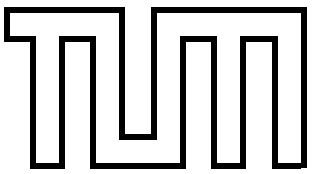


Technische Universität München

Fakultät für Informatik
Forschungs- und Lehreinheit Informatik VI

Übung zur Vorlesung Echtzeitsysteme

Aufgabe 4 – Esterel

Nadine Keddis
keddis@fortiss.org

Dominik Sojer
sojer@in.tum.de

Stephan Sommer
sommerst@in.tum.de

Michael Geisinger
geisinge@in.tum.de

Wintersemester 2011/12

Aufgabe 4: Esterel

In der Vorlesung haben Sie die Programmiersprache *Esterel* kennen gelernt. Das 5. Übungsblatt vertieft dieses Wissen anhand kleiner Aufgaben. Für das Bearbeiten der Aufgaben muss die Syntax von Esterel verwendet werden. Sowohl eine Kurzreferenz als auch eine vollständige Sprachbeschreibung zu Esterel finden Sie im Ordner `Uebung05` auf dem Laufwerk Q:.

Allgemeines

Für die Übungsaufgaben wird der frei verfügbare Esterel Compiler verwendet, der in einer virtuellen Maschine bereits auf den Rechnern installiert ist. Sie können den freien Compiler von der Webseite `http://www-sop.inria.fr/esterel.org/files/Html/Downloads/Downloads.htm` beziehen. Neben dem frei verfügbaren Compiler findet die Esterel-Sprache ebenfalls Verwendung in *Esterel Studio* (Produkt nicht mehr verfügbar) und in *Scade* von Esterel Technologies. *Scade* wird z.B. in der Luftfahrtindustrie zur Komponentenentwicklung (Airbus A380) eingesetzt.

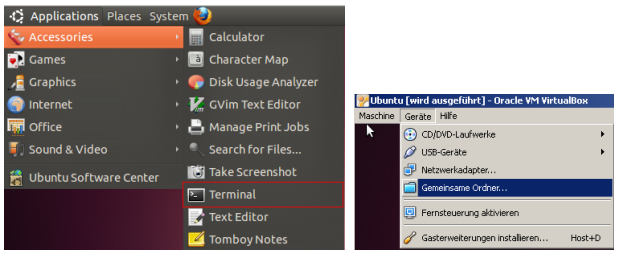
Einrichten der Entwicklungsumgebung

Bevor Sie mit der Aufgabe beginnen können, müssen Sie die Entwicklungsumgebung vorbereiten bzw. starten. Gehen Sie dazu wie folgt vor:

- Starten Sie dazu VirtualBox (*Start* → *Programme* → *Oracle VM VirtualBox* → *VirtualBox*).
- Starten Sie über das Menü *Datei Applianceimportieren* den Importvorgang und wählen Sie im Wizzard die Ubuntu VM *Ubuntu.ovf* aus (aus C:\Esterel). Sie können dann den Importvorgang abschliessen. Nach 4-5 Minuten steht Ihnen Ihre eigene VM inklusive der Esterel Entwicklungstools zur Verfügung.
- Ändern Sie in den Eigenschaften der VM den zugewiesenen Arbeitsspeicher auf 512MB.
- Starten Sie nun die VM und melden Sie sich mit dem Benutzer *Esterel* und dem Passwort *Esterel* an.
- Starten Sie nun eine Console / ein Terminal (siehe Abbildung 1(a)) und wechseln Sie in das Verzeichnis $\$home/esterel/examples/parkhaus$.
- Um Daten zwischen der virtuellen Maschine und Ihrem Host auszutauschen, gehen Sie wie folgt vor:
  - Öffnen Sie den Dialog *Geräte* → *Gemeinsame Ordner* (Abbildung 1(b)).

– Erstellen Sie einen neuen gemeinsamen Ordner, wählen Sie dabei Ihr Netzlaufwerk als Quelle / Ziel aus.

– Mounten Sie in der virtuellen Maschine den Ordner mit *sudo mount -t vboxsf "GEMEINSAMER_ORDNER" /mnt/.*



(a) *Öffnen eines Terminals* (b) *Gemeinsamen Ordner erstellen*

Abbildung 1: Virtual Box

Hinweise

Beachten Sie bitte folgende Hinweise beim Bearbeiten der Aufgaben:

- Das Rahmenprogramm für die Aufgabe befindet sich in der Datei *parkhaus.strl*. Sie können diese Datei mit einem Editor Ihrer Wahl bearbeiten.
- Beachten Sie bei der Implementierung der Aufgaben, dass der freie Compiler, im Gegensatz zur kommerziellen Version, keine Unterstützung für mehrere Module bietet.
- Zum Übersetzen des Programms könne Sie den Aufruf *make parkhaus.xes* verwenden. Wie bereits von anderen Programmiersprachen bekannt, erhalten Sie vom Compiler Hinweise auf mögliche Fehler.
- Sie können Ihr Programm testen, in dem Sie die Simulation mit *./parkhaus.exe* starten. In der Simulation wird der Zustand der Eingangs- und Ausgangssignale angezeigt. Bei *puren* Signalen bedeutet blau, dass das Signal nicht anliegt und rot, dass das Signal anliegt. Da die Simulation keine verbindung mit einem realen Aufbau hat, müssen Sie dem Simulator die Sensorwerte zum geeigneten Zeitpunkt zur Verfügung stellen.
- Über den Button *Tick* können Sie das Programm um einen Schritt weiterlaufen lassen. Im Quelltext-Fenster (Teil der Simulation) wird die aktuelle Position im Programm hervorgehoben. Durch Verändern der Eingangssignale können Sie den Ablauf Ihrer Lösung beeinflussen.

Aufgabe:

In dieser Übungsaufgabe soll mittels der Esterel-Sprache die Logik für ein Parkhaus mit 20 Stellplätzen implementiert werden (siehe Abbildung 2). Da die Zufahrt zum Parkhaus nur einspurig ist, muss dieser Bereich durch eine Ampel abgesichert werden. Dabei gilt, dass im Initialzustand beide Ampeln rot zeigen.

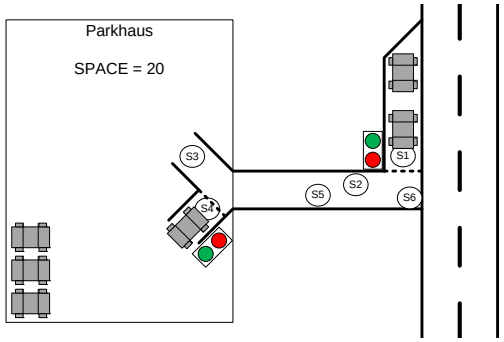


Abbildung 2: *Schematische Darstellung des Parkhauses*

Wenn ein Auto ins Parkhaus fahren will, muss es vor der Ampel warten. Über einen Sensor (*Entry_car_is_waiting*, S1) wird dem Programm die Anwesenheit eines Fahrzeugs mitgeteilt. Wenn die Zufahrt frei und mindestens ein Platz im Parkhaus frei ist, wechselt die Ampel auf grün. Dann darf genau ein Fahrzeug passieren. Das Einfahrende Fahrzeug wird über einen Sensor (*Entry_car_is_going_in*, S2) erkannt, die Ampel wechselt wieder auf rot. Sobald das Fahzeug im Parkhaus angekommen ist, wird dies ebenfalls über einen Sensor detektiert (*Entry_car_is_in*$_P G$, S3). Das Verlassen des Parkhauses funktioniert äquivalent. Die relevanten Sensoren sind dabei *Exit_car_is_waiting* (S4), *Exit_car_is_leaving* (S5), *Exit_car_left_PG* (S6).

a) Machen Sie sich mit dem Rahmenprogramm vertraut und ergänzen Sie die Implementierung für die Ampel bei der Ausfahrt des Parkhauses analog zu der vorhandenen Implementierung für die Ampel bei der Einfahrt.

b) Falls gleichzeitig ein Fahrzeug das Parkhaus verlassen und ein weiteres ins Parkhaus einfahren will muss sichergestellt werden, dass das Fahrzeug, dass das Parkhaus verlassen will bevorzugt wird. Ist dies bei der aktuellen Implementierung der Fall?

c) Erweitern Sie Ihr Programm so, dass beim Ein-/bzw. Ausfahren aus dem Parkhaus sichergestellt ist, dass die Ampel wieder auf rot wechselt, wenn der Sensor *Entry_car_is_going_in* bzw. *Exit_car_is_leaving* nach 10 Ticks nicht auslöst.

d) **Optional:** Erweitern Sie Ihr Programm um einen weiteren Parallelen Strang, der ein einfahrendes Auto implementiert. Überlegen Sie sich, an welchen Stellen Sie das Rahmenprogramm abändern bzw. erweitern müssen.