IL2212 EMBEDDED SOFTWARE

Laboratory 1

System Modelling with ForSyDe: Data Flow and Synchronous Models of Computation

VERSION 3.0 (20250114)

1 Objectives, Requirements, and Bonus Points

1.1 Objectives

The laboratory shall introduce the students to system modelling at a high level of abstraction using the ForSyDe modelling environment. It covers several data flow models of computation and the synchronous model of computation. The laboratory consists of theoretical and practical tasks, which shall be solved and submitted individually by the students via the Canvas page of the course.

1.2 Requirements for Passing the Laboratory

To pass the laboratory, the student must solve and submit all *mandatory tasks* before the deadline. The Canvas page of the course states the deadline and submission instructions. During the laboratory session, the student has to demonstrate and convincingly explain all solutions to the course staff.

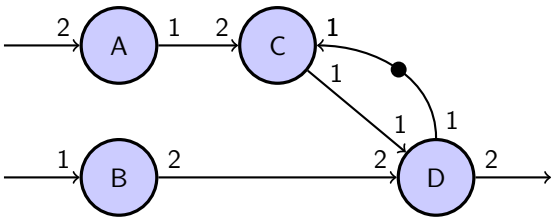
1.3 Bonus Points

Students, who, in addition to the mandatory tasks, solve *optional tasks* before the deadline, can receive bonus points. The bonus points will count towards the total points in the exam. During the laboratory session, the student has to demonstrate and explain all solutions to the course staff.

2 Laboratory Tasks

2.1 Data Flow Model of Computation

1. (**Mandatory**) Given is the following synchronous data flow graph.



   (a) Give a mathematical representation of the SDF graph, and use this representation to derive a *periodic admissible sequential schedule* with the minimal buffer requirements, where you show each step of your calculation. What are the required buffer sizes for this schedule?

   (b) What is the maximum throughput for a single processor, defined as output tokens per time unit, given the following parameters:

   - the execution times for the data flow actors are $C_A = 25$, $C_B = 60$, $C_C = 40$, and $C_D = 50$, and the
   - communication time for sending tokens and receiving can be neglected.

   (c) Give also the necessary input data rate $R_{\text{single},i}$, as tokens per time unit on input arc i, on both input arcs to be able to achieve the maximum throughput for both sub-tasks. What happens, if the input rate is higher or lower than $R_{\text{single},i}$.

   (d) Give a *periodic admissible parallel schedule* for an architecture with two processors aiming to maximise the throughput. Use the same execution times as above and neglect also the communication time for sending and receiving tokens.

   (e) Which throughput can be achieved with this schedule for the two processor architecture? Give also the necessary input data rates $R_{\text{double},i}$ on both input arcs.

2. (**Mandatory**) The following ForSyDe SDF-model uses the same SDF-graph structure as in task 1.

```
module SDF_Application where

import ForSyDe.Shallow

system :: Signal Int -> Signal Int -> Signal Int
system s_in1 s_in2 = s_out where
  s_1 = actor_a s_in1
  s_2 = actor_b s_in2
  s_3 = actor_c s_1 s_4_delayed
  (s_4, s_out) = actor_d s_2 s_3
  s_4_delayed = delaySDF [0] s_4

actor_a :: Signal Int -> Signal Int
actor_a = actor11SDF 2 1 f_1 where
  f_1 [x, y] = [x + y]

actor_b :: Signal Int -> Signal Int
actor_b = actor11SDF 1 2 f_2 where
  f_2 [x] = [x, x+1]

```

```haskell
actor_c :: Signal Int -> Signal Int -> Signal Int
actor_c = actor21SDF (2,1) 1 f_3 where
  f_3 [x, y] [z] = [x + y + z]

actor_d :: Signal Int -> Signal Int -> (Signal Int, Signal Int)
actor_d = actor22SDF (2,1) (1,2) f_4 where
  f_4 [x, y] [z] = ([x + y + z], [x + y, x + y + z])
```

The objective of this task is to get a deeper understanding of ForSyDe SDF-models and their simulation by executing the model with potentially infinite signals and by observing intermediate signals. Study carefully Appendix G of the lecture notes, which also explains the functions `infiniteS` and `takeS`, which are used to define and simulate infinite signals.

(a) (*no written answer required for this sub-task*) Carefully study the SDF model, so that you are able to explain its functionality during the laboratory session.

(b) Define two *infinite* signals `i_1` and `i_2` using the function `infiniteS`, where

- the signal `i_1` shall produce the sequence `{0,1,2,3,4,5,6,7,8,9,...}`, and
- the signal `i_2` shall produce the sequence `{0,10,20,30,40,50,60,70,80,90,...}`.

Use these signals as input signals to simulate the system model and give the first ten values of the output signal.

(c) Change the SDF-model, so that you can observe the intermediate signals `s_1`, `s_2`, `s_3`, `s_4`, and `s_4_delayed`. Give all *finite* intermediate signals for the following *finite* input signals

- `i_1 = {0,1,2,3,4,5,6,7,8,9}`, and
- `i_2 = {0,10,20,30,40,50,60,70,80,90}`.

Simulating the ForSyDe model without a schedule?

An important part of task 1 has been to define a schedule to be able to run the SDF-model on a processor. However, in this task no schedule has been given to the ForSyDe model and still the model can be executed and simulated correctly. Why is this not needed for the ForSyDe simulation in Haskell? How is this possible?

A simplified answer is that the execution semantics of Haskell is data-driven. The schedule is calculated dynamically by the need of data.

This gives a lot of benefits for system modelling, since no schedule has to be derived and only the functionality of the model has to be specified. Once, the structure and the functionality of the model has been defined and validated, the methods used in task 1 can be used to find a schedule and the required buffer sizes for an implementation of the model.

3. (**Mandatory**) Synchronous data flow (SDF) has a lower expressiveness than cyclo-static data flow (CSDF).

   (a) Implement the tutorial CSDF application described in Section G.3 and in Listing G.2 of the lecture notes with only SDF actors using the **`ForSyDe.Shallow`** library. Try to be as close as possible to the original CSDF model regarding the implemented functionality.

   (b) Is it possible to achieve exactly the same functionality as in the original CSDF model? Please give a convincing explanation or a counter-example.

4. (**Mandatory**) Implement the same functionality of the scenario-aware data flow application described in Section G.4 and in Listing G.3 of the lecture notes in **`ForSyDe.Shallow`** only using one detector and one kernel, but no SDF actors.

5. (**Mandatory**) Section 5.2.4 in the lecture notes describes the Boolean data flow (BDF) model of computation (MoC).

   (a) Implement the BDF actors SWITCH and SELECT using the scenario-aware MoC in **`ForSyDe.Shallow`**.

   (b) Implement the BDF graph in the dashed box of Figure 5.20 in **`ForSyDe.Shallow`**. The actors C and D shall implement the identity function (C) and negation function (D). The actors A and B shall be replaced by input signals s_a and s_b, and the actor E by an output signal s_e.

6. (**Optional, 2 Bonus Points**) Implement a scenario-aware data flow application of an adaptive system that uses one detector and one kernel using **`ForSyDe.Shallow`**.

   The kernel shall execute the following operation:

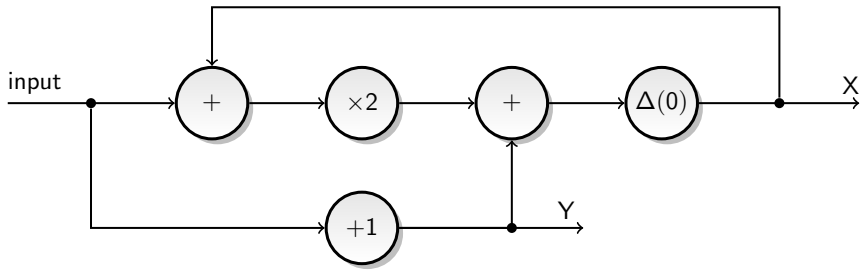
   - In *slow operation* the kernel shall consume 3 input tokens and produce one output token during each firing, which value is the sum of the consumed tokens.
   - In *fast operation* the kernel shall consume 2 input tokens and output one output token during each firing, which value is the sum of the consumed tokens.
   - The kernel shall start in *slow operation* mode.
   - The kernel shall switch from *slow operation* mode to *fast operation* mode, when the sum of the consumed tokens is larger than 20.
   - The kernel shall switch from *fast operation* mode to *slow operation* mode, when the sum of the consumed tokens is less than 10.

   The following simulation example shows the desired functionality.

```
*SADF_Adapter> s_test = signal [4,5,6,8,8,9,9,8,2,4,8,5,2]
*SADF_Adapter> system s_test
{15,25,17,6,15}
```

2.2 Synchronous Model of Computation

7. (**Mandatory**) Given is the following synchronous process network. All processes are synchronous processes, which execute according to the perfect synchrony hypothesis that also forms the base for the synchronous languages.



All processes execute simple mathematical functions except the process $\Delta(0)$. The process $\Delta(0)$ delays an input event one event cycle and has 0 as value for its first output event.

Given are the values of the following sequence of the first three input events: 1, 2, 3.

(a) Is it possible to determine the output sequence with the given information? Motivate!

(b) If it is possible, give also the output sequence. Show also the order relation between the individual output events on X and Y.

(c) Explain the terms *tag*, *value*, *event*, *signal*, and *process* as defined in the *tagged signal model*.

(d) What relation can you establish between the tags of the signals `input`, `X`, and `Y`.

8. (**Mandatory**) Implement the synchronous process network of task 7 in ForSyDe. Create the five processes using ForSyDe process constructors and construct the process network. Simulate its functionality with the input sequence $\{1, 2, 3\}$.

9. (**Mandatory**) Consider the following specification of a vending machine. The vending machine receives coins of the values 5 SEK and 10 SEK. The machine returns a bottle, when the right amount (10 SEK) is inserted. If too much money is inserted, a bottle and a coin (5 SEK) shall be returned. The machine has only a single coin injection slot, so it is not possible to inject two coins at the same time.

Use the **`ForSyDe.Shallow`** library to create the following models for the control system of the vending machine. In all sub-tasks, the synchronous model of computation shall be used.

(a) Create a model that has the following type:

```
vendingMachine :: Signal Bool -- Signal of 5 SEK coins
               -> Signal Bool -- Signal of 10 SEK coins
               -> Signal (Bool, Bool) -- Signal of
                                      -- (Bottle, Return)
```

The system shall have the following behaviour during simulation.

```
*VendingMachine> s_coin5 = signal [False,True,True,
  True,False,False]
*VendingMachine> s_coin10 = signal [True,False,False,
  False,True,False]
```

```
*VendingMachine> vendingMachine s_coin5 s_coin10
{(True,False),(False,False),(True,False),(False,False),
  (True,True),(False,False)}
*VendingMachine> unzipSY $ vendingMachine s_coin5 s_coin10
({True,False,True,False,True,False},
 {False,False,False,False,True,False})
```

(b) Create a model that has the following type:

```
data Coin = C5 | C10 deriving (Show, Eq, Ord)
data Bottle = B deriving (Show, Eq, Ord)
data Return = R deriving (Show, Eq, Ord)

type Coin_Event = AbstExt Coin
type Bottle_Event = AbstExt Bottle
type Return_Event = AbstExt Return

vendingMachine :: Signal Coin_Event -- Signal of Coins
               -> Signal (Bottle_Event, Return_Event)
                  -- Signal of (Bottle, Return)
```

The system shall have the following behaviour during simulation.

```
*VendingMachine> s_coin = signal [Prst C10, Prst C5, Prst C5,
  Prst C5, Prst C10, Abst]
*VendingMachine> vendingMachine s_coin
{(B,_),(_,_),(B,_),(_,_),(B,R),(_,_)}
*VendingMachine> unzipSY $ vendingMachine s_coin
({B,_,B,_,B,_},{_,_,_,_,R,_})
```

10. (**Mandatory**) Implement a C-program that implements a vending machine corresponding to the ForSyDe model of task 9. Your program should read inputs from the terminal window and output the result in the terminal window. An example is given below:

```
Enter an input (10, 5, 0): 10
Output: Bottle
Enter an input (10, 5, 0): 5
Output: Nothing
Enter an input (10, 5, 0): 5
Output: Bottle
Enter an input (10, 5, 0): 5
Output: Nothing
Enter an input (10, 5, 0): 10
Output: Bottle, Return
Enter an input (10, 5, 0): 0
Output: Nothing
```