1 Slicing SLIM Models

1.1 Algorithm in Pseudo-Code

```
 // *** Initialize sets collecting interesting objects ***
interestingDataElements = DataElements ∩ AtomicPropositions
weakInterestingDataElements = ∅
interestingEvents = ∅
interestingModes = Modes ∩ AtomicPropositions
if includeCycles
...then interestingModes ∪= { mode ∈ Modes | mode is on syntcatical cycle }

// *** Fixpoint Iteration ***
repeat
....// Transitions modifing interesting data elements (either by assignment or by
reactivation)
....interestingModes ∪= { transition.sourceMode | transition ∈ Transitions
.........................with (ex. effect in transition.effects: effect.writesTo
∈ interestingDataElements)
.........................or (transition.reactivates ∩ interestingDataElements
≠ ∅) }
....// Flows to interesting data ports
....for flow ∈ Flows with flow.targetPort ∈ interestingDataElements do:
........interestingDataElements ∪= flow.sourcePorts
........interestingModes ∪= flow.inModes ≠ ALL ? flow.inModes : ∅

....// Transitions into interesting modes
....for transition ∈ Transitions with transition.targetMode ∈ interestingModes:
........interestingModes ∪= { transition.sourceMode } // all predecessors
........interestingEvents ∪= { transition.trigger }
........weakInterestingDataElements ∪= transition.guard.readsFrom
........for effect ∈ transition.effects with effect.writesTo ∈ (interestingDataElements
∪ weakInterestingDataElements)
............weakInterestingDataElements ∪= effects.readsFrom
....// Transitions from interesting into still uninteresting moddes
....for transition ∈ Transitions with transition.sourceMode ∈ interestingModes (and
transition.targetMode ∉ interestingModes):
........interestingEvents ∪= { transition.trigger }
........weakInterestingDataElements ∪= transition.guard.readsFrom
........for effect ∈ transition.effects with effect.writesTo ∈ (interestingDataElements)
............weakInterestingDataElements ∪= effects.readsFrom

....// Each weak interesting in data port is automatically interesting
....interestingDataElements ∪= weakInterestingDataElements ∩ InDataPorts
....// Flows to weak interesting out data ports in interesting modes
....for flow ∈ Flows with (flow.targetPort ∈ weakInterestingDataElements)
.........................and (flow.inModes ∩ interestingModes ≠ ∅):
........interestingDataElements ∪= flow.sourcePorts

....// Connections to/from interesting event ports
```

```
....for eventPortCon in EventPortConnections with eventPortCon.sourcePort ∈ interestingEvents
.................................................or eventPortCon.targetPort ∈ interestingEvents
do:
........interestingEvents ∪= { eventPortCon.sourcePort, eventPortCon.targetPort }
........if eventPortCon.inModes ≠ ALL
............then interestingModes ∪= eventPortCon.inModes
....// Transitions with interesting triggers
....for transition ∈ Transitions with transition.trigger ∈ interestingEvents do
........interestingModes ∪= { transition.sourceMode }
until stable

// *** Generate Sliced Specification ***
slicedInDataPorts = InDataPorts ∩ interestingDataElements
slicedOutDataPorts = OutDataPorts ∩ ( interestingDataElements ∪ weakInterestingDataElements
)
slicedDataSubcomponents = DataSubcomponents ∩ ( interesting ∪ weakInterestingDataElements
)
slicedInEventPorts = InEventPorts ∩ interestingEvents
slicedOutEventPorts = OutEventPorts ∩ interestingEvents
slicedModes = (Modes ∩ interestingModes) ∪
..............{ „_SlicedSinkMode (for component)“ | component with component.modes
∉ interestingModes }
for transition ∈ Transitions with transition.sourceMode ∈ interestingModes:
....slicedTransition = copy(transition)
....if slicedTransition.targetMode ∉ interestingModes:
.......slicedTransition.targetMode = „_SlicedSinkMode“ (of resp. component)
....slicedTransition.effects = { effect ∈ transition.effects | effect.writesTo ∈
interestingDataElements or
........(effect.writesTo ∈ weakInterestingDataElements and transition.targetMode
∈ interestingModes) }
slicedFlows = { flows ∈ Flows | flow.targetPort ∈ interestingDataElements or
.................................. (flow.targetPort ∈ weakInterestingDataElements
and flow.inModes ∩ interestingModes ≠ ∅) }
slicedEventPortConnections = { eventPortCon ∈ EventPortConnections | eventPortCon.targetPort
∈ interestingEvents }
```

Our goal is to develop an algorithm for slicing SLIM models with respect to a given property, i.e. a list of atomic propositions over components data ports / subcomponents and nominal/error modes. Slicing should yield a smaller SLIM model which contains only those parts of the original model required by the given property. In other words, all „uninteresting“ parts have been deleted.

1.2 Data Ports

For a first approach to slicing, consider only data elements, i.e. data ports and data subcomponents. Obviously, all data elements occuring in the property are interesting. Furthermore, everything that influences the value of those data elements is interesting as well.

Out Data Ports of a certain component can be modified by:

- Out-out data port connections, thus the out data port of a subcomponent
- In-out flows, thus in data ports of the same component
- Transition effects, thus by own in and out data ports and own data subcomponents. (The data elements in the guard become interesting as well!)
- Resets (on deactivation of connection/flow without assigning transition effect in own component) to default value
- Reset on reactivation of the whole component (if without mode history).

In Data Ports analogously can be modified by:

- In-in data port connection, thus the in data port of the supercomponent
- Out-in data port connection, thus the out data port of a neighbour component
- Resets (on deactivation of connection in supercomponent) to default value

Data subcomponents of a non-data component can be modified by:

- Transition effects (same as for out data ports)
- Reset on reactivation of it inside the component.
- Reset on reactivation of the whole component (if without mode history).

By following these dependencies transitively the set of all interesting data elements can easily be determined by fixpoint iteration.

However, considering only „interesting“ transitions (i.e., those with an assignment to an interesting data element or with an reactivation of a subcomponent under which an interesting data element occurs) is not enough. We have to consider the guards on all transitions on paths leading to the mode with the „interesting“ transitions as well since they determine whether the interesting transition is reachable at all. For example, if `t2` is interesting then the guard of `t1` becomes interesting when `m -t1-> m' -t2-> m''`. This issue will be solved with adding „interesting“ modes to the slicing algorithm and making the **source** mode (implicit by transitive predecessor and target mode) of „interesting“ transitions „interesting“.

What to do with uninteresting data elements? Delet them and all assignments to them. Reading accesses to them should not occure because then they should have become interesting.

1.3 Modes

For even smaller models we do not only want to slice uninteresting data elements but uninteresting modes as well. Several aspects have to be considered with modes:

- The mode info itself or the (un)reachability of modes used in the property.
- The configuration of port connections / flows.
- The sequence of steps changing values of interesting data elements. E.g., with $[](x = 2 \Rightarrow \neg <> (x = 3))$ it is important to distinguish `m1` and `m2` when we have the transitions `m -[then x := 2]-> m1 -[then x := 4]-> m1` and `m -[then x := 0]-> m2 -[then x := 3]-> m2`. Because of this, the **target** mode of transitions changing interesting data elements becomes interesting as well!
- The activation/deactivation of subcomponents is not considered, so far.

1.3.1 Which modes are interesting?

- Modes occuring in the property.
- Target modes of transitions changing interesting data elements or having an interesting trigger.
- Modes in which a connection/flow to an interesting port is active that is not active in every mode.
- Transitive transition predecessors of interesting modes (for reachability of interesting modes and transitions).

Do „negative“ modes need a special treatment? No! They become „interesting“ and then all their predecessors as well. For the remaining „uninteresting“ states we know that they do not contain any of the negative states nor is any of those reachable from them!

1.3.2 What to do with „uninteresting“ modes?

First, we observe that from those modes no „interesting“ transitions (i.e., with an „interesting“ trigger, assigning to an „interesting“ data element or leading to an „interesting“ mode) go out because at least the source mode of every „interesting“ transition is interesting. Secondly, we know that for all interesting data elements the rule to calculate their value does not change in/between uninteresting modes: transitions do not exist and only thos flows/connections are active in uninteresting modes which are active in every mode. So we can merge all „uninteresting“ states to one placeholder sink state `_SlicedSinkState`. QUESTION: What about deadlock properties?

1.4 Event Ports

Properties cannot talk about event ports, thus they become interesting only indirectly by the triggers of „interesting“ transitions. Then the transitive dependencies along event port connection holds as it was the case for data port connections. Additionally, transitions with an „interesting“ event port as trigger become „interesting“ as well and so do their **source** mode and guards (weak!).

Attention: Take care of correctly handling *subcomponent.indataport* in supercomponent!

1.5 Introducing „weak interesting“

After identifying „interesting“ modes we can classify the transitions in four groups by the status of source and target mode:

- **Interesting - Interesting**: Those transitions are kept in the sliced model with the trigger and the guard (both became interesting). The effects can be reduced to assignments to „interesting“ (and later: „weak interesting“) data elements.
- **Interesting - Uninteresting**: Those transitions become a transition with the same trigger and guard but no effects at all the the `_SlicedSinkState`. The effects can be omitted as they obviously do not assign to interesting data elements. The trigger and guard are needed to decide whether in a concrete situation it is possible to enter the uninteresting mode where e.g. the property could be violated. Unfortunately, that makes the trigger and all data elements in the guard interesting as well. Consequently, new modes could become interesting because they have an imcomming transition with an effect changing the newly interesting data elements. However, for the property only changes to the newly interesting data elements up to the transition to the uninteresting modes are needed. Therefore, as an optimisation, we introduced the notion of „weak interesting“, described below.
- **Uninteresting - Uninteresting**: Just deleted as they are totally uninteresting (see SinkMode).
- **Uninteresting - Interesting**: Impossible, as predecessors of interesting modes become interesting!

1.5.1 Fixpoint Iteration for „weak interesting“ data elements

Similar as for normal „interesting“. However, we do not consider all transitions but only transitions between interesting modes and make their RHSs weakly interesting. As soon as we reach an in port to become weakly interesting, it becomes normally interesting, as the notion of weak interesting only makes sense locally inside a component. When an out port becomes weakly interesting, we have to make sure that all ports from which connections/flows in any interesting mode to this port become normal interesting. However, the out port itself does not become normal interesting as then we would consider transitions changing it (note: for out ports no assignments are possible!), possibly adding new modes.

CHANGE TO ALGORITHM: All RHSs/guards become weak interesting (except ports), not interesting!

The idea is that we are not globally interested in the value of weak interesting elements but that we need their value up to the last point where they influence changes to interesting data elements or guard transitions from interesting modes. Changes that happen afterwards can safely be ignored - thus, it is correct to concentrate only on transitions between interesting modes for changes of weak interesting data elements.

1.6 Lifting „interesting“ to „needed“ for complex elements

- Port Connections and Flows are **obviously** needed when their target port is interesting.
- **Transitions** are needed iff they (((have an interesting effect or (which is actually implied) end (thus, start) in an interesting mode or start in an interesting mode (due to interesting trigger) ===>))) start in an interesting mode.
- **Transition Effects** are needed when they assign to an interesting or weak interesting data element on a transition to (thus, from) an interesting mode.

- **Non-Data Components** are needed iff they have a needed non-data subcomponent, an interesting data element, an interesting event port, or ... ???