COMPLEXITÉ ET CALCULABILITÉ

Mini-projet

Rémy KALOUSTIAN

SI4 – G3

Sujet

Les mots de Dyck

Il s'agit ici de traiter les mots de Dyck, autrement dit, les mots bien parenthésés. Notre mission est de réaliser des machines de Turing à une seule bande pour reconnaître les mots de Dyck et en créer.

Pour une définition plus scientifique, un mot de Dyck est :

un mot *w* de longueur *n* (*n*>=0) sur l'alphabet {O, F} - "O" représente "(" (ouvrante) et "F" représente ")" (fermante).

Enfin, notre but est de bien saisir la différence entre une machine EFFICACE et une machine qui ne l'est pas.

NB : Pour des raisons de simplicité, nous utilisons des parenthèses plutôt que O/F afin de pouvoir bien saisir directement la validité du mot. De plus, dans les exemples situés plus loin, nous utilisons b pour montrer un blanc.

Quelques exemples pour votre compréhension :

()()()()() , ((((())))) et (((())()(()))) sont des mots de Dyck car pour chaque parenthèse ouvrante, il y a une parenthèse fermante qui lui correspond.

)()()()(,) et ((())))) ne sont pas des mots de Dyck car il existe des parenthèses fermantes qui n'ont pas d'ouvrantes correspondantes et vice-versa.

Questions posées

Trois questions nous ont été posées.

1) Réaliser une machine simple, qui peut être de complexité élevé, et qui reconnaît les mots de Dyck.

2) Réaliser une machine efficace pour le même problème.

3) Réaliser une machine aussi efficace que possible qui supprime un minimum de lettres du mot pour transformer ce dernier en mot de Dyck.

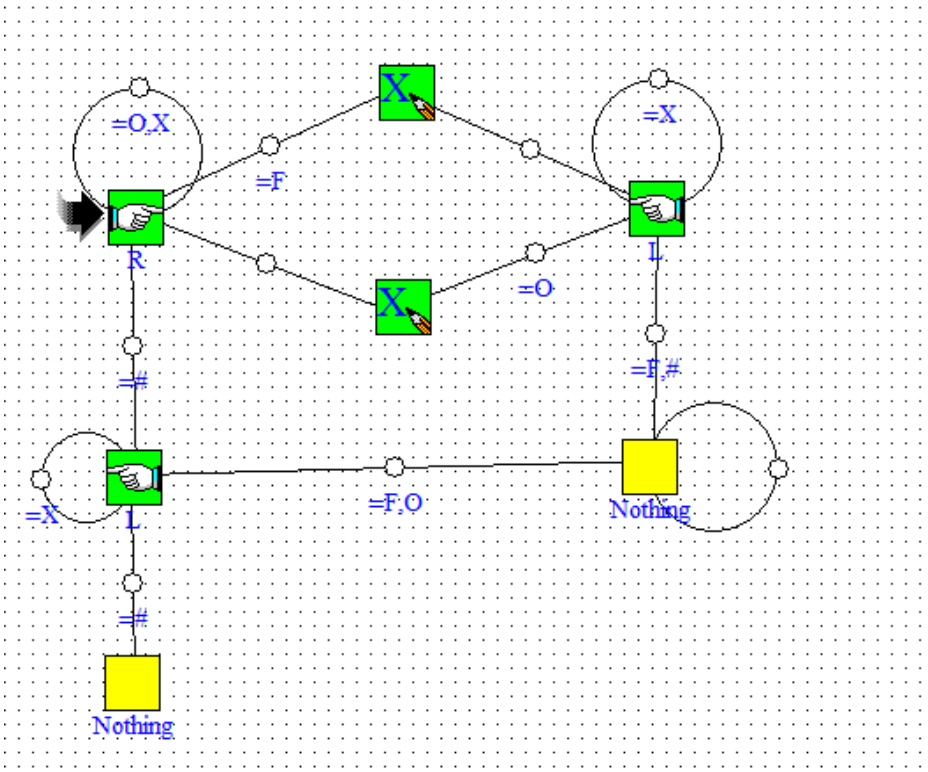
Vous trouverez dans ce document les illustrations des machines de Turing sous Visual Turing, les explications des machines, des exemples pour montrer leur fonctionnement, ainsi qu'une étude de la complexité.

1) Reconnaître les mots de Dyck

Notre but est de reconnaître les mots de Dyck, sans se soucier de la complexité de la machine, dans un premier temps. La construction d'une machine plus optimisée se fera pour la question suivante.

NB : les b correspondent au blanc situé avant et après le mot.

Description de la machine



Nous disposons d'une seule bande pour cette machine. Nous démarrons à gauche. Tant que nous avons des parenthèses gauches ou des parenthèses déjà marquées (symbole $), nous continuons vers la droite, dans le but de trouver une parenthèse fermante. Une fois la parenthèse fermante trouvée, on la marque ($), on revient vers la gauche pour chercher la parenthèse ouvrante correspondante. Une fois cette parenthèse ouvrante trouvée, on la marque($), on repart à droite pour trouver une parenthèse fermante, et une fois trouvée et marquée, on reviendra à gauche pour trouver sa parenthèse ouvrante correspondante, et ainsi de suite jusqu'à avoir un mot entièrement en $ (sauf s'il n'est pas un mot de Dyck).

Exemple N°1, cas valide (la position du curseur est indiquée par une couleur différente)

Mot : (())()

Application de la machine

On démarre à gauche.

(())()

Tant qu'on voit des (on va à droite (état 0).

(())()

On a une), on écrit $ et on va à gauche (état 0 vers état 1).

(($)()

On voit une (, on écrit $ et on va à droite (état 1 vers état 0).

($$)()

On voit un $, on va à droite, pour trouver la prochaine fermante (état 0).

($$)()

On voit une fermante, on écrit $, on va à gauche pour trouver l'ouvrante correspondante (état 0 vers 1).

($$$()

On voit $, on se déplace à gauche jusqu'à la prochaine parenthèse ouvrante (état 1).

($$$()

On voit (, on écrit $ et on va droite pour trouver la prochaine parenthèse fermante (état 1 à état 0).

$$$$()

Tant qu'on voit des $ ou (, on va à droite (état 0).

$$$$()

On voit), on écrit $, on va à gauche (état 0 à état 1).

$$$$($

On voit (, on écrit $, on va à droite pour trouver la prochaine parenthèse fermante (état 1 à état 0).

$$$$$$

Tant qu'on voit des $, on va à droite.

$$$$$$b

On est arrivé au blanc de droite, on écrit b et on va à gauche (état 0 vers état 2).

$$$$$$

Tant qu'on voit des $, on va à gauche (état 2).

b $$$$$$

On arrive au blanc gauche, on va dans l'état final (état 2 vers état 3).

On est bien arrivé à l'état final, la machine a reconnu le mot de Dyck.

Exemple N°2, cas invalide (la position du curseur est indiquée par une couleur différente)

Mot :)()((

Application de la machine

On commence à gauche.

)()((

On voit), on écrit $, on va à gauche (état 0 vers 1).

b$)((

On voit un blanc, on écrit b, on est en Stationnaire (état 1 vers état 4).

On est arrivé dans l'état puis, le mot n'est pas reconnu car il n'est pas un mot de Dyck.

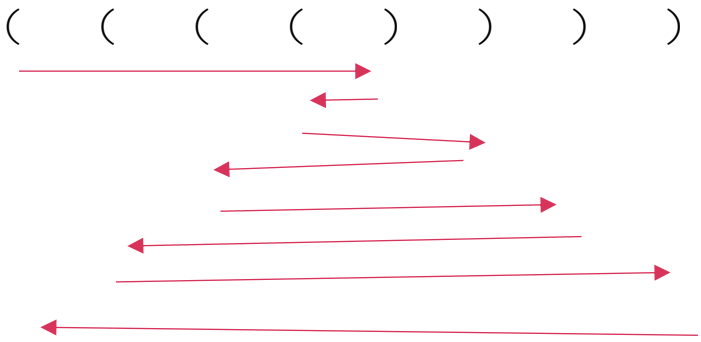
Étude de la complexité

Concentrons-nous sur le pire des cas : (((())))

Ici, on va effectuer N/2 parcours. A chaque parcours, la taille du parcours augmente de 2. On peut exprimer la longueur Lp du parcours de la manière suivante :

$Lp_k = Lp_{k-1} + 2$

Schéma représentatif du parcours :



On arrive alors à une complexité de N pour l'agrandissement du parcours, car à chaque parcours dans un sens, il y a un autre parcours plus grand dans l'autre sens.

Notre complexité générale est :

$N/2*O(N)$

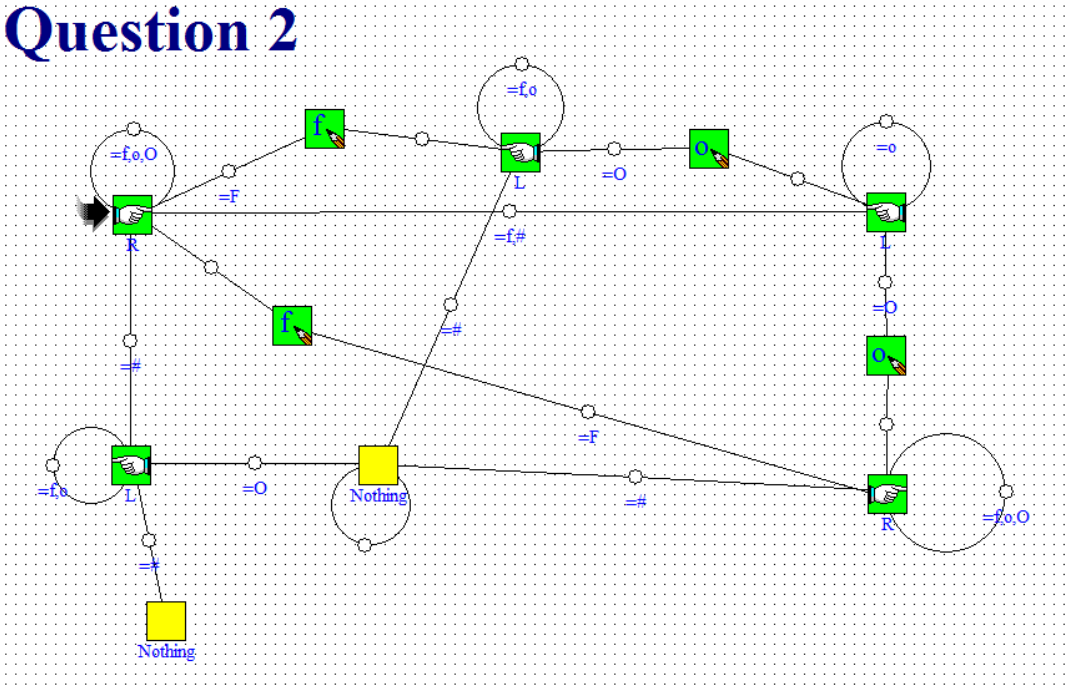
$= O(N^2)$

On se retrouve donc avec une complexité de $O(N^2)$.

2) Reconnaissance efficace

Cette fois-ci, nous devons reconnaître un mot de Dyck grâce à une machine de Turing efficace. Autrement dit, nous devons effectuer moins d'opérations que pour la question précédente. Nous comparerons les deux machines dans la partie consacrée à l'étude de la complexité.

Description de la machine



Cette machine dispose d'un fonctionnement similaire à celle de la question précédente. Toutefois, son comportement nous permet d'économiser beaucoup de déplacements et ainsi réduire le temps de validation du mot.

On commence à gauche, puis on va à droite pour trouver une fermante. Une fois cette fermante marquée, on revient ensuite à gauche pour trouver son ouvrante correspondante. Le mot sera validé lorsque chaque couple ouvrante-fermante sera marqué (O-F) et ne sera pas validé si il reste une parenthèse ouvrante sans fermante correspondante, ou bien une fermante marquée F sans ouvrante correspondante.

Étudions plus en détail cette machine à travers des exemples.

Exemple N°1, cas valide (la position du curseur est indiquée par une couleur différente)

Mot : ()()()

Application de la machine

On commence à gauche.

()()()

On lit (, on va à droite.

()()()

On lit), on écrit f, on va à gauche.

(f()()

On lit (, on écrit o, on va à gauche.

bof()()

On lit b, on va à droite.

of()()

Tant qu'on lit o,f,(, on va à droite.

of()()

On lit), on écrit f, on va à gauche.

of(f()

On lit (, on écrit o, on va à gauche.

ofof()

On lit f, on va à droite.

ofof()

Tant qu'on lit o,f,(, on va à droite.

ofof()

On lit), on écrit f, on va à gauche.

ofof(f

On lit (, on écrit o, on va à gauche.

ofofof

On lit f, on va à droite.

ofofof

Tant qu'on lit o,f, on va à droite.

ofofofb

On lit b, on va à gauche.

ofofof

Tant qu'on lit o,f, on va à gauche.

bofofof

On lit b, on est en stationnaire dans l'état de sortie.

Le mot est reconnu avec succès.

Exemple N°2, cas invalide (la position du curseur est indiquée par une couleur différente)

Mot : ((()))

Application de la machine

On commence à gauche.

((()))

Tant qu'on lit (, on va à droite.

((()))

On lit), on écrit F, on va à gauche.

((f))

On lit (, on écrit o, on va à gauche.

(of))

On lit (, on écrit o, on va à droite.

oof))

Tant qu'on lit o,f, on va à droite.

oof))

On lit), on écrit f, on va à droite.

ooff)

On lit), on écrit f, on va à gauche.

oofff

Tant qu'on lit o,f, on va à gauche.

boofff

On lit b, on va dans l'état poubelle en mode stationnaire.

Le mot n'est pas reconnu, c'est ce que nous attendions.

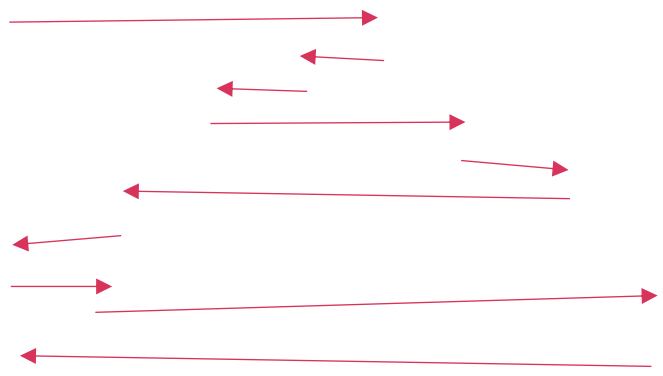
Étude de la complexité

De manière générale, la complexité de cette machine est du même ordre que la machine précédente. Elle demeure tout de même plus efficace

Voyons tout d'abord le pire des cas : ((((()))))

Simulons les parcours effectués

(((( ))))



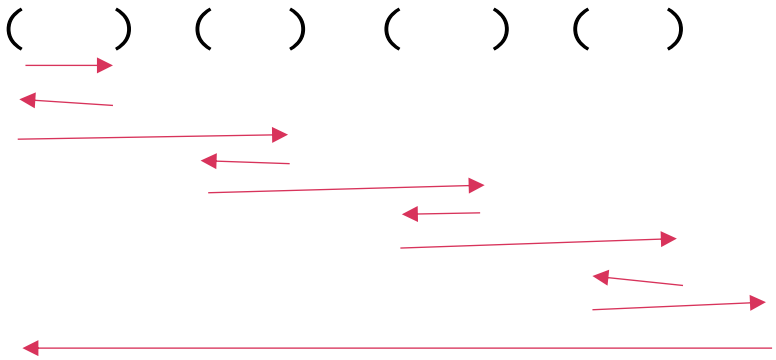
Dans ce cas, on note que certains parcours augmentent à chaque fois et ce, proportionnellement à la longueur N du mot. Ce type de parcours est similaire à celui effectué pour la machine de la question 1. On en déduit que la complexité du pire des cas est donc la même que pour la première machine.

Cependant, plusieurs éléments nous permettent de dire que cette machine est plus efficace que celle de la première question.

Premièrement, concernant le pire des cas, on peut voir qu'il y aura beaucoup de moins de déplacements. En effet, pour la machine de la question 1, chaque déplacement dans un sens se soldait par un autre déplacement dans l'autre sens. Ce n'est pas le cas ici.

Deuxièmement, étudions le meilleur des cas : ()()()()

Simulons le parcours.



On constate que la machine a un comportement similaire à celle du la question précédente. Il y a beaucoup moins de déplacements que dans le pire des cas et seul le dernier parcours dépend de la longueur N du mot. Cela amène à une complexité réduite et valide l'efficacité de la machine. Ainsi en plus d'améliorer l'efficacité de la machine, on conserve la rapidité d'exécution pour le meilleur cas, présent dans la machine précédente.

Enfin, une fois l'analyse du mot terminé, on garde une certaine cohérence, car les parenthèses ouvrantes et fermantes sont toujours discernables, là où elles sont remplacées par $ pour la première machine.

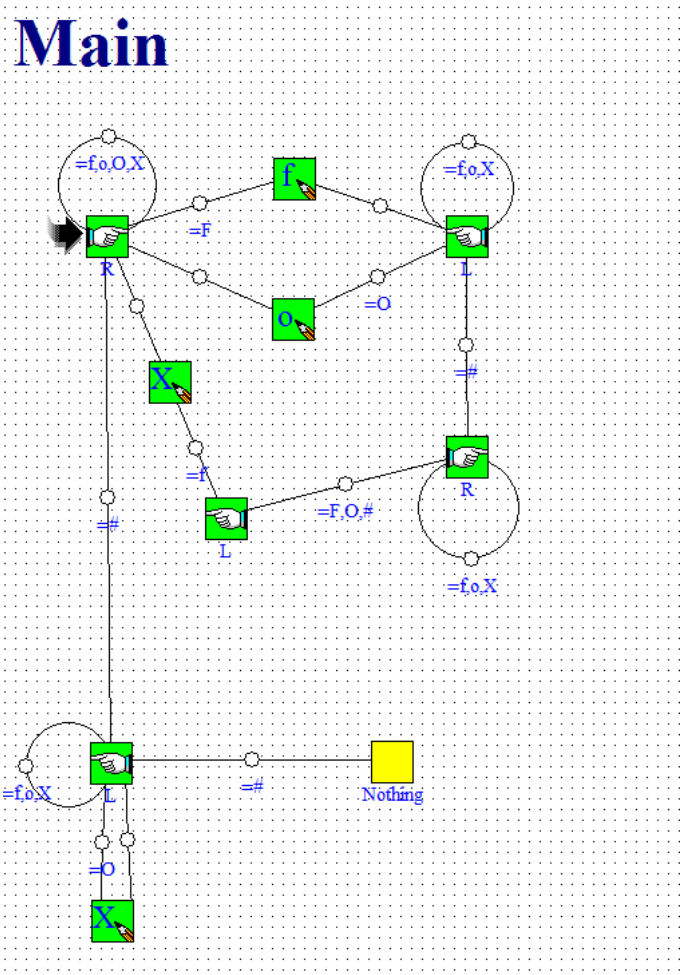
Par conséquent, nous pouvons dire que la deuxième machine est plus efficace que la première machine.

3) Corriger un mot

Notre but est ici de corriger un mot, autrement dit supprimer des parenthèses ouvrantes et/ou fermantes afin de transformer un mot quelconque en mot de Dyck.

Nous devons y arriver en réalisant un minimum de changement. Nous transformons les parenthèses en trop en un autre symbole pour faire du mot un mot de Dyck.

Description de la machine



La machine parcourt d'abord le mot afin d'identifier un couple ouvrante-fermante. Ensuite, lorsqu'on trouve une autre parenthèse, elle vérifie sa validité. Par exemple, si on a trouvé une fermante après avoir trouvé un couple ouvrante-fermante, on revient tout à gauche. L'absence d'ouvrante pour la fermante trouvée nous fera repartir à droite jusqu'au blanc ou une parenthèse non marquée. A gauche de ce blanc/parenthèse non marquée se trouve notre fermante précédemment trouvée, que l'on marquera avec un $, car elle n'a pas d'ouvrante correspondante.

On procède de même pour le reste du mot. L'idée principale est de noter les ouvrantes avec o, les fermantes avec f, afin d'identifier les couples ouvrante-fermante, et de vérifier leur validité. La validation se fera par un parcours tout à gauche, puis un tout à droite. Chaque trouvaille d'une fermante va enclencher un parcours de vérification.

Cet exemple semble pertinent car il ne marchait pas pour la première version de la machine correspondant à cette question. Étudions sa correction (la position du curseur est indiquée par une couleur différente).

Mot : ())))

Application de la machine

Nous démarrons à gauche.

(())))

On lit (, on va à droite.

(()))

On voit), on écrit f, on va à gauche.

(f))

On voit (, on écrit o, on va à droite.

of))

On lit f, on va à droite.

of))

On lit), on écrit f, on va à gauche.

off)

On lit f, on va à gauche

off)

Tant qu'on lit f, on va à gauche

boff)

On lit b, on va à droite.

off)

Tant qu'on lit o ou f, on va à droite.

off)

On lit), on va à gauche.

off)

On lit f, on écrit $, on va à droite.

of$)

On lit), on écrit f, on va à gauche.

of$f

Tant qu'on lit f ou o ou $, on va à gauche.

bof$f

On lit b, on va à droite.

of$f

Tant qu'on lit f ou o ou $, on va à droite.

of$fb

On lit b, on a à gauche.

of$f

On lit f, on écrit $, on va à droite.

of$$b

On lit b, on va à gauche.

of$$

Tant qu'on lit o,f,$, on va à gauche.

bof$$

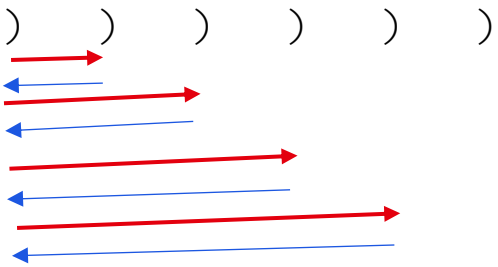
On lit b, on est en stationnaire dans l'état final.

Le mot a été corrigé avec succès.

Étude de la complexité

Dans le pire des cas :))))))) , autrement dit un mot avec seulement des parenthèses fermantes.

Ici pour chaque parenthèse fermante, on va effectuer un parcours pour marquer f, on va revenir tout à gauche, puis repartir tout à droite, trouver une fermante et maquer $ sur la fermante précédente.



Les parcours en bleu sont effectués N fois, tout comme les parcours en rouge. On a donc une complexité de O(N) dans chaque parcours. Pour chaque parcours dans un sens, il y a un autre parcours dans l'autre sens.

Pour un mot de longueur N, la complexité vaut :

N * Pg +N * Pd (avec Pg le parcours vers la gauche, et Pd le parcours vers la droite)

= N*O(N)+ N*O(N)

=2*O(N^2)

= O(N^2)

La complexité du pire des cas est donc de l'ordre de N^2.

Dans le meilleur des cas : ((()))

Ici, à chaque parcours pour trouver les correspondances ouvrante-fermante, on réduit le parcours de 2 unités.

On fait ainsi N/2 parcours. Imaginons k l'indice d'un parcours, on peut exprimer la longueur Lp d'un parcours de la manière suivante :

$Lp_k = Lp_{k-1} - 2$

Malgré tout, le fait que chaque parcours dans un sens soit suivi d'un parcours dans l'autre conserve la complexité à O(N^2). La diminution du parcours rend toutefois une complexité moins importante que dans le pire des cas.

Exemples sur Visual Turing

Les exemples ont été sélectionnés pour représenter des cas différents pour chaque machine, sauf lors de comparaisons des machines pour un même type de mot. Ces exemples ont de plus été choisis car ils modélisent des cas pouvant être source de problème si non pris en compte dans les machines.

Pour la question 1, les exemples dans Visual Turing sont :

OOOFFF, qui est un cas assez normal permettant de se rendre compte du fonctionnement de la machine et de sa longueur de parcours (en effet, ce mot est un des pires cas).

OF, qui permet de voir que la machine marche même pour les mots courts.

OFOFOF, qui permet de comparer cette machine avec celle de la question 2.

OFOFOFFO, qui permet de voir comment la machine se comporte dans un cas incorrect, avec une F au milieu d'un mot de Dyck.

FFFFOOOO, qui permet de voir que la machine ne compte pas juste le nombre de parenthèses mais les associe également.

O, qui permet de voir un cas très spécial.

NB : Les $ sont matérialisés par un X sur Visual Turing. De même pour la troisième machine.

Pour la question 2, les exemples dans Visual Turing sont :

OOOOOFFFFF, qui est le pire des cas et permet de voir la différence de traitement comparé à la machine 1.

F, un autre cas très spécial, pour vérifier que cette machine prend en compte tous les cas.

OFOFOFOF, qui permet de voir qu'on garde à peu près le même comportement que la machine précédente dans ce cas-là.

FFOFOOOFOF, qui permet de vérifier que la machine marche dans un cas incorrect.

Pour la question 3, les exemples dans Visual Turing sont :

OOOOFFF, qui permet de voir le fonctionnement de la machine sur un cas avec que des O à gauche et que des F à droite.

FOFOFOFO, qui est un cas particulier puisqu'il contient un mot de Dyck, mais n'en est pas un, du fait de ses extrémités.

OFOOFF, qui permet de vérifier que la machine ne change pas un mot déjà valide.