

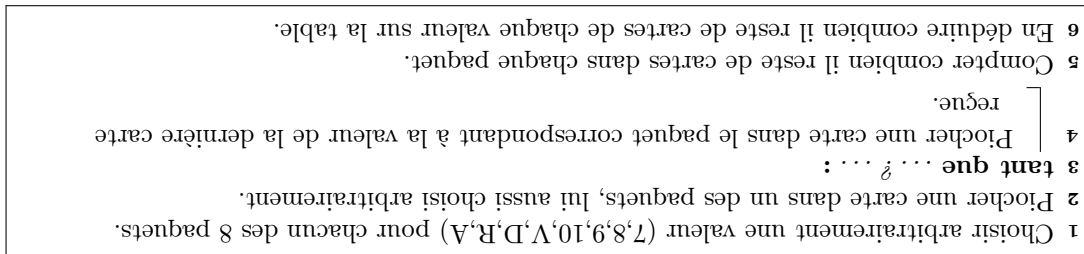
Objectifs

- À la fin de cette séance, vous devriez être capable de :
- concevoir et exploiter un invariant de boucle ;
 - donner un variant de boucle afin de prouver la terminaison d’un algorithme simple ;
 - spécifier un algorithme simple et démontrer sa correction à l’aide d’un invariant.

Exercice 1 : Encore un tour de cartes

Observez attentivement le tour de cartes proposé par votre enseignant.

1. Essayez de découvrir le « truc » et de l’exprimer (informellement).
Vérifiez que votre idée est correcte en essayant de réaliser le tour vous-mêmes.
Si vous n’y arrivez pas, vous pouvez consulter l’algorithme suivant (mais pas avant d’avoir cherché!) :



2. Précisez ce qui se passe lorsque le tour se termine : quelle est la postcondition de l’algorithme (et donc comment le magicien sait-il ce qui reste sur la table) ?
3. Exprimez précisément un invariant du tour de magie.
4. Démontrez que votre invariant en est effectivement un.
5. Votre invariant est-il vérifié lorsque le tour commence ? (c’est-à-dire après avoir pioché la toute première carte)

Sinon, reprenez à l’étape 3 et affinez votre invariant pour tenir compte de la situation initiale.
6. À quel moment le magicien a-t-il intérêt à arrêter de piocher des cartes pour que le tour de magie soit plus simple à réaliser ?

Correction de l’exercice 1

1. Le principe est le suivant : chacun des 8 paquets de cartes correspond à une valeur possible pour les cartes. Le nombre de cartes restant dans un paquet doit correspondre au nombre de cartes restantes de cette valeur.
Ainsi, lorsqu’on pioche une carte d’une certaine valeur, cela indique le prochain paquet dans lequel on doit piocher.
2. Une expression simple pourrait être :
« Le nombre de cartes présent dans chaque paquet est égal au nombre de cartes de ce type présentes sur la table » mais en fait on est décalé d’une itération : le premier paquet dans lequel on se sert ne respecte pas cette propriété (il lui manque une carte) et la dernière carte piochée non plus (le paquet correspondant comporte une carte de trop).
On peut « réparer » l’invariant en précisant ces deux exceptions, ou bien d’une façon un peu plus concise :
« Si je repose la dernière carte piochée sur le premier paquet, alors le nombre de cartes présent dans chaque paquet est égal au nombre de cartes de ce type présentes sur la table. »
3. Supposons l’invariant vérifié en début d’itération, et que la dernière carte piochée est un X : comme on pioche une carte dans le paquet X, on diminue de 1 le nombre de cartes dans le paquet X, mais

on garde également en main une carte X de plus, donc on diminue de 1 le nombre de X sur la table.

Les valeurs de cartes différentes de X ne sont pas affectées par cette itération (en particulier la carte piochée à cette itération n'a pas d'importance puisque l'invariant prévoit de la reposer sur la table avant de compter).

4. La toute première carte piochée doit être prise sur un paquet arbitraire, elle doit donc être considérée comme une initialisation et pas comme une itération. Avant la première itération, si on repose donc cette première carte sur son paquet, il y a bien sur la table 4 cartes de chaque valeur (l'intégralité du paquet).
5. En fin de tour (lorsqu'il s'arrête de piocher), le magicien sait donc exactement combien il reste de cartes de chaque valeur sur la table.

Un bon moment pour arrêter le tour consiste à attendre que « reposer la dernière carte piochée sur le premier paquet » soit une opération triviale, par exemple :

- quand la valeur de la dernière carte piochée est celle du premier paquet ;
- ou encore plus simple, lorsqu'il faudrait piocher une carte dans le premier paquet alors qu'il est déjà épuisé : il est alors clair que le contenu de la table est directement donné par l'état actuel des paquets.

À l'inverse, si une telle situation se produit « trop tôt » dans le tour, on se retrouve avec une situation similaire à la précondition (le nombre de cartes présent dans chaque paquet est égal au nombre de cartes de ce type présentes sur la table). Il suffit de recommencer en piochant une carte dans un autre paquet arbitraire.

Exercice 2 : Preuve d'algorithme

On s'intéresse à l'algorithme suivant :

Data : un entier n
Result : c est une valeur approchée de la racine carrée de n
 $c := 0$
 $s := 1$
tant que $s \leq n$:
 | $c := c + 1$
 | $s := s + 2 * c + 1$

1. Écrire une spécification de l'algorithme (précondition, postcondition) plus précise que ce qui est décrit ici.
2. Démontrer qu'il se termine à l'aide d'un variant de boucle bien choisi.
3. Démontrer que $(c + 1)^2 = s$ est un invariant de la boucle.
4. Quelle partie de la postcondition pouvez-vous démontrer à l'aide de cet invariant ?
5. Quel autre invariant faut-il étudier pour établir complètement la correction de cet algorithme ?

Correction de l'exercice 2

1. Il est clair que la precondition est ici $n \geq 0$ sinon un calcul de racine carrée n'aurait pas de sens. Pour préciser ce qui est dit dans l'énoncé, cet algorithme calcule l'arrondi à l'entier inférieur de la racine carrée de n . Mais pour exprimer la postcondition sans paraphraser et pour éviter de manipuler des racines carrées, on va plutôt démontrer que $c^2 \leq n < (c + 1)^2$.
2. Ici s augmente et ne peut pas dépasser n , on va donc démontrer que $n - s$ est un variant. C'est clairement un entier positif (car $s \leq n$), il reste à démontrer qu'il décroît strictement à chaque itération. Or, dans une itération, s augmente de $2c + 1$. Comme c est lui-même positif, $2c + 1 \geq 1$, et donc $n - s$ diminue au moins de 1.
3. Supposons que $(c + 1)^2 = s$ au début d'une itération. À la fin on a alors $c' = c + 1$ et :

$$\begin{aligned}
 s' &= s + 2 * c' + 1 \\
 &= (c + 1)^2 + 2 * c' + 1 \quad (\text{par hypothèse de début d'itération}) \\
 &= c'^2 + 2 * c' + 1 \\
 &= (c' + 1)^2
 \end{aligned}$$

l'invariant est donc maintenu par une itération.

Remarquons par ailleurs qu'avant d'entrer dans la boucle $c = 0$ et $s = 1$ donc initialement on a bien $s = (c + 1)^2$.

4. En sortie de boucle on aura $s = (c + 1)^2$ et $s > n$, ce qui assure que $(c + 1)^2 > n$ (deuxième inégalité de la postcondition).
5. Il faut encore qu'on montre que $c^2 \leq n$ en fin d'algorithme. Pour cela, on peut simplement prendre cette inégalité elle-même comme invariant. En effet, la condition du Tant que assure que $s \leq n$ en début d'itération, et on a montré précédemment que $s = (c + 1)^2$ à ce moment. On a donc $(c + 1)^2 \leq n$, et à la fin de l'itération $c' = c + 1$ donc $c'^2 \leq n$.

Remarquons qu'ici on n'a pas vraiment utilisé l'hypothèse que l'invariant était vérifié en début d'itération, la condition de la boucle suffit à le vérifier en fin d'itération.

En revanche, l'initialisation de cet invariant ($c^2 \leq n$ avant d'entrer dans la boucle) repose sur la precondition de notre algorithme ($n \geq 0$).