Aufgabe 2a. Sequenzielle Spezifikationen

Verkaufsmaschineprotokoll :

```
% Definition der coins
sort
        Coin = struct _5c | _10c | _20c | _50c | Euro;
map
        value: Coin  -> Int;   % Der Wert des coin als integer deklariert

        % Initialisierung die Werte der deklarierten Coinstypen
eqn
        value(_5c) = 5;
        value(_10c) = 10;
        value(_20c) = 20;
        value(_50c) = 50;
        value(Euro) = 100;

% Definition der Produkte
sort
        Product = struct tea | coffee | cake | apple;
map
        price: Product  -> Int;         % the price of a product as an integer
```

```
        % Initialisierung die Werte der deklarierten Produkttypen
eqn
        price(tea) = 10;
        price(coffee) = 25;
        price(cake) = 60;
        price(apple) = 80;

% Definition der Aktionen
act
        accept: Coin;      % accept a coin inserted into the machine
        return: Coin;           % returns change
        offer: Product;      % offer the possibility to order a certain product
        serve: Product;      % serve a certain product
        returnChange: Int;   % request to return the current credit as  change

% Definition der Prozesse
%Muss Credit kleier als 200,um neue Coints zu akzeptieren
%Das Credit beginnt mit 0 Wert
%Credit soll gleich oder größer als Produktpreis sein
%Produktpreis wird von Credit abgezogen

proc
        VendingMachine = VM(0);

        VM(credit : Int) =
```

```
        sum coin:Coin.(credit<200)->accept(coin).VM(credit+value(coin)) +
        (credit>0)->returnChange(credit).ReturnChange(credit) +
        sum product:Product.(credit>=price(product)) ->
offer(product).serve(product).VM(credit-price(product));

    % Creditwert muss gleich oder größer als gegebene Returnwert
    %Returnwert wird vom Credit abgezogen

    ReturnChange(credit : Int) =
%sum coin:Coin.(credit >= value(coin)) -> return(coin).ReturnChange(credit -
value(coin))
(credit>=value(Euro)) -> return(Euro).ReturnChange(credit-value(Euro)) <>
(credit>=value(_50c)) -> return(_50c).ReturnChange(credit-value(_50c)) <>
(credit>=value(_20c)) -> return(_20c).ReturnChange(credit-value(_20c)) <>
(credit>=value(_10c)) -> return(_10c).ReturnChange(credit-value(_10c)) <>
(credit>=value(_5c)) -> return(_5c).ReturnChange(credit-value(_5c)) <> VM(credit);

% Es wird mit dem Prozess VendingMachine begonnen

init
    VendingMachine
;
```

Beschreibung der Probleme :

1. Sollen verchiedene Coinstypen definiert werden.
2. Der Wert des Coin muss eine Zahl sein.
3. Die Coins müssen zwischen 5 Cent und 1 Euro sein.
4. Sollen verschiedene Produkttypen definiert werden.
5. Jedes Produkt hat ein Preis.
6. Es gibt verschiedene Aktionen, die vom Benutzer ausgeführt werden können.
7. Das Guthaben muss ausreichen, um ein Produkt auszuwählen, und wenn ein Produkt gewählt wurde (*offer: Product* wurde gewählt), wird das Produkt ausgegeben (*serve: Product*) und das Guthaben entsprechend angepasst.
8. Der Automat kann nur bei einem Guthaben kleiner als 200 noch weitere Münzen akzeptieren.
9. Das Guthaben können als Wechselgeld zurückgegeben werden, wenn Credit positiv ist, und das Wechselgeld wird von groß nach klein ausgegeben.

Lösungen der Probleme :

1. Sort Coin = struct _5c | _10c | _20c | _50c | Euro; Mit Sort Coin kann Datentyp Coin mit verschiedenen Namen definiert werden.
2. Map value: Coin -> Int; Mit Map wird den Wert des Coin als Int definiert.
3. Eqn value(Coinname) = x; Mit Eqn kann den Wert des Coin zwischen 5 und 100 initialisiert werden.

4. Sort Product = struct tea | coffee | cake | apple; Mit Sort Product kann Datentyp Product mit verschiedenen Namen definiert werden.
5. Map price: Product -> Int; eqn price(Productname) = x; Mit Map wird den Wert des Produkts als Int definiert, und mit Eqn können die Werte der Produkte initialisiert werden.
6. Act accept: Coin; return: Coin; offer: Product; serve: Product; returnChange: Int;.Es kann mit Act verschiedene Aktionen definiert werden.
7. Proc VendingMachine = VM(0);
   VM(credit : Int) = sum product:Product.(credit>=price(product)) -> offer(product).serve(product).VM(credit-price(product)). Es wird erst überprüft, ob das Guthaben größer als oder gleich das Preis des Produkts ist, wenn passt, dann kann denn Nutzer die Aktion offer für ein Produkt auswählen, danach wird nach dem Serve des ausgewählten Produkts weitergeleitet, wenn Serve Aktion ausgeführt wird,dann wird das preis des Produkts vom Guthaben abgezogen.
8. Proc VendingMachine = VM(0);
   VM(credit: Int)=sum coin:Coin.(credit<200)
   ->accept(coin).VM(credit+value(coin)). Es wird immer vor dem accept Vorgang überprüft, ob das Guthaben kleiner als 200 Cent ist.

9. ReturnChange(credit : Int) = (credit>=value(Euro))->
   return(Euro).ReturnChange(credit-value(Euro))<>
   (credit>=value(_50c)) -> return(_50c).ReturnChange(credit-

value(_50c)) <>(credit>=value(_20c)) ->
return(_20c).ReturnChange(credit-value(_20c))
<>(credit>=value(_10c)) -> return(_10c).ReturnChange(credit-
value(_10c)) <>(credit>=value(_5c)) ->
return(_5c).ReturnChange(credit-value(_5c)) <> VM(credit);

Es wird erst vor jedem return Vorgang überprüft, ob das Guthaben größer als oder gleich den ausgewählten Wert ist, wenn passt, dann wird den ausgewählten Wert vom Guthaben abgezogen.Die Vorgänge beginnen auch hier in einer Reihenfolgen mit einem Wert von Groß nach klein,und durch <> kann nur einen Vorgang durchgeführt werden.