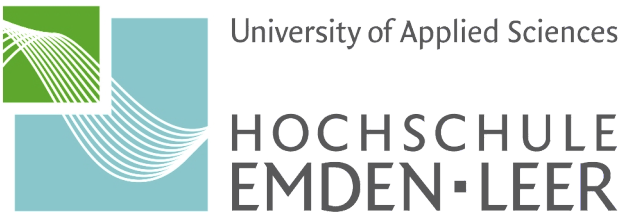
Parallele und verteilte Systeme

Spezifikationen in mCRL2



Gruppenmitglieder: Johannes Theiner
Semester: Sommersemester 2020
letzte Änderung: 20. Januar 2022

A. Sequenzielle Spezifikation

Problemstellung:

Spezifiziert werden soll eine Verkaufsmaschine die Tee, Kaffee, ein Stück Kuchen und Äpfel verkaufen kann. Akzeptiert werden 5, 10, 20, 50 Cent und 1 Euro Münzen. Das maximale Guthaben beträgt zwei Euro, etwaiges Rückgeld soll von groß nach klein zurückgegeben werden.

Lösung:

Zuerst werden die Datentypen *Coin* und *Product* definiert, sowie die zugehörigen Äquivalenzen(Auflistung 1 bzw. 2). Zusätzlich wird die Reihenfolge der Münzen definiert.

```
1
```

Auflistung 1: Spezifikation Münzen

```
1
```

Auflistung 2: Spezifikation Produkte

Definiert wird ein Prozess *VM* (Auflistung 3) der mehrere Möglichkeiten der Fortführung anbietet:

- Münzen akzeptieren und zum Guthaben hinzufügen, wenn dieses kleiner als 2€ ist (Zeile 3-5).
- Geld zurückgeben, wenn Guthaben vorhanden ist (Zeile 7-9).
- Sämtliche Produkte anbieten, die mit dem aktuellen Guthaben bezahlt werden können (Zeile 11-14).

```
1
```

Auflistung 3: Spezifikation Verkaufsmaschine

Zusätzlich wird ein Unterprozess *ReturnChange* (Auflistung 4) definiert, der das Rückgeld in definierter Reihenfolge zurückgibt in dem:

1. die größtmögliche Münze ermittelt wird (Zeile 2,3).
2. diese dem Kunden zurückgegeben wird (Zeile 5).
3. der Prozess wiederholt wird, solange noch Guthaben vorhanden ist (Zeile 6-9).

```
1
```

Auflistung 4: Spezifikation Geldrückgabe

B. Parallele Spezifikationen

B.1. Version 1

Problemstellung:

Spezifiziert werden soll ein Ampelsystem, welches 4 seperate Ampeln enthält. Hier sollen die einzelnen Ampeln unabhängig voneinander die verschiedenen Farbzustände (rot, grün, gelb) durchlaufen.

Lösung:

Definiert werden zuerst die Datentypen *CardinalDirection* und *Colour*, die Reihenfolge der Farben sowie der Aktor *show* (Auflistung 5).

```
1
```

Auflistung 5: Spezifikation Datentypen Ampelsystem

Nachfolgend werden die Prozesse *TF* und *TrafficLight*, welcher für einen einfacheren Aufruf von *TF* sorgt, spezifiziert.

Da hier die Farben nur in einer Endlosschleife durchlaufen werden sollen wird zeigt der show Aktor die aktuelle Konfiguration und darauf folgend wird der Prozesses erneut mit der nächstfolgenden Farbe gestartet (Auflistung 6; Zeile 5,6).

```
1
```

Auflistung 6: Spezifikation unabhängiger Prozesse

B.2. Version 2

Problemstellung:

Zusätzlich zur vorherigen Version soll hier um einen weiteren Prozess *Monitor* erweitert werden, der unsichere Zustände erkennt, eine Warnung ausgibt und das System anhält.

Lösung:

Version 1 wird mit einem neuen Prozess *Monitor* erweitert, der Zustand des Systems in einer Key-Value Map speichert.

Dazu kommunizieren die *TF* Prozesse mit dem Monitor Prozess über die Aktoren *show* und *seeColour* (Auflistung 7; Zeile 7).

```
1
```

Auflistung 7: Spezifikation Kommunikation

Zuerst prüft der *Monitor* Prozess ob der aktuelle Status zulässig ist (Auflistung 8; Zeile 2) und gibt eine Fehlermeldung aus (Zeile 3 & 4), oder beobachet und speichert sämtliche Änderungen.

```
1
```

Auflistung 8: Spezifikation Monitor

Die Überprüfung ob der Zustand sicher ist wird in das Mapping *unsafe* ausgelagert welches in Auflistung 9 definiert ist.

```
1
```

Auflistung 9: Spezifikation unsafe

Zur besseren Umsetzung dieser Überprüfung[1] werden die beiden Mappings *nextDirection* und *oppositeDirection* definiert (Auflistung 10).

```
1
```

Auflistung 10: Spezifikation nächste- & Gegenrichtung

B.3. Version 3

Problemstellung:

In dieser Version soll der *Monitor* nicht mehr nur unsichere Zustände erkennen, sondern auch verhindern.

Lösung:

Dazu wird die Bedingung verlegt um die Überprüfung durchzuführen bevor *seeColour* (Auflistung 11; Zeile 4) aufgerufen wird um die Synchronisierung zu unterbinden.

```
1
```

Auflistung 11: Spezifikation Monitor

[1] hauptsächlich um in Version 3 dies einfacher nutzen zu können

Zusätzlich wird *unsafe* um den Parameter *colour* erweitert, damit sämtliche Kombinationen überprüft werden können (Auflistung 12).

```
1
```

Auflistung 12: Spezifikation unsafe

B.4. Version 4

Problemstellung:

Nun soll die zentrale Instanz *Monitor* entfernt werden und die Ampeln sich untereinander synchronisieren um unsichere Zustände zu vermeiden.

Lösung:

Zuerst wird der Ablauf eines Richtungswechsels definiert (Auflistung 13). Dazu wird zuerst die nachfolgende Richtung freigegeben und darauf folgend pausiert bis eine Freigabe für den eigenen Prozess erteilt wurde.

```
1
```

Auflistung 13: Spezifikation Richtungswechsel

Der Wechsel zur nächsten Farbe wird durchgeführt sobald beide aktive Seiten diesen autorisieren (siehe Auflistung 16; Zeile 15).

```
1
```

Auflistung 14: Spezifikation Farbwechsel

Bei jedem Durchlauf des *TL* Prozesses wird zuerst der aktuelle Zustand über *show* gezeigt und dann falls die Ampel nicht rot zeigt nur zur nächst folgenden Farbe gewechselt, solange auch die andere Seite dies möchte. Ist die Ampel rot wird zuerst gewartet bis beide Ampeln in diesem Zustand sind und dann ein Richtungswechsel ausgeführt.

Um diese Logik nicht beim ersten Start auszuführen (was dazu führen würde das alle vier Richtungen auf Grün umschalten könnten) wird über den Aktor *changeDirection* und die dazugehörige Synchronisation ein beschränkter Bereich definiert welcher zu Beginn für Norden und Süden freigegeben wird.

```
1
```

Auflistung 15: Spezifikation Hauptprozess

```
1
```

Auflistung 16: Spezifikation Kommunikation