Les tests

Crédits

- Sur la nuance entre *erreur* et *défaut*, cet article de Cynthia Lefevre.

Crédits

- Sur la nuance entre *erreur* et *défaut*, cet article de Cynthia Lefevre.
- Ce cours du Labri

Crédits

- Sur la nuance entre *erreur* et *défaut*, cet article de Cynthia Lefevre.
- Ce cours du Labri
- Cet article de Wikipedia.

Crédits

- Sur la nuance entre *erreur* et *défaut*, cet article de Cynthia Lefevre.
- Ce cours du Labri
- Cet article de Wikipedia.
- Ce cours du LRI de Paris-Saclay

Crédits

- Sur la nuance entre *erreur* et *défaut*, cet article de Cynthia Lefevre.
- Ce cours du Labri
- Cet article de Wikipedia.
- Ce cours du LRI de Paris-Saclay
- Un cours du CNAM avec des exemples de normes réclamant les critères étudiés.

Terminologie

Erreur Action humaine produisant un résultat incorrect. ≪ l'erreur est humaine ≫.

Terminologie

Erreur Action humaine produisant un résultat incorrect. ≪ l'erreur est humaine ≫.

Défaut Une imperfection dans un composant (ou un système) qui peut conduire à ce que ce composant (ou système) n'exécute pas les fonctions requises. On cherche les défauts sans manipuler le système.

Terminologie

Erreur Action humaine produisant un résultat incorrect. ≪ l'erreur est humaine ≫.

Défaut Une imperfection dans un composant (ou un système) qui peut conduire à ce que ce composant (ou système) n'exécute pas les fonctions requises. On cherche les défauts sans manipuler le système.

Défaillance ou Panne IEEE 729 : La fin de la capacité d'un système ou d'un de ses composants d'effectuer la fonction requise, ou de l'effectuer à l'intérieur de limites spécifiées. On cherche une défaillance en manipulant le système.

Terminologie

Erreur Action humaine produisant un résultat incorrect. ≪ l'erreur est humaine ≫.

Défaut Une imperfection dans un composant (ou un système) qui peut conduire à ce que ce composant (ou système) n'exécute pas les fonctions requises. On cherche les défauts sans manipuler le système.

Défaillance ou Panne IEEE 729 : La fin de la capacité d'un système ou d'un de ses composants d'effectuer la fonction requise, ou de l'effectuer à l'intérieur de limites spécifiées. On cherche une défaillance en manipulant le système.

Anomalie Un écart entre le résultat attendu et celui obtenu.

Terminologie

Erreur Action humaine produisant un résultat incorrect. ≪ l'erreur est humaine ≫.

Défaut Une imperfection dans un composant (ou un système) qui peut conduire à ce que ce composant (ou système) n'exécute pas les fonctions requises. On cherche les défauts sans manipuler le système.

Défaillance ou Panne IEEE 729 : La fin de la capacité d'un système ou d'un de ses composants d'effectuer la fonction requise, ou de l'effectuer à l'intérieur de limites spécifiées. On cherche une défaillance en manipulant le système.

Anomalie Un écart entre le résultat attendu et celui obtenu.

Remarque

Ce n'est pas parce qu'il y a anomalie qu'il y a défaut. Peut-être le testeur a-t-il compris la spécification d'une façon et le développeur d'une autre.

Défaut VS Défaillance

Il y a un rapport de causalité.

- A retenir : Un défaut (dans le code) ou une erreur (de l'utilisateur du programme) peut causer (un jour) une défaillance.

Défaut VS Défaillance

Il y a un rapport de causalité.

- A retenir : Un défaut (dans le code) ou une erreur (de l'utilisateur du programme) peut causer (un jour) une défaillance.
- On peut trouver un défaut sans manipuler le système (analyse du code).

Défaut VS Défaillance

Il y a un rapport de causalité.

- A retenir : Un défaut (dans le code) ou une erreur (de l'utilisateur du programme) peut causer (un jour) une défaillance.
- On peut trouver un défaut sans manipuler le système (analyse du code).
- Mais on trouve une défaillance en manipulant le système.

Test

- Définition (norme IEEE 729) : Le *test* est un processus manuel ou automatique, qui vise à établir qu'un système vérifie les propriétés exigées par sa spécification, ou à détecter des différences entre les résultats engendrés par le système et ceux qui sont attendus par la spécification.

Test

- Définition (norme IEEE 729) : Le *test* est un processus manuel ou automatique, qui vise à établir qu'un système vérifie les propriétés exigées par sa spécification, ou à détecter des différences entre les résultats engendrés par le système et ceux qui sont attendus par la spécification.
- Le test ne se préoccupe pas des erreurs. Ce n'est pas au testeur de pointer les imperfections humaines.

Test

- Définition (norme IEEE 729) : Le *test* est un processus manuel ou automatique, qui vise à établir qu'un système vérifie les propriétés exigées par sa spécification, ou à détecter des différences entre les résultats engendrés par le système et ceux qui sont attendus par la spécification.
- Le test ne se préoccupe pas des erreurs. Ce n'est pas au testeur de pointer les imperfections humaines.
- Le test met en évidence les défauts afin de prévenir les défaillances.

Tests en général

- Toute fabrication de produit suit les étapes suivantes

Tests en général

- Toute fabrication de produit suit les étapes suivantes
  1. Conception

Tests en général

- Toute fabrication de produit suit les étapes suivantes
  1. Conception
  2. Réalisation

Tests en général

- Toute fabrication de produit suit les étapes suivantes
  1. Conception
  2. Réalisation
  3. Tests

Tests en général

- Toute fabrication de produit suit les étapes suivantes
  1. Conception
  2. Réalisation
  3. Tests
- Test : On s'assure que le produit final correspond à ce qui a été demandé selon divers critères :

Tests en général

- Toute fabrication de produit suit les étapes suivantes
  1. Conception
  2. Réalisation
  3. Tests
- Test : On s'assure que le produit final correspond à ce qui a été demandé selon divers critères :
  - esthétique,

Tests en général

- Toute fabrication de produit suit les étapes suivantes
  1. Conception
  2. Réalisation
  3. Tests
- Test : On s'assure que le produit final correspond à ce qui a été demandé selon divers critères :
  - esthétique,
  - performance,

Tests en général

- Toute fabrication de produit suit les étapes suivantes
  1. Conception
  2. Réalisation
  3. Tests
- Test : On s'assure que le produit final correspond à ce qui a été demandé selon divers critères :
  - esthétique,
  - performance,
  - ergonomie,

Tests en général

- Toute fabrication de produit suit les étapes suivantes
  1. Conception
  2. Réalisation
  3. Tests
- Test : On s'assure que le produit final correspond à ce qui a été demandé selon divers critères :
  - esthétique,
  - performance,
  - ergonomie,
  - fonctionnalité,

Tests en général

- Toute fabrication de produit suit les étapes suivantes
  1. Conception
  2. Réalisation
  3. Tests
- Test : On s'assure que le produit final correspond à ce qui a été demandé selon divers critères :
  - esthétique,
  - performance,
  - ergonomie,
  - fonctionnalité,
  - robustesse etc.

Des Bogues aux conséquences désastreuses

Suède - 1980 776 millions d'euros (275 millions de dollars de 1980).
104 camions semi-remorques de 35 tonnes, c'est la cargaison que le Zenobia emporta par le fond lors de son naufrage, 1 mois seulement après son lancement inaugural.
Pourquoi ? Un bug dans le logiciel de contrôle des ballasts surremplissait les réservoirs par rapport aux indications qui lui étaient envoyées.

Des Bogues aux conséquences désastreuses

Suède - 1980 776 millions d'euros (275 millions de dollars de 1980).
104 camions semi-remorques de 35 tonnes, c'est la cargaison que le Zenobia emporta par le fond lors de son naufrage, 1 mois seulement après son lancement inaugural.
Pourquoi ? Un bug dans le logiciel de contrôle des ballasts surremplissait les réservoirs par rapport aux indications qui lui étaient envoyées.

EADS - 1996 266 millions d'euros (200 millions d'euros de 1996)
Le 4 juin 1996 (1er vol), la fusée Ariane 5 s'est élevée jusqu'à la moitié du ciel, avant que son logiciel ne coupe les moteurs et que le lanceur explose en plein vol.
Pourquoi ? L'accélération maximale de la fusée a dépassé la valeur admissible par le logiciel, générant une erreur générale et donc l'autodestruction. Valeurs recopiées du programme Ariane 4 (moins puissante). La simulation avait été annulée afin d'économiser 800.000 francs (119000 euros).

Coût des tests

- Budget IT : Budget alloué aux technologies de l'information (ou IT pour Information Technology).

Coût des tests

- Budget IT : Budget alloué aux technologies de l'information (ou IT pour Information Technology).
- Dans les entreprises, la part du budget IT consacrée aux tests et à l'assurance qualité augmente de 9 points d'une année sur l'autre selon Capgemini.

Coût des tests

- Budget IT : Budget alloué aux technologies de l'information (ou IT pour Information Technology).
- Dans les entreprises, la part du budget IT consacrée aux tests et à l'assurance qualité augmente de 9 points d'une année sur l'autre selon Capgemini.
- De 26% en 2014, elle passe à 35 % en 2015.

Coût des tests

- Budget IT : Budget alloué aux technologies de l'information (ou IT pour Information Technology).
- Dans les entreprises, la part du budget IT consacrée aux tests et à l'assurance qualité augmente de 9 points d'une année sur l'autre selon Capgemini.
- De 26% en 2014, elle passe à 35 % en 2015.
- Il était prévu qu'en 2018, la part des tests et de la qualité passe à 40 % du total. (Source Silicon)

VVT

- Validation, Vérification et Test logiciel.

VVT

- Validation, Vérification et Test logiciel.
- Démonstration automatique (Hum !) : exhaustive mais considérée comme trop coûteuse.

VVT

- Validation, Vérification et Test logiciel.
- Démonstration automatique (Hum !) : exhaustive mais considérée comme trop coûteuse.
- Model Checking : vérifier si le modèle d'un système satisfait une propriété. Par exemple, on souhaite vérifier qu'un programme ne se bloque pas, qu'une variable n'est jamais nulle, etc. Généralement, la propriété est écrite dans un langage, souvent en *logique temporelle*. La vérification est généralement faite de manière automatique.

VVT

- Validation, Vérification et Test logiciel.
- Démonstration automatique (Hum !) : exhaustive mais considérée comme trop coûteuse.
- Model Checking : vérifier si le modèle d'un système satisfait une propriété. Par exemple, on souhaite vérifier qu'un programme ne se bloque pas, qu'une variable n'est jamais nulle, etc. Généralement, la propriété est écrite dans un langage, souvent en *logique temporelle*. La vérification est généralement faite de manière automatique.
- Test : non exhaustif mais facile à mettre en œuvre (bon rapport qualité/temps).

Cycle en V

Le test commence dès le début !

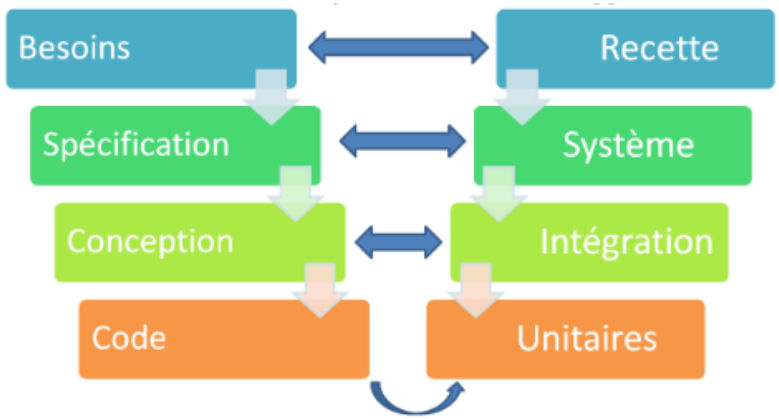


FIGURE – Le cycle en V du Génie Logiciel (d'après Irif)

Classification

selon le niveau au cycle de développement

- Tests de recette : test de réception du logiciel chez le client final

Classification

selon le niveau au cycle de développement

- Tests de recette : test de réception du logiciel chez le client final
- Tests intégration système : test de l'intégration du logiciel avec d'autres logiciels

Classification

selon le niveau au cycle de développement

- Tests de recette : test de réception du logiciel chez le client final
- Tests intégration système : test de l'intégration du logiciel avec d'autres logiciels
- Tests systeme : test d'acception du logiciel avant livraison (nouvelle version par exemple). Généralement effectué par le client dans ses locaux après installation du système ou d'une unité fonctionnelle, avec la participation du fournisseur.

Classification

selon le niveau au cycle de développement

- Tests de recette : test de réception du logiciel chez le client final
- Tests intégration système : test de l'intégration du logiciel avec d'autres logiciels
- Tests systeme : test d'acception du logiciel avant livraison (nouvelle version par exemple). Généralement effectué par le client dans ses locaux après installation du système ou d'une unité fonctionnelle, avec la participation du fournisseur.
- Tests Integration : test de l'intégration des différents composants (avec ou sans hardware)

Classification

selon le niveau au cycle de développement

- Tests de recette : test de réception du logiciel chez le client final
- Tests intégration système : test de l'intégration du logiciel avec d'autres logiciels
- Tests systeme : test d'acception du logiciel avant livraison (nouvelle version par exemple). Généralement effectué par le client dans ses locaux après installation du système ou d'une unité fonctionnelle, avec la participation du fournisseur.
- Tests Integration : test de l'intégration des différents composants (avec ou sans hardware)
- Tests Unitaires : tests élémentaires des composants logiciels (une fonction, un module, ...)

Classification selon la caractéristique

- tests fonctionnels (boîte noire) : destinés à s'assurer que, dans le contexte d'utilisation réelle, le comportement fonctionnel obtenu est bien conforme avec celui attendu.

Classification selon la caractéristique

- tests fonctionnels (boîte noire) : destinés à s'assurer que, dans le contexte d'utilisation réelle, le comportement fonctionnel obtenu est bien conforme avec celui attendu.
- Tests de performance (rapidité, consommation mémoire etc.)

Classification selon la caractéristique

- tests fonctionnels (boîte noire) : destinés à s'assurer que, dans le contexte d'utilisation réelle, le comportement fonctionnel obtenu est bien conforme avec celui attendu.
- Tests de performance (rapidité, consommation mémoire etc.)
- Tests de robustesse (mon serveur va-t-il bien réagir à un afflux de connexion ?)

Classification selon la caractéristique

- tests fonctionnels (boîte noire) : destinés à s'assurer que, dans le contexte d'utilisation réelle, le comportement fonctionnel obtenu est bien conforme avec celui attendu.
- Tests de performance (rapidité, consommation mémoire etc.)
- Tests de robustesse (mon serveur va-t-il bien réagir à un afflux de connexion ?)
- Tests de vulnérabilité : suis-je bien protégé face aux risques d'intrusions ?

Classification selon le niveau d'accessibilité

- tests de type boîte noire (ou fonctionnels) : technique de conception de test qui n'est pas fondée sur l'analyse de la structure interne du composant ou du système mais sur la définition du composant ou du système.

Classification selon le niveau d'accessibilité

- tests de type boîte noire (ou fonctionnels) : technique de conception de test qui n'est pas fondée sur l'analyse de la structure interne du composant ou du système mais sur la définition du composant ou du système.
- tests de type boîte blanche (ou structurel) : technique de conception de test, en général fonctionnel, fondée sur l'analyse de la structure interne du composant ou du système.

Classification selon le niveau d'accessibilité

- tests de type boîte noire (ou fonctionnels) : technique de conception de test qui n'est pas fondée sur l'analyse de la structure interne du composant ou du système mais sur la définition du composant ou du système.
- tests de type boîte blanche (ou structurel) : technique de conception de test, en général fonctionnel, fondée sur l'analyse de la structure interne du composant ou du système.
- Pour comparer les deux.

Divers

Tests de non-régression : à la suite de la modification d'un logiciel (ou d'un de ses constituants), un test de régression a pour but de montrer que les autres parties du logiciel n'ont pas été affectées par cette modification.

En CPGE

- Tests unitaires en général boîte blanche et fonctionnels.

En CPGE

- Tests unitaires en général boîte blanche et fonctionnels.
- Moyen : pour tout exercice, créer un fichier ou une fonction de tests.

En CPGE

- Tests unitaires en général boîte blanche et fonctionnels.
- Moyen : pour tout exercice, créer un fichier ou une fonction de tests.
- Rendu de projets : les tests sont impératifs.

Complémentarité des tests fonctionnels et structurel

En examinant ce qui a été réalisé, on ne prend pas forcément en compte ce qui aurait du être fait

- Les approches structurelles (boîte blanche) détectent plus facilement les erreurs commises

Complémentarité des tests fonctionnels et structurel

En examinant ce qui a été réalisé, on ne prend pas forcément en compte ce qui aurait du être fait

- Les approches structurelles (boîte blanche) détectent plus facilement les erreurs commises
- Les approches fonctionnelles (boîte noire) détectent plus facilement les erreurs d'omission

Partitionnement du domaine d'entrée

- Il s'agit de répartir les données en un nombre fini de classes d'équivalence, sans restreindre le degré d'exigence. Cette méthode est généralement utilisée pour réduire le nombre total de cas de test à un ensemble fini.

Partitionnement du domaine d'entrée

- Il s'agit de répartir les données en un nombre fini de classes d'équivalence, sans restreindre le degré d'exigence. Cette méthode est généralement utilisée pour réduire le nombre total de cas de test à un ensemble fini.
- Exemple : test d'une zone de saisie attendant des nombres de 1 à 1000.

Partitionnement du domaine d'entrée

- Il s'agit de répartir les données en un nombre fini de classes d'équivalence, sans restreindre le degré d'exigence. Cette méthode est généralement utilisée pour réduire le nombre total de cas de test à un ensemble fini.
- Exemple : test d'une zone de saisie attendant des nombres de 1 à 1000.
  - Classe des données valides d'entrée : sélectionner un (ou quelques) nombre.s dans $[\![2, 999]\!]$ (exemple : 50 et 800).

Partitionnement du domaine d'entrée

- Il s'agit de répartir les données en un nombre fini de classes d'équivalence, sans restreindre le degré d'exigence. Cette méthode est généralement utilisée pour réduire le nombre total de cas de test à un ensemble fini.
- Exemple : test d'une zone de saisie attendant des nombres de 1 à 1000.
  - Classe des données valides d'entrée : sélectionner un (ou quelques) nombre.s dans $[\![2, 999]\!]$ (exemple : 50 et 800).
  - Classe des données inférieure à 0 : sélectionner un ou plusieurs nombres négatifs (-1 et -100 par exemple).

Partitionnement du domaine d'entrée

- Il s'agit de répartir les données en un nombre fini de classes d'équivalence, sans restreindre le degré d'exigence. Cette méthode est généralement utilisée pour réduire le nombre total de cas de test à un ensemble fini.
- Exemple : test d'une zone de saisie attendant des nombres de 1 à 1000.
  - Classe des données valides d'entrée : sélectionner un (ou quelques) nombre.s dans $[\![2, 999]\!]$ (exemple : 50 et 800).
  - Classe des données inférieure à 0 : sélectionner un ou plusieurs nombres négatifs (-1 et -100 par exemple).
  - Classe des données supérieures à 1001 : choisir 1050 et 2300 par exemple.

Tests aux limites

Test d'une zone de saisie attendant des nombres de 1 à 1000

En complément du partitionnement du domaine d'entrée, on ajoute les *tests aux limites*

- Ce sont les données encore valides mais au delà desquelles les entrées ne le sont plus.

Tests aux limites

Test d'une zone de saisie attendant des nombres de 1 à 1000

En complément du partitionnement du domaine d'entrée, on ajoute les *tests aux limites*

- Ce sont les données encore valides mais au delà desquelles les entrées ne le sont plus.
- Il faut impérativement tester 1 et 1000.

Tests aux limites

Test d'une zone de saisie attendant des nombres de 1 à 1000

En complément du partitionnement du domaine d'entrée, on ajoute les *tests aux limites*

- Ce sont les données encore valides mais au delà desquelles les entrées ne le sont plus.
- Il faut impérativement tester 1 et 1000.
- On peut y ajouter les valeurs

Tests aux limites

Test d'une zone de saisie attendant des nombres de 1 à 1000

En complément du partitionnement du domaine d'entrée, on ajoute les *tests aux limites*

- Ce sont les données encore valides mais au delà desquelles les entrées ne le sont plus.
- Il faut impérativement tester 1 et 1000.
- On peut y ajouter les valeurs
  - juste en dessous des bornes : 0 et 999,

Tests aux limites

Test d'une zone de saisie attendant des nombres de 1 à 1000

En complément du partitionnement du domaine d'entrée, on ajoute les *tests aux limites*

- Ce sont les données encore valides mais au delà desquelles les entrées ne le sont plus.
- Il faut impérativement tester 1 et 1000.
- On peut y ajouter les valeurs
  - juste en dessous des bornes : 0 et 999,
  - juste au-dessus des bornes : 2 et 1001

Tests aux limites

Test d'une zone de saisie attendant des nombres de 1 à 1000

En complément du partitionnement du domaine d'entrée, on ajoute les *tests aux limites*

- Ce sont les données encore valides mais au delà desquelles les entrées ne le sont plus.
- Il faut impérativement tester 1 et 1000.
- On peut y ajouter les valeurs
  - juste en dessous des bornes : 0 et 999,
  - juste au-dessus des bornes : 2 et 1001
- Si un des tests ne donne pas le résutat attendu, le programme n'est pas correct. En revanche, le programme peut être incorrect même si tous les tests du scénario donnent satisfaction.

Rappel : Tests unitaires ; tests fonctionnels

- En CPGE, on s'intéresse uniquement aux **Tests Unitaires** : tests élémentaires des composants logiciels (une fonction, un module, ...) ;

Rappel : Tests unitaires ; tests fonctionnels

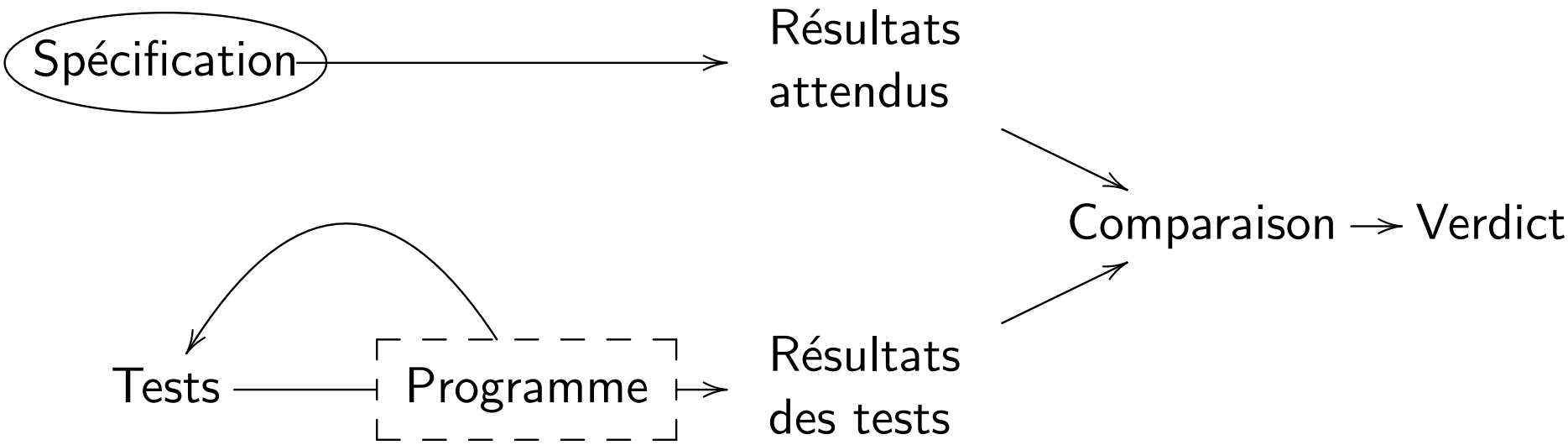
- En CPGE, on s'intéresse uniquement aux **Tests Unitaires** : tests élémentaires des composants logiciels (une fonction, un module, ...) ;
- La plupart des tests que nous avons écrits jusqu'à présent étaient des **Tests fonctionnels** (≪ boîte noire ≫) : destinés à s'assurer que, dans le contexte d'utilisation réelle, le comportement fonctionnel obtenu est bien conforme avec celui attendu.
  On crée des tests sans ce soucier du code écrit.

Tests structurels

dits aussi tests boîte blanche

Dans cette section, on fait des tests en « boîte blanche »

- Sélection de tests à partir de l'analyse du code source du système

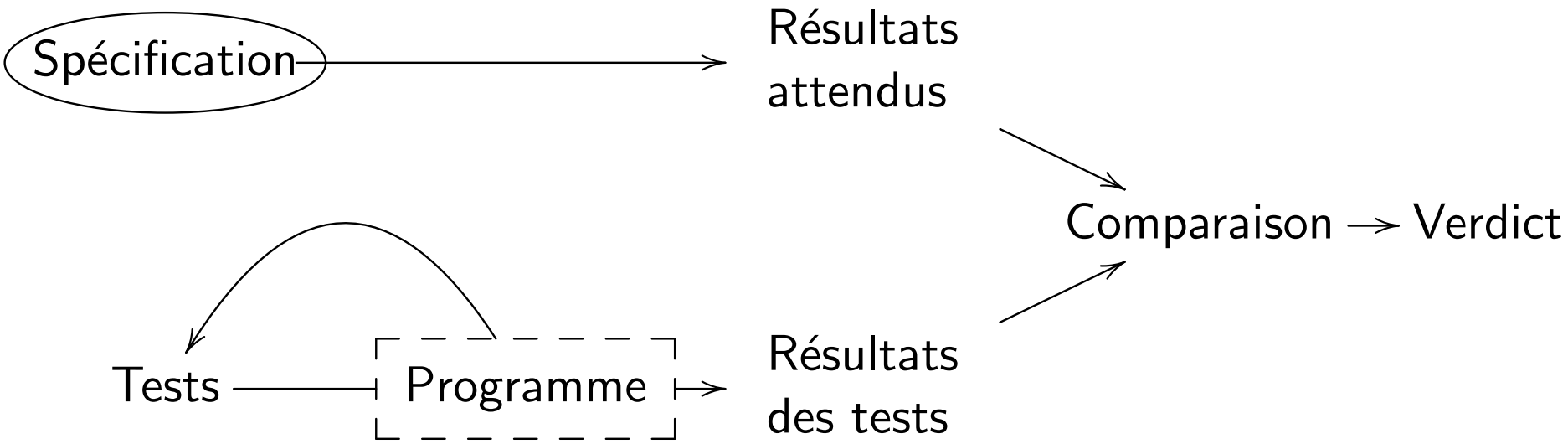


Tests structurels

dits aussi tests boîte blanche

Dans cette section, on fait des tests en « boîte blanche »

- Sélection de tests à partir de l'analyse du code source du système
- Constructions des tests uniquement pour du code déjà écrit.



Test structurel

- Utiliser la structure du code pour dériver des cas de tests.

Test structurel

- Utiliser la structure du code pour dériver des cas de tests.
- Complémentaire des tests fonctionnels car on étudie la réalisation et pas seulement la spécification.

Test structurel

- Utiliser la structure du code pour dériver des cas de tests.
- Complémentaire des tests fonctionnels car on étudie la réalisation et pas seulement la spécification.
- Il y a deux méthodes :

Test structurel

- Utiliser la structure du code pour dériver des cas de tests.
- Complémentaire des tests fonctionnels car on étudie la réalisation et pas seulement la spécification.
- Il y a deux méthodes :
  - À partir du *graphe de flot de contrôle* (GFC) : couverture de toutes les instructions, tous les branchements... (CPGE)

Test structurel

- Utiliser la structure du code pour dériver des cas de tests.
- Complémentaire des tests fonctionnels car on étudie la réalisation et pas seulement la spécification.
- Il y a deux méthodes :
  - À partir du *graphe de flot de contrôle* (GFC) : couverture de toutes les instructions, tous les branchements... (CPGE)
  - À partir du *graphe de flot de données* : couverture de toutes les utilisations d'une variable, de tous les couples définition-utilisation...

Graphe de flot de contrôle (GFC)

- Un GFC est un graphe (souvent simple) orienté connexe avec un sommet initial (ou d'entrée) `E` et un sommet de sortie `S` accessible depuis `E`.

Graphe de flot de contrôle (GFC)

- Un GFC est un graphe (souvent simple) orienté connexe avec un sommet initial (ou d'entrée) `E` et un sommet de sortie `S` accessible depuis `E`.
- Un sommet interne est

Graphe de flot de contrôle (GFC)

- Un GFC est un graphe (souvent simple) orienté connexe avec un sommet initial (ou d'entrée) `E` et un sommet de sortie `S` accessible depuis `E`.
- Un sommet interne est
  - Soit un bloc d'instructions élémentaires (on regroupe les séquences).

Graphe de flot de contrôle (GFC)

- Un GFC est un graphe (souvent simple) orienté connexe avec un sommet initial (ou d'entrée) `E` et un sommet de sortie `S` accessible depuis `E`.
- Un sommet interne est
  - Soit un bloc d'instructions élémentaires (on regroupe les séquences).
  - Soit un sommet de décision étiqueté par un test.

Graphe de flot de contrôle (GFC)

- Un GFC est un graphe (souvent simple) orienté connexe avec un sommet initial (ou d'entrée) `E` et un sommet de sortie `S` accessible depuis `E`.
- Un sommet interne est
  - Soit un bloc d'instructions élémentaires (on regroupe les séquences).
  - Soit un sommet de décision étiqueté par un test.
- Il y a deux sortes d'arcs :

Graphe de flot de contrôle (GFC)

- Un GFC est un graphe (souvent simple) orienté connexe avec un sommet initial (ou d'entrée) `E` et un sommet de sortie `S` accessible depuis `E`.
- Un sommet interne est
  - Soit un bloc d'instructions élémentaires (on regroupe les séquences).
  - Soit un sommet de décision étiqueté par un test.
- Il y a deux sortes d'arcs :
  - Les arcs de sortie de blocs d'instructions : ils représentent le passage d'un bloc d'instructions à une instruction conditionnelle ou à un autre bloc d'instruction. Il ne sont pas étiquetés.

Graphe de flot de contrôle (GFC)

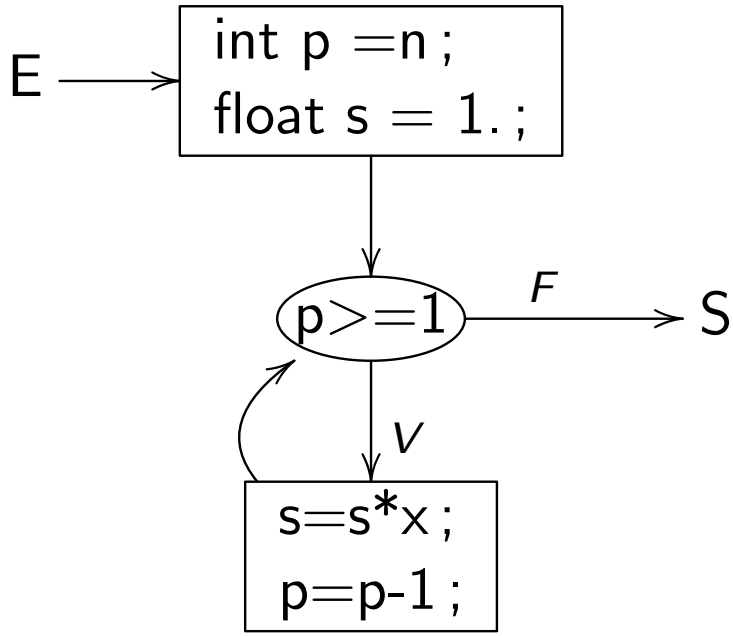
- Un GFC est un graphe (souvent simple) orienté connexe avec un sommet initial (ou d'entrée) `E` et un sommet de sortie `S` accessible depuis `E`.
- Un sommet interne est
  - Soit un bloc d'instructions élémentaires (on regroupe les séquences).
  - Soit un sommet de décision étiqueté par un test.
- Il y a deux sortes d'arcs :
  - Les arcs de sortie de blocs d'instructions : ils représentent le passage d'un bloc d'instructions à une instruction conditionnelle ou à un autre bloc d'instruction. Il ne sont pas étiquetés.
  - les arcs de décision : ce sont les arcs de sortie des sommets de décision. Ils sont étiquetés par les réponses aux tests des expressions conditionnelles.

Graphe de flot de contrôle (GFC)

- Un GFC est un graphe (souvent simple) orienté connexe avec un sommet initial (ou d'entrée) `E` et un sommet de sortie `S` accessible depuis `E`.
- Un sommet interne est
  - Soit un bloc d'instructions élémentaires (on regroupe les séquences).
  - Soit un sommet de décision étiqueté par un test.
- Il y a deux sortes d'arcs :
  - Les arcs de sortie de blocs d'instructions : ils représentent le passage d'un bloc d'instructions à une instruction conditionnelle ou à un autre bloc d'instruction. Il ne sont pas étiquetés.
  - les arcs de décision : ce sont les arcs de sortie des sommets de décision. Ils sont étiquetés par les réponses aux tests des expressions conditionnelles.
- Il y a un seul arc de sortie d'un sommet bloc d'instructions. Et, le plus souvent, deux arcs de sortie (dits *arcs de décision*) des sommets de décision correspondant aux valeurs positives ou négatives des conditions.

Exemple : calcul de X^n

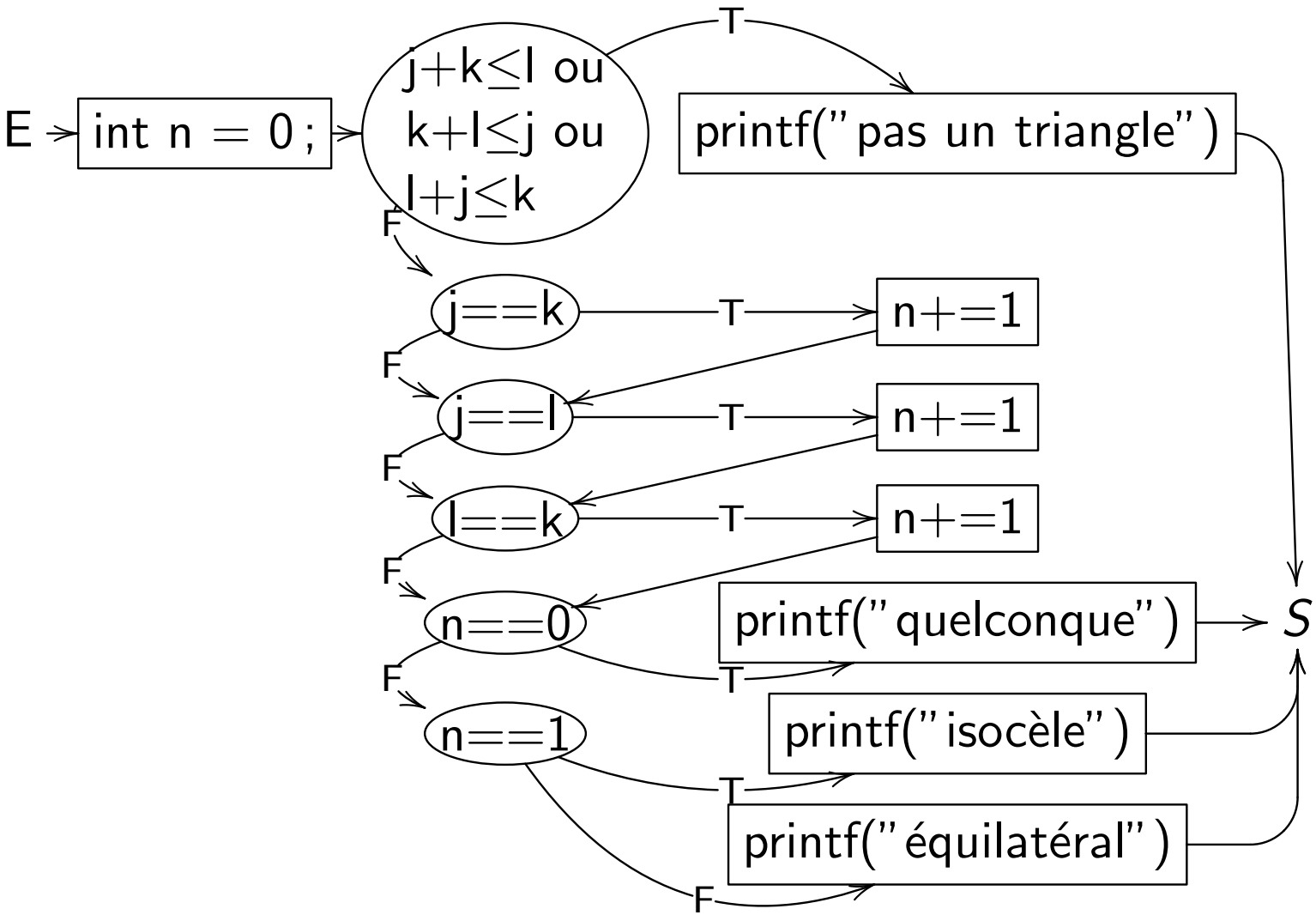
```
1  float power(float x, int n)
       {
2    int p = n; float s = 1. ;
3    while(p>=1){
4      s=s*x;
5      p=p-1;
6    }
7    return s; }
8
```



Exemple des triangles

```
1  void triangle(int j,int k,int l){
2    // j,k, l : les 3 côtés d'un triangle
3    // indique si le triangle est quelconque,
4    // isocèle ou équilatéral
5    int n = 0;// n pour 'nature du triangle'
6    if (j+k<= l || k+l <=j || l+j<= k)
7       printf("pas un triangle\n");
8    else{
9         if (j==k) n += 1;
10        if (j==l) n += 1;
11        if (l==k) n += 1;
12        if (n == 0)
13          printf("quelconque\n");
14        else if (n == 1)
15          printf("isocèle\n");
16        else
17          printf("équilatéral\n");
18      }
19  }
```

Exemple des triangles



Avec `T` pour **True** et `F` pour **False**.

Chemin faisable

- Un *chemin de contrôle* est une suite $x_0, x_1, \ldots, x_n$ de sommets tels que $x_0 = E$, $x_n = S$ et (x_i, x_{i+1}) est un arc pour tout $i < n$.

Chemin faisable

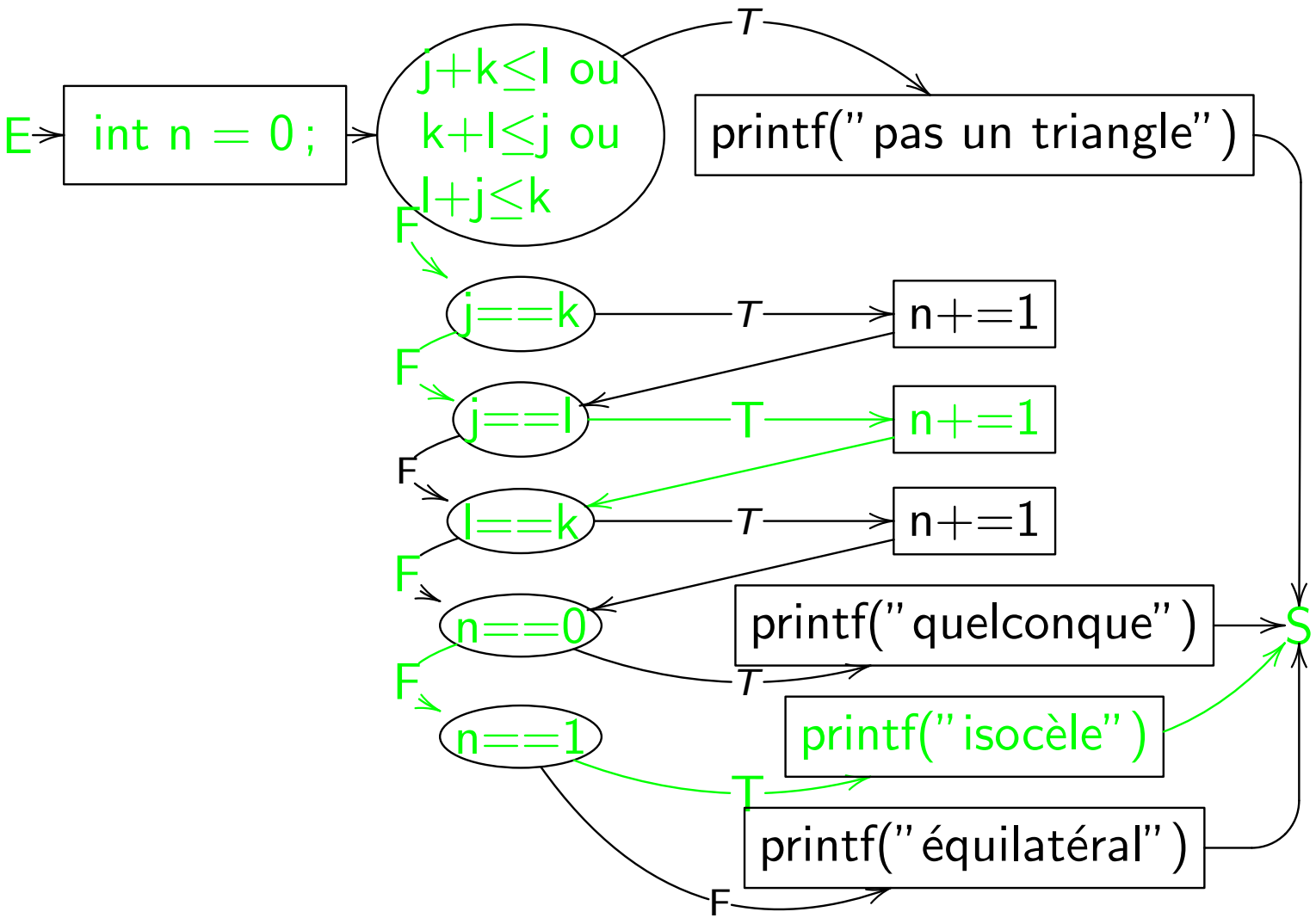
- Un *chemin de contrôle* est une suite $x_0, x_1, \ldots, x_n$ de sommets tels que $x_0 = E$, $x_n = S$ et (x_i, x_{i+1}) est un arc pour tout $i < n$.
- Un chemin est dit *faisable* si il existe un ensemble de conditions sur les variables d'entrées permettant de passer par tous les sommets de ce chemin.

Chemin faisable

- Un *chemin de contrôle* est une suite $x_0, x_1, \ldots, x_n$ de sommets tels que $x_0 = E$, $x_n = S$ et (x_i, x_{i+1}) est un arc pour tout $i < n$.
- Un chemin est dit *faisable* si il existe un ensemble de conditions sur les variables d'entrées permettant de passer par tous les sommets de ce chemin.
- Si un chemin n'est pas faisable, il est dit *infaisable*. La recherche de chemins infaisable est *indécidable* : on ne sait pas, prenant en entrée un GFC quelconque, dire dans tous les cas s'il existe un chemin infaisable.

Exemple des triangles

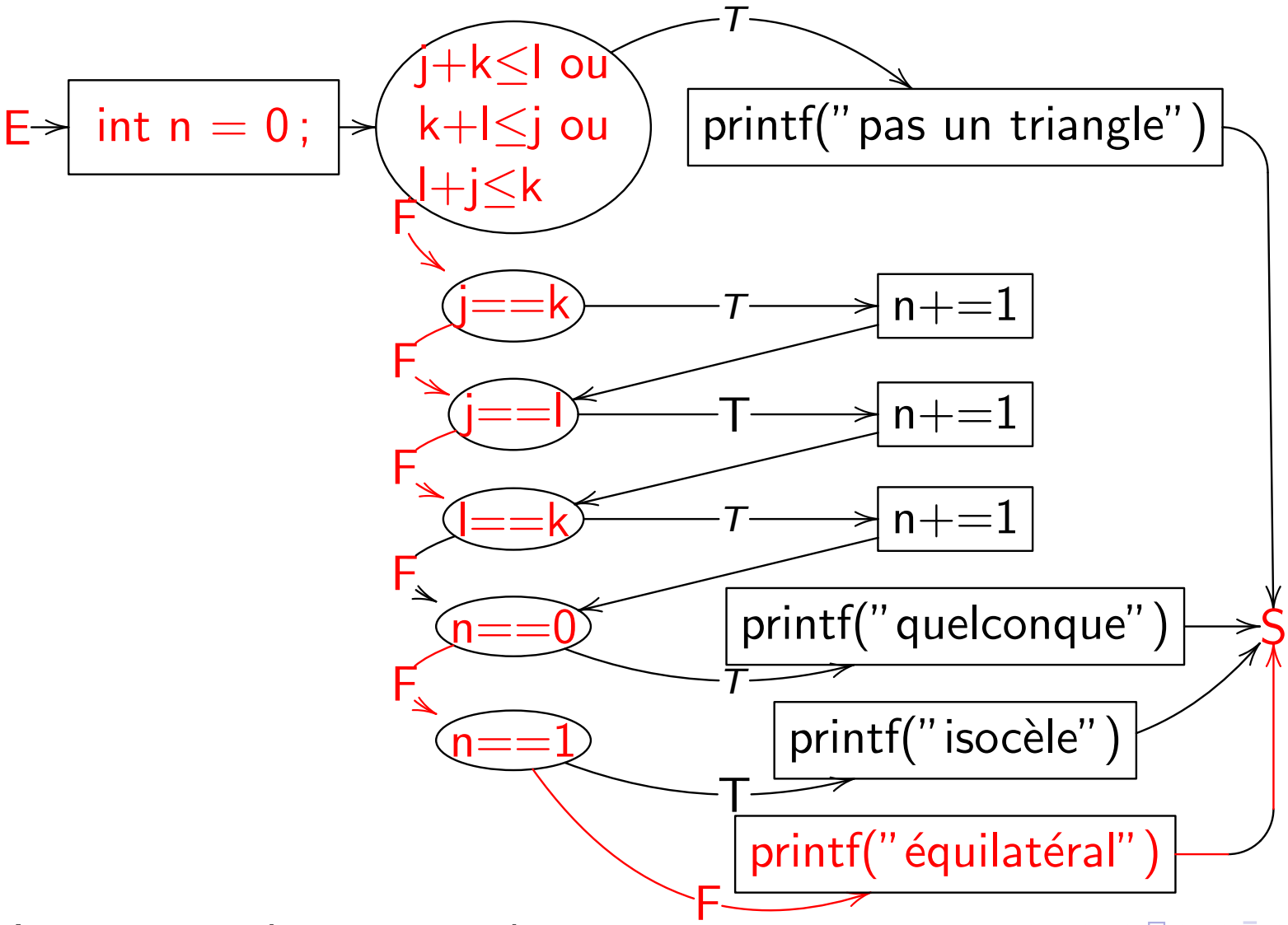
Chemin faisable



Pour $j = 5; k = 3; l = 5$.

Exemple des triangles

Chemin infaisable



Aucune entrée pour ce chemin.

Condition d'un chemin

- La *condition d'un chemin* est la conjonction de tous les sommets de décision portant sur les entrées que traverse ce chemin.

Condition d'un chemin

- La *condition d'un chemin* est la conjonction de tous les sommets de décision portant sur les entrées que traverse ce chemin.
- Exemple : avec $j = 3, k = 5, l = 3$, la condition du chemin vert parcouru est

$$\neg(j+k \leq l \vee k+l \leq j \vee j+l \leq k) \wedge j \neq k \wedge j = l \wedge l \neq k \wedge n \neq 0 \wedge n = 1.$$

Condition d'un chemin

- La *condition d'un chemin* est la conjonction de tous les sommets de décision portant sur les entrées que traverse ce chemin.
- Exemple : avec $j = 3, k = 5, l = 3$, la condition du chemin vert parcouru est

$$\neg(j+k \leq l \vee k+l \leq j \vee j+l \leq k) \wedge j \neq k \wedge j = l \wedge l \neq k \wedge n \neq 0 \wedge n = 1.$$

- La condition du chemin rouge est

$$\neg(j+k \leq l \vee k+l \leq j \vee j+l \leq k) \wedge j \neq k \wedge j \neq l \wedge l \neq k \wedge n \neq 0 \wedge n \neq 1.$$

Or

$$\neg(j+k \leq l \vee k+l \leq j \vee j+l \leq k) \wedge j \neq k \wedge j \neq l \wedge l \neq k \implies n = 0.$$

Condition d'un chemin

- La *condition d'un chemin* est la conjonction de tous les sommets de décision portant sur les entrées que traverse ce chemin.
- Exemple : avec $j = 3, k = 5, l = 3$, la condition du chemin vert parcouru est

$$\neg(j+k \leq l \vee k+l \leq j \vee j+l \leq k) \wedge j \neq k \wedge j = l \wedge l \neq k \wedge n \neq 0 \wedge n = 1.$$

- La condition du chemin rouge est

$$\neg(j+k \leq l \vee k+l \leq j \vee j+l \leq k) \wedge j \neq k \wedge j \neq l \wedge l \neq k \wedge n \neq 0 \wedge n \neq 1.$$

Or
$\neg(j+k \leq l \vee k+l \leq j \vee j+l \leq k) \wedge j \neq k \wedge j \neq l \wedge l \neq k \implies n = 0.$

- Et donc la condition du chemin rouge est

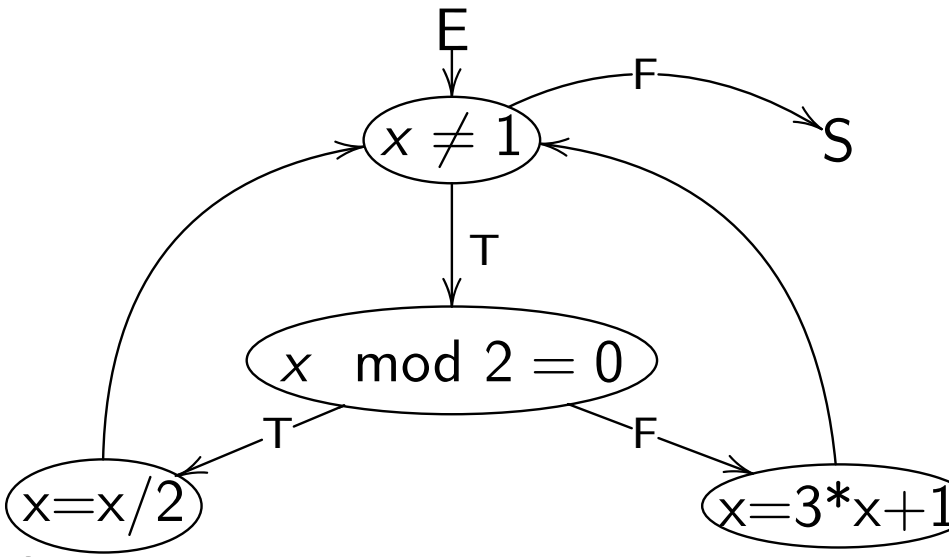
$$\neg(j+k \leq l \vee k+l \leq j \vee j+l \leq k) \wedge j \neq k \wedge j \neq l \wedge l \neq k \wedge 0 \neq 0 \wedge 0 \neq 1.$$

Cette formule est une *antilogie* : le chemin rouge est infaisable.

Suite de Syracuse

```
1  void syracuse(int x){
2    // on ne considère
       que des entiers>0
3    while (x!=1){
4      if (x%2==0) x=x/2;
5      else x = 3*x+1;
6    }
7    printf("fini\n");
8  }
```

La conjecture veut que au bout d'un moment $x = 1$.

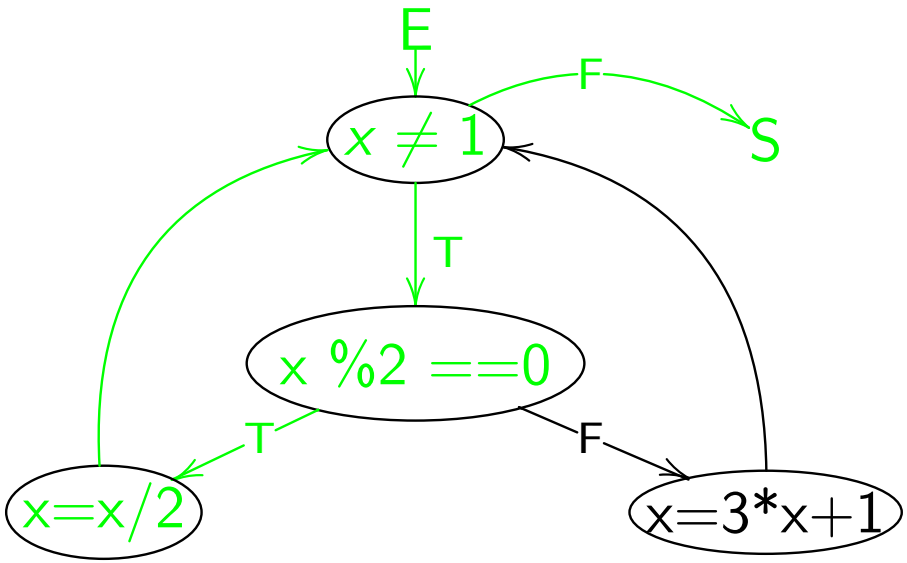


On ne sait pas s'il existe des chemins qui partent de de **E** et n'atteignent pas **S**.

Syracuse : La *discipline de test* (DT) **x=2** *sensibilise* le chemin *C*

Etant donné un chemin faisable, on sait trouver les conditions pour le parcourir.

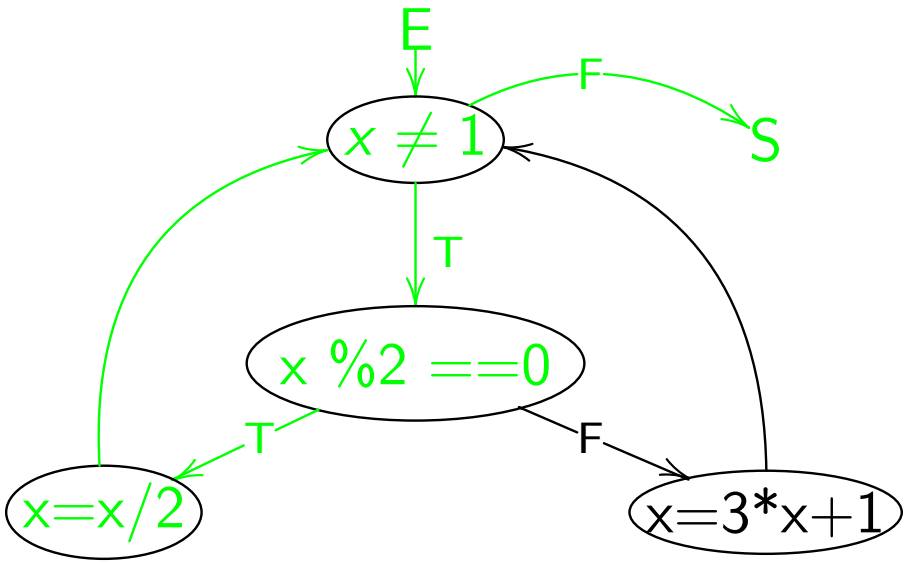
- soit
  le chemin *C* : **E, x !=1, x%2==0,x=x/2,x !=1,S**.



Syracuse : La *discipline de test* (DT) **x=2** *sensibilise* le chemin *C*

Etant donné un chemin faisable, on sait trouver les conditions pour le parcourir.

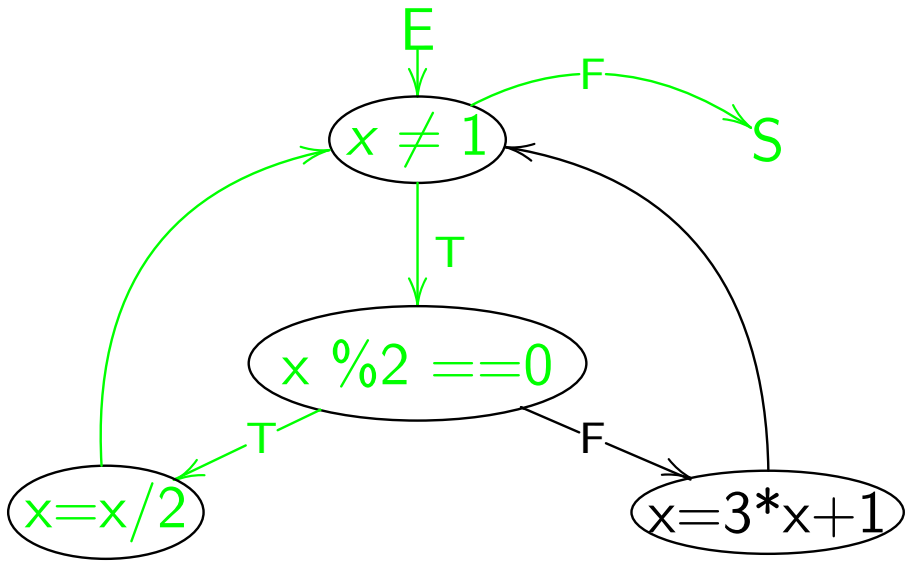
- soit
  le chemin *C* : **E, x !=1, x%2==0,x=x/2,x !=1,S**.
- Quelle condition le vérifie ?



Syracuse : La *discipline de test* (DT) **x=2** *sensibilise* le chemin *C*

Etant donné un chemin faisable, on sait trouver les conditions pour le parcourir.

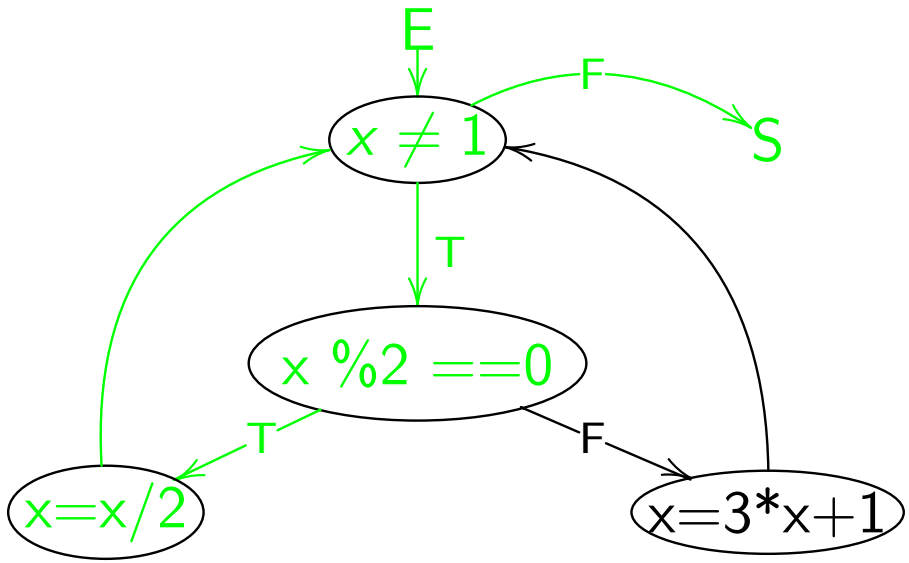
- soit
  le chemin C : **E, x !=1, x%2==0,x=x/2,x !=1,S**.
- Quelle condition le vérifie ?
- $$\begin{array}{c} x \neq 1 \\ x \mod 2 = 0 : x \text{ est pair} \\ x/2 = 1 \end{array}$$



Syracuse : La *discipline de test* (DT) **x=2** *sensibilise* le chemin C

Etant donné un chemin faisable, on sait trouver les conditions pour le parcourir.

- soit
  le chemin C : **E, x !=1, x%2==0,x=x/2,x !=1,S**.
- Quelle condition le vérifie ?
- $$\begin{array}{c} x \neq 1 \\ x \mod 2 = 0 : x \text{ est pair} \\ x/2 = 1 \end{array}$$
- On trouve $x = 2$.



Critère de couverture

- On appelle *critère de couverture* une condition définissant un ensemble de chemins du graphe de flot de contrôle.

Critère de couverture

- On appelle *critère de couverture* une condition définissant un ensemble de chemins du graphe de flot de contrôle.
- Génération de tests grâce à un critère de couverture :

Critère de couverture

- On appelle *critère de couverture* une condition définissant un ensemble de chemins du graphe de flot de contrôle.
- Génération de tests grâce à un critère de couverture :
  - Sélectionner un ensemble minimal de chemins satisfaisant le critère.

Critère de couverture

- On appelle *critère de couverture* une condition définissant un ensemble de chemins du graphe de flot de contrôle.
- Génération de tests grâce à un critère de couverture :
  - Sélectionner un ensemble minimal de chemins satisfaisant le critère.
  - Eliminer les chemins infaisables

Critère de couverture

- On appelle *critère de couverture* une condition définissant un ensemble de chemins du graphe de flot de contrôle.
- Génération de tests grâce à un critère de couverture :
  - Sélectionner un ensemble minimal de chemins satisfaisant le critère.
  - Eliminer les chemins infaisables
  - Pour chaque chemin faisable, on définit un ensemble de conditions associées sur les entrées (c'est à dire un *objectif de test*). Par exemple : entrée $x > 0$

Critère de couverture

- On appelle *critère de couverture* une condition définissant un ensemble de chemins du graphe de flot de contrôle.
- Génération de tests grâce à un critère de couverture :
  - Sélectionner un ensemble minimal de chemins satisfaisant le critère.
  - Eliminer les chemins infaisables
  - Pour chaque chemin faisable, on définit un ensemble de conditions associées sur les entrées (c'est à dire un *objectif de test*). Par exemple : entrée $x > 0$
  - Pour chaque objectif de test (donc pour chaque chemin faisable) choisir un *cas de test* (donc des valeurs d'entrées satisfaisant la condition associée au chemin). Par exemple : entrée $x = 3$

Critère « tous les sommets »

- Critère atteint lorsque tous les sommets du graphe de contrôle sont parcourus.

Critère « tous les sommets »

- Critère atteint lorsque tous les sommets du graphe de contrôle sont parcourus.
- On définit le *taux de couverture des sommets* par

$$\tau_s = \frac{\text{nombre de sommets parcourus}}{\text{nombre de sommets}}$$

Critère « tous les sommets »

- Critère atteint lorsque tous les sommets du graphe de contrôle sont parcourus.
- On définit le *taux de couverture des sommets* par

$$\tau_s = \frac{\text{nombre de sommets parcourus}}{\text{nombre de sommets}}$$

- Exigence minimale pour la certification en aéronautique (norme EUROCAE ED-12B) : $\tau_s = 1$.

Critère « tous les sommets »

- Critère atteint lorsque tous les sommets du graphe de contrôle sont parcourus.
- On définit le *taux de couverture des sommets* par

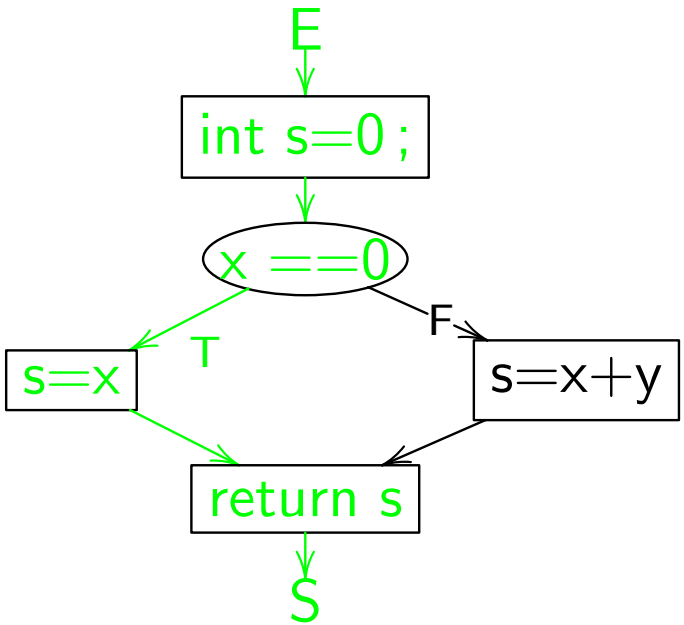
$$\tau_s = \frac{\text{nombre de sommets parcourus}}{\text{nombre de sommets}}$$

- Exigence minimale pour la certification en aéronautique (norme EUROCAE ED-12B) : $\tau_s = 1$.
- Qualification niveau C : Un défaut peut provoquer un problème *majeur* entraînant un dysfonctionnement des équipements vitaux de l'appareil.

Critère « tous les sommets »

```
1  int somme(int x, int y
       ){
2     int s = 0;
3     if (x==0)
4        s=x;
5     else
6        s= x+y;
7     return s;
8  }
```

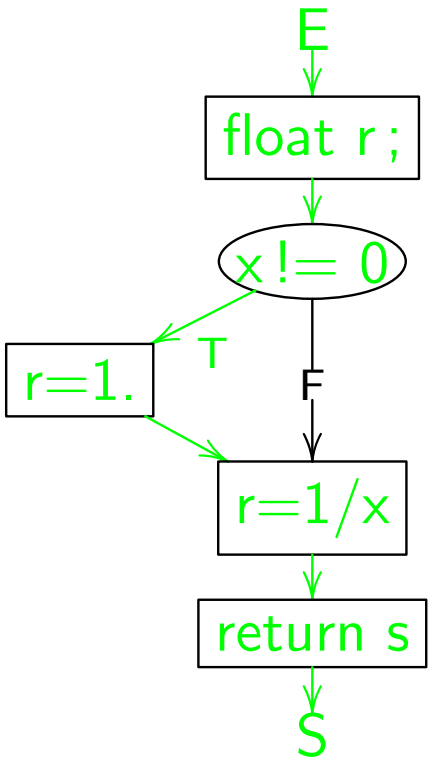
Il y a deux chemins. L'un d'eux permet de détecter le défaut.



Le chemin **E ;int s=0 ;x ==0 ;s=x ;return s ; S** est associé à la condition $x = 0$ satisfaite par les entrées $x = 0, y = 6$. Cela permet de détecter le défaut.

Limite du critère « tous les sommets »

```
1  float div(float x){
2    float r;
3    if (x!=0.)
4      r=1.;
5    r = 1/x;
6    return r;
7  }
8
```



Le chemin **E ;int r ;x !=0 ;r=1 ;return r ; S** (associé à $x = 2$) couvre bien tous les sommets mais la division par zéro n'est pas détectée.

Critère « Tous les arcs »

- Dit aussi critère « *toutes les décisions* ».

Critère « Tous les arcs »

- Dit aussi critère « *toutes les décisions* ».
- Pour chaque décision, un test rend la décision vraie, un autre la rend fausse.

Critère « Tous les arcs »

- Dit aussi critère « *toutes les décisions* ».
- Pour chaque décision, un test rend la décision vraie, un autre la rend fausse.
- *Taux de courverture des arcs* :

$$\tau_a = \frac{\text{nombre d'arcs parcourus}}{\text{nombre d'arcs}}$$

Critère « Tous les arcs »

- Dit aussi critère « *toutes les décisions* ».
- Pour chaque décision, un test rend la décision vraie, un autre la rend fausse.
- *Taux de courverture des arcs* :

$$\tau_a = \frac{\text{nombre d'arcs parcourus}}{\text{nombre d'arcs}}$$

- Norme DO 178B, qualification des systèmes embarqués au niveau B : un défaut peut provoquer un problème *dangereux*

Critère « Tous les arcs »

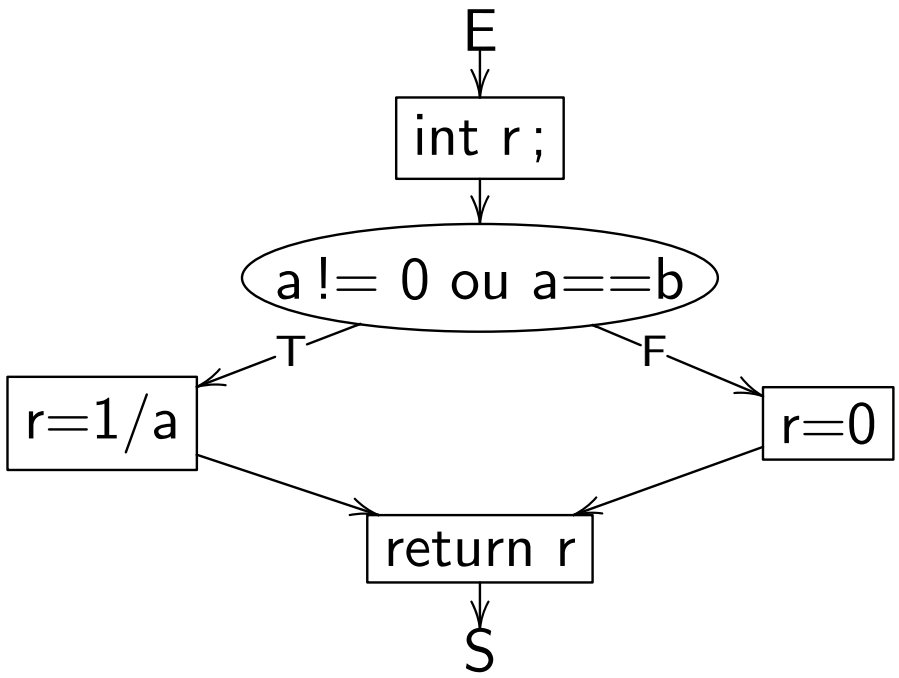
- Dit aussi critère « *toutes les décisions* ».
- Pour chaque décision, un test rend la décision vraie, un autre la rend fausse.
- *Taux de courverture des arcs* :

$$\tau_a = \frac{\text{nombre d'arcs parcourus}}{\text{nombre d'arcs}}$$

- Norme DO 178B, qualification des systèmes embarqués au niveau B : un défaut peut provoquer un problème *dangereux*
- Critère « Tous-les-arcs » entraîne critère « Tous-les-sommets » (puisque le graphe est connexe, chaque sommet est adjacent à un arc au moins).

Limite du critère tous les arcs

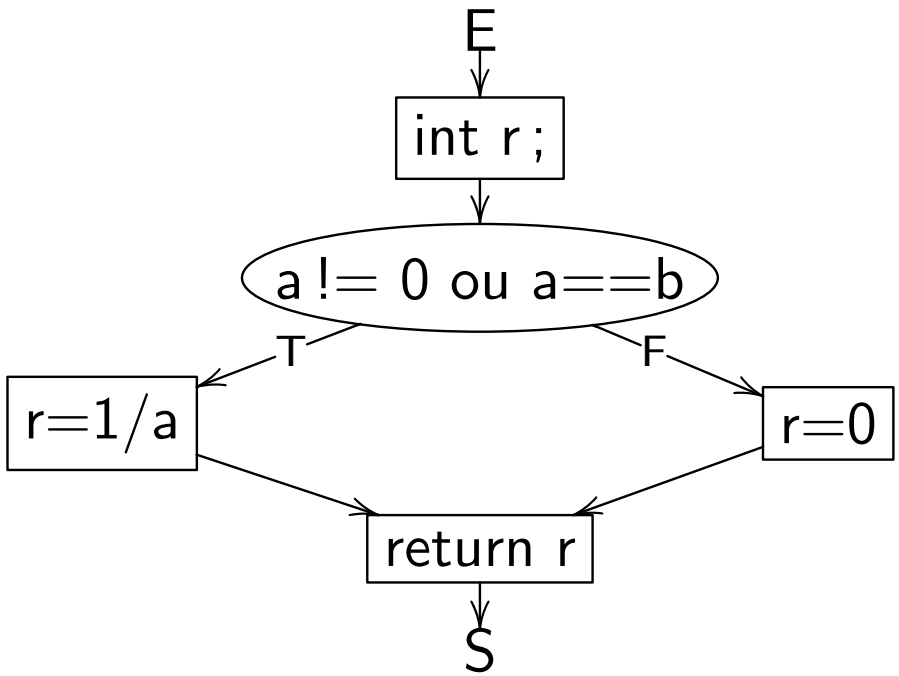
```
1  float F(int a, int b){
2     int r;
3     if (a != 0 || a==b)
          r=1/a;
4     else r=0;
5     return r;
6  }
```



- Critère Toutes-les-Décisions satisfait avec {**a = 0,b = 1**} et {**a = 2,b = 1**}.

Limite du critère tous les arcs

```
1  float F(int a, int b){
2     int r;
3     if (a != 0 || a==b)
          r=1/a;
4     else r=0;
5     return r;
6  }
```



- Critère Toutes-les-Décisions satisfait avec {**a = 0,b = 1**} et {**a = 2,b = 1**}.
- Pourtant, division par zéro non détectée avec {**a = 0,b = 0**}.

Décomposer les conditionnelles

Toutes les conditions multiples

- On a vu que le critère ≪ tous les arcs ≫ pouvait passer à côté d'erreurs.

Décomposer les conditionnelles

Toutes les conditions multiples

- On a vu que le critère ≪ tous les arcs ≫ pouvait passer à côté d'erreurs.
- C'est parce que ce critère est satisfait pour un sommet de décision avec seulement deux jeux d'entrées : un jeu d'entrées rendant la décision vraie et un autre la rendant fausse.

Décomposer les conditionnelles

Toutes les conditions multiples

- On a vu que le critère ≪ tous les arcs ≫ pouvait passer à côté d'erreurs.
- C'est parce que ce critère est satisfait pour un sommet de décision avec seulement deux jeux d'entrées : un jeu d'entrées rendant la décision vraie et un autre la rendant fausse.
- Plutôt que la considérer la décision comme un seul sommet du graphe de contrôle, on introduit les sous-decisions comme nouveaux sommets.

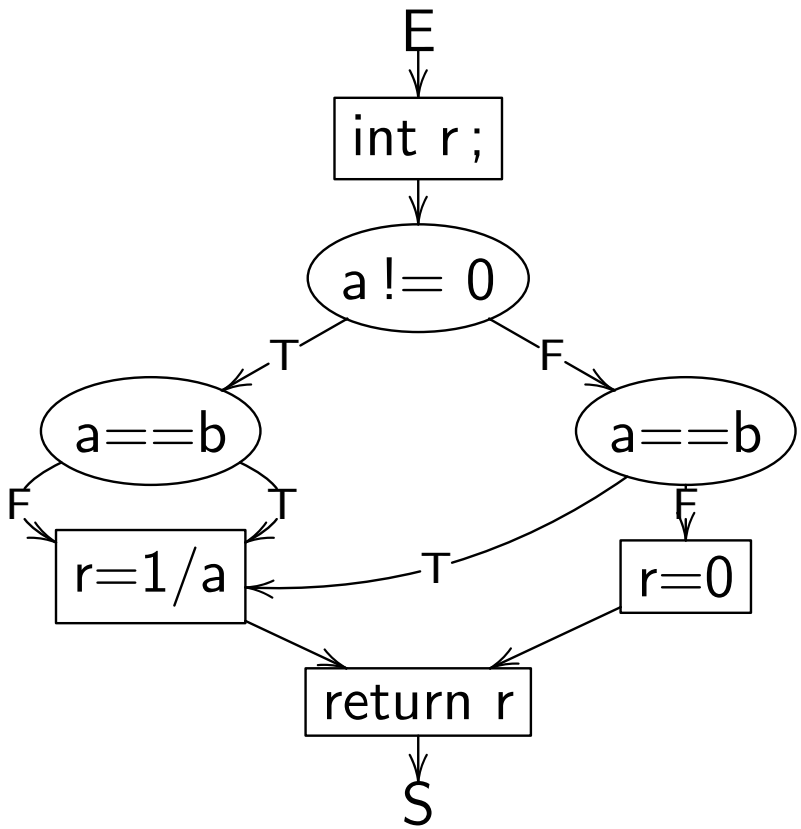
Décomposer les conditionnelles

Toutes les conditions multiples

- On a vu que le critère « tous les arcs » pouvait passer à côté d'erreurs.
- C'est parce que ce critère est satisfait pour un sommet de décision avec seulement deux jeux d'entrées : un jeu d'entrées rendant la décision vraie et un autre la rendant fausse.
- Plutôt que la considérer la décision comme un seul sommet du graphe de contrôle, on introduit les sous-decisions comme nouveaux sommets.
- Ce critère est appelé « toutes les conditions multiples »

Critère toutes les conditions multiples : exemple

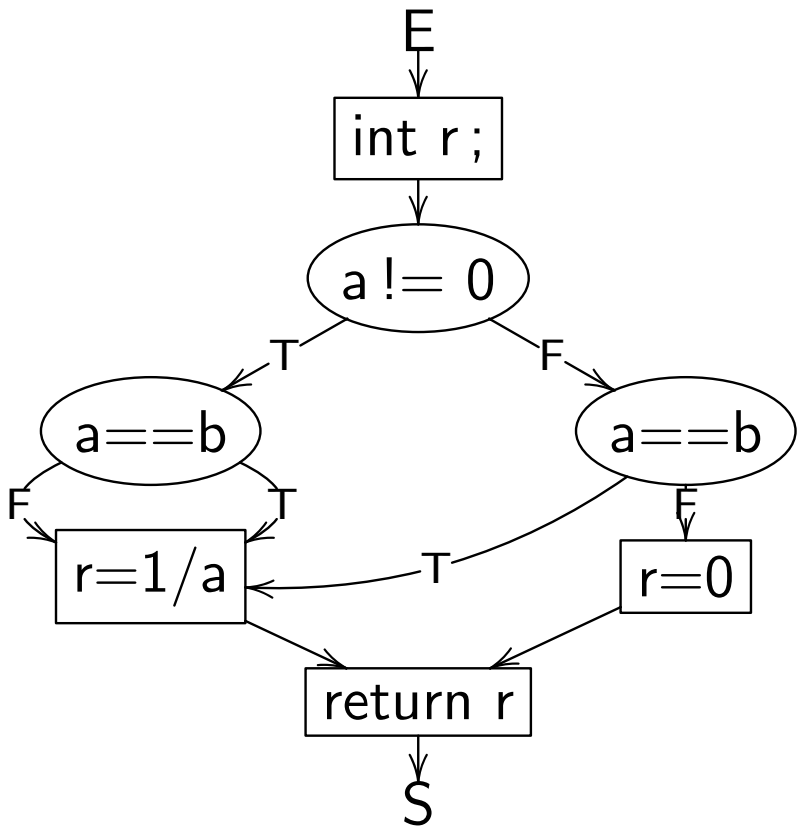
- Le critère Toutes-les-Décisions n'est plus satisfait avec seulement $\{\mathbf{a = 0, b = 1}\}$ et $\{\mathbf{a = 2, b = 1}\}$.



Le graphe simple devient un multigraphe (deux arcs sortant du `a==b` de gauche vers `1/a`).

Critère toutes les conditions multiples : exemple

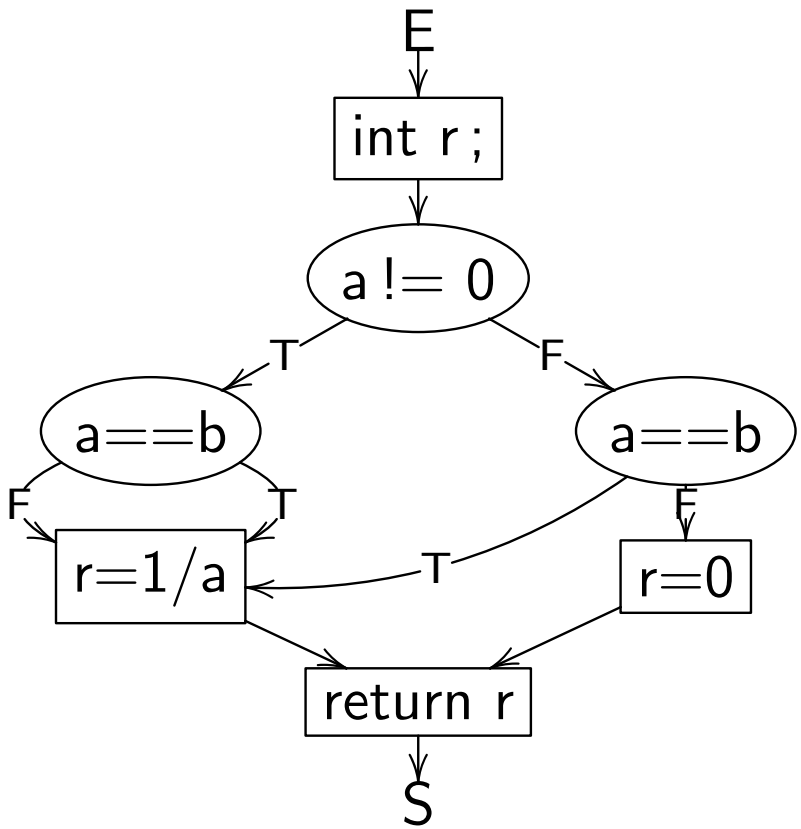
- Le critère Toutes-les-Décisions n'est plus satisfait avec seulement $\{\mathbf{a = 0, b = 1}\}$ et $\{\mathbf{a = 2, b = 1}\}$.
- Le critère Toutes-les-décisions est satisfait avec $\{\mathbf{a = 0, b = 1}\}$; $\{\mathbf{a = 2, b = 2}\}$; $\{\mathbf{a = 2, b = 1}\}$ et enfin $\{\mathbf{a = 0, b = 0}\}$.



Le graphe simple devient un multigraphe (deux arcs sortant du `a==b` de gauche vers `1/a`).

Critère toutes les conditions multiples : exemple

- Le critère Toutes-les-Décisions n'est plus satisfait avec seulement $\{\mathbf{a = 0, b = 1}\}$ et $\{\mathbf{a = 2, b = 1}\}$.
- Le critère Toutes-les-décisions est satisfait avec $\{\mathbf{a = 0, b = 1}\}$; $\{\mathbf{a = 2, b = 2}\}$; $\{\mathbf{a = 2, b = 1}\}$ et enfin $\{\mathbf{a = 0, b = 0}\}$.
- Division par zéro détectée.



Le graphe simple devient un multigraphe (deux arcs sortant du `a==b` de gauche vers `1/a`).

Critère « Toutes les décisions multiples »

Explosion combinatoire

Exemple `if (A && (B || C))`

Toutes-les-conditions

A	B	C	décision
0	1	1	0
1	1	1	1

Toutes-les-conditions-multiples

A	B	C	décision
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

$2^{\text{nb de variables}}$.

Comment limiter la combinatoire ?

Critère Toutes-les-Conditions/Décisions-modifié

MC/DC

- Objectif : améliorer les critères de couverture basés sur les décisions tout en contrôlant la combinatoire

Critère Toutes-les-Conditions/Décisions-modifié

MC/DC

- Objectif : améliorer les critères de couverture basés sur les décisions tout en contrôlant la combinatoire
- Critère MC/DC : *Modified Condition/Decision Coverage*

Critère Toutes-les-Conditions/Décisions-modifié

MC/DC

- Objectif : améliorer les critères de couverture basés sur les décisions tout en contrôlant la combinatoire
- Critère MC/DC : *Modified Condition/Decision Coverage*
- On ne s'intéresse à un jeu de test faisant varier une condition que s'il influe sur la décision.

Critère MC/DC

Exemple `if (A && (B || C))`

- Principe : pour chaque test atomique, trouver deux cas de tests qui changent la décision lorsque les autres conditions sont fixées.

Critère MC/DC

Exemple `if (A && (B || C))`

- Principe : pour chaque test atomique, trouver deux cas de tests qui changent la décision lorsque les autres conditions sont fixées.
- Pour A :

Critère MC/DC

Exemple `if (A && (B || C))`

- Principe : pour chaque test atomique, trouver deux cas de tests qui changent la décision lorsque les autres conditions sont fixées.
- Pour A :
  - $A = 0, B = 1, C = 1$. Décision : 0

Critère MC/DC

Exemple `if (A && (B || C))`

- Principe : pour chaque test atomique, trouver deux cas de tests qui changent la décision lorsque les autres conditions sont fixées.
- Pour A :
  - $A = 0, B = 1, C = 1$. Décision : 0
  - $A = 1, B = 1, C = 1$. Décision : 1

Critère MC/DC

Exemple `if (A && (B || C))`

- Principe : pour chaque test atomique, trouver deux cas de tests qui changent la décision lorsque les autres conditions sont fixées.
- Pour A :
  - $A = 0, B = 1, C = 1$. Décision : 0
  - $A = 1, B = 1, C = 1$. Décision : 1
- Pour B :

Critère MC/DC

Exemple `if (A && (B || C))`

- Principe : pour chaque test atomique, trouver deux cas de tests qui changent la décision lorsque les autres conditions sont fixées.
- Pour A :
  - $A = 0, B = 1, C = 1$. Décision : 0
  - $A = 1, B = 1, C = 1$. Décision : 1
- Pour B :
  - $A = 1, B = 0, C = 0$. Décision : 0

Critère MC/DC

Exemple `if (A && (B || C))`

- Principe : pour chaque test atomique, trouver deux cas de tests qui changent la décision lorsque les autres conditions sont fixées.
- Pour A :
  - $A = 0, B = 1, C = 1$. Décision : 0
  - $A = 1, B = 1, C = 1$. Décision : 1
- Pour B :
  - $A = 1, B = 0, C = 0$. Décision : 0
  - $A = 1, B = 1, C = 0$. Décision : 1

Critère MC/DC

Exemple `if (A && (B || C))`

- Principe : pour chaque test atomique, trouver deux cas de tests qui changent la décision lorsque les autres conditions sont fixées.
- Pour A :
  - $A = 0, B = 1, C = 1$. Décision : 0
  - $A = 1, B = 1, C = 1$. Décision : 1
- Pour B :
  - $A = 1, B = 0, C = 0$. Décision : 0
  - $A = 1, B = 1, C = 0$. Décision : 1
- Pour C :

Critère MC/DC

Exemple `if (A && (B || C))`

- Principe : pour chaque test atomique, trouver deux cas de tests qui changent la décision lorsque les autres conditions sont fixées.
- Pour A :
  - $A = 0, B = 1, C = 1$. Décision : 0
  - $A = 1, B = 1, C = 1$. Décision : 1
- Pour B :
  - $A = 1, B = 0, C = 0$. Décision : 0
  - $A = 1, B = 1, C = 0$. Décision : 1
- Pour C :
  - $A = 1, B = 0, C = 1$. Décision : 1

Critère MC/DC

Exemple `if (A && (B || C))`

- Principe : pour chaque test atomique, trouver deux cas de tests qui changent la décision lorsque les autres conditions sont fixées.
- Pour A :
  - $A = 0, B = 1, C = 1$. Décision : 0
  - $A = 1, B = 1, C = 1$. Décision : 1
- Pour B :
  - $A = 1, B = 0, C = 0$. Décision : 0
  - $A = 1, B = 1, C = 0$. Décision : 1
- Pour C :
  - $A = 1, B = 0, C = 1$. Décision : 1
  - $A = 1, B = 0, C = 0$. Décision : 0. Déjà couvert.

Critère MC/DC

Exemple `if (A && (B || C))`

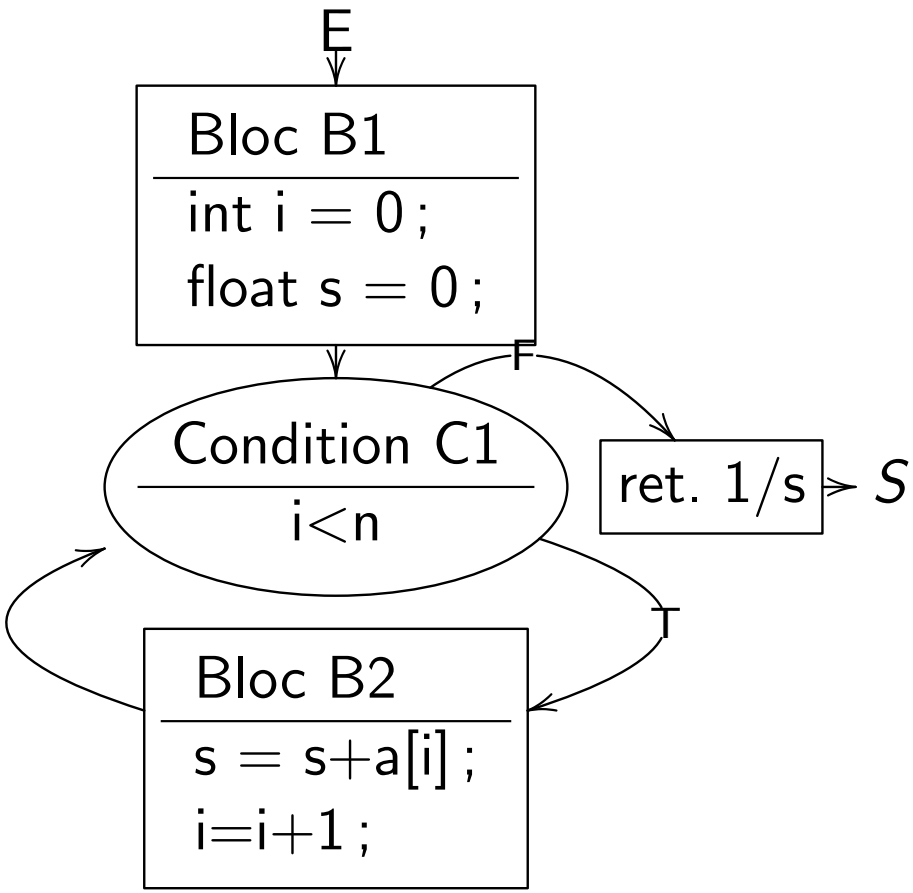
- Principe : pour chaque test atomique, trouver deux cas de tests qui changent la décision lorsque les autres conditions sont fixées.
- Pour A :
  - $A = 0, B = 1, C = 1$. Décision : 0
  - $A = 1, B = 1, C = 1$. Décision : 1
- Pour B :
  - $A = 1, B = 0, C = 0$. Décision : 0
  - $A = 1, B = 1, C = 0$. Décision : 1
- Pour C :
  - $A = 1, B = 0, C = 1$. Décision : 1
  - $A = 1, B = 0, C = 0$. Décision : 0. Déjà couvert.
- Si n conditions : critère MC/DC entraîne au plus $2 \times n$ tests contre 2^n pour le critère Toutes-les-décisions-multiple.

Limite du critère Toutes-les-décisions-multiples

```
1  float invsum(int n,
       float a[]){
2    int i = 0;
3    float s = 0;
4    while(i<n){
5      s=s+a[i];
6      i=i+1;
7    }
8    return 1/s;
9  }
```

Toutes-les-décisions = toutes-les-décisions-multiples

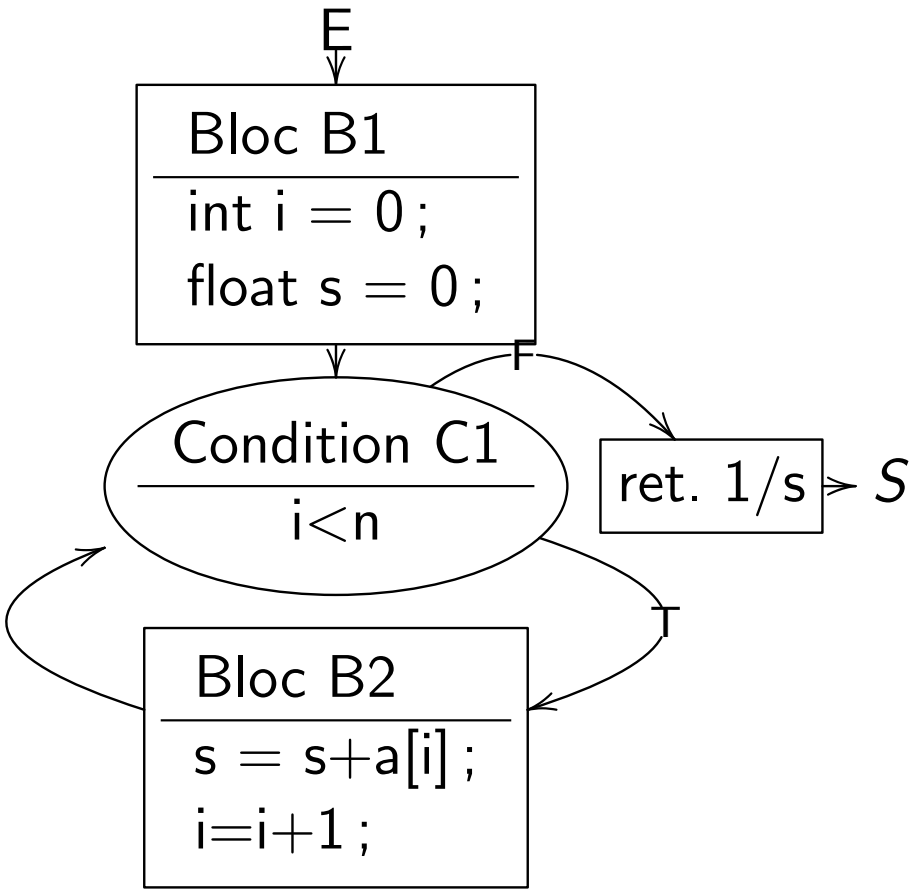
Puisque test atomique



Limite du critère Toutes-les-décisions-multiples

```
1  float invsum(int n,
       float a[]){
2    int i = 0;
3    float s = 0;
4    while(i<n){
5      s=s+a[i];
6      i=i+1;
7    }
8    return 1/s;
9  }
```

Toutes-les-décisions = toutes-les-décisions-multiples
Puisque test atomique



E,B1,C1,B2,C1,ret. 1/s, S
couvre toutes les décisions.
Satisfait par $n = 1$,
$a = \{2., 3., 5.\}$. Défaut non détecté si le tableau est vide

Critère « tous les chemins »

- Critère Tous-les-chemins : parcourir tous les arcs dans chaque configuration possible (et non pas au moins une fois comme dans le critère toutes-les-décisions)

Critère « tous les chemins »

- Critère Tous-les-chemins : parcourir tous les arcs dans chaque configuration possible (et non pas au moins une fois comme dans le critère toutes-les-décisions)
- Impraticable dès qu'il y a des boucles car existence de chemins infinis.

Critère « tous les chemins »

- Critère Tous-les-chemins : parcourir tous les arcs dans chaque configuration possible (et non pas au moins une fois comme dans le critère toutes-les-décisions)
- Impraticable dès qu'il y a des boucles car existence de chemins infinis.
- Affaiblissement possible : « tous les chemins qui passent au plus k fois dans les boucles »

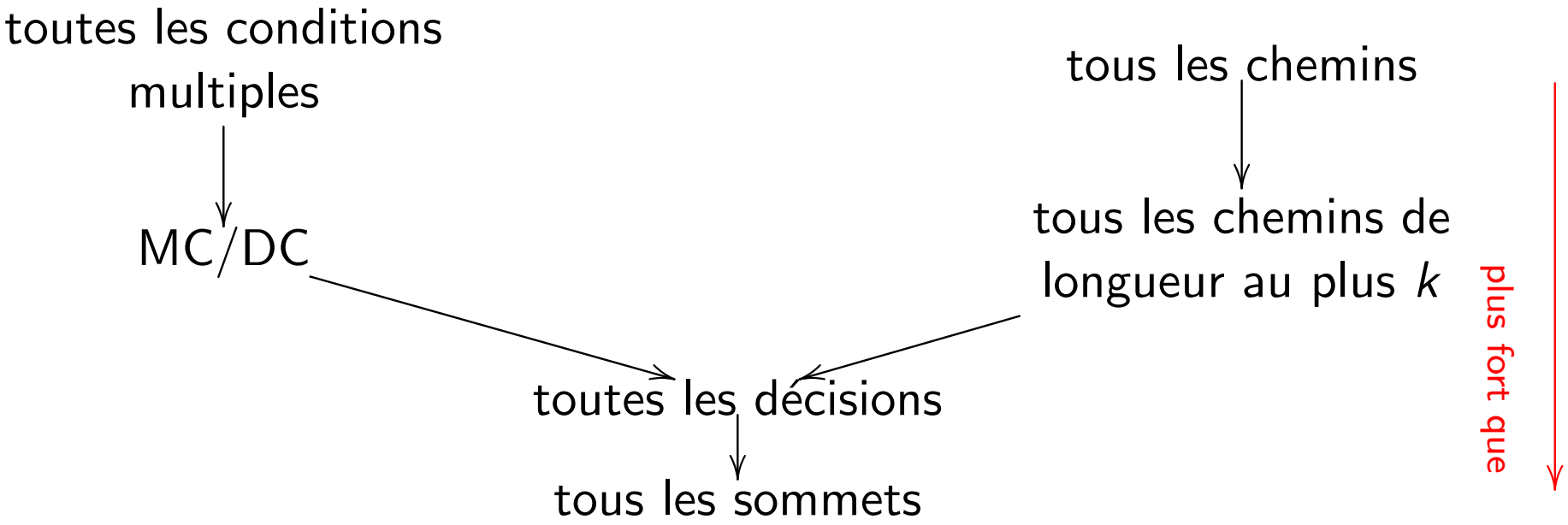
Critère « tous les chemins »

- Critère Tous-les-chemins : parcourir tous les arcs dans chaque configuration possible (et non pas au moins une fois comme dans le critère toutes-les-décisions)
- Impraticable dès qu'il y a des boucles car existence de chemins infinis.
- Affaiblissement possible : « tous les chemins qui passent au plus k fois dans les boucles »
- En pratique on se limite souvent aux cas $k = 1$ ou $k = 2$. Cela permet au moins de détecter les erreurs produites quand on ne rentre pas dans la/les boucle.s.

Critère « tous les chemins »

- Critère Tous-les-chemins : parcourir tous les arcs dans chaque configuration possible (et non pas au moins une fois comme dans le critère toutes-les-décisions)
- Impraticable dès qu'il y a des boucles car existence de chemins infinis.
- Affaiblissement possible : « tous les chemins qui passent au plus k fois dans les boucles »
- En pratique on se limite souvent aux cas $k = 1$ ou $k = 2$. Cela permet au moins de détecter les erreurs produites quand on ne rentre pas dans la/les boucle.s.
- Dans l'exemple précédent, $n = 0, a = \{1.\}$ satisfait le chemin **E,B1,C1,ret . 1/s,S** et détecte la division par zéro.

Hiérarchie des critères



Convention de notation algébrique

- ε pour $\ll$ chemin vide $\gg$

Convention de notation algébrique

- ε pour $\ll$ chemin vide $\gg$
- $A \cdot B$ ou AB pour $\ll$ A suivi de B $\gg$

Convention de notation algébrique

- ε pour « chemin vide »
- $A \cdot B$ ou AB pour « A suivi de B »
- $(CB)^3$ pour « $CBCBCB$ »

Convention de notation algébrique

- ε pour « chemin vide »
- $A \cdot B$ ou AB pour « A suivi de B »
- $(CB)^3$ pour « $CBCBCB$ »
- $A + B$ pour « A ou B »

Convention de notation algébrique

- ε pour « chemin vide »
- $A \cdot B$ ou AB pour « A suivi de B »
- $(CB)^3$ pour « $CBCBCB$ »
- $A + B$ pour « A ou B »
- A^+ pour « $A + A^2 + A^3 + \cdots + A^n + \ldots$ »

Convention de notation algébrique

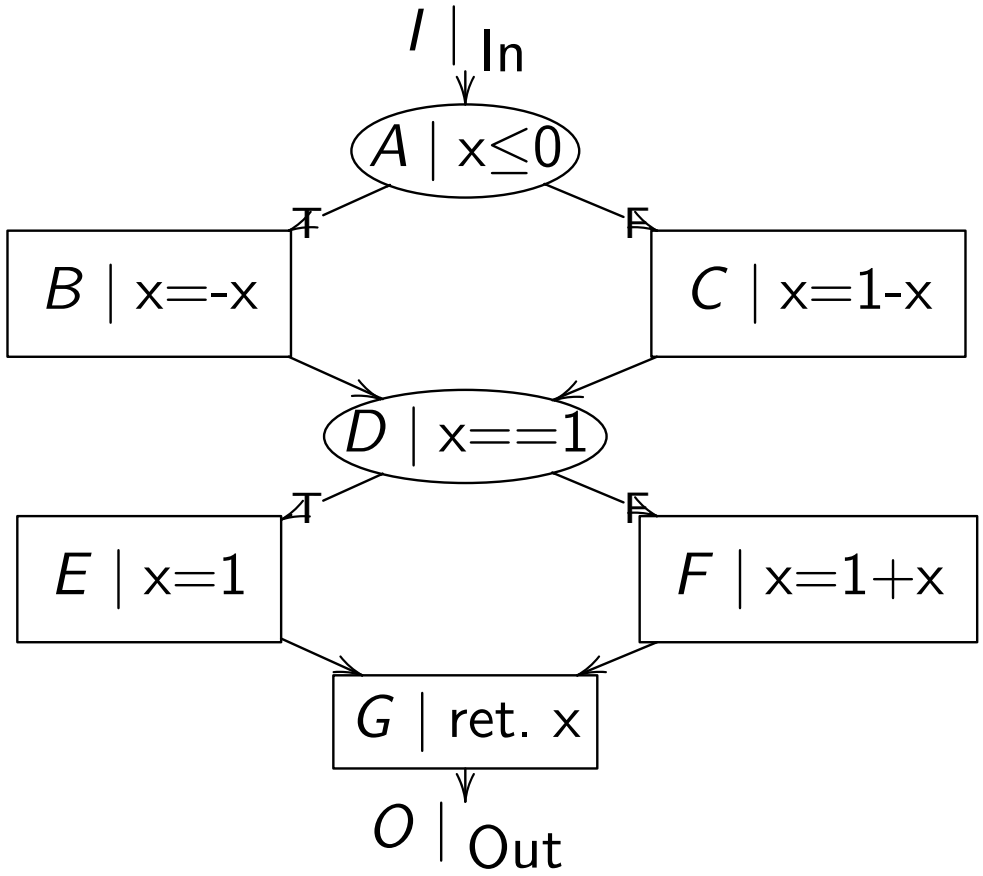
- ε pour « chemin vide »
- $A \cdot B$ ou AB pour « A suivi de B »
- $(CB)^3$ pour « $CBCBCB$ »
- $A + B$ pour « A ou B »
- A^+ pour « $A + A^2 + A^3 + \cdots + A^n + \ldots$ »
- A^* pour « $\varepsilon + A + A^2 + A^3 + \cdots + A^n + \ldots$ »

Convention de notation algébrique

- ε pour « chemin vide »
- $A \cdot B$ ou AB pour « A suivi de B »
- $(CB)^3$ pour « $CBCBCB$ »
- $A + B$ pour « A ou B »
- A^+ pour « $A + A^2 + A^3 + \cdots + A^n + \ldots$ »
- A^* pour « $\varepsilon + A + A^2 + A^3 + \cdots + A^n + \ldots$ »
- $A^{\underline{4}}$ pour $\varepsilon + A + A^2 + A^3 + A^4$

Expression des chemins sous forme algébrique

```
1  void F(int x){
2    if (x<=0)
3      x=-x;
4    else
5      x=1-x;
6    if (x==1)
7      x=1+x;
8    else
9      x=1;
10   return x;
11 }
```

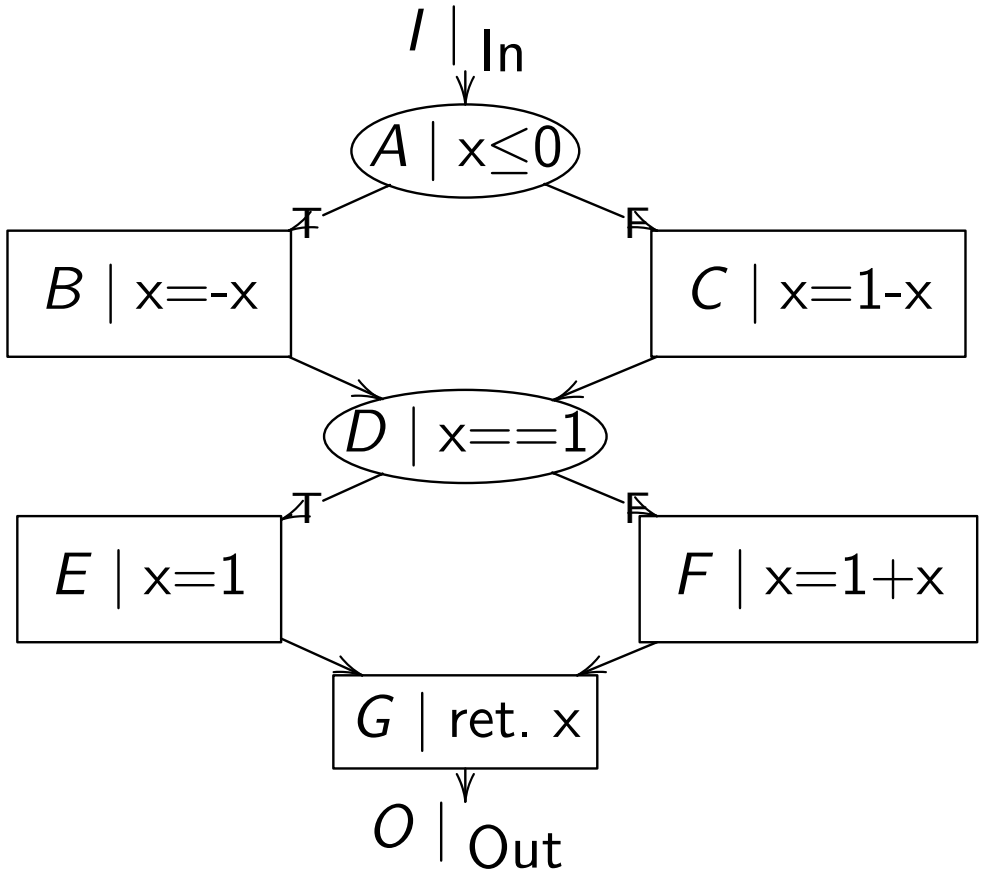


- Les chemins de contrôle possibles sont décrits par

$$IA(B + C)D(E + F)GO$$

Expression des chemins sous forme algébrique

```
1  void F(int x){
2    if (x<=0)
3      x=-x;
4    else
5      x=1-x;
6    if (x==1)
7      x=1+x;
8    else
9      x=1;
10   return x;
11 }
```



- Les chemins de contrôle possibles sont décrits par

$$IA(B + C)D(E + F)GO$$

- Le nombre de chemins de contrôle possible est $1 \cdot 1 \cdot (1 + 1) \cdot 1 \cdot (1 + 1) \cdot 1 = 4$