System-Level Design

2.01.334, Summer 2024

Exercise 2
Models of Computation

Assigned: April 15, 2024
Due: April 22+29, 2024

General Instructions

1. Please submit your solutions via mail to your advisors (mail addresses can be found at the end of this document). Submissions should include a single typewritten PDF file with the writeup and a single Zip or Tar archive for any supplementary files (e.g. source files, which has to be compilable by simply running `make` and should include a README file with instructions for running each model).
2. You may discuss the problems with your classmates but make sure to submit your own independent and individual solutions. You are allowed to work in groups with up to three members. In this case, please put the full names of all three group members on the cover page of you submitted solutions.
3. Some questions might not have a clearly correct or wrong answer. In general, try to write down your arguments and reasoning how you have arrived at your solutions.

Task 1: Non-determinism

a) What is non-determinism?

Determinism describes a system where the same inputs always lead to the same outputs. In a non-deterministic (NDT) system this is not the case. NDT is not to be confused with random behavior, since this can be described statistically.

b) How might non-determinism arise? (give one example not discussed in class)

Some examples for non-determinism are

- *Truly concurrent processes*
- *dynamic scheduling of tasks, reacting to the first task that succeeds, cancelling tasks that cannot lead to a result*

At `https://arxiv.org/abs/2210.15202` *a "A Survey on Parallelism and Determinism" can be found if more background on the discussion of this topic should be of interest.*

c) What are the advantages and disadvantages of having non-determinism in a language or model, i.e. in what circumstances might it be positive/desired or negative/undesired?

The disadvantages of a NDT system are that it is more difficult to analyze and verify. A deterministic system is more difficult to develop, since in the course of the development very strong restrictions must be made. This is often referred to as over-specification.

d) Is a SpecC specification model deterministic? If yes, why? If not, list possible sources of non-determinism and how they can be avoided (i.e. how a SpecC model can be made deterministic?).

A SpecC model is not deterministic because SpecC supports concurrency and NDT automata. If these are not used, the model is deterministic.

Task 2: Synchronous Dataflow (SDF) Synthesis

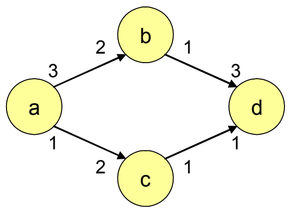


Figure 1: SDF graph

For the SDF graph in Figure :

a) Show that the graph is consistent and that it has a valid schedule.

First, a linear equation system is set up::
$3a = 2b$
$3d = b$
$2c = a$
$c = d$
This can be solved as:
$3(2c) = 2(3d)|c = d$
$6d = 6d$
$=> d = 1$
$=> b = 3, a = 2, c = 1$
The possible solution is not only trivial (i.e. $a = b = c = d = 0$*) and for this reason the SDF graph is consistent and has a valid schedule.*

b) List all possible minimal periodic static schedules.

All minimal periodic schedulings look like this:

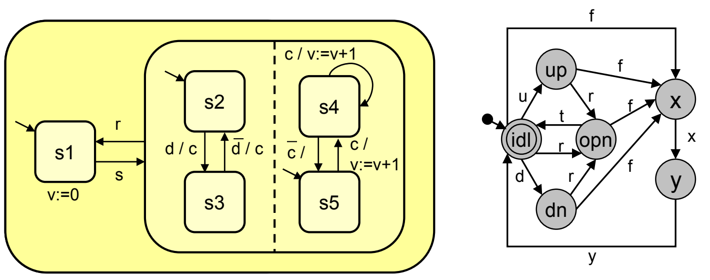
scheduling	*max. buffer a->b*
aabbbcd	6
aabbcbd	6
aabcbbd	6
aacbbbd	6
ababbcd	4
ababcbd	4
abacbbd	4

c) Find the periodic schedule with the lowest token buffer usage. What is the minimum buffer usage?

The solution is shown in the table above. The last three schedulings meet the requirement.

Task 3: State Machines

In class, we have discussed the concepts of extended state machines (FSMs with data), hierarchy (OR state) and concurrency (AND state) for managing complexities in FSMD and HCFSM models.



(a) HCFSMD example (b) HCFSM example

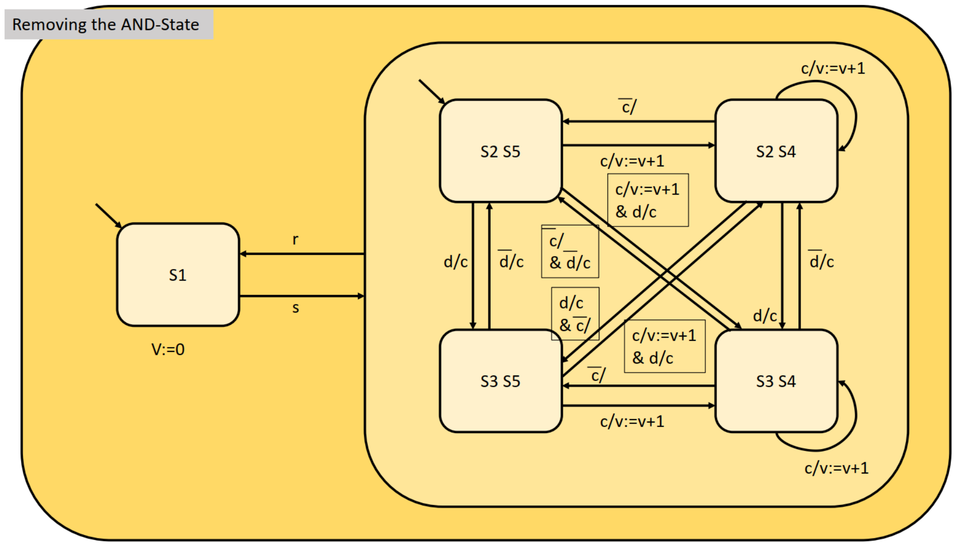
Figure 2: Finite State Machines

a) Convert the HCFSM(D) in Figure 2(a) into an equivalent FSMD. Note that the concurrent (AND) composition of states is communicating through signal c, and that the HCFSM has Mealy semantics. Transitions for unspecified input conditions default to remaining in the same state, where unspecified outputs default to absence of producing any event. Absence is otherwise indicated as negated conditions on signals. What functionality does the state machine actually perform?

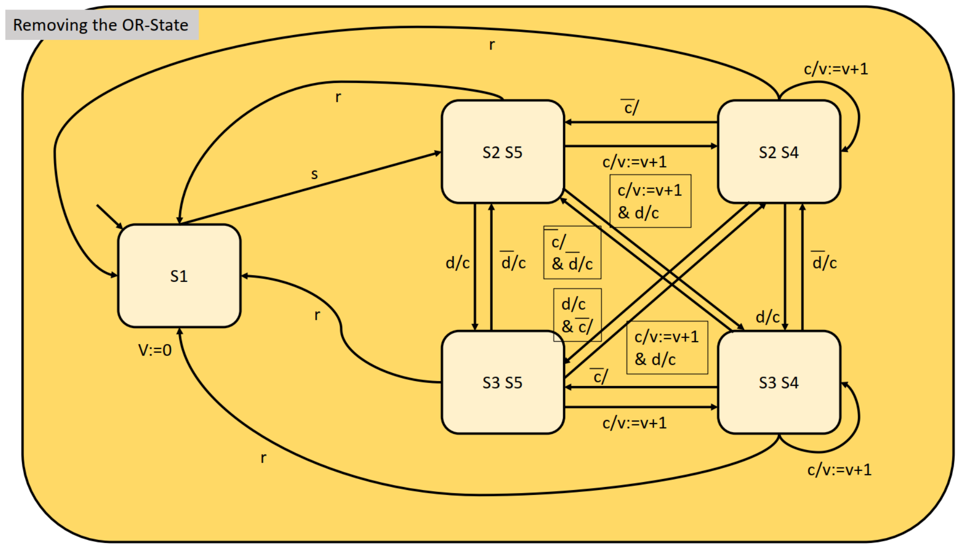
*For the solution of this task it is necessary to proceed according to the slide $3-37$. Here, first the concurrency is eliminated and then the hierarchy. With the elimination of the concurrency it is to be paid attention to the fact that some transition conditions are omitted. Figure **a)** depicts the step by step conversion of the HCFSM(D) in Figure 2(a) into an equivalent FSMD*

b) Try to reduce the complexity of the FSM in Figure 2(b) by converting it into a HCFSM using hierarchy (OR states) and/or concurrency (AND states) wherever possible. What can you generally say about the complexity (number of states and transitions) of an FSM as a function of the size of an equivalent HCFSM?

*In this case, too, we have to proceed according to the slide $3-37$. First the states up, idl, opn and dn are grouped to a superstate. On closer inspection, one recognizes that idl, up and dn also form a superstate in this case. Figure **b)** depicts a step by step reduction (from a to d) of the complexity of the FSM in Figure 2(b) by converting it into a HCFSM.*

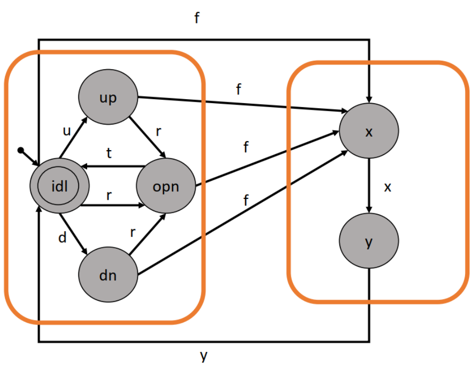


(a) removing the AND state

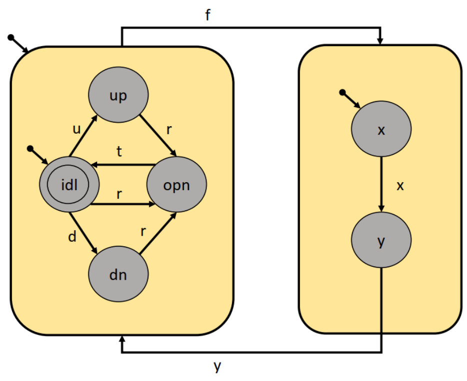


(b) removing the OR state

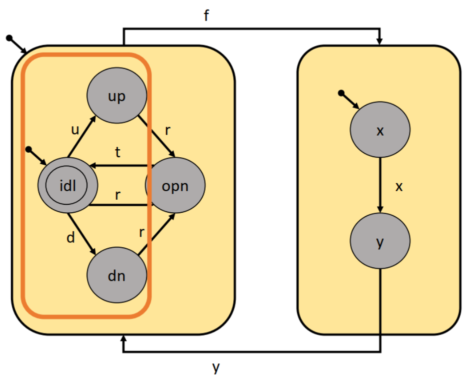
Figure 3: Step by step conversion of the HCFSM(D) in Figure 2(a) into an equivalent FSMD



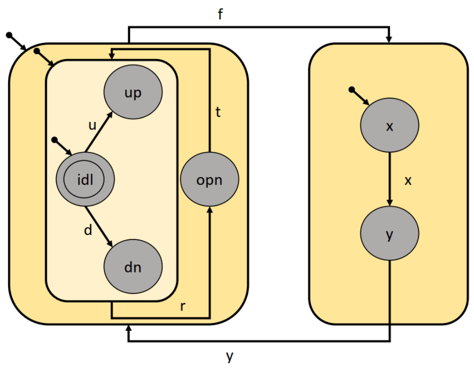
(a) identifying super states



(b) super states



(c) identifying super states



(d) super state

Figure 4: Step by step reduction (from a to d) of the complexity of the FSM in Figure 2(b) by converting it into a HCFSM

Task 4: Models of Computation and Languages
In class, we learned about the different Models of Computation (MoCs): KPN, SDF, FSM(D), HCFSM, PSM.

a) What is the relationship between MoCs and languages?

A model is described by a language. The language must be able to represent the different properties of the model. Example concurrency (HCFSM in SpecC or SystemC).

b) Can SpecC support all these MoCs? If so, briefly sketch for each MoC that you think can be supported how you would represent a corresponding model in SpecC.

model	*support?*	*language construct*
KPN	*yes (partially)*	*Process nodes are represented as parallel composed behaviors. Communication via Channel* `s_queue`*. Anyhow the system can deadlock on full queues or might required and unbounded amount of memory. If Park's algorithm should be applied it has to be added in an additional parallel behavior that knows when KPN nodes are blocked and which is capable of modifying the size of the Queue Channels.* *Another possibility is to calculate a static schedule of the SDF and execute this static schedule as and FSM. Communication between the state of the FSM (that represnet the actors) takes place though local variables. This solution has the disadvantage over above solution that all potential parallelismn is removed from the model.*
SDF	*yes*	*Process nodes are represented as parallel composed behaviors. Communication via Token Channels.*
FSM(D)	*yes*	*Though FSM composition of behaviors. Calculations (i.e. the data path extension) in process possible inside behaviors.*
HCFSM	*yes*	*Possible though a combination of parallel behavior composition that contains FSM behaviros as child behaviros (usage of behavior hierarchy).*
PSM	*yes*	*Basic model behind the bahaviors in SpecC and its composition.*

Task 5: Discrete-Event Semantics
For each of the following code examples, what is the value of `myB` printed at the end of execution and at what simulated time does the program terminate. You are free to run the code on top of the SpecC simulator and observe the program output, but you need to provide an explanation and reasoning of why the program is behaving as it is (e.g. sequence of events happening during simulation):

```
behavior A(int myB) {
  void main(void) {
    myB = 10;
  }
};


behavior B(int myB) {
  void main(void) {
    myB = 42;
  }
};


behavior Main(void) {
  int myB;

  A a(myB);
  B b(myB);

  int main(void) {
    par { a; b; }
    printf("%d", myB);
    return 0;
  }
};
```

Listing 1: **a)**

```
behavior A(int myB) {
  void main(void) {
    waitfor 42;
    myB = 10;
  }
};

behavior B(int myB) {
  void main(void) {
    myB = 42;
  }
};


behavior Main(void) {
  int myB;

  A a(myB);
  B b(myB);

  int main(void) {
    par { a; b; }
    printf("%d", myB);
    return 0;
  }
};
```

Listing 2: **b)**

```
behavior A(int myB) {
  void main(void) {
    myB = 10;
    waitfor 42;
  }
};

behavior B(int myB) {
  void main(void) {
    waitfor 10;
    myB = 42;
  }
};

behavior Main(void) {
  int myB;

  A a(myB);
  B b(myB);

  int main(void) {
    par { a; b; }
    printf("%d", myB);
    return 0;
  }
};
```

Listing 3: **c)**

```
behavior A(int myA, event e) {
  void main(void) {
    myA = 10;
    notify e;
    myA = 11;
    notify e;
    waitfor 10;
  }
};

behavior B(int myA, int myB, event e) {
  void main(void) {
    wait e;
    myB = myA;
  }
};

behavior Main(void) {
  int myA;
  int myB;
  event e;

  A a(myA, e);
  B b(myA, myB, e);

  int main(void) {
    par { a; b; }
    printf("%d", myB);
    return 0;
  }
};
```

Listing 4: **d)**

```
behavior A(int myA, event e) {
  void main(void) {
    myA = 10;
    notify e;
    waitfor 10;
    myA = 11;
    notify e;
  }
};

behavior B(int myA, int myB, event e) {
  void main(void) {
    wait e;
    myB = myA;
  }
};

behavior Main(void) {
  int myA;
  int myB;
  event e;

  A a(myA, e);
  B b(myA, myB, e);

  int main(void) {
    par { a; b; }
    printf("%d", myB);
    return 0;
  }
};
```

Listing 5: **e)**

a) *No statement can be made about which value is at the end in `myB`, because both behaviors are concurrent and no order is given. The simulator chooses either randomly (to simulate concurrency) or based on a predefined order. The simulation needs a time unit, because the behaviors are always executed to the end (wait-statements). (Run-to-completion)*

b) *Since A is paused for 42 time units, this behavior writes its value last and `myB` has the value 10 at the end. The simulation takes 43 time units (Current time units + waiting time).*

c) *In this case, it is the other way around and `myB` is 42. This simulation also takes 43 time units.*

d) *Behavior A describes the value `myA` twice and throws an event e twice. Behavior B reacts to this event and writes the value `myA` into `myB`. However, the assignments in Behavior A runs in one time unit, so `myA` is overwritten with 11. Since Behavior A and B are specifed to execute cuncurrently it is not defined if the assignment `myA=11` in Behavior A or the assignment `myB=myA` in Behavior B is executed first. The event thrown twice has no influence on the activation of Behavior B. For this reason the value of `myB` is undefined (could be 10 or 11). The simulation takes 11 time units.*
This non-determinism can be repaired if we apply a "run-to-completion" semantics in which Behavior A runs until the behavior is finished or until it reaches the next timed wait statement (`waitfor 10` in this case). Under this semantics the first notified event e is discarded/overwritten by the second notified event e. In this case the final value of `myB=11`.

e) *This example is very similar to the previous one with the difference that Behavior A is paused after the assignment of `myA` and the first event. Here Behavior B is activated and the value `myB` is assigned. Since Behavior B has now run, it is not started a second time and the assignment of `myA` and the further event remain unused. The final value of `myB` is 10 and the simulation also takes 11 time units.*

f) What code has to be inserted at the beginning of behavior B (line 13) in (e) to change the output of the program? Give two different options. What must not appear there for the program not to deadlock?

We can either insert at second `waite;` or a `waitfor(10);` in line 13 to change the output from `myB=10` to `myB=11`. Any wait statement larger than 10 time units will lead to a deadlock, since the notification of event e in Behavior A will be discarded after simulation time 10 has passed. Since no further event e is being notified by Behavior A, the `waite` will wait forever and thus deadlock.

g) Why do SpecC events have a semantic in which they can get lost? Under what condition do SpecC events get lost? What type of channel/communication could not be modeled if delivery would always be guaranteed?

Events are only valid in the current time slot (time unit in which is has been notified). If they are not reacted to in this time slot, they are lost. Without this behavior the modeling of asynchronous processes would not be possible.

Administrative

Advisors: Kim Grüttner <kim.gruettner@dlr.de>
Jörg Walter <joerg.walter@offis.de>
Henning Schlender <henning.schlender@dlr.de>
Sven Mehlhop <sven.mehlhop@offis.de>