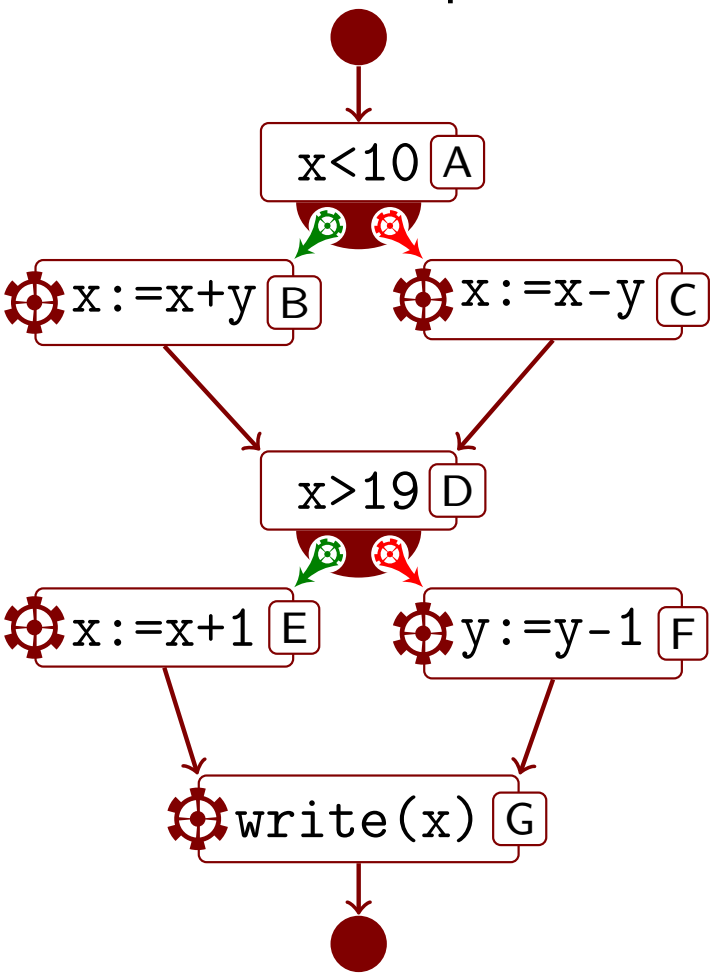
Code

```
y:=y+1;
while x<=y do
                x:=x+3
od;
y:=y-1
```

Graphe de flot de contrôle

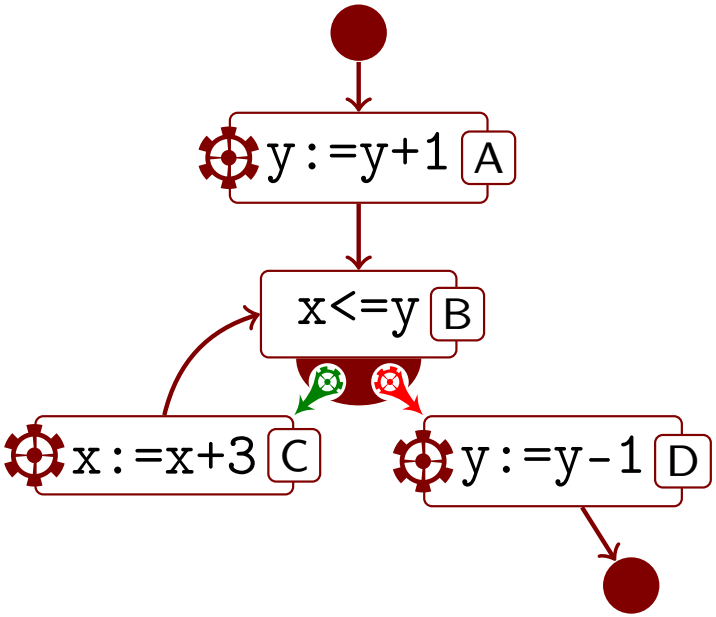


Retour Exemple 1



- Sur l'exemple 1, il y a 4 chemins ABDEG, ACDEG, ABDFG et ACDFG.
- On notera l'ensemble des chemins par P1 = ABDEG + ACDEG + ABDFG + ACDFG.
- Le symbole + signifie « ou »
- On peut factoriser : P1=A(B+C)D(E+F)G

Retour Exemple 2



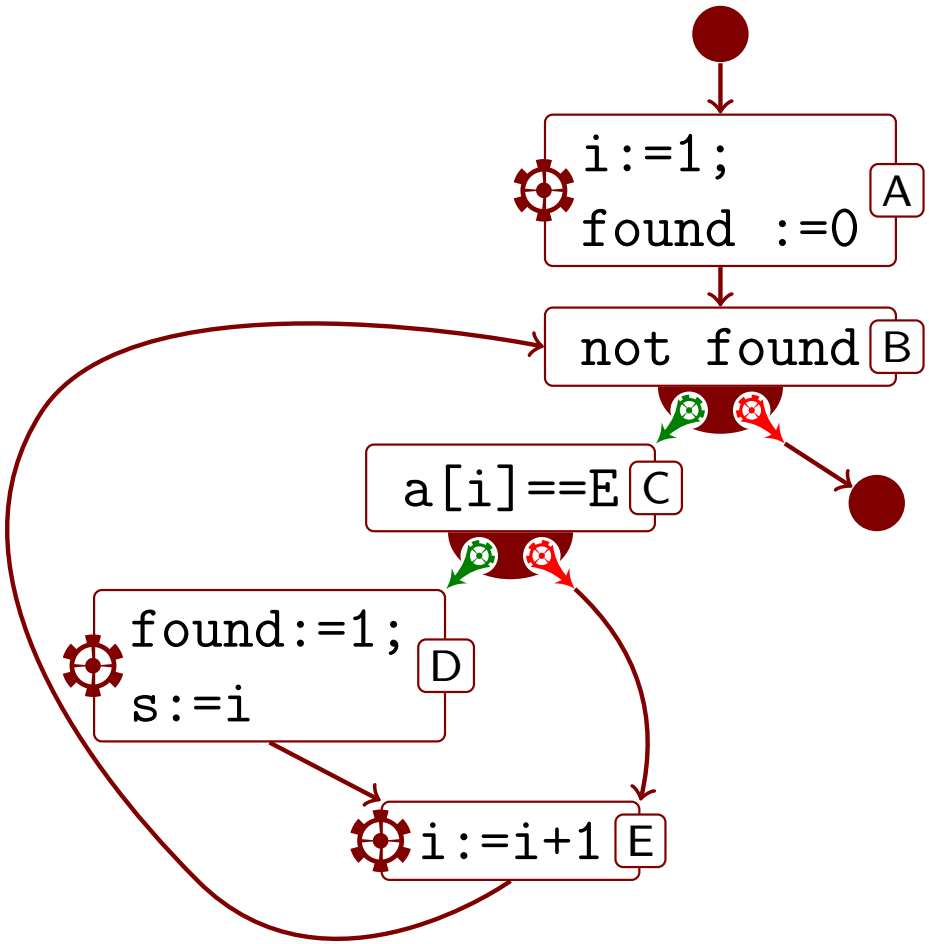
- Sur l'exemple 2, il y a une infinité de chemins
- On notera l'ensemble des chemins par P2 = AB(CB)*D ou A(BC)*BD.
- Le symbole * signifie « plusieurs fois, potentiellement zero fois »

Donne les GFC (graphes de flot de contrôle), et les expressions des chemins des programmes suivants :

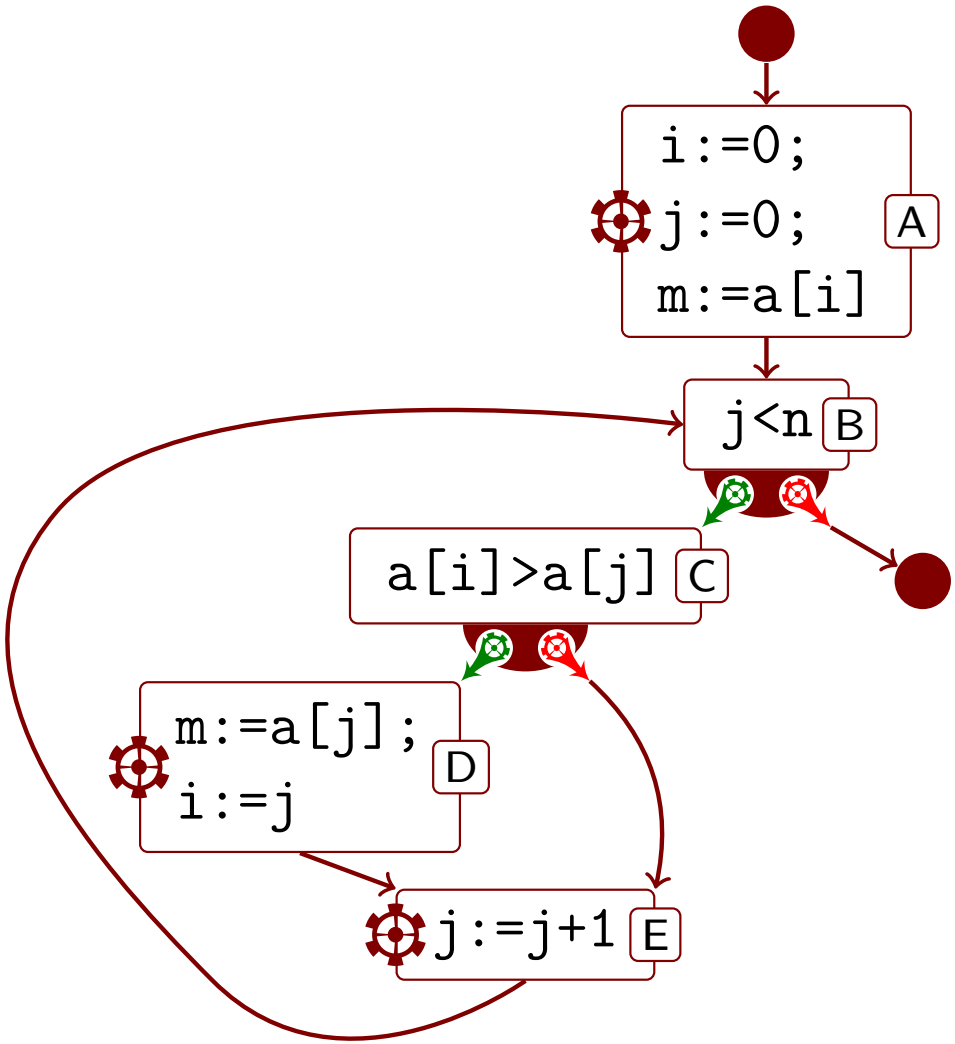
```
i:=1; found :=0;
while not found do
  if a[i]==E then
    found:=1;s:=i
  fi;
  i:=i+1
od;
```

```
i:=0;j:=0;m:=a[i];
while j<n do
  if a[i]>a[j] then
    m:=a[j];i:=j
  fi;
  j:=j+1
od
```

Que calculent ces programmes ? Quels problèmes voyez-vous ?

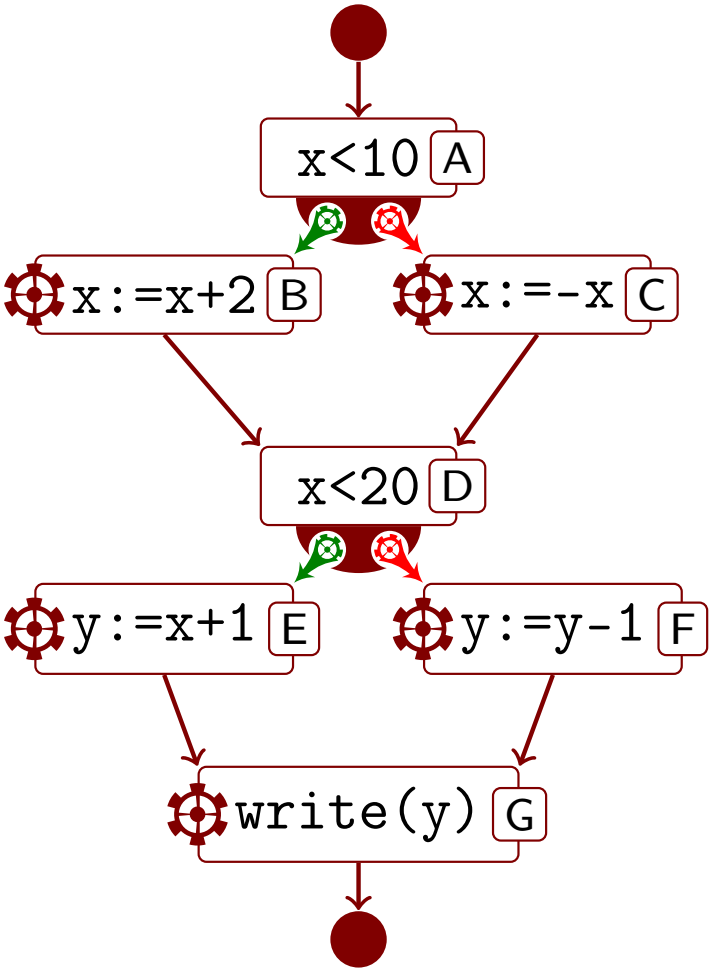


- Le programme calcule l'indice de la première occurrence de E dans le tableau a.
- Chemins P3 = AB(C(E+DE)B)*
- Notation P3 = AB(CD?EB)*
- D? signifie « D ou rien »
- Problème : la boucle ici peut être infinie.
- Remarque : AB appartient à P3, mais aucune entrée ne peut correspondre.



- Le programme calcule le maximum du tableau a.
- Chemins P4 = P3
- Problème : si n vaut 0 a[0] n'est pas défini, il faudrait vérifier que n est bien la taille du tableau.

Variation de l'exemple 1

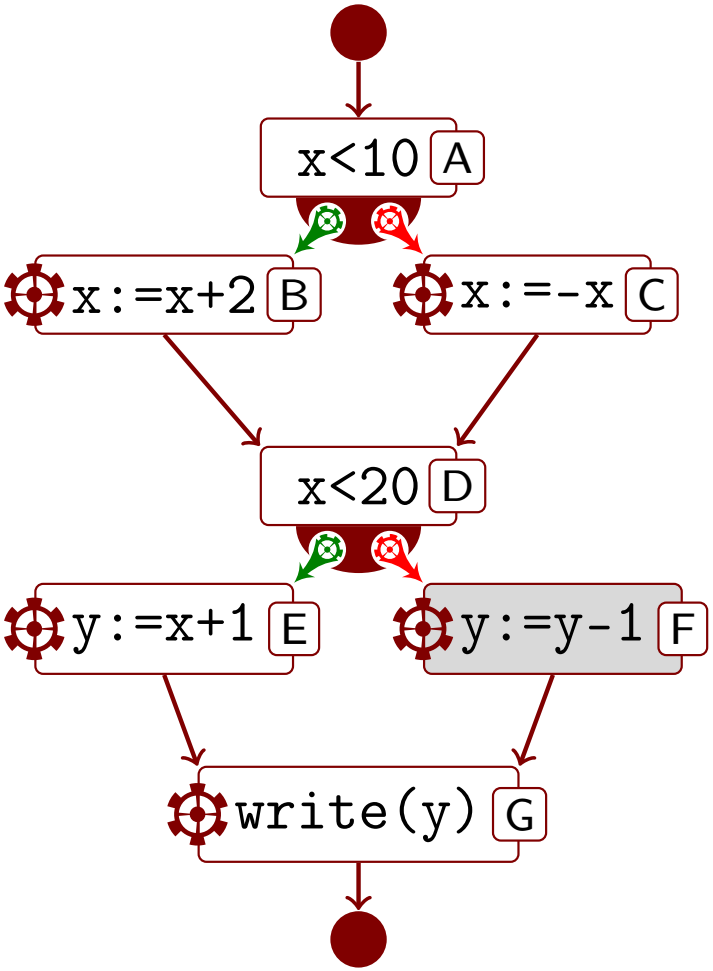


Définition

On appelle données de test, abrévié DT, l'entrée du programme, c'est-à-dire, l'assignation des variables.

Question : trouver une DT pour chacun des chemins ? Spoiler : il y a un problème.

Variation de l'exemple 1



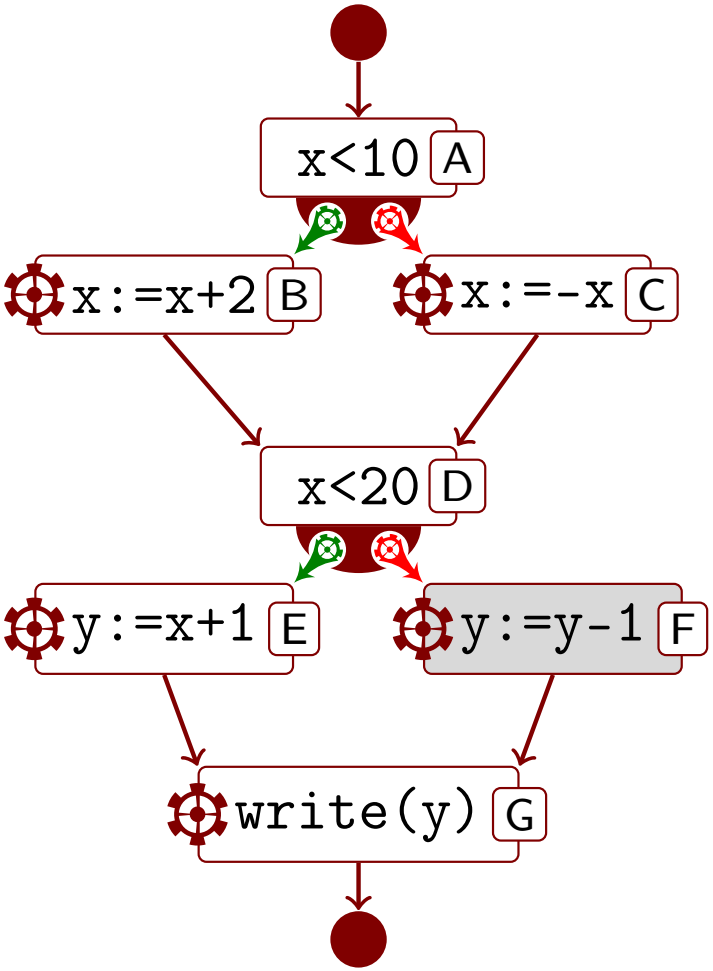
Définition

On appelle données de test, abrévié DT, l'entrée du programme, c'est-à-dire, l'assignation des variables.

Question : trouver une DT pour chacun des chemins ? Spoiler : il y a un problème.

↬ F n'est pas accessible.

Variation de l'exemple 1



Définition

On appelle données de test, abrévié DT, l'entrée du programme, c'est-à-dire, l'assignation des variables.

Question : trouver une DT pour chacun des chemins ? Spoiler : il y a un problème.

F n'est pas accessible.

Définition

Le flot est le chemin emprunté lors de l'exécution.

Définition

Un nœud N est accessible si et seulement si il existe une entrée tel que le chemin correspondant à cette entrée contient N.

Théorème

Le problème d'accessibilité est indécidable.

À par thé

1. Le problème de l'arrêt
2. Équivalence au problème d'accessibilité
3. Dernier détail.

Définition

Soit un programme P et une entrée x, est-ce que P(x) termine ?

Question : Existe-t-il un problème `Halt` qui décide en temps *fini* du problème de l'arrêt ?

L'aventurie.è.r.e [insérer votre nom] est en bien mauvaise posture. Après s'être fait capturer par des robots tueurs, un choix de sentence lui est proposé. E.i.ll.e a droit de prononcer une phrase, si cette phrase est jugée vrai la sentence sera la mort par balle, si cette phrase est jugée fausse la sentence sera la mort par électrocution. Évidement avec ses talents, [insérer votre nom] s'en sort indemne après avoir fait planter les robots. Devinez comment ?

Supposons que `Halt` existe. Considérons alors le programme suivant :

```
diagonale(x):
    si Halt(x, x) est vrai(1)
    alors boucle infinie(2)
    sinon renvoyer vrai(3)
```

Considérer la terminaison de diagonale(diagonale).

- Si diagonale(diagonale) termine (1) alors, on rentre dans une boucle infinie (2), diagonale(diagonale) ne termine pas.
- Mais si diagonale(diagonale) ne termine pas (1) alors diagonale(diagonale) termine(3).

PLANTAGE MACHINE

Définition

Soit P un programme avec $\mathcal{G} = (N, A)$ son graphe de flot de contrôle, $n \in N$ un nœud et x une donnée d'entrée. n est-il atteint lors de l'éxécution $P(x)$?

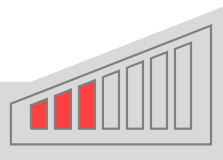
Au lieu prouver que ce problème est indécidable directement, on va créer l'équivalence de ce problème avec celui de l'arrêt. Réduction à partir du problème de terminaison

Accessibilité $\Longrightarrow$ Arrêt : Le noeud de sortie est-il accessible ?

Arrêt $\Longrightarrow$ Accessibilité : Pour tester l'accessibilité du nœud n, on transforme la sortie en boucle infinie, puis on raccorde n à une nouvelle sortie.

Dernier détail

Les problèmes énoncés étaient pour une entrée donnée, or dans notre cas on veut savoir s'il existe une entrée telle qu'un nœud soit atteint.
Question : donner une idée pour résoudre ce dernier détail.



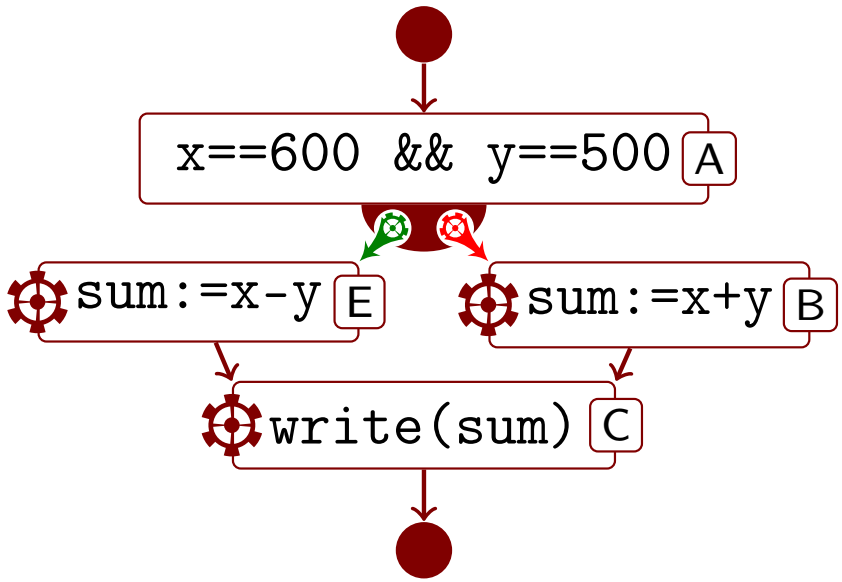
- Dans le cas général, on ne peut pas parcourir tous les chemins (il peut y avoir des boucles par exemple).
- Pour un ensemble de données de test, on va considérer la couverture par nœud ou par arc.
- Un nœud/arc est couvert s'il appartient à un chemin d'une des données de test.
- Le taux de couverture sera alors définie par $\frac{\text{nb de nœuds couverts}}{\text{nb total de nœuds}}$ ou $\frac{\text{nb d'arc couverts}}{\text{nb total d'arc}}$

Addition de deux entiers

Code

```
if x==600 and y==500 then
   sum:=x-y
else
   sum:=x+y
fi
write(sum)
```

Graphe de flot de contrôle



Données de test : DT1=[x=10,y=20], DT2=[x=600,y=500]

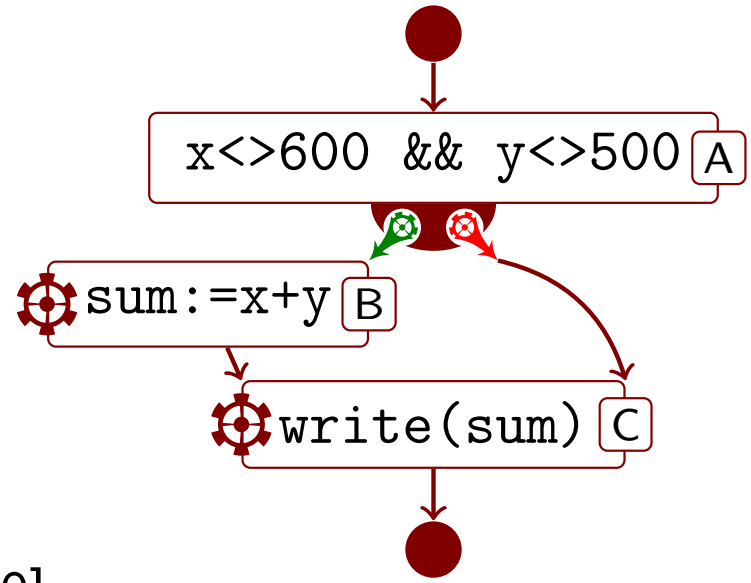
- Tous les nœuds/arcs sont couverts et l'erreur est détectée

Addition de deux entiers

Code

```
if x<>600 and y<>500 then
   sum:=x+y
fi
write(sum)
```

Graphe de flot de contrôle

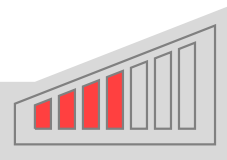


- Données de test : DT1=[x=10,y=20]
  - Tous les nœuds sont couverts mais l'erreur n'est PAS détectée
  - Tous les arcs ne sont pas couverts
- Données de test : DT1=[x=10,y=20], DT2=[x=600,y=500]
  - Tous les arcs sont couverts et l'erreur est détectée

Propriété

Une couverture des arcs implique une couverture des nœuds

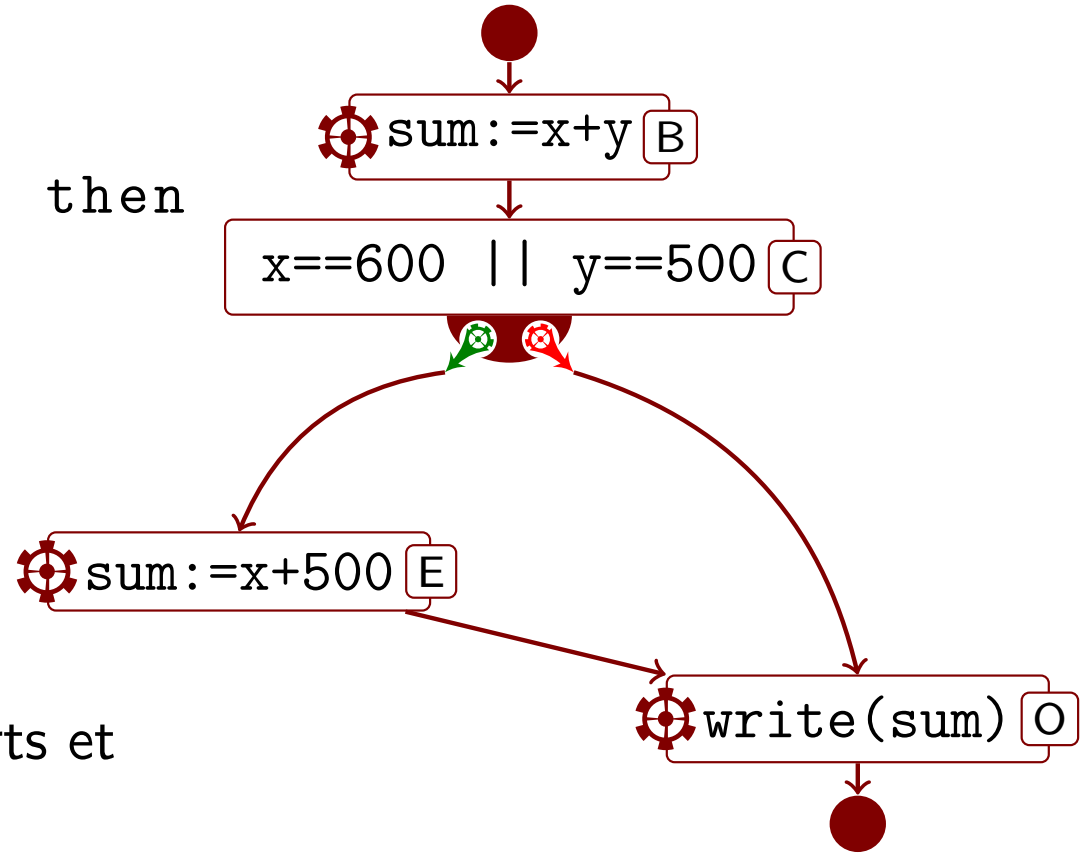
L'inverse n'est pas vrai, mais presque...
Question : comment rendre l'inverse vrai ?



Code

```
sum:=x+y
if x==600 or y==500 then
   sum:=x+500
fi
write(sum)
```

Graphe de flot de contrôle



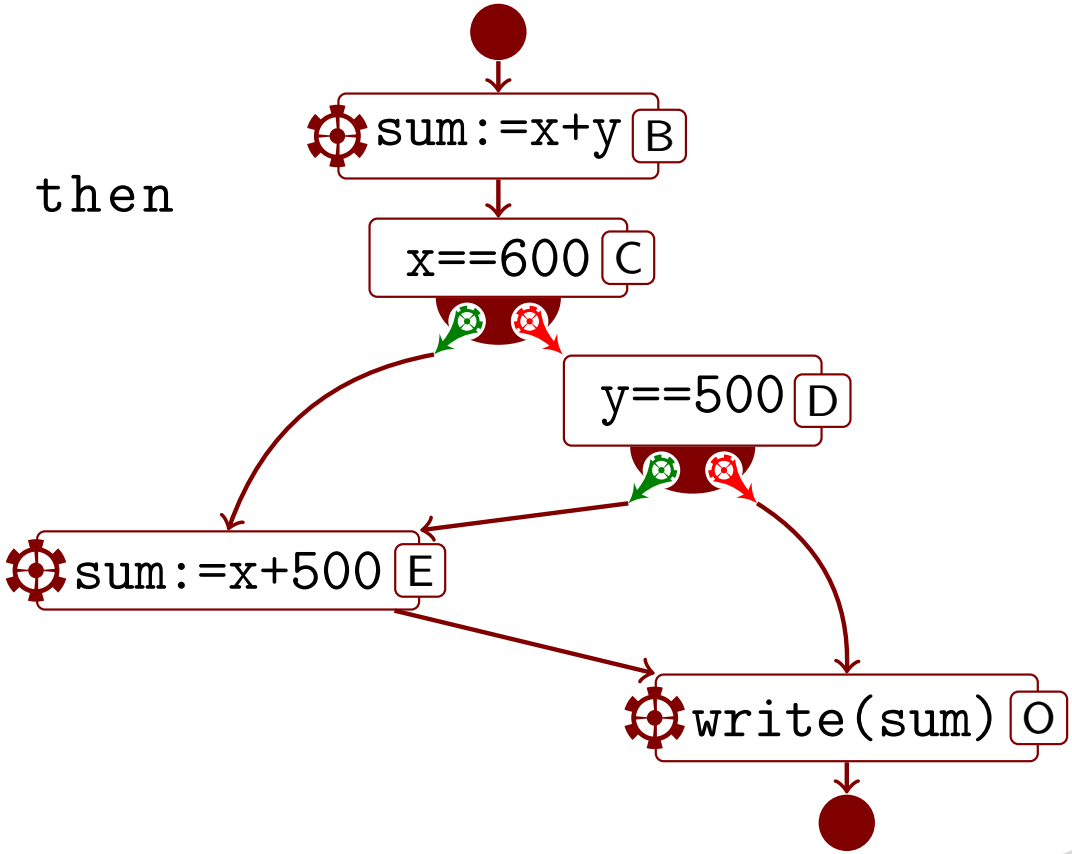
Données de test :
DT1=[x=10,y=20],
DT2=[x=12,y=500]

- Tous les arcs sont couverts et l'erreur n'est PAS détectée
- Il faut scinder la condition

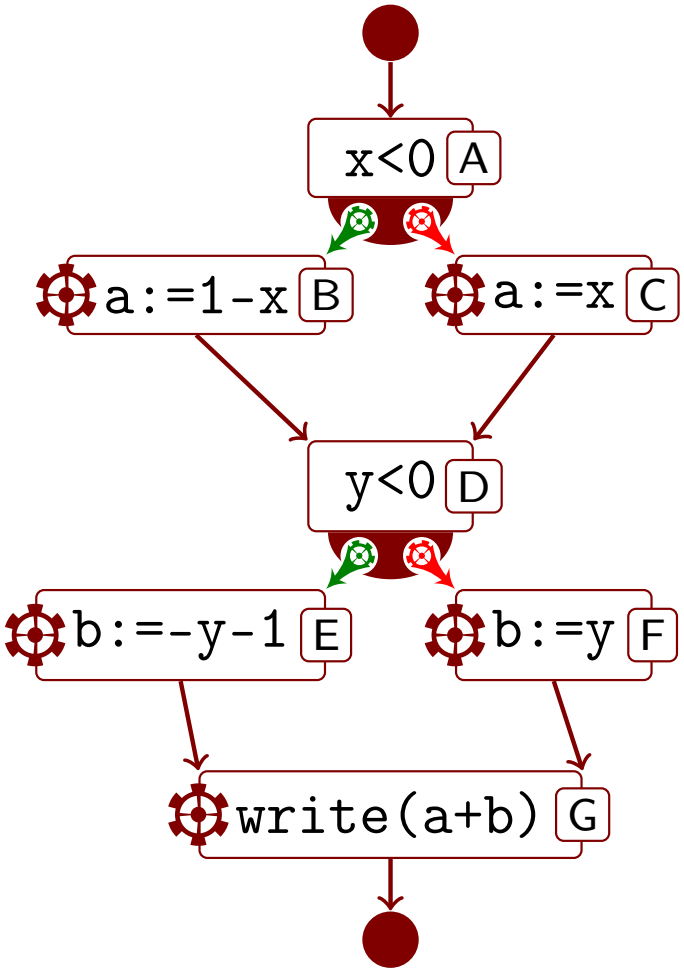
Code

```
sum:=x+y
if x==600 or y==500 then
  sum:=x+500
fi
write(sum)
```

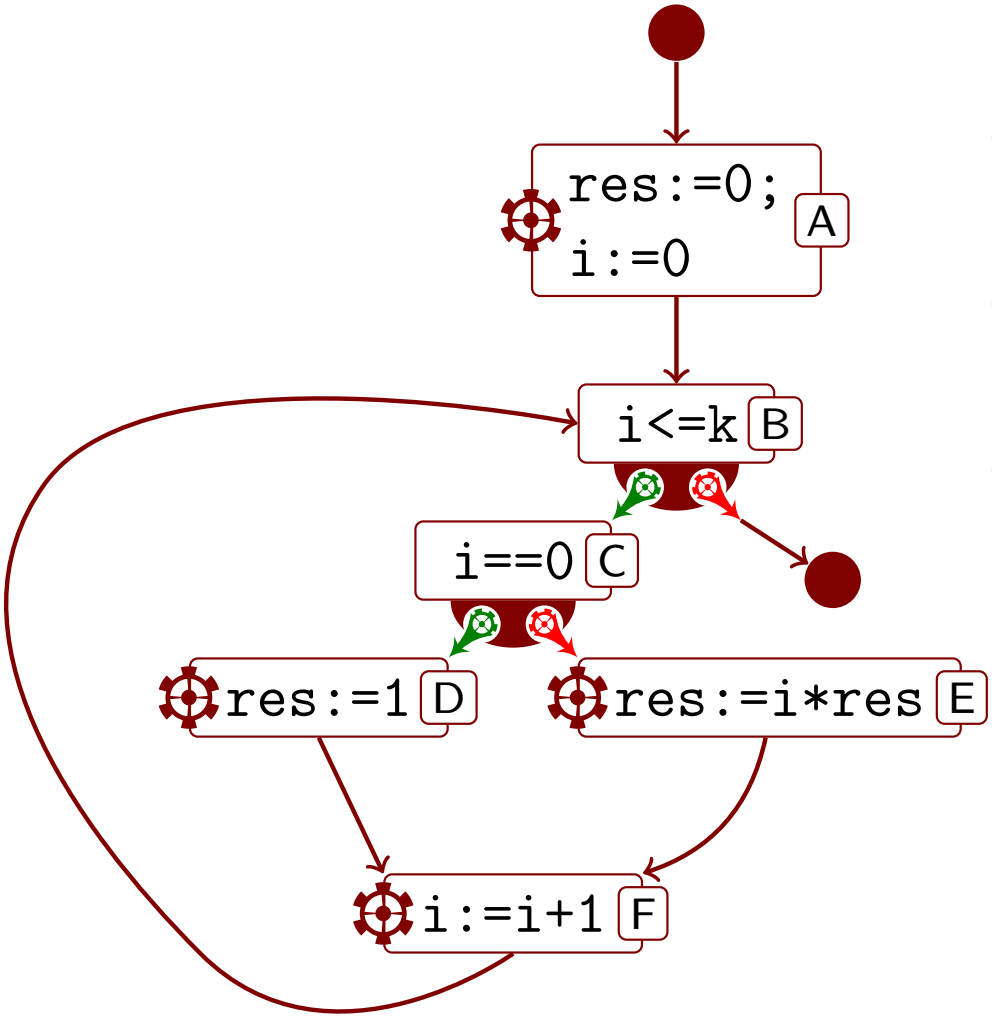
Graphe de flot de contrôle



Graphe de flot de contrôle

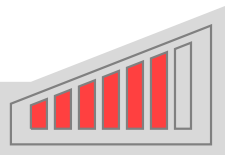


- Dans cet exemple le programme doit calculer la fonction abs(x)+abs(y).
- Il y a effectivement une erreur.
- Données de test :
  DT1=[x=10,y=20],
  DT2=[x=-2,y=-3]
  ⇝ Tous les nœuds/arcs sont couverts mais l'erreur n'est PAS détectée



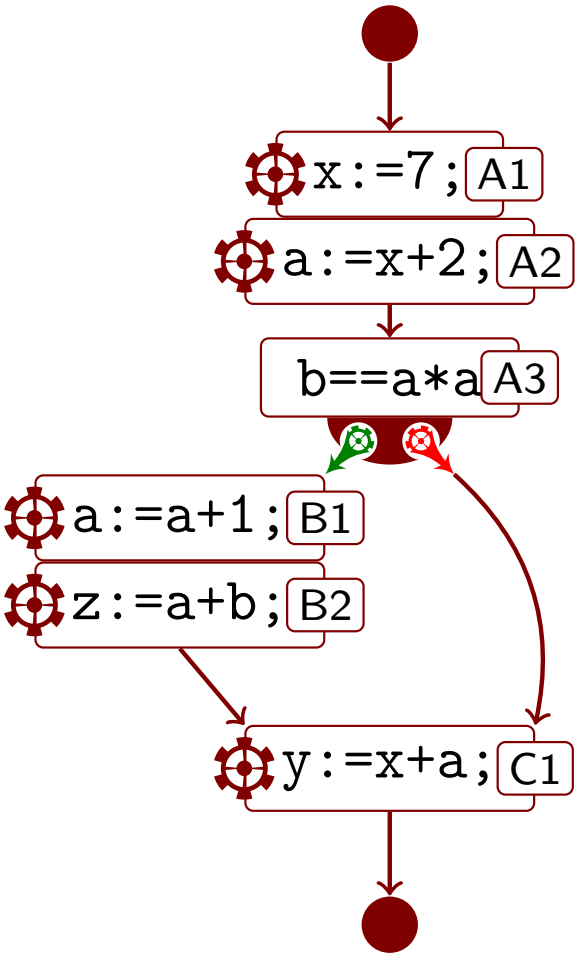
- Le programme calcule `factoriel(k)`.
- Il y a effectivement une erreur.
- Données de test :
  DT1=[k=2]
  ↬ Tous les nœuds/arcs sont couverts mais l'erreur n'est PAS détectée.

On verra une façon de traiter ces problèmes de dépendance dans le prochain chapitre.



Après la structure du graphe, on peut s'intéresser au contenu des nœuds.

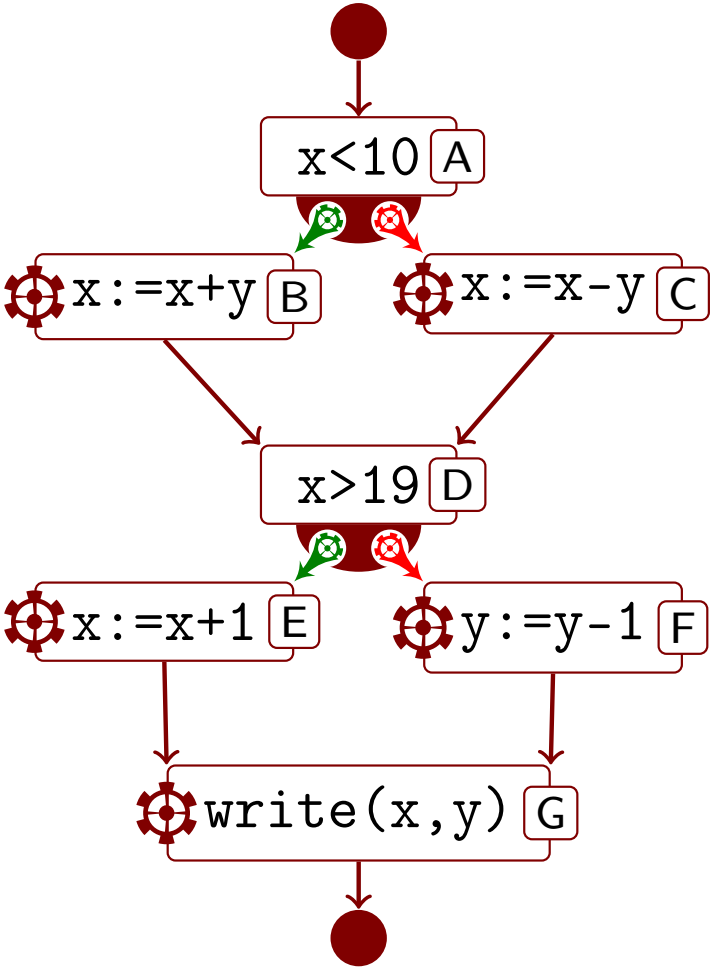
- C'est souvent utilisé pour l'analyse statique.
- On détecte l'utilisation d'une variable non initialisée, ou une variable affectée, mais jamais utilisée par exemple.
- Une variable est **définie** lors d'une instruction si sa valeur est modifiée `a:=...`
- Une variable est **utilisée** lors d'une instruction si sa valeur est lue.



Définition

Un chemin est dr-strict pour une variable x s'il commence par une définition de x, fini par une utilisation de x, et ne contient pas d'autre définition de x.

- [A1, A2, A3, B1, B2, C1] est-il dr-strict pour x ?
- [A2, A3, B1, B2, C1] est-il dr-strict pour a ?
- [A1, A2, A3, B1] est-il dr-strict pour a ?
- [A1, A2, A3, C1] est-il dr-strict pour a ?

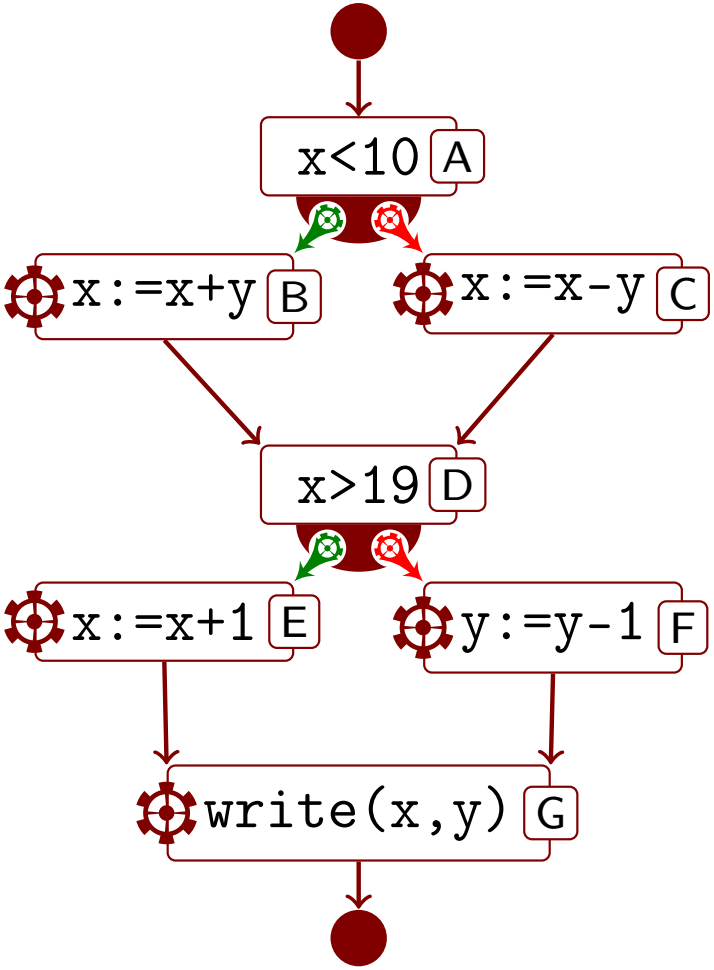


Définition

Pour chaque définition il y a un chemin dr-strict dans un test (c.-à-d. le flôt passe par un tel chemin pour une des DT)

Pour couvrir le critère il faut par exemple les chemins ABDEG et ACDFG

Critère tous-les-utilisateurs



Définition

Pour chaque définition et chaque utilisation accessible à partir de cette déf. il y a un chemin dr-strict dans un test

Pour couvrir le critère il faut par exemple les chemins ABDEG, ACDEG et ACDFG