Form Methods Syst Des (2013) 43:191–232
DOI 10.1007/s10703-012-0167-z

A compositional modelling and analysis framework for stochastic hybrid systems

Ernst Moritz Hahn · Arnd Hartmanns · Holger Hermanns · Joost-Pieter Katoen

Published online: 17 August 2012

Abstract The theory of hybrid systems is well-established as a model for real-world systems consisting of continuous behaviour and discrete control. In practice, the behaviour of such systems is also subject to uncertainties, such as measurement errors, or is controlled by randomised algorithms. These aspects can be modelled and analysed using stochastic hybrid systems. In this paper, we present HMODEST, an extension to the MODEST modelling language—which is originally designed for stochastic timed systems without complex continuous aspects—that adds differential equations and inclusions as an expressive way to describe the continuous system evolution. MODEST is a high-level language inspired by classical process algebras, thus compositional modelling is an integral feature. We define the syntax and semantics of HMODEST and show that it is a conservative extension of MODEST that retains the compositional modelling approach. To allow the analysis of HMODEST models, we report on the implementation of a connection to recently developed tools for the safety verification of stochastic hybrid systems, and illustrate the language and the tool support with a set of small, but instructive case studies.

Keywords Stochastic hybrid automata · Process calculus · Modelling · Analysis · Abstraction

This work has been supported by the DFG as part of SFB/TR 14 AVACS, by the EU FP7 project MoVeS, by the DFG/NWO bilateral research project ROCKS and has received funding from the European Union Seventh Framework Programme under grant agreement number 295261 as part of the MEALS project.

E.M. Hahn · A. Hartmanns (✉) · H. Hermanns
Saarland University – Computer Science, Campus E1 3, 66123 Saarbrücken, Germany
e-mail: arnd@cs.uni-saarland.de

E.M. Hahn
e-mail: emh@cs.uni-saarland.de

H. Hermanns
e-mail: hermanns@cs.uni-saarland.de

J.-P. Katoen
LS2: Software Modelling and Verification, RWTH Aachen University, 52056 Aachen, Germany
e-mail: katoen@cs.rwth-aachen.de

1 Introduction

Embedded software [28, 47] is omnipresent: It controls pacemakers and satellites, drives the climate control in our offices, is at the heart of our financial markets, supports the operation of power and chemical plants, and makes our cars and TVs work. Embedded software is different from traditional software in many respects. Most importantly, it is subject to complex and permanent interactions with its (mostly physical) environment via sensors and actuators. Typically, software in embedded systems does not terminate and interaction usually takes place with multiple concurrent processes at the same time. Reactions to the stimuli provided by the environment should be prompt (timeliness or responsiveness), i.e., the software has to "keep up" with the speed of the processes with which it interacts. As it is executed on (possibly battery-powered) devices where several other activities may go on, nonfunctional properties including the efficient usage of resources (such as power consumption) and robustness are important for dependable operation. Indeed, high requirements are put on performance and dependability, since tuning and maintenance is very difficult for embedded devices.

Embedded software is an important motivation for the development of modelling techniques and tools that provide an easy migration path for design engineers and at the same time support the description and evaluation of quantitative system aspects. This is rooted in the observation that classical models of software that leave out "nonfunctional" aspects such as cost, efficiency and robustness can at most partially address the critical features of embedded systems.

Software development for embedded systems must therefore during all design phases be supported by adequate formal methods to achieve strong results on correctness, performance, cost, and efficiency. Modelling formalisms are needed that are easily accessible, highly expressive, and supported by effective tools. In recent years, two major trends can be observed in this area: On the one hand, we see broad efforts to push further the boundaries of tool-supported quantitative evaluation of models of embedded systems, mostly rooted in model checking advances. For real-time [10] and for probabilistic systems [44], tool support has already reached a considerable degree of maturity, and model checking techniques are now being extended to cover continuous dynamics—as in hybrid automata—or continuous probabilistic phenomena—as in stochastic processes. On the other hand, various extensions of lightweight formal notations, such as SDL (System Description Language), UML (Unified Modelling Language), AADL (Architecture Analysis and Design Language) and SysML (Systems Modelling Language) are adapted to encompass quantitative aspects. In addition, a whole range of more rigorous formalisms based on, e.g., stochastic or hybrid process algebras [41, 42] or appropriate extensions of automata, such as timed automata [7], probabilistic automata [58] and hybrid automata [4] are being investigated. Lightweight notations are typically closer to engineering techniques but lack a formal semantics; rigorous formalisms do have such formal semantics, but their learning curve is typically too steep from a practitioner's perspective and they mostly have a restricted expressiveness.

One of the most expressive languages with a rigorous formal semantics is MODEST, the **mo**delling and **de**scription language for **s**tochastic **t**imed systems [17]. MODEST is a descendant of well-known process algebras, such as CSP and LOTOS, and is capable of expressing not only functional, but real-time, stochastic and probabilistic aspects of embedded software. The semantic domain associated with MODEST is that of stochastic timed automata (STA), basically timed automata [7] extended with data, discrete probabilistic branching, and clocks that may have a random duration. Tool support for MODEST and STA has flourished over the last decade, and has led to applications of various nature such as evaluating and optimising scheduling problems [50], the analysis of self-configuring networks [18], wireless

sensor networks [62], power consumption under drifting clocks [34], embedded transportation safety [8], and power grid stability under substantial renewable energy infeed [13]. Despite the fact that MODEST is a very expressive language, the entirety of its STA model spectrum can by now be analysed by quantitative evaluation.

In this paper we aim at taking up further advances on the evaluation side, in particular with respect to hybrid automata verification [22, 31, 32, 39, 54–56]. The particular feature of a hybrid system is that it superposes discrete state changes and continuously varying quantities. The latter are usually described by differential equations or differential inclusions. From this perspective, a timed automaton [7] is a hybrid automaton where each continuous variable c (a clock in timed automata jargon) follows the simple differential equation $\dot{c} = 1$. Conventional hybrid systems [4] capture many characteristics of real systems (air traffic management, pressure control, braking and acceleration manoeuvres). However, in various application areas, the lack of *randomness* hampers faithful modelling and analysis. This is especially true for wireless sensing and control applications, where message loss probabilities and other random effects (such as node placement, node failure, and battery drain) turn the overall control problem into a quantitative one—the control objective can only be managed with a certain, hopefully sufficiently large, probability.

The idea of combining probabilistic and hybrid systems is not new, and various different models have been proposed, each from its own perspective, see e.g. [2, 3, 20, 21, 59]. The most important difference lies in the place where to introduce randomness. One option is to replace deterministic mode transitions by probability distributions over mode jumps. Alternatively, differential equations that describe the evolution of continuous variables within a mode can be replaced by stochastic differential equations. More general stochastic hybrid models can be obtained by mixing the above two choices, and by combining with memoryless timed probabilistic jumps [14].

In this paper, we enrich stochastic timed automata (STA), the underlying semantical model of MODEST [17], with the possibility to describe continuous flows. This is similar in spirit to ideas behind probabilistic hybrid automata [59], an extension of hybrid automata where mode transitions are governed by discrete probability distributions. We present HMODEST, an extension of MODEST that adds differential equations and inclusions as an expressive way to describe the continuous system evolution. HMODEST is a high-level language with various convenient modelling features that enables the compositional modelling of stochastic hybrid automata. We define the syntax and semantics of HMODEST and show that it is a conservative extension of MODEST. To allow the analysis of HMODEST specifications, we report on the implementation of a connection to recently developed tools for the safety verification of stochastic hybrid systems, and illustrate HMODEST as well as the tool support with three case studies from the hybrid systems area: a water tank, a thermostat that can randomly fail, and a model of headway control in the European Train Control System (ETCS).

Organisation of the paper The paper is organised as follows. Sections 2, 3 and 4 review the syntax and semantics of MODEST extended for hybrid systems. Section 5 describes first prototypical tool support, that is applied in Sect. 6 to a selection of challenging case studies. Section 7 discusses related work, and Sect. 8 concludes the paper.

2 Syntax

We start our discussion by giving the complete grammar for HMODEST processes and process behaviours, extended and revised relative to the one of MODEST [17].

Table 1 Types of variables in HMODEST considered in this paper

Syntax	Type	Domain	Continuous behaviour
bool	Boolean variables	$\{true, false\}$	$\dot{x} = 0$
int	unbounded integers	$\mathbb{Z}$	$\dot{x} = 0$
int($e_1..e_2$)	bounded integers	$\{e_1, \ldots, e_2\}$	$\dot{x} = 0$
real	static real variables	$\mathbb{R}$	$\dot{x} = 0$
clock	clocks	$\mathbb{R}_0^+$	$\dot{x} = 1$
var	continuous variables	$\mathbb{R}$	according to invariants

2.1 Models, processes and declarations

A HMODEST specification consists of a sequence of *declarations* and a *process behaviour*. Declarations are constructed according to the following grammar:

$$
\begin{aligned}
dcl ::= {} & [\,\mathsf{patient} \mid \mathsf{impatient}\,]\ \mathsf{action}\ act;\ \mid && (actions)\\
& \mathsf{exception}\ excp;\ \mid && (exceptions)\\
& type\ var\ [\,= e\,];\ [\,\mathsf{der}(var) = d;\,]\ \mid && (variables)\\
& \mathsf{process}\ ProcName(t_1\ x_1, \ldots, t_k\ x_k)\ \{\, dcl\ P\,\} && (processes)
\end{aligned}
$$

where, for $0 < i \leq k$, *act*, *excp*, *var*, *ProcName* and x_i are identifiers (names), *type* and t_i are types (see Table 1 for the list of types[1]) and P is a process behaviour. A HMODEST specification can thus be treated as a process without parameters; when we refer to a process in the remainder of this paper, we mean a declared process or the specification's unnamed top-level process. The declarations of a process and its parameters define the following (finite) sets associated to the process:

- $\mathsf{Act}_P = \mathsf{PAct}_P \uplus \mathsf{IAct}_P \uplus \mathsf{Excp}_P \uplus \{\tau, \bot, \flat\}$, the set of actions partitioned into patient and impatient actions, exceptions and the error action $\bot$, the break action $\flat$, and the silent action τ;
- Var_P, the set of variables, which contains both declared variables as well as the process' parameters.

Example 1 As a running example, we will consider a standard hybrid system (originating from [29], where it was used without a MODEST specification): A water tank that has a continuous inflow of water, but is not supposed to overflow. To achieve this, a pump that pumps water out of the tank (at a rate faster than the inflow) can be turned on or of. However, the pump will be destroyed if it draws in air, so it must be turned off before the tank runs dry. Figure 1 shows the outline of the HMODEST specification for this system. It consists of two processes `Tank` and `Controller` that model the tank, including the pump, and the (discrete) controller that repeatedly measures the water level to decide when to switch the pump on or off. The two actions `on` and `off` are used by the controller to switch the state of the pump in the tank. The water level in the tank is represented by the continuous variable `level`. This is a global variable (initially set to 6.5) shared between `Tank` and

[1] Our implementation also supports fixed-size arrays and user-defined data structures, which are technical extensions but not conceptually relevant for this paper.

```
action on, off;
var level = 6.5;

process Controller()
{
   real measurement;

   ...
}
```

```
process Tank()
{
   process On()  { ... }
   process Off() { ... }

   Off()
}
```

```
par {
   :: Tank()
   :: Controller()
}
```

Fig. 1 Structure of the HMODEST specification of the water tank system

Table 2 Expression categorisation

Category	Type	Derivatives	Nondeterministic	Sampling
Exp	any	✔	✔	✔
Sxp	any	✗	✔	✔
Ixp	bool	✔	✔	✗
Bxp	bool	✗	✗	✗
Axp	$\mathbb{R}$	✗	✗	✗

`Controller` so as to allow its measurement in the controller. The two processes `Tank` and `Controller` execute in parallel, i.e. asynchronously except when they synchronise on the shared actions `on` or `off`. (A formal semantics of the par construct is provided in Sect. 3.)

2.2 Expressions

Let Exp be the set of expressions containing variables in some set of variables Var. We distinguish four important subsets of Exp based on the expressions' types and whether they can contain references to a variable's first derivative (using the der construct), whether subexpressions with nondeterministic values (using the any construct) are allowed—we call such expressions nondeterministic—and whether sampling of values according to probability distributions is possible. The four subsets are

- Sxp: expressions that do not contain references to derivatives, but may be nondeterministic and use sampling, for example $x + \mathsf{Uniform}(0, 2) + \mathsf{any}(y \mid x + y == z)$ where $\mathsf{Uniform}(x, y)$ denotes sampling from the uniform distribution over the interval $[x, y]$ and $\mathsf{any}(x \mid e)$ nondeterministically selects a value v such that $e[x/v]$ evaluates to *true*;
- Ixp: Boolean expressions, which evaluate to *true* or *false*, may be nondeterministic, and may contain references to derivatives, e.g. $\mathsf{der}(x) \leq 4$ to specify the invariant that the first derivative of x (also denoted $\dot{x}$) must not exceed 4;
- Bxp $\subseteq$ Ixp: simple Boolean expressions such as $i == 1$, without derivatives, nondeterminism or sampling, and
- Axp: arithmetic expressions such as $2.5 + x$ or $\mathsf{ceil}(y)$, which evaluate to values in $\mathbb{R}$ and do neither contain derivatives, nor nondeterminism, nor sampling.

This categorisation of expressions is summarised in Table 2.

Variables can initially be assigned the value of an expression $e \in$ Sxp explicitly; otherwise, they are implicitly initialised to a default value, typically zero. The first derivative of a continuous variable x, denoted $\mathsf{der}(x)$ in HMODEST or $\dot{x}$ in this paper, can also be constrained as part of the declarations, but this is merely a syntactic shortcut: $\mathsf{der}(x) = d;$, where

d is an expression that does not contain sampling, is semantically equivalent to replacing the process behaviour P with $\mathsf{invariant}(\mathsf{der}(x) == d)\ \{\ P\ \}$. Note that the der operator is only valid for continuous variables and, in particular, cannot be used with variables of type clock or real.

We omit a full grammar for expressions in this paper and only point out the possibility of including the any and der operators as well as probability distributions; other than that, MODEST expressions are mostly a subset of the expressions allowed in typical programming languages like C, C# or Java.

Example 2 In the water tank example, the controller measures the water level in the tank in regular intervals. This measurement is subject to a measurement error: The actual value measured by the controller is sampled according to a normal distribution with the actual value as mean and a standard deviation of 1. The corresponding HMODEST code snippet is

$$\texttt{measurement} = \mathsf{Normal}(\texttt{level}, 1).$$

2.3 Process behaviours

The process behaviours are constructed according to the following grammar:

$$\begin{aligned} P ::= {} & \mathit{act} \mid \mathsf{stop} \mid \mathsf{abort} \mid \mathsf{break} \mid P_1;\ P_2 \mid \\ & \mathsf{when}(b)\ P \mid \mathsf{urgent}(b)\ P \mid \mathsf{invariant}(i)\ P \mid \mathsf{invariant}(i)\ \{\ P\ \} \mid \\ & \mathsf{alt}\ \{\ ::P_1 \ldots ::P_k\ \} \mid \mathsf{do}\ \{\ ::P_1 \ldots ::P_k\ \} \mid \mathsf{par}\ \{\ ::P_1 \ldots ::P_k\ \} \mid \\ & \mathit{act}\ \mathsf{palt}\ \{\ :w_1 \colon \mathit{asgn}_1;\ P_1\ \ldots\ :w_k \colon \mathit{asgn}_k;\ P_k\ \} \mid \\ & \mathsf{throw}(\mathit{excp}) \mid \mathsf{try}\ \{\ P\ \}\ \mathsf{catch}\ \mathit{excp}_1\ \{\ P_1\ \}\ \ldots\ \mathsf{catch}\ \mathit{excp}_k\ \{\ P_k\ \} \mid \\ & \mathsf{relabel}\ \{\ I\ \}\ \mathsf{by}\ \{\ G\ \}\ \{\ P\ \} \mid \mathsf{extend}\ \{\ H\ \}\ \{\ P\ \} \mid \\ & \mathit{ProcName}(e_1, \ldots, e_k) \end{aligned}$$

where, with Exp containing the expressions over Var_P, $\mathit{act} \in \mathsf{PAct}_P \cup \mathsf{IAct}_P \cup \{\tau\}$, $\mathit{excp} \in \mathsf{Excp}_P$, $\mathit{excp}_i \in \mathsf{Excp}_P$, $b \in \mathsf{Bxp}$, $i \in \mathsf{Ixp}$, $w_i \in \mathsf{Axp}$, $H \subseteq \mathsf{PAct}_P \cup \mathsf{IAct}_P$ is a set of observable actions, I and G are vectors of equal length which have elements in $\mathsf{Act}_P \setminus \{\flat, \perp\}$ such that all elements in I are pairwise different and not equal to τ, and the asgn_i are type-consistent[2] assignments of the form

$$\{= x_1 = e_1, \ldots, x_k = e_k =\}$$

with $x_i \in \mathsf{Var}$ and $e_i \in \mathsf{Sxp}$ for $1 \le i \le k$. An assignment of the form described above represents a function $f = \{\, x_1 \mapsto e_1, \ldots, x_k \mapsto e_k \,\} \in \mathsf{Asgn}$, where $\mathsf{Asgn} = \mathsf{Var} \to \mathsf{Sxp}$, that maps each x_i to e_i and all other variables x to their original value (i.e. to the expression x). In particular, this means that the values of several variables are changed in one atomic step by an assignment. In order to simplify the semantics of process calls ($\mathit{ProcName}(e_1, \ldots, e_k)$), we assume that every process call corresponds to a unique process declaration, which can be achieved by renaming in any case.

[2]Since we omitted the details of the expression syntax, we assume type correctness in assignments, guards, weights etc. instead of providing the (standard) type checking rules in detail.

```
process On()  { invariant(der(level) == -2) alt { :: on; On() :: off; Off() } }
process Off() { invariant(der(level) ==  1) alt { :: on; On() :: off; Off() } }

Off()
```

Fig. 2 Behaviour of the `Tank` process

The most basic process behaviours are *act*, which simply performs an action and then terminates, stop, which does not perform any activity (resulting in a deadlocked system), abort, which indicates an unhandled error by performing action $\bot$ in an infinite loop, and break, which issues the special $\flat$ action that is used to break out of a do loop. Process behaviours can be combined in a sequential composition using the ; operator: As soon as the left-hand process behaviour terminates successfully (such as *act* by performing its action), the right-hand behaviour takes over. Process behaviours can be decorated with deadlines that restrict the passage of time using the urgent keyword, with guards that represent enabling conditions using the when keyword, and with invariants that control the passage of time and the evolution of continuous variables using the invariant keyword. The alt construct allows the specification of nondeterministic choices between different behaviours; the do construct is a variant of alt that restarts from the beginning when the chosen behaviour terminates successfully as long as no $\flat$ action is issued. The palt construct assigns weights to process behaviours, resulting in a probabilistic choice. Using the parallel composition par, a set of process behaviours can be specified to execute concurrently, possibly synchronising on certain actions. The concept of exceptions, well known from programming languages like Java or C#, is present in MODEST through the throw and try-catch constructs. In order to modify the alphabet of a process behaviour, which is relevant for synchronisation in parallel compositions, the relabel construct can be used to rename actions or exceptions, including hiding of actions by renaming them to τ, and the extend construct can be used to include actions not actually present in a process behaviour to its alphabet. Finally, a named process can be called using the $\mathit{ProcName}(e_1, \ldots, e_k)$ syntax. A formal semantics for all process behaviours will be given in Sect. 3.

Example 3 The process behaviour of the `Tank` process is given in Fig. 2. It alternates between waiting for the `on` signal while the pump is off (thus the water level increases at a rate of 1 unit of water (e.g. 1 litre) per unit of time (e.g. seconds)) and waiting for the `off` signal while the pump is on. We assume the pump to be able to pump off 3 l of water per second, so factoring in the continuous inflow, the water level decreases at a rate of 2 l/s when the pump is on. If an unexpected signal is received, e.g. signal `on` when the pump is already on, that signal is ignored—this ensures *input-enabledness* on signals `on` and `off`, i.e. these signals can always be received and are never blocked by the `Tank` process. Initially, the pump is off.

2.3.1 Shorthands

Two useful shorthands for more complex process behaviours are the hide and delay constructs. Using hide, which is useful to modify the action alphabet and thus the synchronisation interface of a process behaviour for parallel composition, is equivalent to relabelling a set of actions to the silent action τ:

$$\mathsf{hide}\ \{\ H\ \}\ \{\ P\ \} \stackrel{\text{def}}{=} \mathsf{relabel}\ \{\ H\ \}\ \mathsf{by}\ \{\tau, \ldots, \tau\}\ \{\ P\ \}.$$

```
do {
   :: delay(1.0) {= measurement = Normal(level, 1) =};
      delay(0.1) alt {
         :: when(measurement <= 5) off
         :: when(measurement >= 8) on
         :: when(measurement < 8 && measurement > 5) tau
      }
}
```

Fig. 3 Behaviour of the `Controller` process

The idea behind the delay construct is to provide an easy way to specify that a certain process behaviour should be executed after some precise amount of time. It can be expanded to a process call to newly introduced processes D_*:

$$\mathsf{delay}(e_{delay}, e_{cond})\ P \stackrel{\text{def}}{=} D_{e_{cond}}(e_{delay})$$

where e_{delay} is a deterministic expression in Sxp, $e_{cond} \in$ Bxp and process $D_{e_{cond}}$ is implicitly declared as

$$\mathsf{process}\ D_{e_{cond}}(\mathsf{real}\ x)\,\{\,\mathsf{clock}\ c;\ \ \mathsf{when}(c \geq x \wedge e_{cond})\ \mathsf{urgent}(c \geq x \wedge e_{cond})\ P\ \}.$$

Intuitively, $\mathsf{delay}(e_{delay}, e_{cond})\ P$ delays the initial behaviour of P until e_{delay} time units have passed and condition e_{cond} becomes true; at that point, the initial behaviour of P becomes urgent, so that it is performed without further delay. Condition e_{cond} is part of the shorthand because it cannot be added by hand since writing e.g. $\mathsf{urgent}(c \geq x)\ \mathsf{urgent}(e_{cond})$ would be equivalent to $\mathsf{urgent}(c \geq x \vee e_{cond})$.

Example 4 Consider again the water tank example. The `Controller` process uses the delay shorthand to perform a measurement precisely 1 second after the last one was processed. In addition, it specifies that reacting to a new value, i.e. deciding what signal to send to the tank, needs an additional 0.1 s. The resulting process behaviour is shown in Fig. 3. The implemented control strategy is simple: Once the measurement indicates that the tank contains at least 8 l of water, the pump is switched on; it is switched off again as soon as a water level of at most 5 l is measured. If the measured water level is between these two bounds, no signal is sent to the tank, i.e. the pump is left in its current state (action `tau`).

2.4 Properties

Properties, quantities of interest to be computed for a HMODEST specification, may be included as part of a declarations section. In this paper, the only kind of properties we consider are queries for the probability of reaching a set of *unsafe* states, which can be expressed in MODEST's property language as $\mathsf{Pmax}(\Diamond\ e)$ where e is a deterministic expression that characterises the unsafe states. The expression e may be of the form $e' \wedge \mathsf{time} \leq T$ to compute the probability of reaching the states characterised by e' within time bound T. Since our models may be nondeterministic, we ask for Pmax, which denotes the *maximum* probability over all schedulers, i.e. the probability over all possible resolutions of nondeterminism.

Example 5 In our water tank example, we want to make sure that the tank never overflows, and that it never reaches a level so low that the pump is in danger of drawing in air. Due

to the normally distributed measurement error, the model is stochastic, so we consider the probability that this happens. In addition, since measurements could, with a low probability, be very far from the real value, we can be sure that for infinite runs of the system, the water will eventually reach an unsafe level. However, the probability that this happens within the lifetime of the system should be negligible, so for a time bound T, the property of interest is

$$\mathsf{Pmax}(\Diamond\,(\texttt{level} > 12 \vee \texttt{level} < 1) \wedge \mathsf{time} \leq T),$$

assuming that the tank can hold 12 l of water and that 1 l is a safe lower level for operating the pump.

3 Symbolic semantics

In this section, we will define the symbolic semantics of a given HMODEST process, which is the first of two steps in defining a semantics for HMODEST. It consists in transforming the process calculus constructs of a process into a stochastic hybrid automaton, a model described in Sect. 3.1. In this stochastic hybrid automaton, the parts of the model not related to process-calculus-based definitions, such as model variables or assignments, will still be maintained in a symbolic form. In absence of non-tail-recursive process calls, the automaton will stay finite, so that it is possible to build it explicitly. The transformation of this symbolic automaton into a concrete, usually infinite, model will be the second step, presented in Sect. 4.

3.1 Stochastic hybrid automata

Stochastic hybrid automata represent the behaviour of a HMODEST process in a symbolic way. This allows them to remain a finite model, yet represent infinite behaviour. The definition we use is obtained from similar definitions of non-stochastic hybrid automata, like the ones in [4, 31, 38], by adding stochastic behaviour to the edges of automata. Apart from the addition of transition labels and urgency constraints, it equals the definition of [29], which in turn was extended from [63] and [59].

Definition 1 (SHA) A *stochastic hybrid automaton* (SHA) is a tuple

$$(\mathsf{Loc}, \mathsf{Inv}, \ell_0, \mathsf{Act}, \mathsf{Var}, v_0, \rightarrow)$$

where

- Loc is a finite set of *locations*,
- $\mathsf{Inv}\colon \mathsf{Loc} \to \mathsf{Ixp}$ maps each location to an invariant,
- $\ell_0 \in \mathsf{Loc}$ is the initial location,
- $\mathsf{Act} = \mathsf{PAct} \uplus \mathsf{IAct} \uplus \mathsf{Excp} \uplus \{\tau, \bot, \flat\}$ is a finite set of *actions*, partitioned as before,
- Var is a finite set of typed variables,
- $v_0 \in \mathsf{Var} \to \mathsf{Sxp}$ maps each variable to an expression describing its initial value, and
- $\rightarrow \subseteq \mathsf{Loc} \times \mathsf{Act} \times \mathsf{Bxp} \times \mathsf{Bxp} \times \mathsf{Wxp}$ is the finite *edge relation*

with $\mathsf{Wxp} = \mathsf{Asgn} \times \mathsf{Loc} \to \mathsf{Axp}$. The type of a variable v can be retrieved as $\mathit{type}(v)$, which can be extended to apply to expressions in the usual way.

An edge $(\ell, a, g, d, \mathscr{W}) \in \rightarrow$ consists of a *source location* ℓ, an *action label* a, a *guard* g that determines when the edge is enabled (i.e. when it can be taken), a *deadline* (or *urgency*

Table 3 Models overview

	SHA	STA	PHA	PPTA	PTA	PA	SHS	DTSHS	NLMP
discrete stochastics	✔	✔	✔	✔	✔	✔	✔	✔	✔
continuous stochastics	✔	✔	✗	✗	✗	✗	✔	✔	✔
discrete dynamics	✔	✔	✔	✔	✔	✔	✔	✔	✔
real time	✔	✔	✔	✔	✔	✗	✔	✗	✗
differential inclusions	✔	✗	✔	✗	✗	✗	✗	✗	✗
stochastic differential eqs.	✗	✗	✗	✗	✗	✗	✔	✗	✗
discrete nondeterminism	✔	✔	✔	✔	✔	✔	✗	✗	✔
continuous nondeterminism	✔	✔	✔	✔	✔	✗	✗	✗	✔

constraint) d that imposes a condition on the passage of time (time cannot pass when a deadline in the current location is satisfied), and a *target* $\mathscr{W}$ that symbolically represents a distribution over assignments and target locations. We will write $\ell \xrightarrow{a,g,d} \mathscr{W}$ to denote $(\ell, a, g, d, \mathscr{W}) \in \rightarrow$; if $\mathscr{W}(A, \ell) = 1$ and $\mathscr{W}(A', \ell') = 0$ for all $(A', \ell') \neq (A, \ell)$, $\mathscr{W}$ represents a Dirac distribution and we take the liberty to write $\mathscr{W}$ as $\mathscr{D}(A, \ell)$ for brevity.

SHA relates to a wide class of other hybrid formalisms. An overview of these models and their restrictions compared to SHA is given in Table 3. We consider stochastic hybrid automata (SHA, as in this paper, see also [29]), stochastic timed automata (STA [17]; for a similar model also see [45]), the restriction to which we exemplify in definitions 2 and 3, probabilistic hybrid automata (PHA [59, 63]), probabilistic priced timed automata (PPTA [11]), probabilistic timed automata (PTA [46]), probabilistic automata (PA [57]), stochastic hybrid systems (SHS, in the sense of a work by Hu *et al.* [43]), discrete-time stochastic hybrid systems (DTSHS [1]; note that although the model itself does not have continuous dynamics, it is usually used to approximate systems with stochastic differential equations by step transitions with normally distributed targets; there is also an extension with nondeterminism (control) [2]), and nondeterministic labelled Markov processes (NLMP [25, 61], as in Definition 5; there is also the restricted model where nondeterminism occurs only between actions, but not within transitions of the same label [27, 52]). As seen from the table, these models are strictly less expressive than SHA. An exception are the SHS by Hu *et al.*, as they include stochastic differential equations. We did not include these type of dynamics in our framework, because currently it does not allow for the analysis of such models.

Definition 2 (Clock constraints) Consider a set of *clock variables* $Ck \subseteq \mathsf{Var}$, i.e. a set of variables of type clock. *Clock constraints* are Boolean expressions of the form

$$CC ::= b \mid \mathit{true} \mid \mathit{false} \mid CC \wedge CC \mid CC \vee CC \mid \neg CC \mid c \sim e \mid c_1 \sim c_2,$$

where $b \in \mathsf{Bxp}$, $e \in \mathsf{Axp}$, b and e do not contain clock variables, $\sim \in \{>, \geq, <, \leq, =, \neq\}$ and $c, c_1, c_2 \in Ck$. A *clock assignment* is an assignment $A \in \mathsf{Asgn}$ such that if $x \in Ck$, then either $A(x) = 0$ or $A(x) = x$.

Definition 3 (STA) A *stochastic timed automaton* (STA [17]) is a stochastic hybrid automaton $(\mathsf{Loc}, \mathsf{Inv}, \ell_0, \mathsf{Act}, \mathsf{Var}, v_0, \rightarrow)$ with $\mathsf{Var} = Ck \uplus Dv$ where Ck is a set of clock variables and Dv is a set of (non-continuous) real variables such that

– all assignments in $\rightarrow$ are clock assignments,
– guards and urgency constraints are clock constraints,
– there is no occurrence of the any construct in expressions or assignments, and
– for all $\ell \in \mathsf{Loc}$, we have $\mathsf{Inv}(\ell) = \mathit{true}$.

3.2 Semantics of a HMODEST process

The original symbolic semantics of MODEST was given in terms of STA [17]. The main ingredient for the extension to a SHA semantics, which we present here, is the addition of two first-level invariant constructs that replace the previous shorthand of the same name. The shorthand mapped invariant(i) P to

$$\mathsf{alt}\,\{\,::\mathsf{when}(i)\ P\ \ ::\mathsf{urgent}(\neg\, i)\ \mathsf{when}(\mathit{false})\ \mathsf{throw}(\mathit{excp}_{\mathit{invariant}})\,\}$$

where $\mathit{excp}_{\mathit{invariant}}$ was a new exception only used for this purpose. The shorthand took advantage of the fact that guards do not influence deadlines, so the deadline $\neg\, i$ would still take effect even though the edge it is associated to is disabled. Since this construction simulated an invariant using a deadline, it could not be used to represent all possible invariants: there are some invariants that cannot be represented as deadlines [36], including the practically useful case of invariants with a strict less-than comparison of a clock variable and a constant such as $c < 3$.

The semantics presented here also corrects several minor issues with the original STA semantics; for example, assignments are composed using $\circ$ instead of $\cup$ in several places and the semantics for par and palt no longer involves any exceptions, which originally led to the order of the parallel behaviours being relevant for the semantics, which went against the intuition of an associative and commutative parallel composition operator.

Definition 4 (Symbolic semantics) The *symbolic semantics* of a HMODEST process P with process behaviour Q is the SHA $(\mathsf{Loc}, \mathsf{Inv}, \ell_0, \mathsf{Act}, \mathsf{Var}, v_0, \rightarrow)$ where

– Act and Var are the union of the sets Act_P and Var_P of P and the sets Act and Var given by the symbolic semantics of the processes that are called from within P's process behaviour,
– $v_0 = \mathbf{A}(Q) \circ v_{\text{decl}}$ where v_{decl} represents the initial values assigned to all variables in Var according to their declarations, $\mathbf{A}$ is the *assignment collecting function* as defined in Table 5 and the $\circ$ operator denotes the sequential composition of assignments, defined as

$$(A_2 \circ A_1)(x) \stackrel{\text{def}}{=} A_2(x)\big[x_1/A_1(x_1), \ldots, x_n/A_1(x_n)\big]$$

for $\mathsf{Var} = \{x_1, \ldots, x_n\}$.
– $\ell_0 = Q$,
– $\mathsf{Inv}(\ell) = \mathbf{I}(\ell)$ as defined in Table 4,
– the edge relation $\rightarrow$ is given by the inference rules presented below, and
– the set Loc of locations is the set of reachable process behaviours according to $\rightarrow$.

Note that the SHA corresponding to a HMODEST process contains invariants for locations, via the invariant collection function, as well as deadlines on edges, which will both be used in the concrete semantics (Sect. 4) to control the passage of time.

Table 4 The invariant collection function

$\mathbf{I}(P) = i \wedge \mathbf{I}(Q)$	if $P = \mathsf{invariant}(i)\ Q$
$\mathbf{I}(P) = tt$	if $P = act$, $P = act\ \mathsf{palt}\{:w_1\colon asgn_1;\ P_1 \cdots :w_k\colon asgn_k;\ P_k\}$, $P = \mathsf{stop}$, $P = \mathsf{abort}$, $P = \mathsf{break}$ or $P = \mathsf{throw}(excp)$
$\mathbf{I}(P) = \mathbf{I}(P_1)$	if $P = P_1;\ P_2$ or $P = \mathsf{auxdo}\{P_1\}\{P_2\}$
$\mathbf{I}(P) = \bigwedge_{i=1}^{k} \mathbf{I}(P_i)$	if $P = \mathsf{alt}\{::P_1 \ldots ::P_k\}$, $P = \mathsf{do}\{::P_1 \ldots ::P_k\}$ or $P = \mathsf{par}\{::P_1 \ldots ::P_k\}$
$\mathbf{I}(P) = \mathbf{I}(Q)$	if $P = \mathsf{when}(b)\ Q$, $P = \mathsf{urgent}(b)\ Q$, $P = \mathsf{relabel}\{I\}\,\mathsf{by}\{G\}\ Q$, $P = \mathsf{extend}\{H\}\ Q$, $P = \mathsf{try}\{Q\}\ \mathsf{catch}\ excp_1\{P_1\} \ldots \mathsf{catch}\ excp_k\{P_k\}$ or $P = ProcName(e_1, \ldots, e_k)$ and process $ProcName$ is declared as $\mathsf{process}\ ProcName(t_1\ x_1, \ldots, t_n\ x_k)\{Q\}$

Table 5 The assignment collecting function [17]

$\mathbf{A}(P) = \varnothing$	if P has one of the following forms: act, stop, abort, throw($excp$), break or $act\ \mathsf{palt}\{:w_1\colon a_1;\ P_1\ \ldots\ :w_k\colon a_k;\ P_k\}$
$\mathbf{A}(P) = \mathbf{A}(Q)$	if P has one of the following forms: $Q; Q'$, when(b) Q, urgent(b) Q, invariant(i) Q, invariant(i) $\{Q\}$, try $\{Q\}$ catch e_1 $\{P_1\}$ … catch e_k $\{P_k\}$, relabel $\{I\}$ by $\{G\}$ Q or extend $\{H\}$ Q
$\mathbf{A}(P) = \bigcup_{i=1}^{k} \mathbf{A}(P_i)$	if P has one of the following forms: alt $\{::P_1 \ldots ::P_k\}$, do $\{::P_1 \ldots ::P_k\}$, or par $\{::P_1 \ldots ::P_k\}$
$\mathbf{A}(ProcName(e_1, \ldots, e_k)) = \mathbf{A}(Q) \circ \{x_1 = e_1, \ldots, x_k = e_k\}$	if $ProcName$ is declared as process $ProcName(t_1\ x_1, \ldots, t_n\ x_k)\ \{Q\}$

3.3 Inference rules

act, abort, break The inference rules for performing an action, including the special action $\flat$ to break out of a loop with the break construct, are straightforward:

$$\frac{}{act \xrightarrow{act,tt,ff} \mathscr{D}(\varnothing, \checkmark)}\ (\text{act}) \qquad \frac{}{\mathsf{break} \xrightarrow{\flat,tt,ff} \mathscr{D}(\varnothing, \checkmark)}\ (\text{break}) \qquad \frac{}{\mathsf{abort} \xrightarrow{\bot,tt,ff} \mathscr{D}(\varnothing, \mathsf{abort})}\ (\text{abort})$$

The abort process, which *can* be specified syntactically but more usually occurs as the consequence of an unhandled exception (see below), simply performs the unhandled error action $\bot$ over and over again. The *successfully terminated process* $\checkmark$ is only used as part of the semantics and cannot be specified syntactically. There is no inference rule for the stop process since its semantics is precisely to do nothing.

Conditions Any process behaviour can be decorated with a guard using the when construct, with a deadline using the urgent construct, and with an invariant using the invariant construct. Guards, deadlines and the invariant(i) P form of the invariant construct only affect the first, immediate edges resulting from the decorated behaviour and then disappear—

$$\frac{P \xrightarrow{a,g,d} \mathscr{W}}{\mathsf{when}(b)\ P \xrightarrow{a,g\wedge b,d} \mathscr{W}}\ (\text{when}) \qquad \frac{P \xrightarrow{a,g,d} \mathscr{W}}{\mathsf{urgent}(b)\ P \xrightarrow{a,g,d\vee b} \mathscr{W}}\ (\text{urgent}) \qquad \frac{P \xrightarrow{a,g,d} \mathscr{W}}{\mathsf{invariant}(i)\ P \xrightarrow{a,g,d} \mathscr{W}}\ (\text{inv})$$

—while $\mathsf{invariant}(i)\,\{\,P\,\}$ is a static operator, i.e. it does not disappear after following one edge; with $Q(P) = \mathsf{invariant}(i)\,\{\,P\,\}$, its inference rule reads:

$$\frac{P \xrightarrow{a,g,d} \mathscr{W}}{Q(P) \xrightarrow{a,g,d} \mathscr{W} \circ \mathbf{M}_{\mathrm{inv}}^{-1}}\ (\mathrm{sinv}) \qquad \text{where } \mathbf{M}_{\mathrm{inv}}(A, P') \stackrel{\mathrm{def}}{=} \begin{cases} \langle A, Q(P')\rangle & \text{if } P' \neq \checkmark \\ \langle A, P'\rangle & \text{if } P' = \checkmark. \end{cases}$$

The inference rules for invariant ignore the actual invariant expression i because it does not become part of the edge relation, but is instead preserved as part of the function that maps each location to an invariant.

Sequential composition A process behaviour P' can be performed only after another process behaviour P has successfully terminated when they are composed using the ; operator for sequential composition:

$$\frac{P \xrightarrow{a,g,d} \mathscr{W}}{P\,;\,Q \xrightarrow{a,g,d} \mathscr{W} \circ \mathbf{M}_{;}^{-1}}\ (\mathrm{seq}) \qquad \text{where } \mathbf{M}_{;}(A, P') \stackrel{\mathrm{def}}{=} \begin{cases} \langle A,\ P';\ Q\rangle & \text{if } P' \neq \checkmark \\ \langle \mathbf{A}(Q) \circ A,\ Q\rangle & \text{if } P' = \checkmark. \end{cases}$$

Nondeterministic choice A nondeterministic choice between several process behaviours is provided by the alt keyword:

$$\frac{P_i \xrightarrow{a,g,d} \mathscr{W}_i \quad (0 < i \leq k)}{\mathsf{alt}\,\{ :: P_1 \ldots :: P_k\} \xrightarrow{a,g,d} \mathscr{W}_i}\ (\mathrm{alt})$$

Loops The semantics of the do construct is defined via the auxiliary auxdo construct, which is not part of the MODEST syntax. It is used to keep track of the original behaviour of the loop which must be restored after each iteration:

$$\mathsf{do}\,\{ :: P_1 \ldots :: P_k\} \stackrel{\mathrm{def}}{=} \mathsf{auxdo}\,\{\mathsf{alt}\,\{ :: P_1 \ldots :: P_k\}\}\,\{\mathsf{alt}\,\{ :: P_1 \ldots :: P_k\}\}$$

The semantics of auxdo is defined in two inference rules: The first one handles the case that the break action is performed to jump out of the loop, while the second rule defines the semantics of performing a step within the loop, including proceeding to the next iteration once the last step has been performed.

$$\frac{P \xrightarrow{\flat,g,d} \mathscr{W}}{\mathsf{auxdo}\,\{\,P\,\}\,\{\,Q\,\} \xrightarrow{\tau,g,d} \mathscr{D}(\varnothing, \checkmark)}\ (\mathsf{breakout}) \qquad \frac{P \xrightarrow{a,g,d} \mathscr{W} \quad (a \neq \flat)}{\mathsf{auxdo}\,\{\,P\,\}\,\{\,Q\,\} \xrightarrow{a,g,d} \mathscr{W} \circ \mathbf{M}_{\mathrm{do}}^{-1}}\ (\mathsf{auxdo})$$

where

$$\mathbf{M}_{\mathrm{do}}(A, P') \stackrel{\mathrm{def}}{=} \begin{cases} \langle A, \mathsf{auxdo}\,\{\,P'\,\}\,\{\,Q\,\}\rangle & \text{if } P' \neq \checkmark \\ \langle \mathbf{A}(Q) \circ A, \mathsf{auxdo}\,\{\,Q\,\}\,\{\,Q\,\}\rangle & \text{if } P' = \checkmark. \end{cases}$$

Process calls Let process *ProcName* be declared as

$$\mathsf{process}\ \mathit{ProcName}(t_1\ x_1, \ldots, t_n\ x_k)\ \{P\}.$$

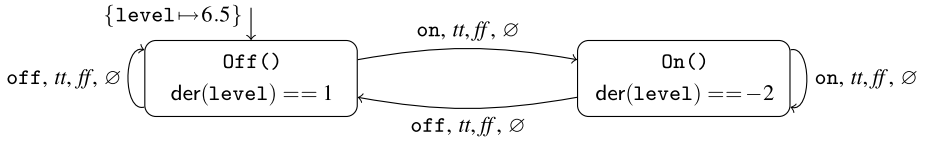

Fig. 4 SHA for the `Tank` process

The process call $ProcName(e_1, \ldots, e_k)$ then behaves like P where the variables $x_1, \ldots, x_k$ are assigned the values of the expressions $e_1, \ldots, e_k$. The assignment of variables is performed just before the process call is executed, which is ensured by the assignment collecting function (see Table 5). The inference rule for process calls is then

$$\frac{P \xrightarrow{a,g,d} \mathscr{W}}{ProcName(e_1, \ldots, e_k) \xrightarrow{a,g,d} \mathscr{W}}\ \text{(call)}\ .$$

We note that this rule does not include any renaming or stacking of variables, but this is only possible because of the assumption of unique process declarations for process calls in the previous section, and it only works correctly for tail-recursive models. To give a semantics for arbitrarily recursive models, we would need to extend the inference rule above and the assignment collecting function to include the necessary renamings there. We would also have to account for (countably) infinite sets of variables in the definition of SHA.

Example 6 The SHA corresponding to process `Tank` in our running example is shown in Fig. 4, where each location is labelled by its process behaviour (first line) and the corresponding invariant according to the invariant collection function **I** (second line). We see that the (symbolic) semantics indeed matches the (intuitive) description of the tank's behaviour that we gave in Sect. 2.3. We can indeed prove the edges using the inference rules that we have seen so far; the proof for the location-changing edge labelled `on` is as follows:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{\texttt{on} \xrightarrow{\texttt{on},tt,ff} \mathscr{D}(\varnothing, \checkmark)}\ \text{(act)}}{\texttt{on};\ \texttt{On()} \xrightarrow{\texttt{on},tt,ff} \mathscr{D}(\varnothing, \texttt{On()})}\ \text{(seq)}}{\mathsf{alt}\,\{::\texttt{on};\ \texttt{On()}\ ::\texttt{off};\ \texttt{Off()}\} \xrightarrow{\texttt{on},tt,ff} \mathscr{D}(\varnothing, \texttt{On()})}\ \text{(alt)}}{\mathsf{invariant}(\texttt{der(level)}==1)\ \mathsf{alt}\,\{::\texttt{on};\ \texttt{On()}\ ::\texttt{off};\ \texttt{Off()}\} \xrightarrow{\texttt{on},tt,ff} \mathscr{D}(\varnothing, \texttt{On()})}\ \text{(inv)}}{\texttt{Off()} \xrightarrow{\texttt{on},tt,ff} \mathscr{D}(\varnothing, \texttt{On()})}\ \text{(call)}$$

The proofs for the other edges of the `Tank` process' behaviour follow the same pattern.

Example 7 The SHA for the `Controller` process is shown in Fig. 5, where P is the original auxdo for the entire original loop, while P' represents the remainder of the loop. We omit proofs for the edges here since they are very similar to those in the previous example. Note, however, how the expansion of the delay shorthand has introduced new variables, guards and deadlines that together make sure that precisely 1 (0.1) time units elapse before a new measurement is taken (before the reaction to the measurement is performed).

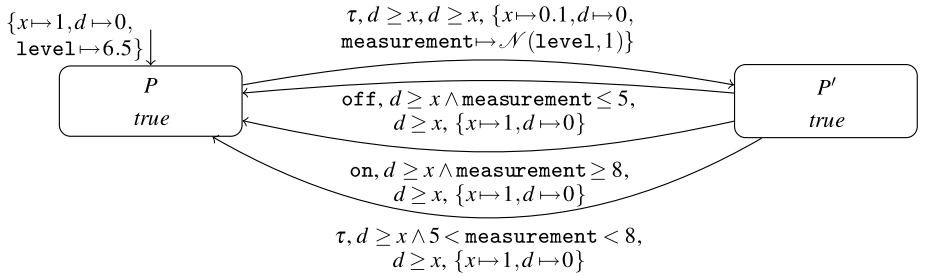

Fig. 5 SHA for the `Controller` process

Probabilistic choice The construct for probabilistic choice, palt, is *action-prefixed*: An action *act* is performed and the target of the edge labelled *act* is then the (symbolic) probability distribution over the weighted alternatives that make up the body of the palt. The inference rule thus reads as follows:

$$\frac{}{act\ \mathsf{palt}\ \{:w_1\colon asgn_1;\ P_1\ \ldots\ :w_k\colon asgn_k;\ P_k\} \xrightarrow{act,tt,ff} \mathscr{W}}\ (\text{palt})$$

where

$$\mathscr{W}\big(\mathbf{A}(P_i)\circ asgn_i,\, P_i\big) \stackrel{\text{def}}{=} \sum_{j=1}^{k} \mathbf{Ind}(i,\, j)\cdot w_j$$

and

$$\mathbf{Ind}(i,\, j) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \mathbf{A}(P_i)\circ asgn_i = \mathbf{A}(P_j)\circ asgn_j \wedge P_i = P_j \\ 0 & \text{otherwise.} \end{cases}$$

Note that $\mathscr{W}$ still is a symbolic function in $\mathsf{Wxp} = \mathsf{Asgn} \times \mathsf{Loc} \to \mathsf{Axp}$. Two problems that are not taken into account by this inference rule are that a weight w_i may be negative, and that the sum of all weights may be zero. These are considered modelling errors, i.e. the semantics of a (syntactically valid) MODEST model that contains one or more such errors is not defined. Tool support will check for this and reject such models.

Exceptions Once declared, an exception can be *thrown* ...

$$\frac{}{\mathsf{throw}(excp) \xrightarrow{excp,tt,ff} \mathscr{D}(\varnothing, \mathsf{abort})}\ (\text{throw})$$

... and either be caught or ignored by enclosing try-catch constructs of the form $Q(P) = \mathsf{try}\ \{P\}\ \mathsf{catch}\ excp_1\ \{P_1\}\ \ldots\ \mathsf{catch}\ excp_k\ \{P_k\}$; if caught, the specified exception handler will be executed:

$$\frac{P \xrightarrow{a,g,d} \mathscr{W} \quad (a \notin \{excp_1, \ldots, excp_k\})}{Q(P) \xrightarrow{a,g,d} \mathscr{W} \circ \mathbf{M}_{\text{try}}^{-1}}\ (\text{try}) \qquad \frac{P \xrightarrow{excp_i,g,d} \mathscr{W} \quad (0 < i \le k)}{Q(P) \xrightarrow{\tau,g,d} \mathscr{D}(\mathbf{A}(P_i),\, P_i)}\ (\text{catch})$$

where

$$\mathbf{M}_{\text{try}}(A,\, P') \stackrel{\text{def}}{=} \begin{cases} \langle A,\, Q(P')\rangle & \text{if } P' \neq \checkmark \\ \langle A, \checkmark\rangle & \text{if } P' = \checkmark. \end{cases}$$

Table 6 The alphabet of a process behaviour [17]

$\alpha(P) = \{act\} \setminus \{\tau\}$	if $P = act$
$\alpha(P) = \varnothing$	if $P = \mathsf{stop}$, $P = \mathsf{break}$, $P = \mathsf{abort}$ or $P = \mathsf{throw}(excp)$
$\alpha(P) = \alpha(Q)$	if $P = \mathsf{when}(b)\ Q$, $P = \mathsf{urgent}(b)\ Q$, $P = \mathsf{invariant}(i)\ Q$, $P = \mathsf{invariant}(i)\ \{Q\}$ or $P = ProcName(e_1, \ldots, e_k)$ and process $ProcName$ is declared as process $ProcName(t_1\ x_1, \ldots, t_n\ x_k)\ \{P\}$
$\alpha(P) = \alpha(P_1) \cup \alpha(P_2)$	if $P = P_1;\ P_2$
$\alpha(P) = \bigcup_{i=1}^{k} \alpha(P_i)$	if $P = \mathsf{alt}\ \{:: P_1 \cdots :: P_k\}$, $P = \mathsf{do}\ \{:: P_1 \cdots :: P_k\}$ or $P = \mathsf{par}\ \{:: P_1 \cdots :: P_k\}$
$\alpha(P) = \alpha(Q) \cup \bigcup_{i=1}^{k} \alpha(P_i)$	if $P = \mathsf{try}\ \{Q\}\ \mathsf{catch}\ excp_1\ \{P_1\}\ \ldots\ \mathsf{catch}\ excp_k\ \{P_k\}$
$\alpha(P) = \alpha(Q)[a_1/a'_1, \ldots, a_k/a'_k] \setminus \{\tau\}$	if $P = \mathsf{relabel}\ \{a_1, \ldots, a_k\}\ \mathsf{by}\ \{a'_1, \ldots, a'_k\}\ Q$
$\alpha(P) = \alpha(Q) \cup \{act_1, \ldots, act_k\}$	if $P = \mathsf{extend}\ \{act_1, \ldots, act_k\}\ Q$
$\alpha(P) = \alpha(act) \cup \bigcup_{i=1}^{k} \alpha(P_i)$	if $P = act\ \mathsf{palt}\ \{:w_1\colon asgn_1;\ P_1\ \ldots\ :w_k\colon asgn_k;\ P_k\}$

Parallel composition The process behaviours in a par construct run concurrently, synchronising on the actions in their common alphabet. The alphabet of a process is computed by function α as defined in Table 6. A parallel composition terminates successfully whenever all its components do so.

To define the semantics of parallel composition, we resort to the auxiliary operator $\|_B$, with $B \subseteq \mathsf{PAct} \cup \mathsf{IAct}$. The par construct is then defined as

$$\mathsf{par}\ \{:: P_1 \ldots :: P_k\} \stackrel{\text{def}}{=} \big(\ldots\big((P_1 \parallel_{B_1} P_2) \parallel_{B_2} P_3\big)\ldots\big) \parallel_{B_{k-1}} P_k$$

with

$$B_j = \left(\bigcup_{i=1}^{j} \alpha(P_i)\right) \cap \alpha(P_{j+1}).$$

The behaviour of $\|_B$ is then formally defined as follows:

Action $a \notin B$ (which is not intended to synchronise) can be performed autonomously, i.e., without the cooperation of the other parallel component:

$$\frac{P_1 \xrightarrow{a,g,d} \mathscr{W} \quad (a \notin B)}{P_1 \parallel_B P_2 \xrightarrow{a,g,d} \mathscr{W} \circ \mathbf{M}^{-1}_{\text{par}\,P_2}}\ (\text{lpar}) \qquad \frac{P_2 \xrightarrow{a,g,d} \mathscr{W} \quad (a \notin B)}{P_1 \parallel_B P_2 \xrightarrow{a,g,d} \mathscr{W} \circ \mathbf{M}^{-1}_{\text{par}\,P_1}}\ (\text{rpar})$$

where

$$\mathbf{M}_{\text{par}\,P}\big(A, P'\big) \stackrel{\text{def}}{=} \big\langle A, P' \parallel_B P \big\rangle \quad \text{and} \quad \checkmark \parallel_B \checkmark = \checkmark.$$

MODEST provides two synchronisation modes which depend on whether the action is patient or impatient. A process that wants to synchronise on a patient action always waits for its partner to be ready. Accordingly, its deadline needs to be relaxed to the requirements of the partner. However, a process that intends to synchronise on an impatient action is not willing to wait for the partner. Therefore, a deadline in an impatient synchronisation should be met as soon as one of the components meets its deadlines. If we let $\otimes_a = \wedge$ if $a \in \mathsf{PAct}$ and $\otimes_a = \vee$ if $a \in \mathsf{IAct}$, the inference rule for synchronisation thus reads:

$$\frac{P_1 \xrightarrow{a,g_1,d_1} \mathscr{W}_1 \quad P_2 \xrightarrow{a,g_2,d_2} \mathscr{W}_2 \quad (a \in B \cap (\mathsf{PAct} \cup \mathsf{IAct}))}{P_1 \parallel_B P_2 \xrightarrow{a,g_1 \wedge g_2,d_1 \otimes_a d_2} (\mathscr{W}_1 \times \mathscr{W}_2) \circ \mathbf{M}^{-1}_{\text{par}}}\ (\text{sync})$$

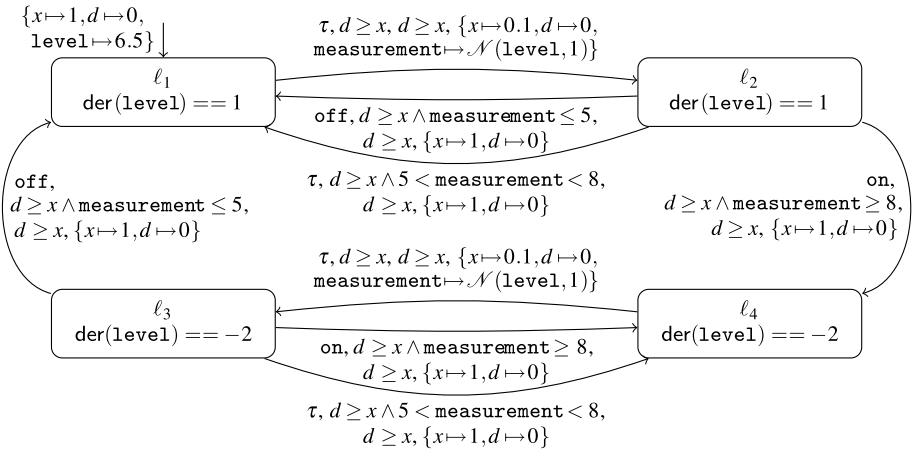

Fig. 6 SHA for the parallel composition of `Tank` and `Controller`

where $(\mathscr{W}_1 \times \mathscr{W}_2)(\alpha_1, \alpha_2) \stackrel{\text{def}}{=} \mathscr{W}_1(\alpha_1) \cdot \mathscr{W}_2(\alpha_2)$ for all α_1 and α_2—corresponding to the product of two probability spaces—and

$$\mathbf{M}_{\text{par}}\big(\langle A_1, P_1'\rangle, \langle A_2, P_2'\rangle\big) \stackrel{\text{def}}{=} \langle A_1 \cup A_2, P_1' \parallel_B P_2'\rangle \quad \text{if } A_1 \text{ and } A_2 \text{ are consistent}$$

where, as before, $\checkmark \parallel_B \checkmark = \checkmark$. The union of functions is defined as

$$(A_1 \cup A_2)(v) \stackrel{\text{def}}{=} \begin{cases} A_1(v) & \text{if } A_2(v) = v \\ A_2(v) & \text{if } A_1(v) = v \end{cases}$$

as long as A_1 and A_2 are *consistent*, i.e. $A_i(x) \neq x \Rightarrow A_{(i \bmod 2)+1}(x) = x$ for all $x \in \mathsf{Var}$. Inconsistent assignments are considered a modelling error.

Example 8 The SHA given as the symbolic semantics for the parallel composition of the behaviours of the `Tank` and `Controller` processes[3] is shown in Fig. 6. For clarity, we have omitted the process behaviours, labelling the locations only with their invariants and identifiers ℓ_i. In locations ℓ_1 and ℓ_2 (ℓ_3 and ℓ_4), the pump in the tank is off (on); the controller is reacting to a measurement in ℓ_2 and ℓ_3, while it waits before taking another measurement in the other two locations.

Alphabet manipulation The alphabet of a process can be modified with the extend and relabel constructs. The extend construct merely extends the alphabet of a process (see Table 6) and may affect behaviour only if it appears within the context of a par construct: For

[3]The semantics of par {::`Tank`() ::`Controller`()} itself contains additional locations because the two process calls are not syntactically equal to the behaviours of the processes called. The semantics shown above can be obtained as the semantics of the entire model by replacing the `do` construct in the `Controller` process by a (tail-)recursive process call and the call `Tank`() in the parallel composition by a direct call to `TankOff`().

$Q(P) = \mathsf{extend}\ \{act_1, \ldots, act_k\}\ P,$

$$\frac{P \xrightarrow{a,g,d} \mathscr{W}}{Q(P) \xrightarrow{a,g,d} \mathscr{W} \circ \mathbf{M}_{\text{ext}}^{-1}}\ (\text{extend}) \quad \text{where } \mathbf{M}_{\text{ext}}\big(A, P'\big) \stackrel{\text{def}}{=} \begin{cases} \langle A, Q(P')\rangle & \text{if } P' \neq \checkmark \\ \langle A, \checkmark\rangle & \text{if } P' = \checkmark. \end{cases}$$

The semantics for the relabel construct is as in traditional process algebra: Observable actions and exceptions are renamed according to a relabelling function, but the behaviour remains otherwise unchanged. For $Q(P) = \mathsf{relabel}\ \{a_1, \ldots, a_k\}\ \mathsf{by}\ \{a'_1, \ldots, a'_k\}\ P$, the inference rule is thus

$$\frac{P \xrightarrow{a,g,d} \mathscr{W} \quad f = [a_1/a'_1, \ldots, a_k/a'_k]}{Q(P) \xrightarrow{f(a),g,d} \mathscr{W} \circ \mathbf{M}_{\text{rel}}^{-1}}\ (\text{relabel})$$

where

$$\mathbf{M}_{\text{rel}}\big(A, P'\big) \stackrel{\text{def}}{=} \begin{cases} \langle A, Q(P')\rangle & \text{if } P' \neq \checkmark \\ \langle A, \checkmark\rangle & \text{if } P' = \checkmark. \end{cases}$$

4 Concrete semantics

The symbolic SHA semantics of the previous section defined the semantics of the process algebra constructs of HMODEST. It did however not give a meaning to assignments to model variables, and also did not consider the continuous behaviour of a HMODEST specification. In this section, we will give a semantics to SHA which takes these parts into account, thus giving a meaning to assignments and invariants. In contrast to the previous section however, which often gives a finite semantic model, the concrete semantics will usually be uncountably large. This is especially the case if non-trivial timed behaviour is specified. Thus, it is in general not possible to construct the concrete semantics explicitly by applying the second step of the transformation. In Sect. 6, we will give a sketch how to anyway prove properties of such models.

In the following, after a brief recap of the necessary definitions from stochastics, we will introduce the concrete model of nondeterministic labelled Markov processes and consider the semantics of types, variables and the expressions, including assignments and invariants, which constitute a SHA. We will then specify the timed and untimed transitions of the NLMP semantics based on these component semantics, and finally combine all of these steps to form a concrete NLMP semantics for SHA.

4.1 Stochastics recap

To define the concrete semantics, we need to shortly recap some notions and definitions from stochastics. A family Σ of subsets of a set Ω is a *σ-algebra*, provided $\Omega \in \Sigma$, and Σ is closed under complement and σ-union (countable union). A set $B \in \Sigma$ is then called *measurable*. Given a family of sets $\mathscr{A}$, by $\sigma(\mathscr{A})$ we denote the σ-algebra *generated* by $\mathscr{A}$, that is the smallest σ-algebra containing all sets of $\mathscr{A}$. The *Borel σ-algebra* over Ω is generated by the open subsets of Ω, and it is denoted $\mathscr{B}(\Omega)$. The pair (Ω, Σ) is called a *measurable space*.

A function $\mu\colon \Sigma \to [0, 1]$ is called *σ-additive* if $\mu(\biguplus_{i\in I} B_i) = \sum_{i\in I} \mu(B_i)$ for countable index sets I. We speak of a *probability measure* if $\mu(\Omega) = 1$. The *Dirac probability measure*

$\mathscr{D}(\omega)$ for $\omega \in \Omega$ is 1 only for $\{\omega\} \in \Sigma$. A function $f\colon \Omega_1 \to \Omega_2$ is Σ_1-Σ_2-*measurable* if every preimage of a measurable set is measurable, i.e. $f^{-1}(B) \in \Sigma_1$ for all $B \in \Sigma_2$.

We denote the set of probability measures on (Ω, Σ) by $\Delta(\Omega)$. It can be endowed with the σ-algebra $\Delta(\Sigma)$ [33] generated by the measures that, when applied to $B \in \Sigma$, give a value greater than $q \in \mathbb{Q} \cap [0, 1]$:

$$\Delta(\Sigma) \stackrel{\text{def}}{=} \sigma\big(\big\{\Delta^{>q}(B) \mid B \in \Sigma \wedge q \in \mathbb{Q}\big\}\big) \quad \text{where } \Delta^{>q}(B) \stackrel{\text{def}}{=} \big\{\mu \mid \mu(B) > q\big\}.$$

Note that $\Delta(\Sigma)$ is a set of sets of probability measures. Together with $\Delta(\Omega)$, it forms the measurable space $(\Delta(\Omega), \Delta(\Sigma))$.

Given a σ-algebra Σ, we define the *hit* σ*-algebra* [61] over Σ by

$$\mathscr{H}(\Sigma) \stackrel{\text{def}}{=} \sigma\big(\{H_B \mid B \in \Sigma\}\big) \quad \text{where } H_B \stackrel{\text{def}}{=} \{C \in \Sigma \mid C \cap B \neq \emptyset\}.$$

Thus, H_B consists of all measurable sets C which have a nonempty intersection with B. $\mathscr{H}(\Sigma)$ is then generated by all sets of sets $\{C_1, \ldots, C_n\}$ such that there is a set B which "hits" all C_i.

Given a finite index set I and a family $(\Sigma_i)_{i\in I}$ of σ-algebras, the *product* σ*-algebra* $\bigotimes_{i\in I} \Sigma_i$ is defined as

$$\bigotimes_{i\in I} \Sigma_i \stackrel{\text{def}}{=} \sigma\left(\left\{\bigtimes_{i\in I} B_i \;\middle|\; \forall i \in I\colon B_i \in \Sigma_i\right\}\right),$$

while for a family $(\mu_i)_{i\in I}$ of measures on Σ_i, the *product measure* is the uniquely defined probability measure $\bigotimes_{i\in I} \mu_i \in \Delta(\bigotimes_{i\in I} \Sigma_i)$ such that

$$\left(\bigotimes_{i\in I} \mu_i\right)\left(\bigtimes_{i\in I} B_i\right) \stackrel{\text{def}}{=} \prod_{i\in I} \mu_i(B_i) \quad \text{for all } B_i \in \Sigma_i,\ i \in I.$$

We extend this definition to families of sets of distributions $(M_i)_{i\in I}$ with $M_i \in \Delta(\Sigma_i)$ by

$$\bigotimes_{i\in I} M_i \stackrel{\text{def}}{=} \left\{\bigotimes_{i\in I} \mu_i \;\middle|\; \mu_i \in M_i \text{ for all } i \in I\right\}.$$

We use $\otimes$ as an infix operator on two σ-algebras, probability measures or sets of probability measures.

4.2 Nondeterministic labelled Markov processes

The semantics of a SHA is a nondeterministic labelled Markov process (similar to the definition by Wolovick [61], extended from [25]).

Definition 5 (NLMP) A *nondeterministic labelled Markov process (NLMP)* is a tuple of the form $(S, \Sigma_S, s_0, L, \Sigma_L, T)$ where

- S is a (possibly uncountable) set of *states*,
- Σ_S is a σ-algebra over S,
- $s_0 \in \Delta(\Sigma_S)$ is the *initial choice*,
- L is a (possibly uncountable) set of *labels*,
- Σ_L is a σ-algebra over L,

– $T\colon S \to (\Sigma_L \otimes \Delta(\Sigma_S))$ is the Σ_S-$\mathscr{H}(\Sigma_L \otimes \Delta(\Sigma_S))$-measurable *transition function*.

By T_a for $a \in L$, we denote the function $T_a\colon S \to \Delta(\Sigma_S)$ with $T_a(s) \stackrel{\text{def}}{=} T(s)_{|a}$.

Notice that in contrast to [61] and [25], we do not have a single initial state, but a set of probability measures representing a nondeterministic choice over initial distributions. This is necessary because the definition of HMODEST specifications allows for both nondeterministic as well as stochastic assignments over the initial variable values.

We need to equip the labels with a σ-algebra to ensure measurability of T. Even if for all labels a we have that T_a is measurable, it does not follow that their combination T is measurable, if the set of labels is uncountably large. For an example of an NLMP in which each individual T_a is indeed measurable, but T itself is not, see [61, Example 4.10]. However, from the measurability of the transition function T it follows that each T_a is Σ_S-$\mathscr{H}(\Delta(\Sigma_S))$-measurable [61]. Written without measurability requirements, T_a is a function with signature $S \to \mathscr{P}(\Delta(S))$. Thus, for each state $s \in S$ we obtain a set of distributions in $T_a(s)$. We write $s \overset{a}{\hookrightarrow}_T \mu$ if $\mu \in T_a(s)$ and $s \overset{a}{\hookrightarrow}_T s'$ if $\mathscr{D}(s') \in T_a(s)$. We call the latter kind of transitions *trivial*. If T is clear from the context, we write $\hookrightarrow$ instead of $\hookrightarrow_T$.

In a previous publication [29], we considered measurability requirements on the components of a SHA to ensure measurability of the automaton as a whole. We did however not base the components on a process calculus, as we do here. As a semantical model, we used a model which is almost equal to NLMP, except that transitions were not labelled. In the following, we will consider similar measurability restrictions on the model components to ensure the measurability of the SHA semantics of a given HMODEST specification.

4.3 Component semantics

For the definition of the symbolic semantics in Sect. 3, it was sufficient to treat variables and expressions as merely symbols. In this section, we give a meaning to these symbols and specify some auxiliary functions on them.

Assume we are given a SHA $(\mathsf{Loc}, \mathsf{Inv}, \ell_0, \mathsf{Act}, \mathsf{Var}, v_0, \to)$, and let $\mathsf{Var}' \subseteq \mathsf{Var}$. We start by defining the semantics of types and thus the possible values that a variable in Var can take:

Definition 6 (Type semantics) Let $Ty \stackrel{\text{def}}{=} \{type(x) \mid x \in \mathsf{Var}\}$. The *type semantics* $[\![t]\!]$ is a set of possible values for each $t \in Ty$. As a shortcut, the *variable domain Dom* for $x \in \mathsf{Var}$ is $Dom(x) \stackrel{\text{def}}{=} [\![type(x)]\!]$. We let $Dom(\mathsf{Var}) \stackrel{\text{def}}{=} \bigcup_{x \in \mathsf{Var}} Dom(x)$. We assume the existence of a σ-algebra Σ_x over $Dom(x)$ for each $x \in \mathsf{Var}$ which is equal to the corresponding σ-algebra Σ_t of its type $t = type(x)$. We let $\Sigma_{\mathsf{Var}'} \stackrel{\text{def}}{=} \bigotimes_{x \in \mathsf{Var}'} \Sigma_x$. If the type of variable x is clock, var or real, we require $\Sigma_x = \mathscr{B}(\mathbb{R})$.

We can now use these definitions to specify the semantics of variable evaluations, which assign values to variables in accordance with the variables' types:

Definition 7 (Variable evaluation) A *variable evaluation* is a function $v\colon \mathsf{Var}' \to Dom(\mathsf{Var})$ such that $v(x) \in Dom(x)$ for all $x \in \mathsf{Var}'$. By fixing an order on the variables, we can identify variable evaluations with $|\mathsf{Var}'|$-dimensional tuples. By $\mathsf{Val}(\mathsf{Var}')$ we denote the set of all variable evaluations on Var'. We write Val for $\mathsf{Val}(\mathsf{Var})$.

The *restriction* of an evaluation $v\colon \mathsf{Var} \to \mathit{Dom}(\mathsf{Var})$ to Var' is

$$v|_{\mathsf{Var}'}\colon \mathsf{Var}' \to \mathit{Dom}\big(\mathsf{Var}'\big) \quad \text{where } \forall x \in \mathsf{Var}'\colon v|_{\mathsf{Var}'}(x) \stackrel{\text{def}}{=} v(x).$$

Given two disjoint subsets $\mathsf{Var}_1, \mathsf{Var}_2 \subseteq \mathsf{Var}$ of variables and the evaluations $v_1\colon \mathsf{Var}_1 \to \mathit{Dom}(\mathsf{Var}_1)$, $v_2\colon \mathsf{Var}_2 \to \mathit{Dom}(\mathsf{Var}_2)$, their *combination* is

$$(v_1 \uplus v_2)\colon (\mathsf{Var}_1 \uplus \mathsf{Var}_2) \to \mathit{Dom}(\mathsf{Var}_1 \uplus \mathsf{Var}_2) \quad \text{with } (v_1 \uplus v_2)(x) \stackrel{\text{def}}{=} v_i(x) \text{ if } x \in \mathsf{Var}_i.$$

For the invariant semantics, we need to distinguish between continuous and discrete variables, where the values of continuous variables can change over time, the possible changes being governed by invariants, while the values of discrete variables can only be changed explicitly through assignments:

Definition 8 (Continuous and discrete variables) With $\mathsf{Cont}(\mathsf{Var}') \subseteq \mathsf{Var}'$ we denote the *continuous variables* of Var', i.e. the variables which are of type clock or var. The complement of $\mathsf{Cont}(\mathsf{Var}')$ is the set of *discrete variables* $\mathsf{Disc}(\mathsf{Var}') \stackrel{\text{def}}{=} \mathsf{Var}' \setminus \mathsf{Cont}(\mathsf{Var}')$. For $\mathsf{Var}' = \mathsf{Var}$, we only write Cont and Disc.

Expressions $e \in \mathsf{Sxp}$ are used in variable assignments. The semantics of such an expression maps variable evaluations to a set of distributions over the possible successor values of the variable, thereby combining nondeterminism (from the any keyword) and stochastic choices. Expressions $e \in \mathsf{Axp}$ are used as edge weights. Their semantics maps variable valuations to a single nonnegative number. Expressions $e \in \mathsf{Bxp}$ simply map variable evaluations to Boolean values, which will be used to decide the validity of guards, deadlines and invariants. Formally:

Definition 9 (Expression semantics) The semantics $[\![e]\!]$ of an expression e over variables in Var with $\mathit{type}(e) = t$ is such that

- $[\![e]\!]\colon \mathsf{Val} \to \Delta(\Sigma_t)$ is a Σ_{Var}-$\mathscr{H}(\Delta(\Sigma_t))$-measurable function if $e \in \mathsf{Sxp}$,
- $[\![e]\!]\colon \mathsf{Val} \to \mathbb{R}_0^+$ is a Σ_{Var}-$\mathscr{B}(\mathbb{R}_0^+)$-measurable function if $e \in \mathsf{Axp}$, and
- $[\![e]\!]\colon \mathsf{Val} \to \{\mathit{true}, \mathit{false}\}$ is a Σ_{Var}-$\mathscr{P}(\{\mathit{true}, \mathit{false}\})$-measurable function if $e \in \mathsf{Bxp}$.

The semantics of an assignment combines all the possible distributions to the model's variables. Notice that for each variable $x \in \mathsf{Var}$ we have that $[\![a(x)]\!]\colon \mathsf{Val} \to \Delta(\Sigma_t)$ represents the values which the assignment can assign to variable x for a given variable valuation (cf. Sect. 2.3 and the expression semantics).

Definition 10 (Assignment semantics) The semantics $[\![a]\!]$ of an assignment $a \in \mathsf{Asgn}$ is such that $[\![a]\!]\colon \mathsf{Val} \to \Delta(\Sigma_{\mathsf{Var}})$, and it is defined by $[\![a]\!](v) \stackrel{\text{def}}{=} \bigotimes_{x \in \mathsf{Var}} [\![a(x)]\!](v)$.

The measurability restrictions on the expressions guarantee that the semantics of an assignment is Σ_{Var}-$\mathscr{H}(\Delta(\Sigma_{\mathsf{Var}}))$-measurable.

An invariant specifies the valid behaviours of the continuous variables for all combinations of the remaining variables. Because we want to be able to specify systems in which the continuous behaviour is dependent on non-continuous variables, the domain of an invariant semantics is $\mathsf{Val}(\mathsf{Disc})$. As we want to be able to restrict both the values as well as the derivatives of the continuous variables, the range of an invariant semantics is $\Sigma_{\mathsf{Cont}} \otimes \Sigma_{\mathsf{Cont}}$. The

first component of $\Sigma_{\mathsf{Cont}} \otimes \Sigma_{\mathsf{Cont}}$ will later on be used for the values of variables, whereas the second component will handle their derivatives. The invariant semantics is formally defined as follows:

Definition 11 (Invariant semantics) The semantics $[\![e]\!]$ of an invariant $e \in \mathsf{Ixp}$ is of the form $[\![e]\!] : \mathsf{Val}(\mathsf{Disc}) \to \Sigma_{\mathsf{Cont}} \otimes \Sigma_{\mathsf{Cont}}$. We require $[\![e]\!]$ to be Σ_{Disc}-$\mathscr{H}(\Sigma_{\mathsf{Cont}} \otimes \Sigma_{\mathsf{Cont}})$-measurable. If $[\![e]\!](v) = \bigtimes_{x_i \in \mathsf{Cont}} A_i \times \bigtimes_{x_i \in \mathsf{Cont}} A'_i$, for each $x_i \in \mathsf{clock}$ we must have $A'_i = \{1\}$.

The restriction $A'_i = \{1\}$ for $x_i \in \mathsf{clock}$ is used to ensure that clock variables actually behave as clocks, i.e. that their derivative is one, so if one waits for time t, their value increases by t.

The σ-algebras in the definitions above are needed to ensure that the complete NLMP semantics, which we obtain in the end, is well-defined. For variables with real values, we use Borel σ-algebras, as they suffice for all models and properties one might want to specify. For countable data types, the same holds for the power sets of the domains. For more complex data types, like records and arrays, the σ-algebra to be used depends on the model and properties to be specified. We do not provide a detailed discussion here, but in general σ-algebras for complex data structures can be derived by combining the σ-algebras of the constituent elements. For instance, we could use the σ-algebra $\sigma(\bigcup_{n\geq 1} \bigotimes_{i=1}^{n} \mathscr{B}(\mathbb{R}))$ for arrays with real-valued elements.

Example 9 Consider the SHA for the parallel composition of `Tank` and `Controller` in Fig. 6. We have $\mathsf{Var} = \{x, d, \texttt{level}, \texttt{measurement}\}$ with $type(x) = \mathsf{real}$, $type(d) = \mathsf{clock}$, $type(\texttt{level}) = \mathsf{var}$, $type(\texttt{measurement}) = \mathsf{real}$ and thus two continuous and two discrete variables: $\mathsf{Cont} = \{d, \texttt{level}\}$, $\mathsf{Disc} = \{x, \texttt{measurement}\}$. We have $Dom(\cdot) = [\![type(\cdot)]\!] = \mathbb{R}$ for all variables, and the σ-algebra used for all variables is $\Sigma_{\cdot} = \mathscr{B}(\mathbb{R})$, such that we have $\Sigma_{\mathsf{Var}} = \bigotimes_{i=1}^{4} \mathscr{B}(\mathbb{R}) = \mathscr{B}(\mathbb{R}^4)$.

A possible variable evaluation to give concrete values to the variables is $v : \mathsf{Var} \to \mathbb{R}$ with $v(x) = 0.1$, $v(d) = 2.3$, $v(\texttt{level}) = 7.9$ and $v(\texttt{measurement}) = 7.4$, which can be written as $v = (0.1, 2.3, 7.9, 7.4)$ in tuple form. We can then consider $v|_{\{x,d\}} : \{x, d\} \to \mathbb{R}$, the restriction to the variables x and d, with $v|_{\{x,d\}} = (0.1, 2.3)$. Given $v' : \{\texttt{level}, \texttt{measurement}\} \to \mathbb{R}$ with $v'(\texttt{level}) = 11.8$ and $v'(\texttt{measurement}) = 12.1$, we can define $v'' : \mathsf{Var} \to \mathbb{R}$ as the combination $v'' = v|_{\{x,d\}} \uplus v'$, such that we have $v''(x) = 0.1$, $v''(d) = 2.3$, $v''(\texttt{level}) = 11.8$ and $v''(\texttt{measurement}) = 12.1$.

The semantics $[\![1]\!] : \mathsf{Val} \to \Delta(\mathscr{B}(\mathbb{R}))$ of the expression $1 \in \mathsf{Sxp}$ is $[\![1]\!](v) = \{\mathscr{D}(1)\}$ for all variable evaluations $v : \mathsf{Var} \to \mathbb{R}$; if we considered 1 as an arithmetic expression instead, we would have $[\![1]\!](v) = 1$. The semantics of the right-hand side of the assignment that measures the water level is $[\![\mathscr{N}(\texttt{level}, 1)]\!](v) = \{\mathscr{N}(v(\texttt{level}), 1)\}$ where $\mathscr{N}(x, y^2)$ denotes a normal distribution with expected value x and variance y^2. For the deadlines, the semantics $[\![d \geq x]\!] : \mathsf{Val} \to \{true, false\}$ is such that $[\![d \geq x]\!](v) = true$ iff $v(d) \geq v(x)$.

For the assignment

$$a = \{x \mapsto 1, d \mapsto 0, \texttt{measurement} \mapsto \mathscr{N}(\texttt{level}, 1)\}$$

we have $[\![a]\!] : \mathsf{Val} \to \mathscr{B}(\mathbb{R}^4)$ as follows:

$$\begin{aligned}[\![a]\!](v) &= [\![1]\!](v) \otimes [\![0]\!](v) \otimes \{v(\texttt{level})\} \otimes [\![\mathscr{N}(\texttt{level}, 1)]\!](v) \\ &= \{\mathscr{D}(1)\} \otimes \{\mathscr{D}(0)\} \otimes \{\mathscr{D}(v(\texttt{level}))\} \otimes \{\mathscr{N}(v(\texttt{level}), 1)\}.\end{aligned}$$

Thus, the assignment sets x and d to constant values, `level` maintains its previous value, and `measurement` is sampled according to a normal distribution with expected value `level` and variance 1. Note that we treat all expressions as sampling expressions here, even those evaluating to constant values, since $\mathsf{Asgn} = \mathsf{Var} \to \mathsf{Sxp}$.

The semantics of the invariant

$$e' = \big(\mathsf{der}(\texttt{level}) == -2\big)$$

is such that $[\![e']\!] \colon \mathsf{Val}(\mathsf{Disc}) \to \mathscr{B}(\mathbb{R}^4)$. We have $[\![e']\!](v) = \mathbb{R}^2 \times \{(1, -2)\}$ for all variable evaluations $v \colon \mathsf{Disc} \to \mathbb{R}$, which means that in this example the flow of the continuous variables is independent of the discrete variables. In the semantics of the invariant, there are no direct restrictions on the values that d and `level` may reach. The restriction of the derivative of `level` is specified by the invariant, whereas the restriction of the derivative of d to 1 results from the fact that d is a clock variable.

4.4 Continuous concrete semantics

Based on the previous definitions, the possible behaviours of the continuous flow that happens *within* one location as time advances can now be defined. We do this by introducing a predicate $\mathsf{Witness}$ that precisely characterises those behaviours, the general idea being that if $\mathsf{Witness}(v, \ell, t, v')$ holds in the current location ℓ and variable evaluation v, it is possible to reach variable evaluation v' (in the same location) by waiting for t time units. As we allow differential *inclusions*, there may be arbitrarily many $v' \neq v''$ for which $\mathsf{Witness}(v, \ell, t, v'')$ also holds, such that it is not clear where exactly one ends up by waiting for time t—this is another place where nondeterministic decisions appear in our model.

Definition 12 (Continuous behaviour) Let $(\mathsf{Loc}, \ell_0, \mathsf{Act}, \mathsf{Var}, v_0, \to)$ be a SHA, $\ell \in \mathsf{Loc}$ a location, $v, v' \in \mathsf{Val}(\mathsf{Var})$ variable evaluations, and $t > 0$ a real-valued duration. The predicate $\mathsf{Witness}(v, \ell, t, v')$ holds if there exists a *witness function* $f \colon [0, t] \to \mathsf{Val}(\mathsf{Cont})$ such that

- $f(0) = v|_{\mathsf{Cont}}$,
- $f(t) \uplus v|_{\mathsf{Disc}} = v'$,
- f is right (left) differentiable in 0 (t) and differentiable in t' for all $t' \in (0, t)$,
- $(f(t'), \dot{f}(t')) \in [\![\mathsf{Inv}(\ell)]\!](v|_{\mathsf{Disc}})$ for all $t' \in [0, t]$, and
- for all $t' \in [0, t)$,

$$\left[\!\!\left[\bigvee_{(\ell, a, g, d, \mathscr{W}) \in \to} d \right]\!\!\right] \big(f(t') \uplus v|_{\mathsf{Disc}}\big) = \mathit{false}.$$

Witness functions are only defined over values of continuous variables Cont. This is done to ensure that the values of discrete variables remain constant as time passes. The definition of $\mathsf{Witness}(v, \ell, t, v')$ establishes that the values of these variables in the successor evaluation v' are indeed the same as in v. The requirement that $[\![\bigvee_{(\ell, a, g, d, W) \in \to} d]\!](f(t') \uplus v_{|\mathsf{Disc}})$ is *false* for $t' \in [0, t')$ is used to ensure that time cannot pass when some edge's deadline has become *true*, forcing an edge—which will not necessarily be the one that has become urgent—to be taken instead of letting time pass. Notice that in the definition we require $t > 0$ rather than $t \geq 0$. This is to ensure that the requirements of f being differentiable are well defined. Also, if $t = 0$ would be allowed, timed steps (of length 0) would be possible even in situations in which a deadline is fulfilled.

As before, some measurability restrictions are needed to guarantee the measurability of the complete semantics of a SHA which is specified later on:

Measurability restrictions For all ℓ, v, t we require $\{v' \mid \mathsf{Witness}(v, \ell, t, v')\} \in \Sigma_{\mathsf{Var}}$ and $\{(t', v') \mid \mathsf{Witness}(v, \ell, t', v')\} \in (\mathscr{B}(\mathbb{R}_0^+) \otimes \Sigma_{\mathsf{Var}})$. We further require

$$h\colon \big(\mathsf{Val}(\mathsf{Var}) \times \mathsf{Loc}\big) \to \big(\mathscr{B}(\mathbb{R}) \otimes \Sigma_{\mathsf{Var}}\big), \quad h(v,\ell) \stackrel{\text{def}}{=} \big\{\big(t, v'\big) \mid \mathsf{Witness}\big(v, \ell, t, v'\big)\big\}$$

to be $(\Sigma_{\mathsf{Var}} \otimes \mathscr{P}(\mathsf{Loc}))$-$(\mathscr{H}(\mathscr{B}(\mathbb{R}_0^+) \otimes \Sigma_{\mathsf{Var}}))$-measurable and, for all $t \in \mathbb{R}_0^+$, we require

$$h_t\colon \big(\mathsf{Val}(\mathsf{Var}) \times \mathsf{Loc}\big) \to \Sigma_{\mathsf{Var}},\ h_t(v,\ell) \stackrel{\text{def}}{=} \big\{v' \mid \mathsf{Witness}\big(v, \ell, t, v'\big)\big\}$$

to be $(\Sigma_{\mathsf{Var}} \otimes \mathscr{P}(\mathsf{Loc}))$-$\mathscr{H}(\Sigma_{\mathsf{Var}})$-measurable.

Example 10 Consider location ℓ_3 of the SHA in Fig. 6 and let $e = (\mathsf{der}(\texttt{level}) == -2)$ be its invariant. To show that $\mathsf{Witness}(v, \ell_3, t, v')$ holds, we have to find a corresponding witness function f. As discussed in the previous example, we have $[\![e]\!](v) = \mathbb{R}^2 \times \{(1, -2)\}$ for all valuations v of the discrete variables. This suggests to define $f\colon [0, t] \to \mathbb{R}^2$ such that $f(t') = (v(d) + t', v(\texttt{level}) - 2t')$ for all t' with $0 \le t' \le t$. From the deadline $d \ge x$, it follows that if $v(d) \le v(x)$ then $v(d) + t' \le v(x)$ must hold for all t' on which f is defined, yielding $t \le v(x) - v(d)$.

Thus, for $\mathsf{Witness}(v, \ell_3, t, v')$ to hold there are two cases. If $v(d) < v(x)$, we must have $0 < t \le v(x) - v(d)$, $v'(x) = v(x)$, $v'(\texttt{measurement}) = v(\texttt{measurement})$, $v'(d) = v(d) + t$ and $v'(\texttt{level}) = v(\texttt{level}) - 2t$. In case $v(d) \ge v(x)$, there are no trajectories, because the requirements on the deadlines cannot be fulfilled.

4.5 Discrete concrete semantics

The discrete transitions *between* locations, which do not take time to be performed, can be described by a predicate $\mathsf{Jump}(v, \ell, \mathscr{W}, \mu)$, which asserts that in location ℓ and variable valuation v, we can perform a discrete jump where the target location and variable evaluation is distributed according to μ (which is derived from $\mathscr{W}$):

Definition 13 (Discrete behaviour) Consider a SHA $(\mathsf{Loc}, \ell_0, \mathsf{Act}, \mathsf{Var}, v_0, \to)$, a location $\ell \in \mathsf{Loc}$, a variable evaluation $v \in \mathsf{Val}$, a set of weight expressions

$$\mathscr{W} = \big\{(a_1, \ell_1, w_1), \ldots, (a_n, \ell_n, w_n)\big\} \subseteq \mathsf{Wxp}$$

(where $a_i \in \mathsf{Asgn}$ is the assignment, l_i is the target location and $w_i \in \mathsf{Axp}$ is the weight) and a distribution $\mu \in \Delta(\mathsf{Val}(\mathsf{Var}) \times \mathsf{Loc})$. For $1 \le i \le n$, we let

$$M_i \stackrel{\text{def}}{=} [\![a_i]\!](v) \otimes \big\{\mathscr{D}(\ell_i)\big\}$$

be the set of probability measures resulting from assignment a_i, extended to include the target locations. Let

$$w \stackrel{\text{def}}{=} \sum_{i=1}^{n} [\![w_i]\!](v) \in \mathbb{R}^+$$

be the total sum of the weights. Then the predicate $\mathsf{Jump}(v, \ell, \mathscr{W}, \mu)$ holds if there are $\mu_1 \in M_1, \ldots, \mu_n \in M_n$ such that μ is the weighted sum of the μ_i, i.e.

$$\forall A \in \Delta(\Sigma_{\mathsf{Var}}) \otimes \mathscr{P}(\mathsf{Loc}) \colon \mu(A) = \sum_{i=1}^{n} \frac{[\![w_i]\!](v)}{w} \mu_i(A).$$

The distribution μ over the jump targets is specified such that the successor location is chosen according to the relative weights of the weight expressions. For a fixed successor location ℓ_i, the distribution over the variable assignments results from a nondeterministic choice between the possible distributions represented by a_i. Note that discrete jumps are not influenced by the invariants of the target locations. It is therefore possible that a transition leads to states in which the invariant is immediately violated, the only consequence being that no time can pass in such states, just as if a deadline evaluated to *true*. This is a *weak invariants* semantics, which is more natural for probabilistic systems—where transitions can have more than a single target state—than a *strong invariants* semantics as employed in e.g. timed automata, which removes transitions leading to states where the invariant immediately evaluates to *false*. If such a behaviour is intended, it can be achieved with weak invariants by suitably strengthening the guards of the edges. In *well-formed* specifications, where the guards of all edges imply the invariants of all possible successor locations, these issues do not arise.

Example 11 Consider the SHA in Fig. 6. In this example, there are no discrete probabilities, such that all weight functions equal 1. In ℓ_1, there is a single edge to ℓ_2. For $\mathsf{Jump}(v, \ell_1, \mathscr{W}, \mu)$ to hold, we must have the following. At first, the guard of the only emanating transition must be fulfilled, i.e. $[\![d \geq x]\!](v) = \mathit{true}$, which for the reachable states is the case if $v(d) \geq 1$. Further, $\mu \colon \mathscr{B}(\mathbb{R}^4) \times \mathscr{P}(\{\mathit{true}, \mathit{false}\}) \to [0, 1]$ is the uniquely defined probability measure such that for $a, b \in \mathbb{R}$ with $a < b$ we have

$$\mu\big(\{0.1, 0, v(\texttt{level}), \ell_2\}\big) \times [a, b]\big) = \int_a^b \exp\left(-\frac{1}{2}\big(x - v(\texttt{level})\big)^2\right) \mathrm{d}x.$$

4.6 Semantics of a stochastic hybrid automaton

With these ingredients, we can define the behaviour of a SHA.

Definition 14 (Semantics of SHA) The *semantics of a SHA* $A = (\mathsf{Loc}, \ell_0, \mathsf{Act}, \mathsf{Var}, v_0, \to)$ is the NLMP $[\![A]\!] = (S, \Sigma_S, s_0, L, \Sigma_L, T)$ where

- $S = \mathsf{Val} \times \mathsf{Loc}$,
- $\Sigma_S = \Sigma_{\mathsf{Var}} \otimes \mathscr{P}(\mathsf{Loc})$,
- $s_0 = (\bigtimes_{x \in \mathsf{Var}} [\![v_0]\!], \ell_0)$,
- $L = \mathsf{Act} \uplus \mathbb{R}^+$, and
- $\Sigma_L = \sigma(\mathscr{B}(\mathbb{R}^+) \uplus \mathscr{P}(\mathsf{Act}))$,

and $\hookrightarrow_T$ is the smallest relation satisfying the inference rules

$$\frac{\ell \xrightarrow{a,g,d} \mathscr{W} \quad \mathsf{Jump}(v, \ell, \mathscr{W}, \mu) \quad [\![g]\!](v) \text{ holds}}{(v, \ell) \overset{a}{\hookrightarrow}_T \mu} \quad \text{and} \quad \frac{\mathsf{Witness}(v, \ell, t, v')}{(v, \ell) \overset{t}{\hookrightarrow}_T \mathscr{D}((v', \ell))}.$$

As stated before, the measurability restrictions on the different parts used to construct the NLMP ensure the validity of the measurability restrictions of the NLMP itself (shown in [29] for a similar model). Definition 14 ensures that timed transitions are simple, as in TPTS (see Definition 15). We also have a weak form of time additivity:

$$s \overset{t+t'}{\hookrightarrow} s' \Rightarrow s \overset{t}{\hookrightarrow} s'' \wedge s'' \overset{t'}{\hookrightarrow} s'$$

As mentioned, because the invariants allow to specify differential inclusions and not only differential equations, we might have several trajectories of the same duration starting in the same state. Thus, in general, we do not have time determinism.

Example 12 The semantics of the SHA of Fig. 6 is the NLMP $(S, \Sigma_S, s_0, L, \Sigma_L, T)$ with

- $S = \mathsf{Val} \times \{\ell_1, \ell_2, \ell_3, \ell_4\}$,
- $\Sigma_S = \mathscr{B}(\mathbb{R}^4) \times \mathscr{P}(\{\ell_1, \ell_2, \ell_3, \ell_4\})$,
- $s_0 = \{\mathscr{D}(1, 0, 6.5, m, \ell_1) \mid m \in \mathbb{R}\}$,
- $L = \{\tau, \texttt{off}, \texttt{on}\} \uplus \mathbb{R}^+$, and
- $\Sigma_L = \sigma(\mathbb{R}^+ \uplus \{\tau, \texttt{off}, \texttt{on}\})$.

As seen from Examples 10 and 11, some of the transitions of this NLMP are

- $(x, d, \texttt{level}, \texttt{measurement}, \ell_3) \overset{t}{\hookrightarrow} (x, d + t, \texttt{level} - 2t, \texttt{measurement}, \ell_3)$ if $0 < t \leq x - d$ and
- $(x, d, \texttt{level}, \texttt{measurement}, \ell_1) \overset{\tau}{\hookrightarrow} \mu$ if $d \geq x$ holds and μ is as described in Example 11.

As noted in Sect. 3, we previously considered a restricted version of MODEST which was restricted to specifying stochastic timed automata [17]. The semantics of such a restricted model leads to a special case of NLMP:

Definition 15 (TPTS [17]) A *timed probabilistic transition system* (TPTS) is a NLMP $(S, \Sigma_S, s_0, L, \Sigma_L, T)$ where $L = \mathsf{Act} \uplus \mathbb{R}^+$ for a finite set of *actions* Act, $\Sigma = \mathscr{B}(S)$, and every transition labelled with $t \in \mathbb{R}^+$ is trivial and satisfies

- $s \overset{t+t'}{\hookrightarrow} s' \Leftrightarrow s \overset{t}{\hookrightarrow} s'' \wedge s'' \overset{t'}{\hookrightarrow} s'$ *(time additivity)* as well as
- $s \overset{t}{\hookrightarrow} s' \wedge s \overset{t}{\hookrightarrow} s'' \Rightarrow s' = s''$ *(time determinism).*

for $t, t' \in \mathbb{R}^+$.

Theorem 1 *Let* $A = (\mathsf{Loc}, \ell_0, \mathsf{Act}, \mathsf{Var}, v_0, \rightarrow)$ *be a SHA resulting from a* HMODEST *specification without variables of type* var *and in which the keywords* invariant *and* any *are not used. Then the semantics of A is a TPTS, and it is equivalent to the semantics of* MODEST *in* [17].

5 Tool support

The idea behind the MODEST language is to provide a unifying modelling language for a most general class of systems in such a way that interesting subclasses are easy to identify, ideally on the syntactic level, and that existing tools that handle particular subclasses can be

used for analysis, where available. The aim of this *single-formalism, multi-solution* approach is to reduce the learning curve and modelling effort required from the end-user as well as to reduce the implementation effort by leveraging existing and proven tools where possible. The MODEST TOOLSET currently consists of MCPTA [36, 37], which allows the analysis of models corresponding to the PTA subset of MODEST using PRISM [44] as a backend, and MODES [16, 36], a discrete-event simulator for deterministic STA that—in contrast to most other simulators—can also handle certain nondeterministic models in a sound way. A connection to the UPPAAL model checker for timed automata [7, 10] has recently been added as well [15], and all of these analysis tools are integrated in MIME, a graphical modelling and analysis environment.

In order to allow the analysis of HMODEST specifications, we have now rewritten PROHVER [29], which computes safe upper bounds for probabilistic reachability probabilities, to use MODEST as its input language and be a part of the MODEST TOOLSET. After extending the existing codebase for parsing, syntax representation and symbolic semantics to support the language extensions introduced in Sect. 2 (in particular, the der keyword and the promotion of the invariant shorthand to a regular language construct) and the modifications in the symbolic semantics (Sect. 3), we could reuse these features to transform, as a first step, a textual HMODEST specification into a single SHA.

PROHVER will use existing tools for non-stochastic hybrid systems to analyse the SHA. However, several conversion steps are necessary to first obtain an input that can be used with such an existing non-stochastic tool as well as the necessary information to reintroduce the stochastic aspects into the tool's output, and then a stochastic analysis has to be performed to actually compute probabilities on the combined result. Currently, our analysis is based on PHAVER [31], but the method can in principle be adapted to other solvers such as HSOLVER [56] or D/DT [24] as well, the necessary modifications in the steps described in the remainder of this section depending on the tool considered.

Typical hybrid solvers such as PHAVER support location invariants, but not deadlines, which we thus convert into invariants as the next step. Deadlines are particularly powerful—and difficult to transform into invariants—when synchronisation between the components of e.g. a network of automata is involved. As we already start with a single SHA, these issues of composition do not arise; however, there are still deadlines that cannot be expressed as invariants. For a detailed comparison of deadlines and invariants, we refer the interested reader to [36], where a transformation from deadlines to invariants is described as well. PROHVER uses this transformation.

We then use discrete probability distributions to overapproximate continuous ones by introducing additional nondeterminism: We divide the support (that is, the possible successor states) of a continuous probability distribution into a number of intervals, such that the probability of each interval is known, and replace the continuous distribution by a probabilistic choice over the intervals, followed by a nondeterministic choice of where to move exactly within the chosen interval. The exact method was described in [29].

Example 13 Consider again the SHA of Fig. 6. We can overapproximate the normal distribution at the edge from ℓ_1 to ℓ_2, as depicted in Fig. 7, by the edge $\ell_1 \xrightarrow{\tau, d \geq x, d \geq x} \mathscr{W}$ with

$$\mathscr{W} =$$

$$\big\{\big(\{x \mapsto 0.1, d \mapsto 0, \texttt{measurement} \mapsto \mathsf{any}(y \mid y \leq \texttt{level} - 0.84)\}, \ell_2, 1\big),$$

$$\big(\{x \mapsto 0.1, d \mapsto 0, \texttt{measurement} \mapsto \mathsf{any}(y \mid \texttt{level} - 0.85 \leq y \leq \texttt{level} - 0.25)\}, \ell_2, 1\big),$$

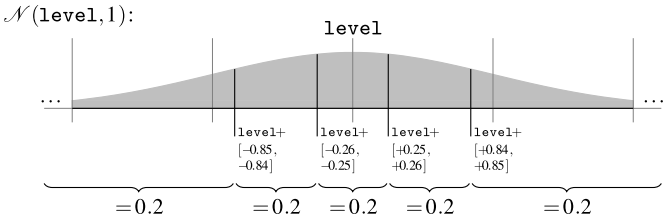

Fig. 7 Overapproximation of a continuous distribution using a discrete one and nondeterministic choices

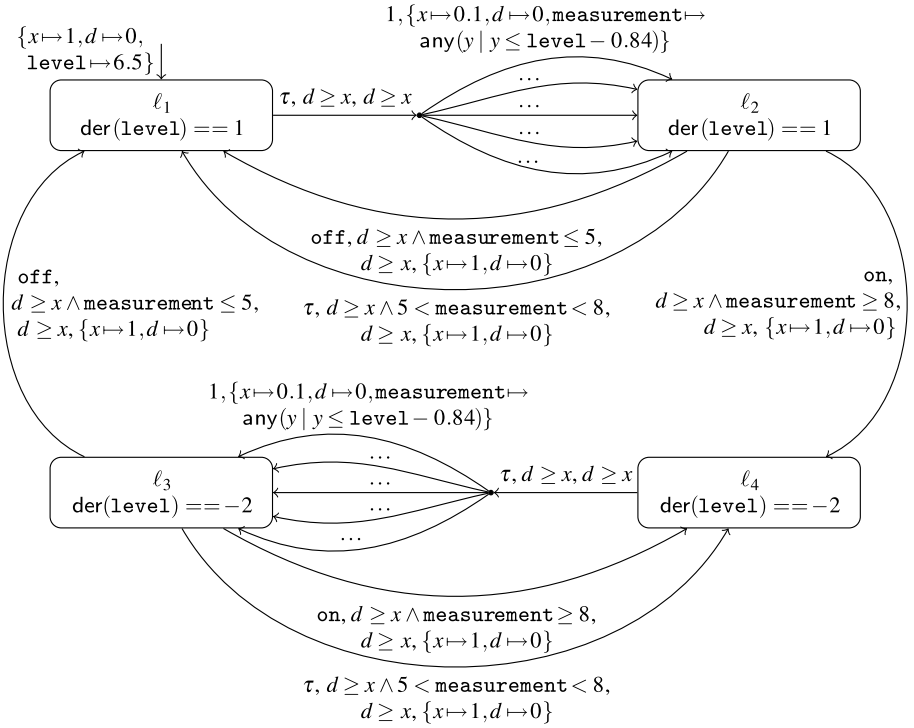

Fig. 8 Overapproximation of the SHA of Fig. 6 by discrete distributions

$$\big(\{x\mapsto 0.1, d\mapsto 0, \texttt{measurement}\mapsto \mathsf{any}(y \mid \texttt{level} - 0.26 \leq y \leq \texttt{level} + 0.26)\}, \ell_2, 1\big),$$
$$\big(\{x\mapsto 0.1, d\mapsto 0, \texttt{measurement}\mapsto \mathsf{any}(y \mid \texttt{level} + 0.25 \leq y \leq \texttt{level} + 0.85)\}, \ell_2, 1\big),$$
$$\big(\{x\mapsto 0.1, d\mapsto 0, \texttt{measurement}\mapsto \mathsf{any}(y \mid \texttt{level} + 0.84 \leq y)\}, \ell_2, 1\big)\},$$

dividing the support of the distribution into equally likely parts. Notice that the parts here are overapproximated and overlap with each other, as is allowed by our method. The same transformation can be applied to the edge from ℓ_4 to ℓ_3. The resulting probabilistic hybrid automaton is given in Fig. 8. In this model, we have $\mathsf{Jump}(v, \ell_1, \mathscr{W}, \mu)$ if there are

$$s_1 \in \big\{\big(0.1, 0, y, v(\texttt{level})\big) \mid y \leq \texttt{level} - 0.84\big\},$$

$$s_2 \in \big\{(0.1, 0, y, v(\texttt{level})) \mid \texttt{level} - 0.85 \le y \le \texttt{level} - 0.25\big\},$$
$$s_3 \in \big\{(0.1, 0, y, v(\texttt{level})) \mid \texttt{level} - 0.26 \le y \le \texttt{level} + 0.26\big\},$$
$$s_4 \in \big\{(0.1, 0, y, v(\texttt{level})) \mid \texttt{level} + 0.25 \le y \le \texttt{level} + 0.85\big\} \quad \text{and}$$
$$s_5 \in \big\{(0.1, 0, y, v(\texttt{level})) \mid \texttt{level} + 0.84 \le y\big\}$$

such that $\mu(\{s_1\}) = \mu(\{s_2\}) = \mu(\{s_3\}) = \mu(\{s_4\}) = \mu(\{s_5\}) = \frac{1}{1+1+1+1+1} = 0.2$.

Since PHAVER only supports a set of modes (or locations) plus continuous variables, we remove non-continuous variables by encoding the possible evaluations into modes. This is possible only if there are finitely many such evaluations. However, there are hybrid solvers which are capable of handling complex discrete structures efficiently [30], thus this step might become optional and solver-dependent in the future.

After these steps, the only kind of stochastic behaviour that remains are the finite-support probability distributions represented by the targets of the edges. We currently require all weights in these targets to be constants: probabilities depending on continuous variables are not yet supported. The result of the transformations at this point is therefore a PHA.

We can now transform this PHA into a non-probabilistic hybrid automaton using the method described in [63]: Each edge

$$l \xrightarrow{a,g,d} \big\{(a_1, \ell_1, w_1), \ldots, (a_n, \ell_n, w_n)\big\}$$

induces a set of non-probabilistic edges

$$\big\{\ell \xrightarrow{a,g,d} (a_1, \ell_1), \ldots, \ell \xrightarrow{a,g,d} (a_n, \ell_n)\big\}.$$

We maintain a mapping between probabilistic and induced non-probabilistic edges. After transforming the resulting non-probabilistic hybrid automaton into the PHAVER input language, PHAVER can be used to analyse it. The result of such an analysis is a finite labelled transition system (LTS) which overapproximates the behaviour of the non-probabilistic hybrid automaton. Using the mapping of edge sets, we can construct a finite Markov decision process (MDP, essentially equivalent to PA) which overapproximates the semantics of the original probabilistic model. Because of this, by analysing this model, we obtain safe upper bounds for probabilistic reachability probabilities in the original semantics.

This chain of analysis steps, implemented in PROHVER, is summarised in Fig. 9.

Example 14 Consider the PHA of Fig. 8. By applying the steps described above, we obtain the non-probabilistic hybrid automaton of Fig. 10. Notice that we have already transformed the deadlines into invariants. In Table 7, we give the mapping between probabilistic and non-probabilistic edges in terms of a mapping between the edge labels in the PHA and the corresponding distributions. In the left part of Fig. 11, we give a part of a labelled transition system which could result from computing an abstraction by a non-probabilistic hybrid solver. Initial states are marked by an incoming arrow without a source state. Distribution 0 with the single label l_0 corresponds to the timed transitions $t \in \mathbb{R}^+$. The state drawn with double lines corresponds to an abstract state containing some unsafe concrete states. In the right part, we give the MDP obtained by using the mapping of Table 7 to convert the labels back to probability distributions. The transformation of distribution 0 is trivial. For distribution 1, we obtain two different probabilistic choices, because l_1 occurs twice. Because l_2, l_3

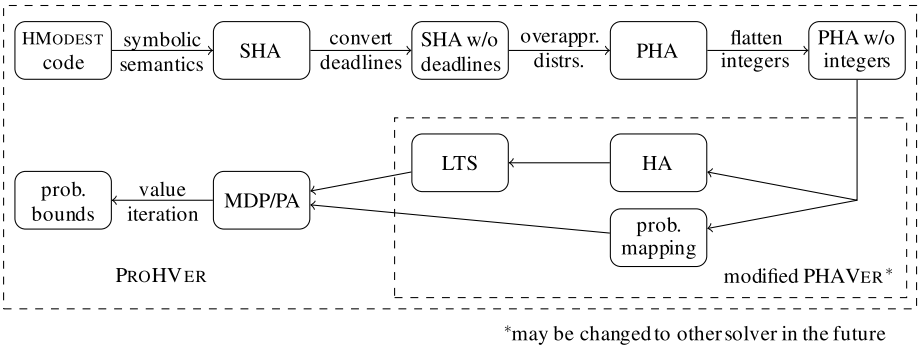

Fig. 9 Overview of the tool chain

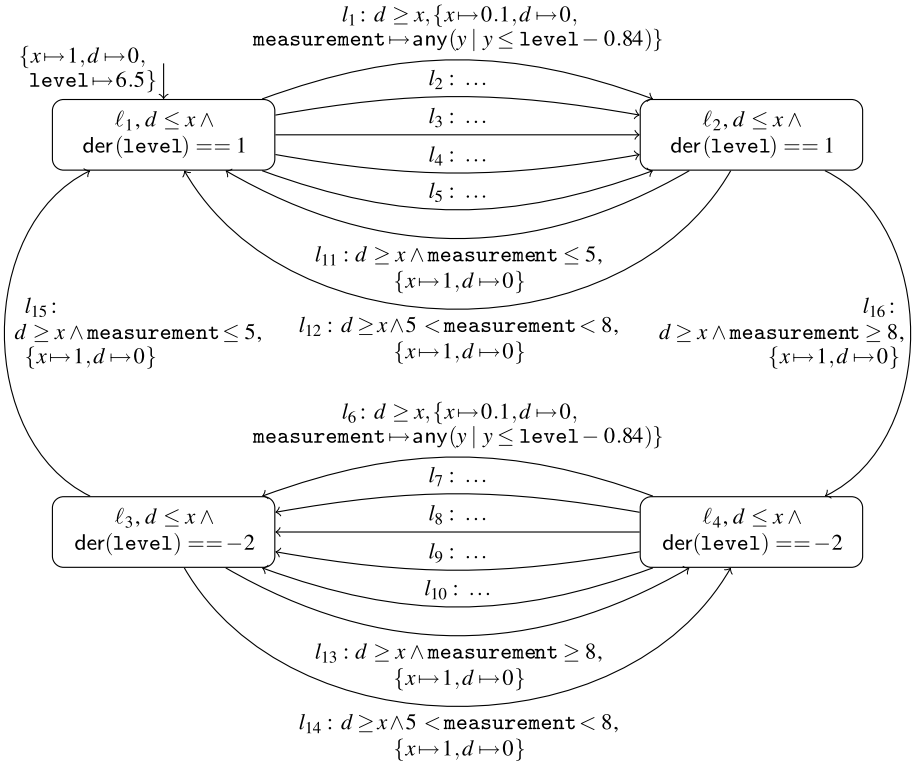

Fig. 10 Induced non-probabilistic hybrid automaton for the PHA of Fig. 8

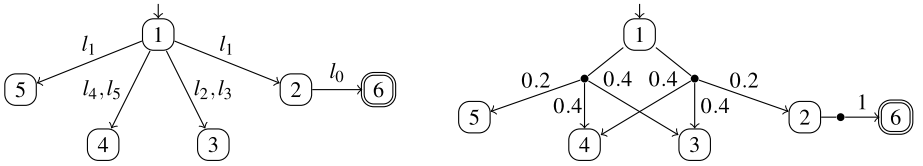

Fig. 11 Part of the LTS abstraction of Fig. 10 (*left*) and MDP after reconstruction of distributions (*right*)

Table 7 Mapping of labels to distributions for hybrid automaton of Fig. 10

Label	Distribution	Probability
l_0	0	1
l_1, l_2, l_3, l_4, l_5	1	0.2
$l_6, l_7, l_8, l_9, l_{10}$	2	0.2
$l_{11} \mid l_{12} \mid l_{13} \mid l_{14} \mid l_{15} \mid l_{16}$	3 | 4 | 5 | 6 | 7 | 8	1

and l_4, l_5 have the same target state, we have to sum up their probabilities. The maximum probability of reaching the unsafe abstract state 6 from the initial abstract state 1 in this MDP is 0.2. The paths in the abstract model leading to this probability correspond to paths in the concrete model in which one first has a severe measurement error (state 1 to 2), followed by an overflow of the tank (state 2 to 6). If this abstract model was indeed a complete abstraction for the whole state space, we could derive that the probability of an overflow (or underflow) of the tank is no larger than 0.2.

Aside from model-checking support for SHA via PROHVER, extending MODEST to HMODEST also benefits MODES: A subclass of SHA that are amenable to and highly useful for discrete-event simulation are models with rewards (or costs), in which continuous variables with constant derivatives are allowed as long as the actual values of these variables are only references in properties. Rewards allow a variety of new properties, such as throughput or availability, to be analysed, and this is now fully supported by MODES. As future work, we also consider extending MODES to—or providing a separate tool for—full hybrid simulation.

The MODEST TOOLSET is available for download on its website, which also includes a list of publications, case studies and documentation, at www.modestchecker.net.

It is cross-platform (tested on various versions of Windows, Linux, Mac OS as well as FreeBSD); on Windows, it requires the Microsoft .NET Framework 4.0, which is available as a free download and is also distributed via Windows Update, while a recent version of the Mono runtime[4] (at least version 2.10.1) is required on other systems.

6 Case studies

We have applied the tool chain discussed in the previous section on three case studies: Our running example, the *water tank*, a *thermostat* that can fail, and a model of headway control in the European Train Control System (*ETCS*). Since the underlying systems have already been studied in previous publications [29, 63], we present them only briefly, focusing on the modelling aspects with HMODEST instead. The three cases present a progression in terms of modelling and analysis complexity: The water tank example relies on sampling from continuous probability distributions, but the invariants are linear; the thermostat contains nonlinear invariants, but only discrete, finite-support probabilistic decisions; the ETCS example finally combines all of these aspects in a model with complex behaviour.

Since our current tool chain relies on PHAVER for part of the analysis, the invariants in all examples are necessarily restricted to *affine* differential inclusions [31], that is conjunc-

[4] http://www.mono-poject.org/.

Table 8 Water level control results

T	Prob.	Time	States
20 s	0.04652	26 s	2354
30 s	0.06929	52 s	4904
40 s	0.09162	115 s	9357
50 s	0.11344	189 s	14106
60 s	0.13475	329 s	19794

tions of formulae of the form

$$c_1 \cdot x_1(t) + d_1 \cdot \dot{x}_1(t) + \cdots + c_n \cdot x_n(t) + d_n \cdot \dot{x}_n(t) \preceq D$$

with $c_i, d_i \in \mathbb{R}$ for all $i \in \{1, \ldots, n\}$, $D \in \mathbb{R}$, and $\preceq \in \{\leq, <\}$, where the x_i are the continuous variables.

6.1 Water tank

The first case study [29] is based on our running example of a water tank with a pump, specified as the parallel composition of a `Tank` and a `Controller` process as presented in the previous sections. In this model, all invariants are *linear*, that is, they are conjunctions of formulae of the form $c_i \cdot \dot{x}_i(t) \preceq d_i$ and linear formulae not containing derivatives $\dot{x}_i(t)$ of continuous variables. The probabilistic behaviour results from the measurement of the water level, which is given by a continuous distribution. This imprecision of measurement might lead to an incorrect reaction of the controller, which in turn is the reason why it is at all possible to reach an unsafe state.

We consider the property presented in Example 5: the maximum probability that an over- or underflow of the tank (more than 12 l or less than 1 l of water)—this characterises the set of unsafe states—occurs within a given time bound T. Table 8 shows the analysis performance and results for this property. Column "prob." gives the (upper) bound on the reachability probability computed by ProHVer, "time" indicates how much time was needed to construct the abstraction by PHAVer[5] (which consistently and significantly dominates the overall analysis time) and "states" gives the number of states of the abstraction computed. To obtain these figures, we have overapproximated the continuous distribution describing the measurements of the filling level by a probabilistic one as described in Sect. 5. The intervals used were $w + \{[-1.5, 1.5], [-2, -1.5], [1.5, 2], (-\infty, -1.9], [1.9, \infty)\}$, w being the actual water level. In this case, a manual analysis has shown that it is only because of this overapproximation that the values provided are upper bounds on the reachability probability and not exact values. In general, the computation of an abstraction of the hybrid behaviour by PHAVer can also lead to values larger than the real probability.

6.2 Thermostat

As a second example, we consider a thermostat with failures, extended from a model by Alur *et al.* [6]. For this system, we chose to highlight a method to transform a mode-based description of a hybrid system into HModest code in a straightforward, mechanical way: Every mode is represented as a process consisting of a nondeterministic choice (`alt`) over the

[5]Computations were performed on an AMD Athlon II X4 620 system with 4 GB RAM.

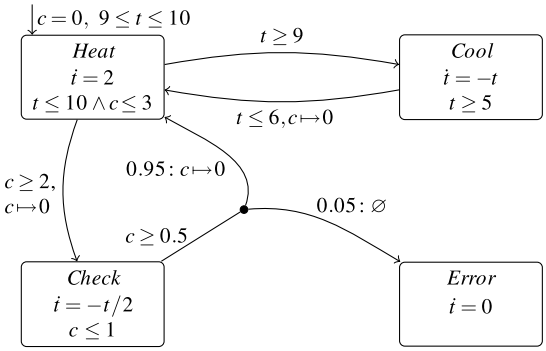

Fig. 12 Mode-based description of the thermostat example

outgoing edges. In case there is only one edge, the alt construct can, of course, be omitted. The location invariant is preserved through an invariant construct on that nondeterministic choice, while every outgoing edge $(\ell, a, g, d, \mathscr{W})$ can be encoded as when(g) urgent(d) a followed by a probabilistic choice palt over the destination locations—i.e. tail-recursive calls to the respective processes—and assignments according to $\mathscr{W}$. The process behaviour of the entire MODEST specification is then simply a call of the process corresponding to the initial mode.

The mode-based description of the thermostat is taken from [63] and shown in Fig. 12. There are four operational modes: *Cool*, *Heat*, *Check* and *Error*. The *Error* mode models the occurrence of a failure, where the temperature sensor gets stuck at the last checked temperature. The set of variables is $\{c, t\}$ where t represents the temperature and c represents a local timer whose derivative is implicitly 1 in all modes; we can consequently model c as a clock variable in HMODEST. The invariants of the modes constrain the evolution of the temperature, which increases linearly when the heat is on and decreases exponentially in the two other modes of regular operation, as well as the passage of time (via c): The system cannot be continuously in heating mode for more than three time units, and checking in mode *Check* takes between 0.5 and 1 time units. We see that the timed behaviour of this case study is more complex than in the water tank example, as it contains affine dynamics which are not linear. On the other hand, the only probabilistic decision occurs when leaving mode *Check*, where we move to the *Error* mode with a probability of 0.05 but continue regular operation in mode *Heat* with probability 0.95.

The HMODEST code resulting from transforming the mode-based description as outlined above is shown in Fig. 13; the correspondence of modes to processes is readily visible except for the *Error* mode: the case of an error occurring is modelled by throwing exception `error`. In order to have no outgoing edges from the location reached after the error occurred, the exception is caught and the exception handler is the stop behaviour. The property we are interested in, which is declared right above process `Heat`, is the maximum probability of reaching mode error—which is indicated by the Boolean variable `err`—within T time units.

In Table 9, we give the bounds for this probability computed by PROHVER and performance statistics for different time bounds.[6] Again, the probabilities provided are not exact, but may be larger than the reachability probabilities in the actual semantics. Because the model only contains discrete distributions, we do not have to apply the step in which continuous distributions are overapproximated. The imprecision in the model is thus only caused

[6]Computations were performed on an Intel Core i7 860 system with 8 GB RAM.

Fig. 13 HMODEST code of the thermostat example

```
exception error;
bool err;
clock c;
var t = any(x | 9 <= x && x <= 10); // temperature

const real T; // time bound
property P_Unsafe = Pmax(<> err && time <= T);

process Heat()
{
   invariant(der(t) == 2 && t <= 10 && c <= 3)
   alt {
      :: when(c >= 2) tau {= c = 0 =}; Check()
      :: when(t >= 9) tau; Cool()
   }
}

process Cool()
{
   invariant(der(t) == -t && t >= 5)
   when(t <= 6) tau {= c = 0 =}; Heat()
}

process Check()
{
   invariant(der(t) == -t/2 && c <= 1)
   when(c >= 0.5) tau palt {
      :95: {= c = 0 =}; Heat()
      : 5: {= err = true =}; urgent throw error
   }
}

try { Heat() }
catch error { stop }
```

Table 9 Thermostat results

T	Prob.	Time	States
2	0	0 s	11
4	0.05	0 s	48
5	0.098	0 s	65
20	0.370	15 s	1544
40	0.642	47 s	4861
80	0.884	151 s	16178
120	0.941	140 s	16130
160	0.986	720 s	55642
180	0.986	815 s	61405

by the overapproximation in the abstraction computed by PHAVER. The coarseness of the abstraction can be specified by a parameter; a more precise abstraction (more abstract states, each of which subsumes fewer concrete states) usually results in better probability bounds, but also increases runtime and memory usage.

6.3 European Train Control System

As an example of a stochastic hybrid system that contains both complex timed behaviour as well as continuous and discrete distributions, we present a model of headway control in the railway domain [29, 40]. In contrast to fully automated transport, which is in general simpler to analyse (as the system is completely under the control of the embedded systems), our sample system implements safeguarding technology that leaves trains under full human control provided safety is not at risk.

Our model implements safe interlocking of railway track segments by means of a "moving block" principle of operation. While conventional interlocking schemes in the railway domain lock a number of static track segments in full, the moving block principle enhances traffic density by reserving a section of track ahead of the train which moves smoothly with the train. This moving block is large enough to guarantee safety even in cases requiring emergency stops, i.e. the block length changes dynamically depending on the current speed of the train and its braking capabilities. We use the European Train Control System (ETCS) Level 3 protocol, in which train separation depends on the absolute braking distance (i.e. it should be safe even if the train ahead comes to a standstill in no time from any speed, e.g. due to a collision) where the distance between two trains must be larger than or equal to the braking distance of the second train plus an additional safety distance (here given as $sd = 400$ m).

Our simplified model, the HMODEST specification of which is shown in Fig. 14, consists of a leader train (process `Leader`), a follower train (process `Follower`) and a moving-block control regularly measuring the leader train position and communicating the corresponding *movement authority* to the follower (process `MovingBlock`). The leader train is freely controlled by its operator within the physical limits of the train (it is assumed not to move backwards, though), while the follower train may be forced to perform a controlled braking if it comes close to the leader. The control principle is as follows:

1. 8 seconds after communicating the last movement authority, the moving-block control takes a fresh measurement m of the leader train position s_l. This measurement may be noisy.
2. Afterwards, a fresh movement authority derived from this measurement is sent to the follower. The movement authority is the measured position m minus the length l of the leader train and further reduced by the safety distance sd. Due to an unreliable communication medium (in practice: GSM-R), this value may reach the follower (in which case its movement authority $auth$ is updated to $m - l - sd$) or not. In the latter case, which occurs with probability 0.1, the follower's movement authority stays as is.
3. Based on the movement authority, the follower continuously checks the deceleration required to stop exactly at the movement authority. Due to PHAVER being confined to linear arithmetic, this deceleration is conservatively approximated as $a_{\text{req}} = \frac{v \cdot v_{\max}}{2(s-auth)}$, where v is the actual speed, $v_{\max}$ the (constant) top speed, and s the current position of the follower train, rather than the physically more adequate, yet non-linear, $a_{\text{req}} = \frac{v^2}{2(s-auth)}$ of the original model [40]. Notice that a_{req} is a negative value, because it describes a deceleration.
4. The follower applies automatic braking whenever the value of a_{req} falls below a certain threshold b_{on}. In this case, the follower's brake controller applies maximum deceleration $a_{\min}$, leading to a stop before the movement authority as $a_{\min} < b_{\text{on}}$. Automatic braking ends as soon as the necessary deceleration a_{req} exceeds a switch-off threshold $b_{\text{off}} > b_{\text{on}}$. The thresholds b_{on} and b_{off} are separate to prevent the automatic braking system from

```
const real SIGMA; // standard deviation of position measurements

const real SD = 400; // safety distance
const real L = 200; // length of train
const real V_MAX = 83.4; // maximum speed of train
const real A_MAX = 0.7; // maximum acceleration of train
const real B_OFF = -0.3; // switch-off threshold
const real B_ON = -0.7; // normal braking deceleration of train
const real A_MIN = -1.4; // maximum deceleration of train

var s_l = 1400; // position of leader train
var s_f = 200; // position of follower train

real auth = 800; // movement authority of follower train

const real T; // time bound
property P_Crash = Pmax(<> (s_f >= s_l - L) && time <= T);

process Leader()
{
   var a; // acceleration
   var v = 0; der(v) = a; // speed

   // The leader train can exhibit any behaviour that is within its
   // acceleration and max. speed constraints, except for driving backwards
   invariant(der(s_l) == v
      && A_MIN <= a && a <= A_MAX && 0 <= v && v <= V_MAX) stop
}

process Follower()
{
   var a; // acceleration
   var v = 0; der(v) = a; // speed

   invariant(der(s_f) == v && 0 <= v && v <= V_MAX) {
      do {
         :: // train is running normally
            invariant(A_MIN <= a && a <= A_MAX
               && v * V_MAX <= 2 * B_ON * (s_f - auth))
            when(v * V_MAX >= 2 * B_ON * (s_f - auth)) tau;
            // forced braking by ETCS system
            invariant(a == A_MIN
               && v * V_MAX >= 2 * B_OFF * (s_f - auth))
            when(v * V_MAX <= 2 * B_OFF * (s_f - auth)) tau
      }
   }
}

process MovingBlock()
{
   real m;

   // measure position of leader train
   delay(8) {= m = Normal(s_l, SIGMA) =};
   // update follower's movement authority
   urgent palt {
      :9: {= auth = m - L - SD  =}; MovingBlock()
      :1: {= /* message lost */ =}; MovingBlock()
   }
}

par { :: Leader()
      :: Follower()
      :: MovingBlock() }
```

Fig. 14 HMODEST code of the European Train Control System case study

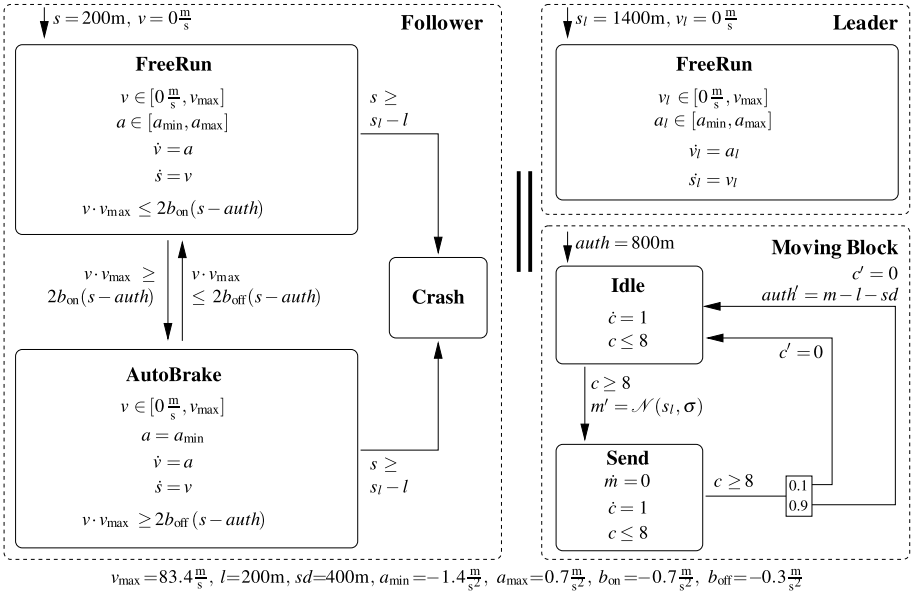

Fig. 15 Mode-based description of the European Train Control System case study [29]

Table 10 Train control results

T	Probability			Time	States
	$\sigma = 10$	$\sigma = 15$	$\sigma = 20$		
60 s	$7.110 \cdot 10^{-19}$	$6.216 \cdot 10^{-9}$	$2.141 \cdot 10^{-5}$	65 s	572
80 s	$1.016 \cdot 10^{-18}$	$8.879 \cdot 10^{-9}$	$3.058 \cdot 10^{-5}$	169 s	1441
100 s	$1.219 \cdot 10^{-18}$	$1.066 \cdot 10^{-8}$	$3.669 \cdot 10^{-5}$	282 s	2399
120 s	$1.524 \cdot 10^{-18}$	$1.332 \cdot 10^{-8}$	$4.587 \cdot 10^{-5}$	1100 s	4537
140 s	$1.727 \cdot 10^{-18}$	$1.509 \cdot 10^{-8}$	$5.198 \cdot 10^{-5}$	2257 s	6569

repeatedly engaging and disengaging in intervals of approximately 8 seconds when the leading train is moving.

In the HMODEST code, points 1 and 2 can be seen implemented in the `MovingBlock` process, while points 3 and 4 are part of the `Follower` process. For this case study, we chose not to use any direct transformation from a mode-based description to build the model, but instead try to make best use of the HMODEST language's features in order to arrive at a concise, yet readable model. For reference, the original mode-based description of this case study as shown in [29] can be found in Fig. 15.

We consider the probability to reach a state in which the trains have collided, i.e. the position of the follower train is greater than or equal to the position of the leader train minus its length, within T seconds. In Table 10, we give the probability bounds and performance results determined by PROHVER.[7] We modelled the measurement error using a normal

[7]Computations were performed on an Intel Core i7 860 system with 8 GB RAM.

distribution with expected value s_l, i.e. the current position of the leader train, and considered different standard deviations σ of the measurement. The abstraction used for each of them can be obtained using structurally equivalent Markov decision processes, only with different probabilities. Thus, we only needed to compute the abstraction once for all deviations, and just had to change the transition probabilities before obtaining probability bounds from the abstraction. We split the normal distributions into the two intervals $s_l + \{(-\infty, 91], [81, \infty)\}$ to obtain a discrete distribution.

7 Related work

Various modular and hierarchical description formalisms for hybrid systems have been proposed. A hierarchical and visual modelling language has been provided in [35], whereas the framework of hybrid I/O automata allows for parallel composition, and supports a rather rich theory of refinement [49]. The language CHARON [5] focuses on the hierarchical modelling of embedded systems, and is supported by analysis tools such as simulators. In the field of process algebras, the compositional specification frameworks for *hybrid systems* has received quite some attention. Examples of such algebraic approaches are HYBRID CHI [9], hybrid process calculus [19], and the hybrid process algebra HYPA [23]. Whereas the work on HYPA and the hybrid process calculus is mainly focused on bisimulation notions, congruence results with respect to parallel composition, as well as axiomatisations, HYBRID CHI is intended as a user-friendly language for industrial systems in the field of mechanical engineering. HYBRID CHI has several features in common with MODEST, such as shared variables, patient and impatient actions, parallel composition using handshaking, and process invocations. It has been equipped with a notion of (stateless) bisimulation, which is a congruence [9]. One of the main differences with our approach is that HYBRID CHI, HYPA and the hybrid process calculus do neither support discrete probabilistic branching nor random delays.

Extensions of the aforementioned with probabilistic branching, covering probabilistic hybrid automata, or random delays, like in stochastic hybrid systems, have received scant attention. A notable exception is stochastic CHARON [12] by Bernadsky *et al.*, an extension of CHARON [5] in which the continuous evolution is governed by stochastic differential equations (SDEs). This modular modelling formalism allows for the parallel composition, hiding, and process instantiation of agents, basically processes in our setting. Agents have local, output, and input variables, and their behaviour is described using locations (called modes). As long as the location invariant is satisfied, the evolution of continuous variables is governed by SDEs. On violating the location invariant, a discrete transition is triggered that is specified by a discrete probability distribution over the target locations—where weights may depend on state variables—and continuous distributions (often referred to as resets) over the updated states. Communication between agents is established using shared variables, i.e., handshaking is not supported. Due to the presence of SDEs, stochastic CHARON covers piecewise deterministic Markov processes (PDMPs, [26]), stochastic diffusion processes, and stochastic hybrid systems in the sense of Hu *et al.* [43]. Platzer [54] has developed a compositional approach for stochastic hybrid systems without nondeterminism but with stochastic differential equations. He defines a notion of stochastic hybrid programs together with a corresponding stochastic logic to express required properties of these models. As a solution method, he puts forward a proof calculus rather than model checking methods.

MODEST does not support SDEs, but covers continuous nondeterminism, guards and deadlines (in addition to invariants), and more importantly, is equipped with a compositional

semantics covering all modelling features. Whereas the tool support for stochastic CHARON purely focuses on simulation, we focus in addition on model checking of several sub-models as supported by PRISM (probabilistic timed automata), PHAVER (hybrid automata), and UPPAAL (timed automata). A related tool for the verification of discrete-time stochastic hybrid systems is FHP-Murphi [53].

A compositional modelling approach tailored to nondeterministic PDMPs has been proposed in [60]. Their work extends interactive Markov chains with general continuous dynamics in the continuous variables with time-dependent transition rates like in time-inhomogeneous Markov chains, and stochastic resets of continuous state variables at transition times. Parallel composition of PDMPs is defined in a CSP-like manner while supporting active (controlling) and passive (observing) actions and is extended (in the usual way) with value passing. Shared variables and other composition operators such as hiding, alternative composition, etc. are not considered. Strubbe and van der Schaft [60] define four semantical levels (as opposed to two in this paper); their "lowest" semantical model closely resembles NLMPs.

Meseguer and Sharykin propose an object-based framework for the modelling and analysis of (distributed) stochastic hybrid systems that interact using asynchronous communication [51]. Their modelling approach is based on the use of probabilistic rewrite rules. Models can be simulated by translating them into Maude rewriting logic specifications, and subsequently applying statistical model checking [48]. The modelling language is based on rewriting, and has similar expressiveness as stochastic CHARON.

8 Conclusion

This paper has presented a modelling and analysis framework with a focus on the compositional modelling of stochastic hybrid systems. Our approach is based on an extension of MODEST [17], a modular formalism for stochastic timed systems, with continuous variables whose evolvement is described by ordinary differential (in)equations (specified as invariants in the language). The paper presents a two-step semantics: a symbolic semantics using stochastic hybrid automata, and a concrete semantics in terms of nondeterministic labelled Markov processes. The symbolic semantics is a small twist of the semantics of MODEST, indicating the straightforward extensibility of its compositional operational semantics. Due to the more complex continuous evolvement of variables, the concrete semantics is substantially more involved, but shown to be a conservative extension of that for MODEST. Three case studies show the usability of the modelling language, and the link to safety verification of (classes of) stochastic and probabilistic hybrid systems. Future work includes the treatment of stochastic differential equations, and the study of behavioural congruence relations.

Acknowledgements The authors thank Pedro D'Argenio for discussions on the language design and Nicolás Wolovick (both from University of Cordoba, Argentina) for his support in the development of the concrete semantics.

References

1. Abate A, Katoen J, Lygeros J, Prandini M (2010) Approximate model checking of stochastic hybrid systems. Eur J Control 16(6):624–641
2. Abate A, Prandini M, Lygeros J, Sastry S (2008) Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. Automatica 44(11):2724–2734

3. Altman E, Gaitsgory V (1997) Asymptotic optimization of a nonlinear hybrid system governed by a Markov decision process. SIAM J Control Optim 35(6):2070–2085
4. Alur R, Courcoubetis C, Halbwachs N, Henzinger TA, Ho PH, Nicollin X, Olivero A, Sifakis J, Yovine S (1995) The algorithmic analysis of hybrid systems. Theor Comput Sci 138:3–34
5. Alur R, Dang T, Esposito JM, Hur Y, Ivancic F, Kumar V, Lee I, Mishra P, Pappas GJ, Sokolsky O (2003) Hierarchical modeling and analysis of embedded systems. Proc IEEE 91(1):11–28
6. Alur R, Dang T, Ivancic F (2006) Predicate abstraction for reachability analysis of hybrid systems. ACM Trans Embed Comput Syst 5(1):152–199
7. Alur R, Dill DL (1994) A theory of timed automata. Theor Comput Sci 126(2):183–235
8. Baró Graf H, Hermanns H, Kulshrestha J, Peter J, Vahldiek A, Vasudevan A (2011) A verified wireless safety critical hard real-time design. In: IEEE int symp on a world of wireless, mobile and multimedia networks (WoWMoM). IEEE Press, New York
9. van Beek DA, Man KL, Reniers MA, Rooda JE, Schiffelers RRH (2006) Syntax and consistent equation semantics of hybrid Chi. J Log Algebr Program 68(1–2):129–210
10. Behrmann G, David A, Larsen KG (2004) A tutorial on UPPAAL. In: Formal methods for the design of real-time systems (SFM-RT). LNCS, vol 3185. Springer, Berlin, pp 200–236
11. Berendsen J, Jansen DN, Katoen JP (2006) Probably on time and within budget: on reachability in priced probabilistic timed automata. In: Quantitative evaluation of systems (QEST). IEEE Comput Soc, Los Alamitos, pp 311–322
12. Bernadsky M, Sharykin R, Alur R (2004) Structured modeling of concurrent stochastic hybrid systems. In: Formal modelling and analysis of timed systems, and formal techniques in real-time and fault-tolerant systems (FORMATS/FTRTFT). LNCS, vol 3253. Springer, Berlin, pp 309–324
13. Berrang P, Bogdoll J, Hahn EM, Hartmanns A, Hermanns H (2012) Dependability results for power grids with decentralized stabilization strategies. Reports of SFB/TR 14 AVACS 83, SFB/TR 14 AVACS, ISSN: 1860-9821. www.avacs.org
14. Blom H, Lygeros J (2006) Stochastic hybrid systems: theory and safety critical applications. Lecture notes in control and information sciences, vol 337. Springer, Berlin
15. Bogdoll J, David A, Hartmanns A, Hermanns H (2012) mctau: bridging the gap between Modest and UPPAAL. In: Model checking software—19th international workshop, SPIN 2012, Oxford, UK, July 23–24. LNCS, vol 7385. Springer, Berlin. ISBN 978-3-642-31758-3
16. Bogdoll J, Fioriti LMF, Hartmanns A, Hermanns H (2011) Partial order methods for statistical model checking and simulation. In: Formal techniques for distributed systems (FMOODS/FORTE). LNCS, vol 6722. Springer, Berlin, pp 59–74
17. Bohnenkamp HC, D'Argenio PR, Hermanns H, Katoen JP (2006) MoDeST: a compositional modeling formalism for hard and softly timed systems. IEEE Trans Softw Eng 32(10):812–830
18. Bohnenkamp HC, Gorter J, Guidi J, Katoen JP (2005) Are you still there?—A lightweight algorithm to monitor node presence in self-configuring networks. In: Dependable systems and networks (DSN). IEEE Comput Soc, Los Alamitos, pp 704–709
19. Brinksma E, Krilavicius T, Usenko YS (2005) A process-algebraic approach to hybrid systems. In: 16th IFAC world congress. IFAC, Laxenburg
20. Bujorianu ML (2004) Extended stochastic hybrid systems and their reachability problem. In: Hybrid systems: computation and control (HSCC). LNCS, vol 2993. Springer, Berlin, pp 234–249
21. Bujorianu ML, Lygeros J, Bujorianu MC (2005) Bisimulation for general stochastic hybrid systems. In: Hybrid systems: computation and control (HSCC). LNCS, vol 3414. Springer, Berlin, pp 198–214
22. Clarke E, Fehnker A, Han Z, Krogh B, Stursberg O, Theobald M (2003) Verification of hybrid systems based on counterexample-guided abstraction refinement. In: Tools and algorithms for the construction and analysis of systems (TACAS). LNCS, vol 2619. Springer, Berlin, pp 192–207
23. Cuijpers PJL, Reniers MA (2005) Hybrid process algebra. J Log Algebr Program 62(2):191–245
24. Dang T, Maler O (1998) Reachability analysis via face lifting. In: Hybrid systems: computation and control (HSCC). LNCS, vol 1386. Springer, Berlin, pp 96–109
25. D'Argenio PR, Wolovick N, Terraf PS, Celayes P (2009) Nondeterministic labeled Markov processes: bisimulations and logical characterization. In: Quantitative evaluation of systems (QEST). IEEE Comput Soc, Los Alamitos, pp 11–20
26. Davis MHA (1993) Markov models and optimization. Chapman & Hall, London
27. Desharnais J, Edalat A, Panangaden P (2002) Bisimulation for labelled Markov processes. Inf Comput 179(2):163–193
28. Edwards S, Lavagno L, Lee EA, Sangiovanni-Vincentelli A (1997) Design of embedded systems: formal models, validation, and synthesis. Proc IEEE 85(3):366–390
29. Fränzle M, Hahn EM, Hermanns H, Wolovick N, Zhang L (2011) Measurability and safety verification for stochastic hybrid systems. In: Hybrid systems: computation and control (HSCC). ACM, New York, pp 43–52

30. Fränzle M, Herde C, Teige T, Ratschan S, Schubert T (2007) Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. J. Satisf. Boolean Model. Comput. 1(3–4):209–236
31. Frehse G (2008) Phaver: algorithmic verification of hybrid systems past HyTech. Int J Softw Tools Technol Transf 10(3):263–279
32. Frehse G, Guernic CL, Donzé A, Cotton S, Ray R, Lebeltel O, Ripado R, Girard A, Dang T, Maler O (2011) Spaceex: scalable verification of hybrid systems. In: Computer-aided verification (CAV). LNCS, vol 6806. Springer, Berlin, pp 379–395
33. Giry M (1982) A categorical approach to probability theory. In: Categorical aspects of topology and analysis. Springer, Berlin, pp 68–85
34. Groß C, Hermanns H, Pulungan R (2007) Does clock precision influence Zigbee's energy consumptions? In: Principles of distributed systems (OPODIS). LNCS, vol 4878. Springer, Berlin, pp 174–188
35. Grosu R, Stauner T (2002) Modular and visual specification of hybrid systems: an introduction to HyCharts. Form Methods Syst Des 21(1):5–38
36. Hartmanns A (2010) Model-checking and simulation for stochastic timed systems. In: FMCO. LNCS, vol 6957. Springer, Berlin, pp 372–391
37. Hartmanns A, Hermanns H (2009) A Modest approach to checking probabilistic timed automata. In: Quantitative evaluation of systems (QEST). IEEE Comput Soc, Los Alamitos, pp 187–196
38. Henzinger TA (1996) The theory of hybrid automata. In: IEEE symp on logic in computer science (LICS), pp 278–292
39. Henzinger TA, Ho PH, Wong-Toi H (1997) HYTECH: a model checker for hybrid systems. Int J Softw Tools Technol Transf 1(1–2):110–122
40. Herde C, Eggers A, Fränzle M, Teige T (2008) Analysis of hybrid systems using HySAT. In: International conference on systems (ICONS). IEEE Comput Soc, Los Alamitos, pp 196–201
41. Hermanns H, Herzog U, Katoen JP (2002) Process algebra for performance evaluation. Theor Comput Sci 274(1–2):43–87
42. Hillston J (1994) A compositional approach to performance modelling. PhD thesis, Univ of Edinburgh
43. Hu J, Lygeros J, Sastry S (2000) Towards a theory of stochastic hybrid systems. In: Hybrid systems: computation and control (HSCC). LNCS, vol 1790. Springer, Berlin, pp 160–173
44. Kwiatkowska M, Norman G, Parker D (2011) PRISM 4.0: verification of probabilistic real-time systems. In: Computer aided verification (CAV'11). LNCS, vol 6806. Springer, Berlin, pp 585–591
45. Kwiatkowska M, Norman G, Segala R, Sproston J (2000) Verifying quantitative properties of continuous probabilistic timed automata. In: Concurrency theory (CONCUR'00). LNCS, vol 1877. Springer, Berlin, pp 123–137
46. Kwiatkowska MZ, Norman G, Segala R, Sproston J (2002) Automatic verification of real-time systems with discrete probability distributions. Theor Comput Sci 282(1):101–150
47. Lee EA (2002) Embedded software. In: Zelkowitz M (ed) Advances in computers, vol 56. Academic Press, San Diego
48. Legay A, Delahaye B, Bensalem S (2010) Statistical model checking: an overview. In: Runtime verification (RV). LNCS, vol 6418. Springer, Berlin, pp 122–135
49. Lynch NA, Segala R, Vaandrager FW (2003) Hybrid i/o automata. Inf Comput 185(1):105–157
50. Mader A, Bohnenkamp HC, Usenko YS, Jansen DN, Hurink J, Hermanns H (2010) Synthesis and stochastic assessment of cost-optimal schedules. Int J Softw Tools Technol Transf 12(5):305–318
51. Meseguer J, Sharykin R (2006) Specification and analysis of distributed object-based stochastic hybrid systems. In: Hybrid systems: computation and control (HSCC). LNCS, vol 3927. Springer, Berlin, pp 460–475
52. Panangaden P (2008) Labelled Markov processes. World Scientific, Singapore
53. Penna GD, Intrigila B, Melatti I, Tronci E, Zilli MV (2006) Finite horizon analysis of Markov chains with the Murphy verifier. Int J Softw Tools Technol Transf 8(4–5):397–409
54. Platzer A (2011) Stochastic differential dynamic logic for stochastic hybrid programs. In: Bjørner N, Sofronie-Stokkermans V (eds) CADE. LNCS, vol 6803. Springer, Berlin, pp 446–460
55. Preußig J, Kowalewski S, Wong-Toi H, Henzinger T (1998) An algorithm for the approximative analysis of rectangular automata. In: Formal techniques in fault tolerant and real time systems (FTRTFT). LNCS, vol 1486. Springer, Berlin, pp 228–240
56. Ratschan S, She Z (2007) Safety verification of hybrid systems by constraint propagation-based abstraction refinement. ACM Trans Embed Comput Syst 6(1):8
57. Segala R (1995) Modelling and verification of randomized distributed real-time systems. PhD thesis, MIT, Cambridge, MA, USA
58. Segala R, Lynch NA (1995) Probabilistic simulations for probabilistic processes. Nord J Comput 2(2):250–273

59. Sproston J (2000) Decidable model checking of probabilistic hybrid automata. In: Formal techniques in real-time and fault-tolerant systems (FTRTFT). LNCS, vol 1926. Springer, Berlin, pp 31–45
60. Strubbe S, van der Schaft A (2006) Compositional modelling of stochastic hybrid systems. In: Cassandras CG, Lygeros J (eds) Stochastic hybrid systems. Control engineering series. Taylor & Francis, London, pp 47–77
61. Wolovick N (2012) Continuous probability and nondeterminism in labeled transition systems. PhD thesis, FaMAF, UNC, Córdoba, Argentina
62. Yue H, Bohnenkamp HC, Kampschulte M, Katoen JP (2011) Analysing and improving energy efficiency of distributed slotted aloha. In: Smart spaces and next generation wired/wireless networking (NEW2AN). LNCS, vol 6869. Springer, Berlin, pp 197–208
63. Zhang L, She Z, Ratschan S, Hermanns H, Hahn E (2010) Safety verification for probabilistic hybrid systems. In: Computer aided verification. LNCS, vol 6174. Springer, Berlin, pp 196–211