Rapport Mega Machine à Caoua

Paul Ecoffet Mathieu Seurin

Vendredi 28 Novembre 2014

Nous allons ici détailler les fonctions utilisées dans notre programme (signature et axiomes) ainsi que faire l'analyse de leur complexité en notation **O**. Après chaque méthode, nous indiquerons le fichier correspondant aux tests que nous avons effectués, ainsi que les noms des tests.

Fonctions de Machine, mode fonctionnement

order :

1. Signature : (Monnaie, Commande) $\Rightarrow$ (Boisson$\cup\emptyset\times$Monnaie)$\cup$Exception
2. Axiomes :
   - $\forall$commande $\in$ Drink, Drink l'ensemble de tous les drinks possibles tel que $\forall$ type $\in$ Commande.stock, commande.stock[type] $\leq$ Machine.Stock[type]
     $\forall$monnaie $\in$ Coins tel que monnaie.compute_surplus(Machine.MaxCashInput) $\neq$ Error
     et monnaie.value $>$ commande.price order(commande, monnaie) $\Rightarrow$ Drink(commande), change
   - $\forall$commande $\notin$ Drink, Drink l'ensemble de tous les drinks possibles
     $\forall$monnaie $\in$ Coins
     order(commande, monnaie) $\Rightarrow$ InvalidOrderException
   - $\forall$Commande $\in$ Drink, Drink l'ensemble de tous les drinks possibles
     $\forall$Monnaie $\in$ Coins
     tel que Monnaie.compute_surplus(Machine.MaxCashInput) = NoChangePossibleException
     order(commande, monnaie) $\Rightarrow$ None, monnaie
   - $\forall$commande $\in$ Drink, Drink l'ensemble de tous les drinks possibles tel que $\forall$ type $\in$ Commande.stock, commande.stock[type] $>$ Machine.Stock[type] order(commande, monnaie) $\Rightarrow$ NotEnoughStockException

3. Complexité : Max(∀ functions ∈ order: Complexité(functions)) = $O(2^n)$ complexité de Coins.compute_surplus, avec n le nombre de pièces dans coins.
4. Test : *test_machine.py*
    - test_order_simple()
    - test_order_complex()
    - test_order_fail_not_enough_cash()
    - test_order_fail_not_drink()
    - test_order_fail_no_stock()
    - test_order_cant_get_maxcash()

Fonctions de Machine, mode Maintenance

reset :

1. Signature : $\emptyset \Rightarrow \emptyset$
2. Axiomes :

$$\text{machine.reset()} \Rightarrow \begin{cases} \text{machine.\_stocks[type]} = 0 & \forall \text{type} \in \text{machine.StocksType} \\ \text{machine.\_coins[type]} = 0 & \forall \text{type} \in \text{machine.CoinssType} \\ \text{machine.\_cash} = 0 \end{cases}$$

edit_prices :

1. Signature : (dictionnaire_prix) $\Rightarrow \emptyset \cup$ TypeError
2. Axiomes :
    - ∀ stock ∈ {'thé', 'café', 'lait', 'chocolat'},
      ∀ prix ≥ 0
      edit_prices(stock=prix) ⇒ machine._stock_prices[stock] = prix
    - Si stock = 'sucre' ∀ $0 \leq \text{prix}_i \leq \text{prix}_{i+1}$, i ∈ $[0, 3]$ edit_prices(sucre= [prix_i])
      machine._stock_prices[stock] = [(prix_i)] ∀i ∈ [0,3]
3. Complexité : $O(n)$ avec n le nombre de produits payant
4. Test : *test_machine.py*
    - test_edit_prices

edit_stocks :

1. Signature : (dictionnaire_stocks) $\Rightarrow \emptyset$
2. Axiomes :
   - $\forall$ stock $\in$ {'thé', 'café', 'lait', 'chocolat', 'sucre'},
     $\forall$ machine.quantite[stock] $<$ quantite $\leq$ machine.quantite_max[stock]
     machine.edit_stock(stock=quantite) $\Rightarrow$ machine.quantite[stock] = quantite
   - $\forall$ stock $\in$ {'thé', 'café', 'lait', 'chocolat', 'sucre'},
     $\forall$ quantite $\leq$ machine.quantite[stock] ou quantite $>$ machine.quantite_max[stock]
     machine.edit_stock(stock, quantite) $\Rightarrow$ machine.quantite[stock] = machine.quantite[stock]
3. Complexité : $O(n)$ avec n le nombre de stocks différents
4. Test : *test_machine.py*
   - test_edit_stocks

refill_stocks :

1. Signature : $\emptyset \Rightarrow \emptyset$
2. Axiomes :
   - $\forall$ stock $\in$ Machine.StocksType,
     machine.refill_stock() $\Rightarrow$ machine.quantite[stock]
     = machine.quantite_max[stock]
3. Complexité : $O(n)$ avec n le nombre de stocks différents
4. Test : *test_machine.py*
   - test_edit_prices

edit_coins :

1. Signature : coins $\Rightarrow \emptyset$
2. Axiomes :
   - $\forall$ pieces $\in$ machine.CoinsType et pieces $\in$ coins,
     $\forall$ 0 $\leq$ coins[pieces] $\leq$ machine._max_coins[pieces]
     edit_coins[coins] $\Rightarrow$ machine._coins[pieces] = coins[pieces], $\forall$pieces
3. Complexité : $O(n)$ avec n le nombre de types de pièces différentes gérées par la machine
4. Test : *test_machine.py*
   - test_edit_prices

refill_coins :

1. Signature : $\emptyset \Rightarrow \emptyset$
2. Axiomes :
   - machine.refill_coins() $\Rightarrow \forall$ valeur $\in$ Machine.CoinsType, machine.coins[valeur] = machine.max_coins[valeur]
3. Complexité : $O(n)$ avec n le nombre de types de pièces différents
4. Test : *test_machine.py*
   - test_edit_prices

remove_stocks :

1. Signature : stock_dict $\Rightarrow \emptyset$
2. Axiomes :
   - $\forall$ A = (stock_type, value)$_i$, $i \in \mathbb{N}$, stock_type$_i$ $\in$ Machine.StocksType
     machine.remove(A) $\Rightarrow \forall$ stock_type, value $\in A$, machine._stocks[stock_type] = machine._stocks[stock_type] - value
3. Complexité : $O(n)$ avec n le nombre de types de stocks gérés par la machine
4. Test : *test_machine.py*
   - test_remove_stocks

add_to_cash :

1. Signature: Coins $\Rightarrow \emptyset$
2. Axiomes :
   - $\forall$coins $\in$ Coins,
     machine.add_to_cash(coins) $\Rightarrow \forall$ type, quantite $\in$ coins,
     machine.cash[type] = machine.cash[type] + quantite

Fonctions de Coins

Coins hérite de collections.Counter.

compute_surplus

1. Signature: value ⇒ change ∈ Coins ∪ NoChangePossibleException
2. Axiomes:
   - ∀ coins *in* Coins, coins.value ≥ value,

     $$\text{coins.compute\_surplus(x)} \Rightarrow \begin{cases} \text{(coins - change).value = value} & \text{si possible} \\ \text{NoChangePossibleException} & \text{si impossible} \end{cases}$$

   - ∀ coins *in* Coins, coins.value ≤ value,
     coins.compute_surplus(x) ⇒ NoChangePossibleException
3. Complexité: $O(2^n)$, n le nombre de pièces dans coins.
4. Tests: *test_coins.py*
   - test_compute_surplus

compute_change

1. Signature : change_value ⇒ Coins
2. Axiomes :
   - ∀ coins *in* Coins, coins.value ≥ value,

     $$\text{coins.compute\_change(x)} \Rightarrow \begin{cases} \text{(change).value = change\_value} & \text{si possible} \\ \text{NoChangePossibleException} & \text{si impossible par division} \end{cases}$$

   - ∀ change, change > coins.value,
     coins.self_compute(change) ⇒ NoChangePossibleException
3. Complexité: $O(n)$ avec n le nombre de types de pièces dans coins.
4. Tests: *test_coins.py*
   - test_compute_change
   - test_compute_change_not_enough_cash
   - test_compute_change_impossible

value

1. Signature : ∅ ⇒ ℕ
2. Axiomes :
   - ∀ coins *in* Coins, coins = (valeur, quantite)$_{n\in\mathbb{N}}$,

     $$\text{coins.value} = \sum_{i=1}^{n} \text{valeur}_i \times \text{quantite}_i$$

3. Complexité: $O(n)$ avec n le nombre de types de pièces dans coins.
4. Tests: *test_coins.py*
    - test_coins_value

Méthodes de Drink

price

1. Signature : $\emptyset \Rightarrow \mathbb{N}$
2. Axiomes :
    - $\forall$ stock_i $\in$ drink.stocks, $\forall$ price_i $\in$ drink.stock_prices, i $\in\{1, \ldots, n\}$

$$\text{drink.price} = \sum_{i=1}^{n} \text{stock}_i \times \text{price}_i$$

3. Complexité : $O(n)$ avec n le nombre de types de stock différents dans drink.
4. Tests: *test_drink.py*
    - test_drink_price

has_beverage

1. Signature: $\emptyset \Rightarrow T, F$
2. Axiomes :
    - $\forall$ drink avec $\exists x \in$ {'chocolate','tea','coffee'}, $x \in$ drink.stock, drink.has_beverage $\Rightarrow$ T
    - $\forall$ drink avec $\forall x \in$ {'chocolate','tea','coffee'}, $x \notin$ drink.stock, drink.has_beverage $\Rightarrow$ F
3. Complexité : $O(1)$
4. Tests: *test_drink.py*
    - test_drink_has_beverage

Méthodes des Logs

Log.message

1. Signature: $\emptyset \Rightarrow \emptyset$

2. Axiomes: ∀ log._message
   log.message = log._message
3. Complexité : $O(1)$
4. Tests: *test_logs.py*
   - test_log

StockLog.message

1. Signature: $\emptyset \Rightarrow \emptyset$
2. Axiomes:
   - ∀ (stock, quantity)$_n$ ∈ prev_stock,
     ∀ (n_stocks, n_quantite)$_n$ ∈ log.cur_stock log.message ⇒

     $\sum_{i=1}^{n}$ "{p_stock_i} : {$p\_quantite_i$} → {$n\_quantité_i$}{$n\_quantité - p\_quantite$}"
3. Complexité : $O(1)$
4. Tests: *test_logs.py*
   - test_stock_log_no_changes
   - test_stock_log_message
   - test_stock_log_str

OrderLog.message

1. Signature: $\emptyset \Rightarrow \emptyset$
2. Axiomes:
   - ∀ commande ∈ Drink, Drink l'ensemble des Drinks possible
     ∀ monnaie ∈ Coins
     Orderlog.message ⇒ "{commande} which cost {commande.price}
     the customer gave {monnaie.value}"
3. Complexité : $O(1)$
4. Tests: *test_logs.py*
   - test_cash_log_message

EndOrderLog.message

1. Signature: $\emptyset \Rightarrow \emptyset$
2. Axiomes:
   - EndOrderLog.message() ⇒ "That's all folks"

3. Complexité : $O(1)$
4. Tests: *test_logs.py*
    - test_end_order_log_message

CoinsLog

CoinsLog est un alias de StockLog avec un intitulé différent.
Test : *test_logs.py*
* test_coins_log_message

CashLog.message

1. Signature: $\emptyset \Rightarrow \emptyset$
2. Axiomes:
    - $\forall$ monnaie$_i \in$ Coins CashLog.message $\Rightarrow$
      "monnaie$_i \rightarrow$ monnaie$_{i-1}$(monnaie$_i -$ monnaie$_{i-1}$)"
3. Complexité : $O(1)$
4. Tests: *test_logs.py*
    - test_cash_log_message