Semantics of the System-Level Integrated Modeling (SLIM) Language

Contents

Chapter 1

Semantic Domains

1.1 Basic Sets

The following basic sets are used throughout this document:

Ide: the set of *identifiers*

Nam: the set of *names*, given by $Nam := Ide.Ide$

Cat: the set of *component categories*, given by

$$Cat := \{\texttt{process}, \texttt{thread group}, \texttt{thread}, \texttt{data}, \texttt{processor}, \texttt{memory}, \texttt{bus}, \texttt{device}, \texttt{system}\}$$

Typ: the set of *(component and data) types*, given by

$$Typ := Ide \cup \{\texttt{[}l\,..\,u\texttt{]} \mid l, u \in \mathbb{Z}, l \leq u\} \cup \{\texttt{bool}, \texttt{clock}, \texttt{continuous}, \texttt{enum(}e_1\texttt{,}\ldots\texttt{,}e_n\texttt{)}, \texttt{int}, \texttt{real}\}$$

where $n \geq 1$ and $e_i \in Ide$ for each $i \in [n]$.

1.2 Components

Every system specification S describes a hierarchy of components where each component is defined by its type and its implementation. In particular, the component implementation describes its structure as an assembly of subcomponents where each subcomponent is referenced by an identifier. Thus each component in S is uniquely identified by a sequence of subcomponent identifiers (which corresponds to a path in the tree structure of S). The following recursive definition formally specifies

- the set of *components*, $Cmp \subseteq Ide^+$,
- the *category* of each component c, $cat(c) \in Cat$,
- its *type (identifier)*, $typ(c) \in Typ$, and
- its *implementation (name)*, $imp(c) \in Nam$.

Here we assume that `main` is the given *main component* with category $cat(\texttt{main}) \in Cat$, type $typ(\texttt{main}) \in Typ$ and *implementation* $imp(\texttt{main}) \in Nam$.

- $\texttt{main} \in Cmp$;
- whenever $c \in Cmp$ such that the specification of $imp(c)$ contains

  $$\textbf{subcomponents} \ \ldots \ sc\texttt{:} \ cc \ tp.im \ \ldots\texttt{;} \ldots,$$

  then
  - $c' := c.sc \in Cmp$,[1]
  - $cat(c') := cc$,
  - $typ(c') := tp$, and
  - $imp(c') := tp.im$ (if $cat(c') = \textbf{data}$, then we let $imp(c') := tp$).

  Moreover we distinguish the following *component classes*:
- *data components*: $DCmp := \{c \in Cmp \mid cat(c) = \textbf{data}\}$,
- *control components*: $CCmp := Cmp \setminus DCmp$, and
- *atomic control components*: $ACmp := \{c \in CCmp \mid c.Ide^+ \cap CCmp = \emptyset\} \subseteq CCmp$.

Example 1.1 For the specification (cf. [COM09, Ex. 4.4/4.6])

```
process Cruise
  features
    engage: in event port;
    brake: in event port;
    speed: in data port real;
    throttle: out data port real;
end Cruise;
process implementation Cruise.Impl
  subcomponents
    input: thread Input.Impl;
    output: thread Output.Impl;
  connections
    event port engage -> input.engage;
    event port brake -> input.brake;
    data port speed -> input.speed;
    data port input.control -> output.control;
    data port output.throttle -> throttle;
end Cruise.Impl;

thread Input
  features
```

[1] If subcomponent identifiers (sc) are unique and if each component implementation is used only once in S, then it suffices to consider $Cmp := Ide$ by letting $sc \in Cmp$.

```
    engage: in event port;
    brake: in event port;
    speed: in data port real;
    control: out data port int default 0;
end Input;
thread implementation Input.Impl
  subcomponents
    timer: data clock in modes (enabled);
    set: data real in modes (enabled);
  modes
    idle: activation mode;
    enabled: mode while timer <= 100;
  transitions
    idle -[engage then timer := 0; set := speed]-> enabled;
    enabled -[when speed<set then control := 1]-> enabled;
    enabled -[when speed>set then control := -1]-> enabled;
    enabled -[brake]-> idle;
    enabled -[when timer >= 100]-> idle;
end Input.Impl;

thread Output
  features
    control: in data port int;
    throttle: out data port real;
end Output;
thread implementation Output.Impl
  ...
end Output.Impl;
```

we obtain (assuming that `Output.Impl` does not define any subcomponents):

$$\begin{aligned} Cmp &= \{\texttt{main}, \texttt{main.input}, \texttt{main.input.timer}, \texttt{main.input.set}, \texttt{main.output}\} \\ DCmp &= \{\texttt{main.input.timer}, \texttt{main.input.set}\} \\ CCmp &= \{\texttt{main}, \texttt{main.input}, \texttt{main.output}\} \\ ACmp &= \{\texttt{main.input}, \texttt{main.output}\} \end{aligned}$$

As all component identifiers are unique, it suffices to set

$$Cmp := \{\texttt{main}, \texttt{input}, \texttt{timer}, \texttt{set}, \texttt{output}\}$$

which yields

$c \in Cmp$	$cat(c)$	$typ(c)$	$imp(c)$
main	**process**	Cruise	Cruise.Impl
input	**thread**	Input	Input.Impl
timer	**data**	**clock**	**clock**
set	**data**	**real**	**real**
output	**thread**	Output	Output.Impl

1.3 Modes

Modes can be attached to control components to define an automata-like behavior, and to model dynamic reconfiguration of the system. In our formal semantics, they are represented by associating the following information with each $c \in CCmp$:

- the (finite) set of its *modes*, $Mod(c) \subseteq Ide$,
- its *starting mode*, $stm(c) \in Mod(c)$,
- the *invariant* of each mode, $inv(c, m)$,
- the set of its (direct) subcomponents which are *active* in the respective mode $m \in Mod(c)$, $Act(c, m) \subseteq Ide$, and
- the *binding relations*
  - $Acc(c, m) \subseteq Act(c, m) \times Act(c, m)$ (**accesses**),
  - $Run(c, m) \subseteq Act(c, m) \times Act(c, m)$ (**running on**), and
  - $Sto(c, m) \subseteq Act(c, m) \times Act(c, m)$ (**stored in**).

  This is accomplished as follows.

- Whenever the specification of $imp(c)$ contains

  $$\textbf{modes} \ \ldots \ m\texttt{:}\ tp\ \textbf{mode while}\ iv\texttt{;}\ \ldots,$$

  then
  - $m \in Mod(c)$,
  - $stm(c) := m$ if $tp \in \{\textbf{initial}, \textbf{activation}\}$, and
  - $inv(c, m) := iv$;

  and

- whenever the specification of $imp(c)$ contains

  $$\textbf{subcomponents}\ \ldots\ sc\texttt{:}\ cc\ im\ bn\ sc'\ \ldots\ \textbf{in modes}\texttt{(}m_1\texttt{,}\ldots\texttt{,}m_n\texttt{);}\ \ldots$$

  where $sc, sc' \in Ide$, $cc \in Cat$, $im \in Nam$, $bn \in \{\textbf{accesses}, \textbf{running on}, \textbf{stored in}\}$, and $n \geq 1$, then for every $i \in [n]$

  $$sc \in Act(c, m_i)$$

  and

  $$(sc, sc') \in \begin{cases} Acc(c, m_i) & \text{if } bn = \textbf{accesses} \\ Run(c, m_i) & \text{if } bn = \textbf{running on} \\ Sto(c, m_i) & \text{if } bn = \textbf{stored in} \end{cases}$$

Note that, according to the syntactic restrictions imposed in [COM09, Sct. 4.3.1], each binding relation of a component $c \in CCmp$ in mode $m \in Mod(c)$ only relates subcomponents that are active in m.

If the implementation of a control component $c \in CCmp$ does not employ any modes, then we assume that $Mod(c) = \{stm(c)\}$ for some **initial** mode $stm(c) = m_0$. As abbreviations we use $CSub(c) := \bigcup_{m \in Mod(c)} \{sc \in Act(c,m) \mid c.sc \in CCmp\}$ for the set of (direct) control subcomponents of c[2], $DAct(c,m) := \{sc \in Act(c,m) \mid c.sc \in DCmp\}$ for the set of data subcomponents of c that are active in mode m, and $Mod := \bigcup_{c \in CCmp} Mod(c)$ for the set of all modes occurring in the specification.

Example 1.2 For the specification in Example 1.1, we obtain

$$Mod(\texttt{main}) = \{\underline{m_0}\} \quad Mod(\texttt{input}) = \{\underline{\texttt{idle}}, \texttt{enabled}\} \quad Mod(\texttt{output}) = \ldots$$

where the respective starting mode $stm(c)$ is underlined, and

$$\begin{aligned} inv(\texttt{main}, m_0) &= \textbf{true} & Act(\texttt{main}, m_0) &= \{\texttt{input}, \texttt{output}\} \\ inv(\texttt{input}, \texttt{idle}) &= \textbf{true} & Act(\texttt{input}, \texttt{idle}) &= \emptyset \\ inv(\texttt{input}, \texttt{enabled}) &= (\texttt{timer <= 100}) & Act(\texttt{input}, \texttt{enabled}) &= \{\texttt{timer}, \texttt{set}\} \\ inv(\texttt{output}, \ldots) &= \ldots & Act(\texttt{output}, \ldots) &= \ldots \end{aligned}$$

1.4 Ports

Let $c \in CCmp$. The *ports* of c are denoted as follows. Whenever the specification of $typ(c)$ contains **features** $\ldots$ p: pc; $\ldots$ with

- $pc =$ **in event port**, then $p \in IEPrt(c)$,
- $pc =$ **out event port**, then $p \in OEPrt(c)$,
- $pc =$ **in data port** tp, then $p \in IDPrt(c)$ and $typ(c,p) := tp$, and
- $pc =$ **out data port** $tp \ldots$, then $p \in ODPrt(c)$ and $typ(c,p) := tp$.

A subset of ports $OEPrtNB(c) \subseteq OEPrt(c)$ is defined for ports declared **nonblocking**, i.e. if $pc =$ **out event port nonblocking**, then $p \in OEPrtNB(c)$.

As abbreviations we use:[3]

$$\begin{aligned} EPrt(c) &:= IEPrt(c) \cup OEPrt(c), \\ EPrt &:= \bigcup_{c \in CCmp} EPrt(c), \\ DPrt(c) &:= IDPrt(c) \cup ODPrt(c), \text{ and} \\ DPrt &:= \bigcup_{c \in CCmp} DPrt(c). \end{aligned}$$

[2]Note that every subcomponent of c has to be active in at least one of the modes of c.

[3]Note that according to the syntactic restrictions imposed in [COM09, Sct. 4.2.2], $typ(c, DPrt(c)) \subseteq \{[l\,..\,u] \mid l, u \in \mathbb{Z}, l \leq u\} \cup \{\texttt{bool}, \texttt{enum}, \texttt{int}, \texttt{real}\}$.

Example 1.3 For the specification in Example 1.1, we obtain

$c \in Cmp$	$IEPrt(c)$	$OEPrt(c)$	$IDPrt(c)$	$ODPrt(c)$
`throttle`}				
`main`	{`engage`, `brake`}	$\emptyset$	{`speed`}	{`throttle`}
`input`	{`engage`, `brake`}	$\emptyset$	{`speed`}	{`control`}
`output`	$\emptyset$	$\emptyset$	{`control`}	{`throttle`}

with

$$typ(\texttt{main}, \texttt{speed}) = typ(\texttt{main}, \texttt{throttle}) = typ(\texttt{input}, \texttt{speed}) = typ(\texttt{output}, \texttt{throttle}) = \textbf{real}$$

and

$$typ(\texttt{input}, \texttt{control}) = typ(\texttt{output}, \texttt{control}) = \textbf{int}.$$

1.5 Port Connections

Let $c \in CCmp$. The *event port connections* of c are denoted as follows. Whenever the specification of $imp(c)$ contains

connections ... **event port** $sc_1.p_1$ -> $sc_2.p_2$ **in modes**(m_1, ..., m_n) [**passive**]; ...

(where, for each $i \in [2]$ and $j \in [n]$, $sc_i \in Act(c, m_j) \cup \{\varepsilon\}$), then

$$(sc_1.p_1, sc_2.p_2, b) \in ECon(c, m_j)$$

for each $j \in [n]$ where $b = \mathsf{true}$ if the **passive** attribute is present and $b = \mathsf{false}$ otherwise. Thus, according to the syntactic restrictions imposed in [COM09, Sct. 4.3.2],

$$\begin{aligned} ECon(c,m) \subseteq\ & IEPrt(c) \times IES(c,m) \times \{\mathsf{false}\} \\ \cup\ & OES(c,m) \times OEPrt(c) \times \{\mathsf{false}\} \\ \cup\ & OES(c,m) \times IES(c,m) \times \mathbb{B} \end{aligned}$$

where $IES(c,m) := \{sc.p \mid sc \in Act(c,m), p \in IEPrt(c.sc)\}$ and $OES(c,m) := \{sc.p \mid sc \in Act(c,m), p \in OEPrt(c.sc)\}$.

Analogously we can define the set of *data port connections*

$$\begin{aligned} DCon(c,m) \subseteq\ & IDPrt(c) \times IDS(c,m) \\ \cup\ & ODS(c,m) \times ODPrt(c) \\ \cup\ & ODS(c,m) \times IDS(c,m) \end{aligned}$$

where $IDS(c,m) := \{sc.p \mid sc \in Act(c,m), p \in IDPrt(c.sc)\}$ and $ODS(c,m) := \{sc.p \mid sc \in Act(c,m), p \in ODPrt(c.sc)\}$ (without requiring the additional Boolean **passive** attribute). Note that $typ(c, p_1) = typ(c, p_2)$ for each $(p_1, p_2) \in DCon(c,m)$.

Example 1.4 For the specification in Example 1.1, we obtain

$$\begin{aligned} ECon(\texttt{main}, m_0) &= \{(\mathsf{engage}, \texttt{input.engage}, \mathsf{false}), \\ &\quad\ (\mathsf{brake}, \texttt{input.brake}, \mathsf{false})\} \\ DCon(\texttt{main}, m_0) &= \{(\texttt{speed}, \texttt{input.speed}), \\ &\quad\ (\texttt{input.control}, \texttt{output.control}), \\ &\quad\ (\texttt{output.throttle}, \texttt{throttle})\}. \end{aligned}$$

The remaining sets are empty.

1.6 Flows

Let $c \in CCmp$. The *flows* of c are denoted as follows. Whenever the specification of $imp(c)$ contains

$$\texttt{flows} \ \ldots \ d \ \texttt{:=} \ a \ \texttt{in modes(}m_1\texttt{,}\ldots\texttt{,}m_n\texttt{);}\ \ldots$$

(where $d \in ODPrt(c)$ and a is an expression over $IDPrt(c)$), then

$$(a, d) \in Flw(c, m_i)$$

for each $i \in [n]$.

1.7 Data Elements

Together, the data subcomponents and the data ports constitute the *data elements* of a control component $c \in CCmp$ in mode $m \in Mod(c)$:[4]

$$Dat(c, m) := DAct(c, m) \cup DPrt(c) \subseteq Ide.$$

Again we set $Dat(c) := \bigcup_{m \in Mod(c)} Dat(c, m)$ and $Dat := \bigcup_{c \in CCmp} Dat(c)$.

We distinguish the following kinds of data elements:

- *clocks*: $Clk(c, m) := \{d \in DAct(c, m) \mid typ(c, d) = \texttt{clock}\}$
- *continuous variables*: $Cnt(c, m) := \{d \in DAct(c, m) \mid typ(c, d) = \texttt{continuous}\}$
- *discrete data elements*: $Dsc(c, m) := Dat(c, m) \setminus (Clk(c, m) \cup Cnt(c, m))$ [5]

Again we allow to use notations such as $Clk(c)$.

Using the **default** attribute, a (control) component specification can assign default values to both (incoming and outgoing) data ports and data subcomponents. In our semantics, this assignment is represented by a partial mapping $dfl(c, d)$ which associates constant values to (some of) the data elements $d \in Dat(c)$. Here we assume that $dfl(c, d) = 0$ for every clock $d \in Clk(c)$.

1.8 Mode Transitions

The set of *mode transitions* of a control component $c \in CCmp$, $MTr(c)$, is defined as follows: whenever the specification of $imp(c)$ contains an entry of the form

$$\texttt{transitions} \ldots m \texttt{ -[}p \texttt{ when } g \texttt{ then } f\texttt{]->} \ m'\texttt{;} \ldots$$

where $m, m' \in Mod(c)$, $p \in \{sc.p \mid sc \in Act(c, m) \cup \{\varepsilon\}, p \in EPrt(c.sc)\} \cup \{\texttt{reset}, \tau\}$, g denotes a guard, and f is an effect (as defined in [COM09, Sct. 4.3.4]), then

$$(m, p, g, f, m') \in MTr(c).$$

[4] Note that data components have neither subcomponents nor ports, and that the set of (data) ports does not depend on the current mode.

[5] Note that according to the syntactic restrictions in [COM09, Sct. 4.2.2], clocks and continuous variables cannot be used as data ports, i.e., $DPrt(c) \subseteq Dsc(c)$.

Example 1.5 For the specification in Example 1.1, we obtain

$$\begin{aligned}
MTr(\texttt{main}) &= \emptyset \\
MTr(\texttt{input}) &= \{(\texttt{idle}, \texttt{engage}, \texttt{true}, (\texttt{timer := 0; set := speed}), \texttt{enabled}), \\
&\quad (\texttt{enabled}, \tau, \texttt{speed < set}, \texttt{control := 1}, \texttt{enabled}), \\
&\quad (\texttt{enabled}, \tau, \texttt{speed > set}, \texttt{control := -1}, \texttt{enabled}), \\
&\quad (\texttt{enabled}, \texttt{brake}, \texttt{true}, \varepsilon, \texttt{idle}), \\
&\quad (\texttt{enabled}, \tau, \texttt{timer >= 100}, \varepsilon, \texttt{idle})\} \\
MTr(\texttt{output}) &= \ldots
\end{aligned}$$

1.9 Reactivation Transitions

The set of *reactivation transitions* of a control component $c \in CCmp$, $RTr(c)$, is defined as follows: whenever the specification of $imp(c)$ contains an entry of the form

$$\texttt{transitions} \ldots m \texttt{ -[@activation then } f\texttt{]-> } m'\texttt{;} \ldots$$

where $m, m' \in Mod(c)$ and f is an effect (as defined in [COM09, Sct. 4.3.4]), then

$$(m, f, m') \in RTr(c).$$

Chapter 2

Semantics of Nominal Specifications

This chapter defines the operational semantics of a SLIM specification under the following restrictions:

- Error behavior is ignored ("nominal").
- Events are not queued but have to be transmitted in a handshaking operation ("synchronous"), in correspondence with AADL's handling of events that trigger mode transitions [SAE08, p. 170].

2.1 Formalizing the Local Behavior of Components

The semantics is specified by first introducing a new automata model, called event-data automata, which is employed to formalize the local behavior of a component. We then show how to represent a SLIM component specification as an event-data automaton.

2.1.1 Event-Data Automata

An *event-data automaton (EDA)* is a tuple of the form

$$\mathfrak{A} = (M, m_0, X, v_0, \chi, \varphi, E, \longrightarrow, \rightrightarrows)$$

where

- M is a finite set of *modes*,
- $m_0 \in M$ denotes the *starting mode*,
- X is a finite set of *variables*, partitioned into
  - *input variables*, IX,
  - *output variables*, OX, and
  - *local variables*, LX,
- $v_0 \in V_X$ is the *initial valuation* where V_X denotes the set of all *valuations*, that is, partial functions that assign values to the elements of X,

- $\chi : M \to (V_{LX} \to \mathbb{B})$ specifies the *mode constraints* (where we assume that $\chi(m_0)(v_0|_{LX}) = \mathsf{true}$),
- $\varphi : M \to (LX \to \mathbb{R})$ specifies the *trajectory equations* by associating with each local variable its derivative in the current mode[1],
- E is a finite set of *events*, partitioned into
  - *input events*, IE, and
  - *output events*, OE, and
- $\longrightarrow \subseteq M \times E_\tau \times (V_X \to \mathbb{B}) \times (V_X \to V_X) \times M$ is a finite *(mode) transition relation* where $E_\tau := E \cup \{\tau\}$. Transitions are represented in the form $m \xrightarrow{e,g,f} m'$, and e, g, and f are called the *trigger*, the *guard*, and the *effect*, respectively. Here f is allowed to modify only output and local variables, that is, $f(v)(x) = v(x)$ for each $v \in V_X$ and $x \in IX$.
- $\rightrightarrows \subseteq M \times (V_X \to V_X) \times M$ is a finite *reactivation transition relation*. Transitions are represented in the form $m \overset{f}{\rightrightarrows} m'$, and f is called the *effect*. Here f is allowed to modify only output and local variables, that is, $f(v)(x) = v(x)$ for each $v \in V_X$ and $x \in IX$.

2.1.2 Semantics of Event-Data Automata

The operational semantics of an EDA is given as a labeled transition system whose states, called *configurations*, are pairs of modes and valuations. Transitions either model the passing of time, involving an update of the non-discrete variables, or are internally triggered by events, including the invisible event τ. The second case requires the guard of the respective transition to be enabled, and then modifies the valuation of the variables according to the transition effect.

The definition of the semantics employs the following notation. Given a valuation $v \in V_X$, a time delay $t \in \mathbb{R}_{>0}$, and a mapping $\varphi : LX \to \mathbb{R}$ of trajectory equations, the notation $v + t \cdot \varphi$ denotes the corresponding temporal modification of the local variables, that is, for each $x \in X$,

$$(v + t \cdot \varphi)(x) := \begin{cases} v(x) + t \cdot \varphi(x) & \text{if } x \in LX \\ v(x) & \text{otherwise} \end{cases}$$

Formally, the *semantics* of an EDA is given by the labeled transition system

$$(Cnf, \kappa_0, L, \longrightarrow)$$

with

- the set of *(local) configurations* $Cnf := M \times V_X$,
- the *initial configuration* $\kappa_0 := (m_0, v_0) \in Cnf$,
- the set of *transition labels* $L := \mathbb{R}_{>0} \cup E_\tau$, and

[1] If $\varphi(m)(x) = 0$, then x is a discrete variable, if $\varphi(m)(x) = 1$, then it is a clock, and otherwise it is a hybrid variable.

- the *(local) transition relation* $\longrightarrow \subseteq \mathit{Cnf} \times L \times \mathit{Cnf}$, given by
  - *time transition*: $(m, v) \xrightarrow{t} (m, v + t \cdot \varphi(m))$ if
    - $t \in \mathbb{R}_{>0}$ and
    - the invariant stays valid for t time units[2]: $\chi(m)(v|_{LX} + t \cdot \varphi(m)) = \mathsf{true}$.
  - *internal or event transition*: $(m, v) \xrightarrow{e} (m', f(v))$ if
    - $e \in E_\tau$ and
    - in the current mode m, an e-transition is enabled where the invariant of the target mode is valid after applying the transition effect: there exists $m \xrightarrow{e,g,f} m'$ in $\mathfrak{A}$ such that $g(v) = \mathsf{true}$ and $\chi(m')(f(v)|_{LX}) = \mathsf{true}$.
  - *reactivation transition*: $(m, v) \rightrightarrows (m', f(v))$ if
    - in the current mode m, an reactivation-transition is enabled where the invariant of the target mode is valid after applying the transition effect: there exists $m \overset{f}{\rightrightarrows} m'$ in $\mathfrak{A}$ such that $\chi(m')(f(v)|_{LX}) = \mathsf{true}$.

2.1.3 Representing SLIM Components as Event-Data Automata

Given a SLIM component specifications as represented by the sets and mappings that were introduced in Chapter 1, we can now define an EDA that represents the local behavior of the component. Here we denote the value of a given Boolean expression b (such as a mode invariant or a transition guard) with respect to a valuation by $[\![b]\!] : V_X \to \mathbb{B}$. Likewise, $[\![a]\!](v)$ denotes the value of an expression a with respect to the valuation $v \in V_X$.

The definition is based on the following associations:

- The meaning of modes in the SLIM component and in the EDA is identical.
- Incoming and outgoing data ports are interpreted as input and output variables, respectively, and data subcomponents are interpreted as local variables.
- Events in the EDA are either SLIM event ports, or are used to represent the event communication between a supercomponent and one of its (active) subcomponents. In the second case, they are of the form *sc.p* where *sc* is the identifier of the subcomponent, and p its event port. Here an incoming event port in the subcomponent gives rise to an output event in the EDA of the supercomponent, and vice versa.
- Initial valuations, mode constraints and trajectory equations are directly taken from the SLIM specification.
- (Reactivation) Transition effects are determined as follows:
  - incoming data ports are not modified,
  - outgoing data ports are updated according to the SLIM transition effect, if given, and not modified otherwise, and

[2]Owing to the linearity of constraints, it suffices to check them in the target valuation only.

– data subcomponents which are active in the target mode of the transition are updated according to the SLIM transition effect, if given, or else reset to their default value if they were inactive in the source mode, and not modified otherwise.

Formally, the association is defined as follows. For each control component $c \in CCmp$, $\mathfrak{A}_c = (M, m_0, X, v_0, \chi, \varphi, E, \longrightarrow, \rightrightarrows)$ is given by letting

- $M := Mod(c)$,
- $m_0 := stm(c)$,
- $X := IX \cup OX \cup LX$ where
  - $IX := IDPrt(c)$,
  - $OX := ODPrt(c)$, and
  - $LX := \bigcup_{m \in Mod(c)} DAct(c, m)$,
- $v_0 := dfl(c)$,
- for every $m \in Mod(c)$, $\chi(m)$ is determined by the constraints occurring in $inv(c, m)$,
- for every $m \in Mod(c)$, $\varphi(m)$ is determined by the trajectory equations occurring in $inv(c, m)$,
- $E := IE \cup OE$ where[3]
  - $IE := IEPrt(c) \cup \{sc.p \mid sc \in CSub(c), p \in OEPrt(c.sc)\}$ and
  - $OE := OEPrt(c) \cup \{sc.p \mid sc \in CSub(c), p \in IEPrt(c.sc)\}$, and
  - $OEnb := OEPrtNB(c)$
- $\longrightarrow := \{(m, p, [\![g]\!], [\![f]\!], m') \mid (m, p, g, f, m') \in MTr(c)\}$ where $[\![f]\!] : V_X \to V_X$ is defined as follows: $[\![f]\!](v) := v'$ with
  - for each $d \in IDPrt(c)$, $v'(d) := v(d)$,
  - for each $d \in ODPrt(c)$,

$$v'(d) := \begin{cases} [\![a]\!](v) & \text{if } f \text{ contains assignment } d := a \\ v(d) & \text{otherwise} \end{cases}$$

  - for each $d \in DAct(c, m')$,

$$v'(d) := \begin{cases} [\![a]\!](v) & \text{if } f \text{ contains assignment } d := a \\ dfl(c, d) & \text{else if } d \notin DAct(c, m) \\ v(d) & \text{otherwise} \end{cases}$$

- $\rightrightarrows := \{(m, [\![f]\!], m') \mid (m, f, m') \in RTr(c)\}$ where $[\![f]\!] : V_X \to V_X$ is defined the same as for $\longrightarrow$.

[3]Here the construction can be optimized by only considering those events that occur as transition labels in c-transitions.

2.2 Formalizing the Global Behavior of Systems

Now we have to specify how the EDAs that represent single components interact with each other. This interaction is highly dynamic; local transitions can cause subcomponents to become (in-)active, and can change the topology of event and data port connections and flows. On the level of the formal model this means that both the activation of the component EDAs and their interconnection depend on the modes of the EDAs.

2.2.1 Networks of Event-Data Automata

A *network of event-data automata (NEDA)* is a tuple of the form

$$\mathfrak{N} = ((\mathfrak{A}_i)_{i\in[n]}, \alpha, EC, DD)$$

where

- each $\mathfrak{A}_i$ is an EDA of the form $\mathfrak{A}_i = (M_i, m_0^i, X_i, v_0^i, \chi_i, \varphi_i, E_i, \longrightarrow_i, \rightrightarrows_i)$ $(i \in [n])$,
- $\alpha : M \to 2^{[n]}$ is the *activation mapping* (where $M := \prod_{i=1}^n M_i$ denotes the set of *global modes*),
- $EC : M \to \{i.e \mid i \in [n], e \in E_i\}^2 \times \mathbb{B}$ is the *event connection mapping*, and
- $DD : M \to (\{i.x \mid i \in [n], x \in IX_i \cup OX_i\} \dashrightarrow \{j.a \mid j \in [n], a \in Exp(IX_j) \cup OX_j\})$ is the *data dependence mapping* where $Exp(IX_j)$ denotes the set of all expressions over IX_j.

2.2.2 Semantics of Networks of Event-Data Automata

The *semantics* of a NEDA is given by the labeled transition system

$$(Cnf, \kappa_0, L, \Longrightarrow)$$

which is defined in terms of the local transition systems $(Cnf_i, \kappa_0^i, L_i, \longrightarrow_i)$ $(i \in [n])$ of the constituent EDAs as follows (please refer to the next list for the definitions of the auxiliary functions):

- the set of *(global) configurations* is given by $Cnf := \prod_{i=1}^n Cnf_i$,
- the *initial configuration* is $\kappa_0 := (\kappa_0^1, \ldots, \kappa_0^n)$,
- the set of *transition labels* is $L := \mathbb{R}_{>0} \cup \{\tau\} \cup E_1$, and
- the *(global) transition relation*, $\Longrightarrow \subseteq Cnf \times L \times Cnf$, is given by
  - *time transition*: models the passing of time involving all active EDAs.

    $$\kappa = (\kappa_1, \ldots, \kappa_n) \overset{t}{\Longrightarrow} (\kappa_1', \ldots, \kappa_n')$$

    if there exists $t \in \mathbb{R}_{>0}$ such that for each $i \in [n]$, $\kappa_i \overset{t}{\longrightarrow} \kappa_i'$ if $i \in \alpha(mod(\kappa))$, and $\kappa_i' = \kappa_i$ otherwise.

– *internal transition*: represents an invisible step of an active EDA.

$$\kappa = (\kappa_1, \ldots, \kappa_n) \stackrel{\tau}{\Longrightarrow} cns_\kappa(nxt(\kappa, \{(i, \kappa_i')\}))$$

if there exist $i \in \alpha(mod(\kappa))$ and $\kappa_i' \in Cnf_i$ such that $\kappa_i \stackrel{\tau}{\longrightarrow}_i \kappa_i'$.

– *multiway communication transition*: an output event is sent from an active EDA to all active EDAs that offer a corresponding input transition, involving at least one non-passive event port connection.

$$\kappa = (\kappa_1, \ldots, \kappa_n) \stackrel{\tau}{\Longrightarrow} cns_\kappa(nxt(\kappa, \{(j, \kappa_j') \mid j \in I \cup R \cup \{i\}\}))$$

if there exist $i \in \alpha(mod(\kappa))$, $oe \in OE_i$, and $\kappa_i' \in Cnf_i$ such that $\kappa_i \stackrel{oe}{\longrightarrow}_i \kappa_i'$ and

$$I := \{j \in \alpha(mod(\kappa)) \setminus \{i\} \mid \text{ex. } ie \in IE_j \text{ s.t. } (i.oe, j.ie, b_j) \in EC(mod(\kappa)) \text{ and } \kappa_j \stackrel{ie}{\longrightarrow}_j \kappa_j'\} \neq \emptyset,$$

where $b_j = \mathsf{false}$ for at least one $j \in I$, and

$$R := \{r \in \alpha(mod(config(\kappa, \{(j, \kappa_j') \mid j \in I \cup \{i\}\}))) \setminus \alpha(mod(\kappa)) \mid \kappa_r \rightrightarrows_r \kappa_r'\}$$

are the possible reactivations.

– *nonblocking event transition*: a nonblocking output event is sent from an active EDA, without requiring any active EDA to offer a corresponding input transition.

$$\kappa = (\kappa_1, \ldots, \kappa_n) \stackrel{\tau}{\Longrightarrow} cns_\kappa(nxt(\kappa, \{(j, \kappa_j') \mid j \in I \cup R \cup \{i\}\}))$$

if there exist $i \in \alpha(mod(\kappa))$, $oe \in OEnb_i$, and $\kappa_i' \in Cnf_i$ such that $\kappa_i \stackrel{oe}{\longrightarrow}_i \kappa_i'$ and

$$I := \{j \in \alpha(mod(\kappa)) \setminus \{i\} \mid \text{ex. } ie \in IE_j \text{ s.t. } (i.oe, j.ie, \mathsf{true}) \in EC(mod(\kappa)) \text{ and } \kappa_j \stackrel{ie}{\longrightarrow}_j \kappa_j'\}$$

where I consists only of passive connections or is the empty set, and

$$R := \{r \in \alpha(mod(config(\kappa, \{(j, \kappa_j') \mid j \in I \cup \{i\}\}))) \setminus \alpha(mod(\kappa)) \mid \kappa_r \rightrightarrows_r \kappa_r'\}$$

are the possible reactivations.

– *input transition*: the first EDA provides an input event that is connected to at least one EDA (including the first one) in which a corresponding transition is enabled.

$$\kappa = (\kappa_1, \ldots, \kappa_n) \stackrel{ie}{\Longrightarrow} cns_\kappa(nxt(\kappa, \{(i, \kappa_i') \mid i \in I \cup R\}))$$

if there exists $ie \in IE_1$ such that

$$I := \{j \in \alpha(mod(\kappa)) \mid \text{ex. } ie' \in IE_j \text{ s.t. } (1.ie, j.ie', \mathsf{false}) \in EC(mod(\kappa)) \text{ and } \kappa_j \stackrel{ie'}{\longrightarrow}_j \kappa_j'\} \neq \emptyset.$$

and

$$R := \{r \in \alpha(mod(config(\kappa, \{(j, \kappa_j') \mid j \in I\}))) \setminus \alpha(mod(\kappa)) \mid \kappa_r \rightrightarrows_r \kappa_r'\}$$

are the possible reactivations.

– *output transition*: some EDA provides an output event that is enabled, and that is connected to an output event of the first EDA.

$$\kappa = (\kappa_1, \ldots, \kappa_n) \stackrel{oe}{\Longrightarrow} cns_\kappa(nxt(\kappa, \{(i, \kappa_i')\}))$$

if there exists $oe' \in OE_i$ such that $\kappa_i \xrightarrow{oe'}_i \kappa_i'$ and $(i.oe', 1.oe, \mathsf{false}) \in EC(mod(\kappa))$.

The definition employs the following auxiliary functions:

- $mod : Cnf \to M$, extracting the mode information from a given global configuration:

$$mod(\kappa_1, \ldots, \kappa_n) := (mod(\kappa_1), \ldots, mod(\kappa_n)) \text{ with } mod(m, v) := m.$$

- $config : Cnf \times 2^{\bigcup_{i\in[n]}\{i\}\times Cnf_i} \to Cnf$, applying a set of local configurations to the global configuration:

$$\begin{aligned} config(\kappa, \emptyset) &:= \kappa \\ config((\kappa_1, \ldots, \kappa_i, \ldots, \kappa_n), \{(i, \kappa_i')\} \cup N) &:= config((\kappa_1, \ldots, \kappa_i', \ldots, \kappa_n), N) \end{aligned}$$

- $nxt : Cnf \times 2^{\bigcup_{i\in[n]}\{i\}\times Cnf_i} \to Cnf$, which reflects the impact of mode transitions occurring in the constituent EDAs, taking the current glocal configuration (first parameter) and the new local configurations (second parameter) into account. This impact is defined as follows:

  – Each EDA occurring in the set enters the new configuration.
  – Next, the impact of the mode transitions on the (direct and indirect) subcomponents of the affected components is determined as follows. Each subcomponent that is re-activated in the transition (that is, it is inactive in the source mode but active in the target mode), that does not support mode history (that is, its starting mode carries the **`activation`** attribute), and has no active reactivation transition, is restarted. This means that it enters its starting mode, and that its data elements obtain their default values.

  Formally, this is described as follows.

$$nxt(\kappa, N) := restart_\kappa(config(\kappa, N))$$

where

$$restart_\kappa(\kappa_1', \ldots, \kappa_n') := (\kappa_1'', \ldots, \kappa_n'')$$

with

$$\kappa_i'' := \begin{cases} \kappa_0^i & \text{if } i \in \alpha(mod(\kappa_1', \ldots, \kappa_n')) \setminus \alpha(mod(\kappa)), \kappa_i \not\rightrightarrows_i \kappa_i', \\ & mod(\kappa_0^i) \ \textbf{`activation`} \text{ mode} \\ \kappa_i' & \text{otherwise} \end{cases}$$

- $cns_\kappa : Cnf \to Cnf$, making a global configuration consistent by taking the (unique) solution of the equation system that is implied by the data dependence mapping. In addition, input or output variables that have been disconnected in the transition (that is, the variable occurs as a target in the data dependence relation of the old mode but in no data dependence of the new mode) are reset to their default values:

$$cns_\kappa((m_1, v_1), \ldots, (m_n, v_n)) := ((m_1, v_1'), \ldots, (m_n, v_n'))$$

if, for each $i \in [n]$ and $x \in IX_i \cup OX_i$,

$$v_i'(x) = \begin{cases} [\![a]\!](v_j') & \text{if } DD(m_1, \ldots, m_n)(i.x) = j.a \\ v_0^i(x) & \text{if } DD(m_1, \ldots, m_n)(i.x) \text{ undefined, } DD(mod(\kappa))(i.x) \text{ defined} \\ v_i(x) & \text{if } DD(m_1, \ldots, m_n)(i.x), DD(mod(\kappa))(i.x) \text{ both undefined} \end{cases}$$

2.2.3 Representing SLIM Specifications as Networks of Event-Data Automata

We now extend the mapping as introduced in Section 2.1.3 by also taking the activation and interconnection structures between and inside components into account. To do so, we assume without loss of generality that the set of control components is given by $CCmp = \{c_1, \ldots c_n\}$ with main component c_1.

Apart from the activation mapping which can directly be taken from the SLIM specification, the essential idea is to analyze the connection and flow structure of event and data ports in the SLIM specification for generating the corresponding NEDA. For the event part this means that, for a given global mode of the system, all end-to-end (that is, multistep) connections between event ports are determined, and are taken into account in the event connection (EC) mapping. Here the following cases need to be considered:

- multistep out-to-in connections, involving zero or more direct out-to-out, one direct out-to-in, and zero or more direct in-to-in connections (where the middle step yields the value of the **`passive`** attribute),
- (implicit) event connections between a supercomponent and one of its direct subcomponents,
- multistep in-to-in connections, originating in the main component and involving zero or more direct in-to-in connections, and
- multistep out-to-out connections, ending in the main component and involving zero or more direct out-to-out connections.

Note that only the first case can involve passive event port connections.

The data dependence (DD) mapping can directly be determined from the data port connections and flows as defined in the SLIM specification.

Formally, given the collection of components in the SLIM specification S, the association of a corresponding NEDA,

$$\mathfrak{N}_S = ((\mathfrak{A}_i)_{i\in[n]}, \alpha, EC, DD),$$

can be defined as follows:

- each $\mathfrak{A}_i := \mathfrak{A}_{c_i}$ $(i \in [n])$ is constructed as described in Section 2.1.3,
- the activation mapping $\alpha : M \to 2^{[n]}$ is derived from Act as follows: for each $(m_1, \ldots, m_n) \in M$,
  - $1 \in \alpha(m_1, \ldots, m_n)$ and
  - whenever $i \in \alpha(m_1, \ldots, m_n)$, then $Act(i, m_i) \subseteq \alpha(m_1, \ldots, m_n)$,
- for each $(m_1, \ldots, m_n) \in M$,

$$\begin{aligned} EC(m_1, \ldots, m_n) := \\ & \{(i.op, j.ip, b) \mid i, j \in [n], op \in OE_i, ip \in IE_j, (c_i.op, c_j.ip) \in ECon^+\} \\ \cup\ & \{(i.(sc.ip), j.ip, \mathsf{false}) \mid i, j \in [n], sc \in Act(c_i, m_i), c_i.sc = c_j, sc.ip \in OE_i\} \\ \cup\ & \{(j.op, i.(sc.op), \mathsf{false}) \mid i, j \in [n], sc \in Act(c_i, m_i), c_i.sc = c_j, sc.op \in IE_i\} \\ \cup\ & \{(1.ip, i.ip', \mathsf{false}) \mid i \in [n], ip \in IE_1, ip' \in IE_i, (c_1.ip, c_i.ip') \in ECon^*\} \\ \cup\ & \{(i.op', 1.op, \mathsf{false}) \mid i \in [n], op \in OE_1, op' \in OE_i, (c_i.op', c_1.op) \in ECon^*\} \end{aligned}$$

  and

- for each $(m_1, \ldots, m_n) \in M$, $i \in [n]$, and $x \in IX_i \cup OX_i$,

$$DD(m_1, \ldots, m_n)(i.x) := \begin{cases} j.y & \text{if } (y, sc.x) \in DCon(c_j, m_j) \text{ and } c_j.sc = c_i \\ & \text{or } (sc_1.y, sc_2.x) \in DCon(c_k, m_k) \text{ and} \\ & \quad c_k.sc_1 = c_j, c_k.sc_2 = c_i \\ & \text{or } (sc.y, x) \in DCon(c_i, m_i) \text{ and } c_i.sc = c_j \\ i.a & \text{if } (a, x) \in Flw(c_i, m_i) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Here the notation $(c_i.op, c_j.ip) \in ECon^+$ means that, in the global mode $(m_1, \ldots, m_n)$, there is a multistep connection from the output port op of component c_i to the input port ip of component c_j, in the order

(a) (zero ore more) out-to-out connections,

(b) (exactly one) out-to-in connection, and

(c) (zero ore more) in-to-in connections.

In particular, the Boolean **`passive`** attribute b is determined by the connection taken in step (b).

Similarly, the notations $(c_1.ip, c_i.ip') \in ECon^*$ and $(c_i.op', c_1.op) \in ECon^*$ refer to a (possibly) empty sequence of in-to-in and out-to-out connections, respectively.

Chapter 3

Semantics of Error Models

3.1 Additional Semantic Domains

The association between a control component and its *error model implementation* is given by the partial mapping

$$err : CCmp \dashrightarrow Nam,$$

as defined in the user interface of the toolset.

3.1.1 Error Types and Implementations

Let $r := err(c)$. Whenever the specification of tp contains

$$\texttt{features} \ \ldots \ s\texttt{:} \ st \ \texttt{state;} \ \ldots$$

then

$$s \in Stt(r)$$

collects the *error states* of r. If $st \in \{\texttt{initial}, \texttt{activation}\}$, then

$$sts(r) := s$$

denotes the *starting state* of tp.

Likewise, the incoming and outgoing *error propagations* of r are collected in the sets

$$IPrp(r), OPrp(r) \subseteq Ide,$$

respectively, and again we let $Prp(r) := IPrp(r) \cup OPrp(r)$. Moreover,

$$Evt(r) \subseteq Ide$$

contains the *error events* of r.

With each outgoing error propagation and each error event an *occurrence distribution* is associated:

$$dst(r) : OPrp(r) \cup Evt(r) \to Dst(Stt(r))$$

where $Dst(Stt(r))$ denotes the set of probability distributions over $Stt(r)$.

Finally each error model implementation specifies a set of *error transitions*

$$ETr(r) \subseteq Stt(r) \times (Evt(r) \cup Prp(r) \cup \{\texttt{reset}\}) \times Stt(r).$$

As an abbreviation we allow to access the elements of an error model by the name of the component using that model, that is, we let $Stt(c) := Stt(err(c))$ etc.

3.1.2 Fault Injection

The effect of faults is specified in the user interface by defining *failure effects* which modify the nominal behavior of a component in dependence of its error state. This impact, the so-called *fault injection*, is defined by a list of assignments to the component's data elements (which have to be active in each of its modes) that overrides the nominal transition effects as specified in Section 1.8 in the presence of an error state. Formally, this yields a *failure mapping* $\mathit{flr}(c, s)$ which defines the failure effect for component $c \in \mathit{CCmp}$ in error state $s \in \mathit{Stt}(c)$ in the form of assignments to the data elements of c. We assume that $\mathit{flr}(c, \mathit{sts}(c)) = \varepsilon$.

3.2 Model Extension

The integration of the nominal and the error model, the so-called *(fault) model extension*, works similarly to the procedure described in [BV07]. It modifies each nominal control component model for which a (non-trivial) error model is defined by enriching it by the error model specification, thus producing the extended model which represents both the nominal and the failure behavior. Informally, the extended model is obtained as follows:

- For each control component in the system, (an instantiation of) the associated error model is attached as a new **`system`** subcomponent to the nominal specification.
- The starting mode of the error subcomponent is defined to be the starting state of the error model. It is an **`initial`**/**`activation`** mode if the starting state is of type **`initial`**/**`activation`**, respectively.
- Each state transition of the error model gives rise to a mode transition of the error subcomponent.
- The set of event ports of the nominal model is extended by adding all error propagations (*propagation ports*) of the respective error model, in order to simulate the forwarding of error information via propagations by event communication.
- Correspondingly, the set of event port connections has to be extended by *propagation port connections* which support the information flow as described in [COM09, Sct. 4.4].
- The other elements (set of components, component activation, data elements/ports/-connections, mode invariants, ...) are not affected.

Formally, the *extended model* is defined by the following modifications. For each $c \in \mathit{CCmp}$, let $r := \mathit{err}(c)$, let $c' := c.r$ be the corresponding new subcomponent of c, and let $\mathit{CCmp}' := \mathit{CCmp} \cup \{c'\}$. The new component c' and the modifications of c are formally defined as follows.

- The modes of c' are exactly the error states of r: $\mathit{Mod}'(c') := \mathit{Stt}(r)$;
- its original starting state becomes the starting mode: $\mathit{stm}'(c') := \mathit{sts}(r)$;
- error events and propagations of r and the special **`reset`** signal are turned into event ports:

$$\begin{aligned} \mathit{IEPrt}'(c) &:= \mathit{IEPrt}(c) \uplus \mathit{IPrp}(r), \\ \mathit{OEPrt}'(c) &:= \mathit{OEPrt}(c) \uplus \mathit{OPrp}(r), \\ \mathit{IEPrt}'(c') &:= \mathit{IPrp}(r) \uplus \{\texttt{reset}\}, \text{ and} \\ \mathit{OEPrt}'(c') &:= \mathit{Evt}(r) \uplus \mathit{OPrp}(r); \end{aligned}$$

- a new outgoing data port `errorState` with

$$typ(c, \texttt{errorState}) := typ(c', \texttt{errorState}) := \textbf{enum}(Stt(r))$$

is introduced to notify c about the current error state of c', and the set of incoming data ports is preserved:

$$\begin{aligned} ODPrt'(c) &:= ODPrt(c) \uplus \{\texttt{errorState}\}, \\ ODPrt'(c') &:= \{\texttt{errorState}\}, \\ IDPrt'(c) &:= IDPrt(c), \text{ and} \\ IDPrt'(c') &:= \emptyset; \end{aligned}$$

- *propagation port connections* are added to c as new event port connections: for each $m \in Mod(c)$,

$$ECon'(c, m) := ECon(c, m) \uplus PCon(c, m)$$

where propagations for component bindings, from the super- to the subcomponent and vice versa, and from and to the environment have to be considered:

$$\begin{aligned} PCon(c, m) := \{&(sc_1.p, sc_2.p, \mathsf{false}) \mid (sc_2, sc_1) \in Acc(c, m) \cup Acc(c, m)^{-1} \cup \\ &\qquad Run(c, m) \cup Sto(c, m), \\ &\qquad p \in OPrp(c.sc_1) \cap IPrp(c.sc_2)\} \\ \cup\ &\{(r.p, sc.p, \mathsf{false}) \mid sc \in Act(c, m), p \in OPrp(r) \cap IPrp(c.sc)\} \\ \cup\ &\{(sc.p, r.p, \mathsf{false}) \mid sc \in Act(c, m), p \in OPrp(c.sc) \cap IPrp(r)\} \\ \cup\ &\{(p, r.p, \mathsf{false}) \mid p \in IPrp(r)\} \\ \cup\ &\{(r.p, p, \mathsf{false}) \mid p \in OPrp(r)\} \end{aligned}$$

- the `errorState` data port connection is added to c: for each $m \in Mod(c)$,

$$DCon'(c, m) := DCon(c, m) \uplus \{(r.\texttt{errorState}, \texttt{errorState})\};$$

- error state transitions of r are turned into mode transitions of c', additionally setting the value of the `errorState` data port and adding dummy **`reset`** transitions if required: whenever $(s, e, s') \in ETr(r)$ (where $s, s' \in Stt(r)$ and $e \in Evt(r) \cup Prp(r) \cup \{\textbf{reset}\}$), then

$$(s, e, \mathsf{true}, \mathit{flr}(c, s'); \texttt{errorState} := s', s') \in MTr'(c').$$

Moreover, for every $s \in Stt(r)$ without an outgoing **`reset`** transition, the dummy transition

$$(s, \textbf{reset}, \mathsf{true}, \varepsilon, s) \in MTr'(c')$$

is added;

- mode transitions in c need to take into account the failure effects. Moreover **`reset`** transitions have to be linked to the error subcomponent, and error event transitions need to be added in order to immediately apply fault effects:

$$\begin{aligned} MTr'(c) := \{&(m, p, g \wedge \texttt{errorState} = s, \mathit{flr}(c, s) \rhd f, m') \mid \\ &\qquad (m, p, g, f, m') \in MTr(c), p \in EPrt(c.sc), s \in Stt(r)\} \\ \cup\ &\{(m, r.\textbf{reset}, g \wedge \texttt{errorState} = s, \mathit{flr}(c, s) \rhd f, m') \mid \\ &\qquad (m, \textbf{reset}, g, f, m') \in MTr(c), s \in Stt(r)\} \\ \cup\ &\{(m, r.e, \texttt{errorState} = s, \mathit{flr}(c, s'), m) \mid (s, e, s') \in ETr(r)\}. \end{aligned}$$

Here the notation $f_1 \rhd f_2$ refers to the combination of the assignments in f_1 and f_2, giving higher priority to those in f_1 in case of name clashes.

Bibliography

[BV07] M. Bozzano and A. Villafiorita. The FSAP/NuSMV-SA Safety Analysis Platform. *Int. J. on Software Tools for Technology Transfer*, 9(1):5–24, 2007.

[COM09] Model-based analysis and verification: Potential solutions. Technical Note D1-2, Issue 4.7, COMPASS Project, November 2009. `http://compass.informatik.rwth-aachen.de/internal/doclist/D1/`.

[SAE08] Architecture Analysis and Design Language (AADL) V2. SAE Draft Standard AS5506 V2, International Society of Automotive Engineers, March 2008.