AUFGABE 7: ABSTRAKTE INTERPRETATION

In dieser Aufgabe werden Sie zunächst ein digitales α-β-Filter, einen Sensorstub und eine Datenstruktur implementieren. Die Korrektheit aller Komponenten soll anschließend mit Astrée nachgewiesen werden.

Achtung: Die zu implemntierende Datenstruktur unterscheidet sich nach grundlegenden und erweiterten Übung. Es gibt also einen gemeinsamen Übungsteil und zwei sich gegenseitig ausschließende Aufgabenstellungen!

Wenden Sie auch in dieser Aufgabe wieder die *objektbasierte Entwurfsmethode* an und verpacken Sie konzeptionell unabhängige Elemente in separate *Module*.

Hinweis: Es bietet sich an, Arbeitspakete für die einzelnen Gruppenteilnehmer zu vereinbaren. So kann beispielsweise ein Gruppenteilnehmer das Filter implementieren, während ein anderer schon die Datenstruktur erstellt.

1 Gemeinsame Übung

1. Einführende Frage: Was ist der Unterschied zwischen den Astrée-Konzepten der *Known Facts* und der *Assertions*?

.....
.....

Wann sollte welches dieser beiden Konzepte zum Einsatz kommen?

.....
.....

2. Sensor-Stub: Implementieren Sie einen Stub für einen Sensor. Dieser soll bei jedem Aufruf einen simulierten Messwert vom Datentyp `float` liefern. Sorgen Sie dabei dafür, dass für Astrée klar ist, dass es von den auf den Übungsfolien angegebenen einschränkenden Annahmen bezüglich des konkreten Messwerts ausgehen darf.

☞ volatile

Wenn Sie möchten, können Sie hier die in der Datei `sensor.dat` enthaltenen Messwerte verwenden um Ihre Implementierung später testen zu können. *Welche Teile Ihrer Implementierung sollte Astrée sehen? Wieso?*

☞ fscanf(3p)

.....
.....

3. Filter: Beantworten Sie zunächst die in diesem Abschnitt gestellten Fragen und notieren Sie sich die Antworten. Anschließend sollen Sie das in der Tafelübung vorgestellte α-β-Filter für den Datentyp `float` implementieren und dessen Korrektheit in Astrée nachweisen. *Welche Nachbedingung muss für die Filterinitialisierung gelten, damit das Filter stabil ist?*

.....
.....
.....

Für welche Werte von λ sind diese Nachbedingungen erfüllt?

.....
.....

Welche Vorbedingungen ergeben sich dadurch für die Filterinitialisierung in Bezug auf die Prozessvarianz, Rauschvarianz und das Abtastintervall?

.....
.....

Annotieren Sie sowohl die Nachbedingungen als auch die Vorbedingungen der Filterinitialisierung für Astrée, sobald Sie die Initialisierung implementiert haben! *Was kann in jedem Filterschritt schiefgehen?*

.....
.....

Wie können Sie erkennen, dass das Filter in einen unbrauchbaren Zustand übergeht?

.....
.....

Treffen Sie bei der Implementierung Maßnahmen, die das Filter in kontrollierter Art und Weise wieder in einen brauchbaren Zustand überführen!

Hinweis: Die Berechnung der *Filterparameter* für den *eingeschwungenen Zustand* sollten Sie mit Hilfe des Datentyps `double` vornehmen. Die Berechnung des Filterschritts und der Parameter für den transienten Filterzustand soll jedoch *auf jeden Fall* mit dem Datentyp `float` erfolgen. Dieses Vorgehen wäre auch in einem industriellen Projekt sinnvoll. *Wieso würde man hier nicht einfach die gesamte Implementierung mit dem Datentyp* `double` *vornehmen?*

.....
.....

Implementieren Sie zunächst das Filter für den eingeschwungenen Zustand, annotieren Sie die Filterinitialisierung und weisen Sie mit Hilfe von Astrée die Korrektheit dieser Initialisierung nach! Fangen Sie nun die Fälle ab, in denen das Filter im laufenden Betrieb in einen unbrauchbaren Zustand übergehen würde und implementieren Sie das in den Übungsfolien beschriebene Erholungsverfahren für diesen Fall. Auch während der Erholungsphase müssen die Filterparameter α_n und β_n sich im stabilen Bereich befinden. Weisen Sie dies mit Hilfe von Astrée nach.

☞ src/ab_filter.c

Achten Sie darauf, dass Sie die Korrektheit ihrer Implementierung nicht nur für bestimmte Filterparametersätze nachweisen, sondern für den gesamten Definitionsbereich! Diese Teilaufgabe gilt als erfolgreich bearbeitet, wenn alle Vor- und Nachbedingungen annotiert sind und Astrée keine Fehler mehr meldet.

Grundlegende Übungen

4. Ringpuffer: Moderne Mikrocontroller stellen häufig eine DMA-Einheit zur Verfügung, die Peripheriedaten *asynchron* zum Prozessor beschaffen kann. Deswegen sollen in dieser Aufgabe die von der Sensorhardware gelieferten Daten in Schüben verarbeitet werden. Implementieren Sie hierzu einen *Ringpuffer*, der diese Datenschübe aufnimmmt und weisen Sie dessen Korrektheit in Astrée nach. Pro Verarbeitungsrunde soll jeweils ein Messwert beim Sensor-Stub abgefragt und in den Ringpuffer geschrieben werden.

☞ src/main.c

5. Integration Fügen Sie die bisher entwickelten Teilmodule zu einem Progamm zusammen. Ihr Ringpuffer soll zunächst komplett mit Sensorwerten aus einem ihrer Sensorstubs gefüllt werden. Der Filter soll immer auf allen im Ringpuffer enthaltenen Werten ausgeführt werden. Sind alle Werte gefiltert, wird der Ringpuffer erneut befüllt und die Abarbeitung beginnt von vorne. Sie können sich das Ergebnis Ihrer Filterung ausgeben lassen und beispielsweise mit `gnuplot` oder `qtiplot` visualisieren.

☞ src/main.c

Erweiterte Übungen

6. Warteschlange für Arbeitsaufträge: In dieser Aufgabe sollen Sie wiederum Ihre Warteschlangenimplementierung aus Aufgabe 5 wiederverwenden. Kopieren Sie diese hierfür in die dafür vorgesehenen Verzeichnisse unserer Vorgabe. In dieser Aufgabe soll die Warteschlange keine Ganzzahlen mehr, sondern Arbeitsaufträge verwalten. Passen Sie die Warteschlange hierzu so an, dass sie Zeiger auf Funktionen verwaltet, die keine Argumente übernehmen und keinen Rückgabewert haben.

☞ typedef Job

☞ include/queue_*element.h*

Leider ist Astrée nicht in der Lage mit `malloc` und `free` umzugehen. Deswegen sollen Sie einen Speicherallokator für die Elemente Ihrer Warteschlange implementieren. Dieser soll Zeiger auf freie Elemente eines global angelegten Speicherpools zurückliefern. Welche Elemente frei sind, soll hierbei in einer Bitmaske vermerkt werden. Ihre Warteschlange soll in der Lage sein 512 Prioritätsstufen zu verwalten.

☞ `src/queue_alloc.c`
☞ `include/queue_alloc.h`

Implementieren Sie ausserdem eine Funktion, die Elemente wieder als für die Allokation verfügbar markiert. Entwerfen Sie auch für den Allokator Verträge und zeigen Sie die Korrektheit von Allokator und Warteschlange mit Hilfe von Astrée.

Je nach Implementierung Ihrer Warteschlange benötigen sie eventuell auch Allokatoren für Iteratoren und die Warteschlangen. Diese sind jedoch mit dem beschriebenen Muster und einer modularen Implementierung des ersten Allokators leicht zu entwickeln. Da wir in dieser Aufgabe nur jeweils eines dieser Objekte verwenden müssen, können Sie diese auch statisch anlegen und auf das Entwickeln weiterer Allokatoren verzichten.

Welche Verträge (Vorbedingungen, Nachbedingungen, Schleifeninvarianten) sind für Ihre Warteschlangenimplementierung und den Allokator sinnvoll? Geben Sie hierbei für jede Funktion, die Teil der öffentlichen Schnittstelle ist mindestens eine nicht-triviale Vor- und Nachbedingung an, sowie für jede Schleife in Ihrer Implementierung mindestens eine nicht-triviale Invariante. Annotieren Sie diese Verträge *anschliessend* in Ihrem Quellcode und weisen Sie die Korrektheit mit Hilfe von Astrée nach!

7. Integration: Setzen Sie nun die einzelnen Bestandteile Ihres Programms zu einem Gesamtprogramm zusammen. Mehrere Sensoren sollen Messwerte liefern, die von Filtern verarbeitet und schliesslich in eine Datei geschrieben werden. Die Arbeitsaufträge, die diese Aktionen verrichten, sollen hierbei in der Initialisierungsphase erstellt, in eine Warteschlange eingereiht und dann in einer Schleife immer wieder – gemäß ihrer Priorität – abgearbeitet werden. Um das Ergebnis zu visualisieren, können Sie z. B. `gnuplot` oder `qtiplot` benutzen. *Welche Annotationen sind für das Gesamtprogramm sinnvoll?* Weisen Sie die Korrektheit des Gesamtprogramms mit Hilfe von Astrée nach!

☞ `src/main_ext.c`

Hinweise

- Bearbeitung: Gruppe mit je zwei bis drei Teilnehmern.
- Abgabezeit: die Woche vom 02.06.2015 bis zum 08.06.2015
- Fragen bitte an `i4ezs@lists.cs.fau.de`