**Nom et prénom :** Mamadou Amate FALL

**Matière** : Sécurité informatique et méthodes formelles

**Session** : Automne 2023

**Professeur** : Kamel Adi

Introduction à Z3

**Introduction**

Z3 est un solveur de contraintes open source développé par Microsoft Research. Il est principalement utilisé pour résoudre des problèmes complexes impliquant des contraintes logiques, arithmétiques et algébriques. Z3 prend en charge plusieurs langages de spécification, tels que SMT-LIB (Syntaxe pour les solveurs SMT) et propose des liaisons pour divers langages de programmation, notamment C, C++, Python et .NET.

Les solveurs de contraintes, comme Z3, sont utilisés dans divers domaines tels que la vérification de logiciels, la synthèse de programmes, la sécurité informatique et d'autres domaines de l'informatique formelle. Ces outils aident Z à automatiser la résolution de problèmes complexes en manipulant des expressions symboliques sous contraintes.

Elle permet de connaitre la satisfiabilité de certaines conditions et de pouvoir générer des cas de test comme :

* Des détections de code mort
* Des générations de cas de test
* Vérification d’invariant

En résumé, Z3 est un outil puissant utilisé dans le domaine de la vérification formelle et de la résolution de problèmes impliquant des contraintes logiques.

Dans les lignes suivantes nous allons tous d’abord présenter un algorithme et ensuite montrer les différents éléments de Z3 qui vont nous permettre de faire l’analyse avec python et enfin effectuer une série de test sur l’algorithme défini plus en haut.

**Plan d’étude**

Table des matières

[1 – Présentation de l’algorithme à étudier 3](#_Toc152423747)

[2 – Présentation de Z3 4](#_Toc152423748)

[A – Installation 4](#_Toc152423749)

[B – Les différentes classes utiles 4](#_Toc152423750)

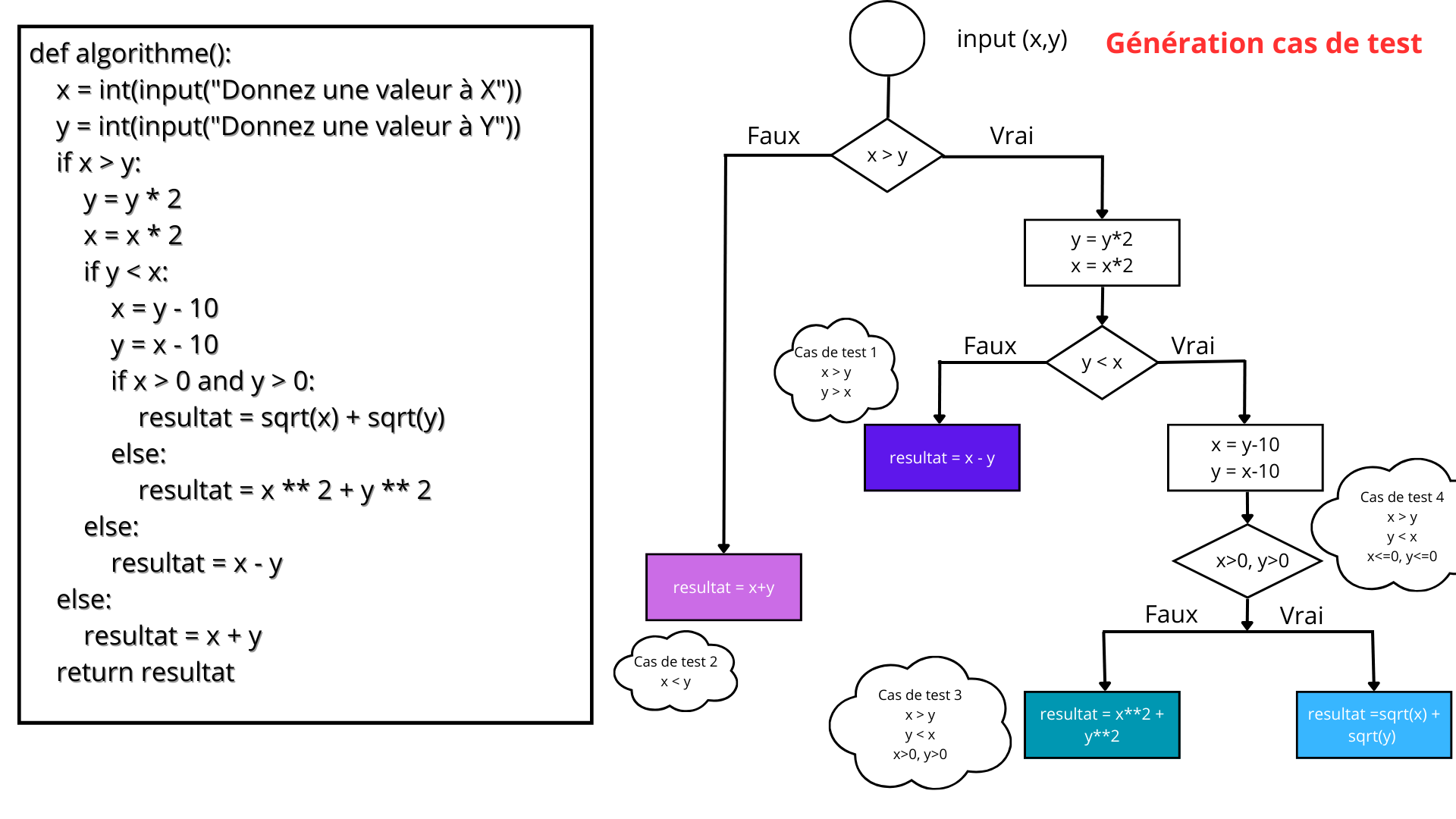
[3 – Cas de test 5](#_Toc152423751)

[A- La détection de code mort 5](#_Toc152423752)

[B- Les cas de test 6](#_Toc152423753)

[C- Cas de vérification d’invariant 7](#_Toc152423754)

# 1 – Présentation de l’algorithme à étudier



L’algorithme à étudier se présente comme suit. Chaque condition IF conduit à deux scénario vrai ou faux selon une condition bien définie.

* IF X > Y entraine à deux situations :
  + FAUX : L’algorithme se termine et donne le résultat X + Y
  + VRAI : L’algorithme se poursuit avec des opérations et tombe dans un autre IF
* IF Y < X entraine à deux situations
  + FAUX : L’algorithme se termine et donne le résultat X - Y
  + VRAI : L’algorithme se poursuit avec des opérations et tombe dans un autre IF
* IF Y > 0 & X > 0 entraine à deux situations
  + FAUX : L’algorithme se termine et donne le résultat X \*\* 2 + Y \*\*2
  + VRAI : L’algorithme se termine et donne le résultat SQRT(X)+ SQRT(Y)

Ces différentes conditions nous permettent de faire une serie de test quand on étudiera dans les parties suivantes

# 2 – Présentation de Z3

## A – Installation

Dans notre étude, nous utiliserons le langage python pour faire différentes analyses.

Pour l’installer, nous utilisons les commandes suivantes :

* PIP INSTALL Z3
* PIP INSTALL Z3-SOLVER

## B – Les différentes classes utiles

Au cours de notre projet nous avons utilisés différentes classes de Z3 pour effectuer nos tests. Nous commençons tout d’abord par importer la bibliothèque Z3 sur notre fichier python en faisant :

* from z3.z3 import \*

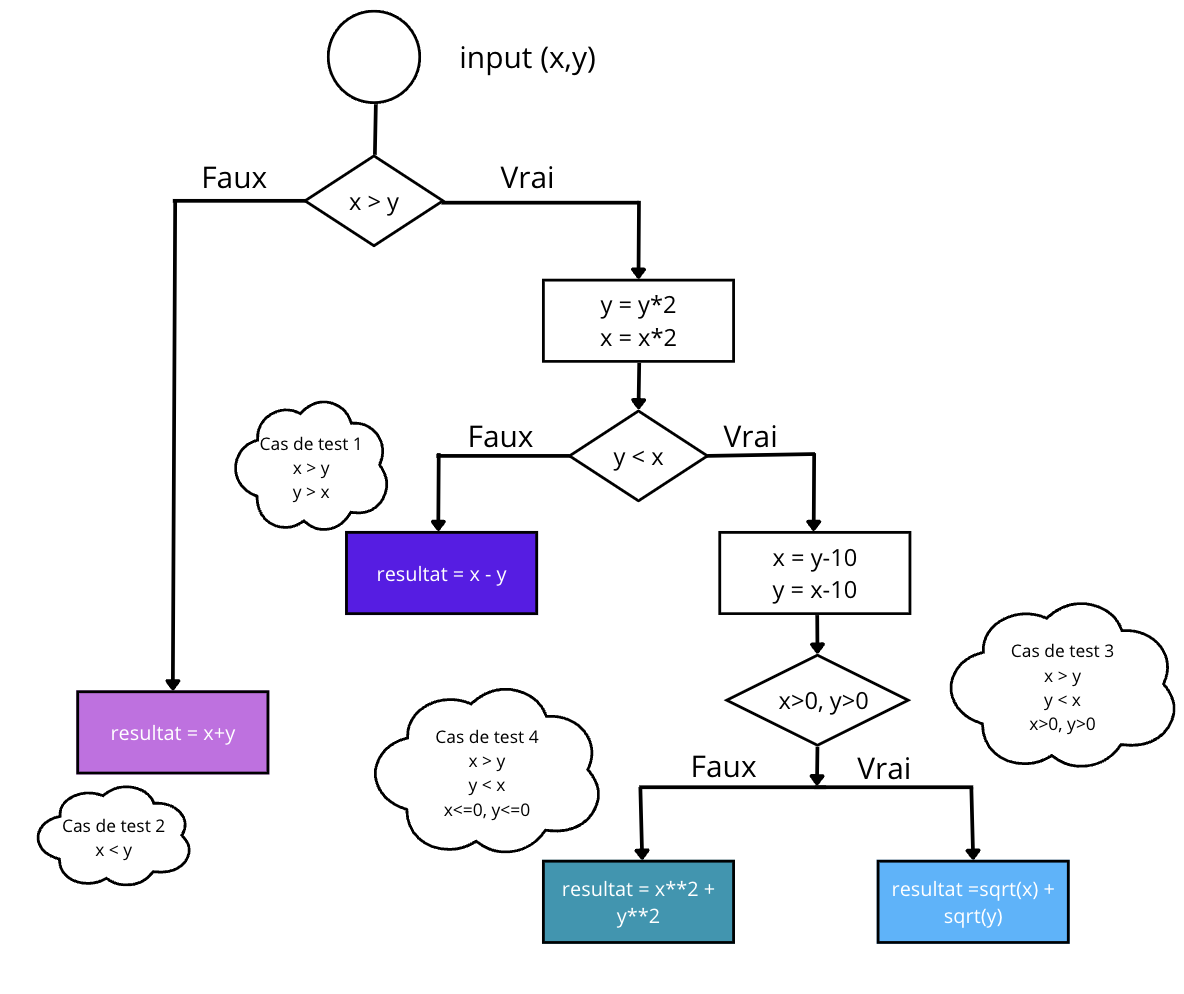
Cette ligne nous permet d’importer toutes les classes de Z3 pour qu’on puisse les utiliser dans notre projet. Voici les principales qu’on a eu à utiliser :

* Int() : permet de créer une instance Z3 de type integer
* Solver() : permet de créer des un solver qui permet de résoudre les équations logique
* Solver.add() : permet d’ajouter des conditions qui doivent résulter par True or False et qui doivent pris en compte par Z3
* Solver.check() : permet de résoudre l’équation logique et donne comme réponse ‘sat’ si la/les condition(s) sont satisfaite(s) ou ‘unsat’ au cas contraire
* Solver.Model() : donne les valeurs des variables Z3 permettant a ce que la/les condition(s) soit satisfaite(s)

# 3 – Cas de test

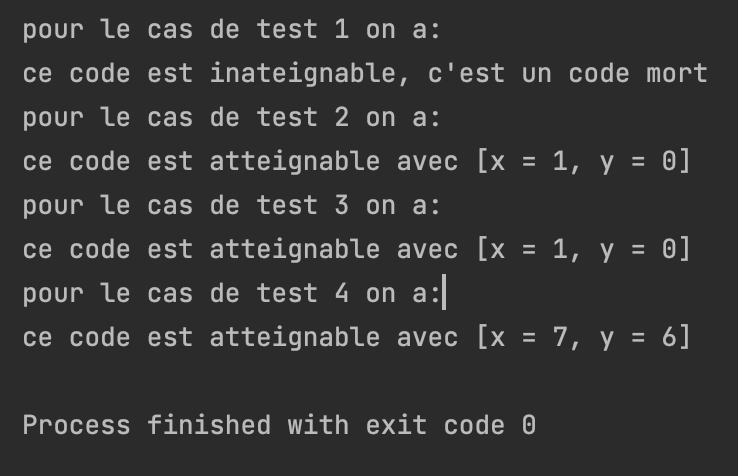
## La détection de code mort

La détection de code mort consiste à vérifier dans notre programme les branche qui ne seront jamais exécuté quel que soit la valeur d’entrée.



Dans notre exemple nous mettons 4 cas de test différents qui représente les branches que l’on veut vérifier si elles sont atteignables.

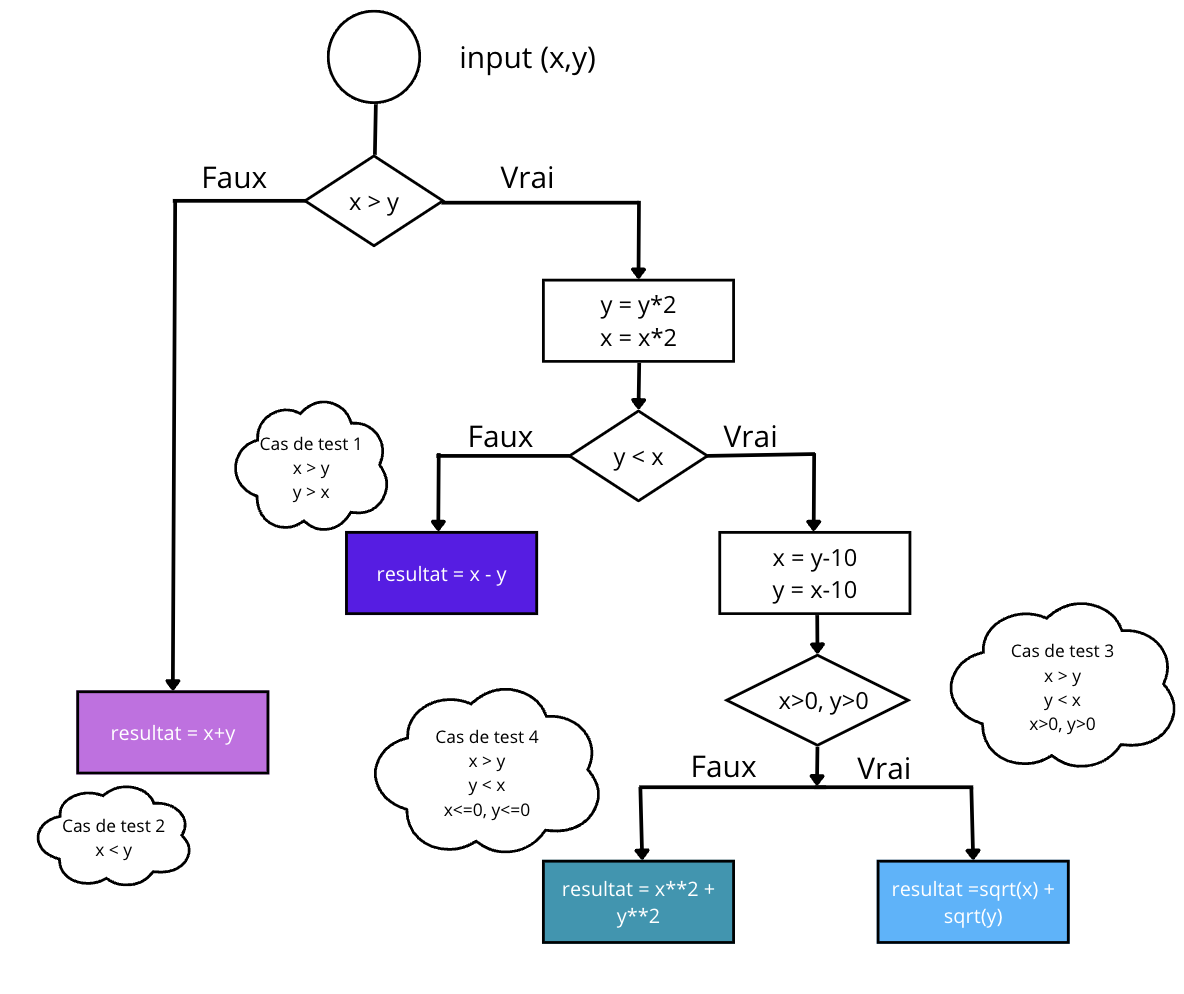
Après vérification nous avons les résultats suivants :



Ainsi comme on le voit, le cas de test 4 sur la détection de code mort montre que cette partie est inatteignable vu la condition car si x > y ---> x \* 2 > y\*2

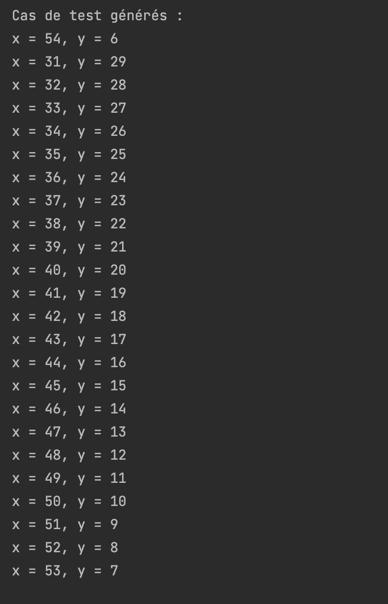
## Les cas de test

Les cas de test permettent de vérifier toutes les valeurs qui permettent de remplir les conditions données à Z3



Dans notre exemple, nous allons tester le cas de test 3. Dans le cadre de la génération de cas de test avec Z3, nous allons rajouter une condition à savoir *x + y =100* à ce cas de test pour avoir un cas de test fini car la condition initiale amène à un cas infini de solution.

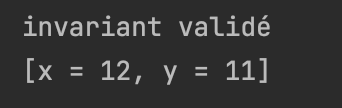
Nous avons le résultat suivant :



## Cas de vérification d’invariant

Ce cas de test permet de vérifier si une condition donnée pour une branche donnée peut être respecté tout le long de l’exécution.

Dans notre exemple nous allons faire tester l’invariant suivant : x > 10 et y > 10 sur le cas de test 3. On a également ajouté quelles sont les valeurs de x et de y en entrée qui respecte cette invariant. Voici le résultat obtenu.



**Conclusion**

Z3 nous permet de faire des vérifications poussées sur différents types de codes et ainsi prévenir des cas de blocages. Son utilisation pourrait être essentiel sur la vérification de systèmes critiques dans des domaines comme l’aérospatiale et le nucléaire pour garantir la pérennité de notre algorithme et prévenir tous les cas possibles à travers les vérifications que proposent Z3