Expressing Realtime Properties in VSE-II

Werner Stephan **Andreas Nonnengart** **Georg Rock**

German Research Center for Artificial Intelligence (DFKI GmbH)
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
{*stephan,nonnengart,rock*}*@dfki.de*

1 INTRODUCTION

It is well known that in many cases severe errors occur in the early stages of the software development process. Following this insight, considerable emphasis was put on requirements engineering. Formal methods provide means for the abstract yet precise description of required system properties. However, specialized description techniques are needed to treat certain aspects adequately. Moreover, complex systems often make it necessary to cover several of these views.

Among the most critical and most difficult (to describe) aspects of systems behaviors are realtime properties. A major problem in this area lies in the complexity of timing constraints and the variety of situations that can appear in such systems. It turned out that timed automata or (linear) hybrid automata serve as a comprehensive mean to express temporal behavior.

Verification Support Environment (VSE) supports abstract system specifications, the formulation and proof of safety and security properties, and refinements. The formal development method of VSE is based on Abstract Data Types and Temporal Logic. The aim of the work presented here is to integrate hybrid automata as a means for comprehensively describing realtime properties into the VSE method where an abstract system specification serves as the starting point of a formal refinement process. Hybrid automata (translated into VSE) provide a realtime view on these systems. Additional requirements specifications representing different views of the behavior can be ”linked” to the same specification.

As running example, we use the well-known gasburner scenario. The overall safety requirement of the gasburner is that the gas concentration should not overtake a certain critical threshold. For the sake of simplicity we abstract from continuous mathematics in the way that we do not consider the gas concentration but the time unlit gas flows out of the nozzle of the gasburner. For details, see the following two sections.

2 VSE AND HYBRID AUTOMATA

In what follows we present the Verification Support Environment and the hybrid automata used in our approach.

2.1 VSE-Verification Support Environment

In this section we describe shortly the methodology VSE-II is based on. The VSE-II system is a tool for the formal development of software systems. It consists of a basic system for editing and type checking specifications and implementations written in the specification language VSE-SL, a facility to display the development structure, a theorem prover for treating the proof obligations arising from development steps as for example refinements, a central database to store all aspects of the development and an automatic management of dependencies between development steps.

Compared to VSE-I [4, 5], which was based on a simple, non-compositional approach for state based systems, VSE-II [6] is extended with respect to comprehensive methods in order to deal with *distributed* and *concurrent systems* [8] and with respect to an even more efficient and uniform proof support which makes use of implicit structuring of the arising proof obligations. The basic formalism used in VSE-II is close to TLA (Temporal Logic of Actions) [7]. A refined *correctness management* allows for an evolutionary software development.

VSE-II is based on a methodology to use the structure of a given specification (e.g. parameterization, actualization, enrichment, or modules) to distribute also the deductive reasoning into local theories [9]. Each theory is considered as an encapsulated unit, which consists of its local signature and axioms. Relations between different theories, as they are given by the model-theoretic structure of the specification, are represented by different links between theories. Each

theory maintains its own set of consequences or lemmata obtained by using local axioms and other formulas included from linked theories.

This method of a structured specification *and* verification is reflected in the central data structure of a *development graph* (see Figure 1), the nodes of which correspond to the units mentioned above. It also provides a graphical interface for the system under development.

2.1.1 *Concurrent System Specifications*

Huge specifications are structured in several subcomponents which constitute the complete system specification including the environment specification. In the description of these units elementary specifications and different structuring operators for state based systems are used.

Elementary Specifications For the specification of state transition systems a specification language close to TLA [1, 7, 2] is used. In addition to the theory of compositional development presented in [2], which covers the composition of systems using input and output variables, *shared variables* are supported by the structuring operators in VSE-II. Furthermore the form of a specification of a component, also discussed in [2], is $\exists\, x_1, \ldots, x_n.(\mathrm{INIT} \wedge \Box[\text{SYS-STEPS}]_{\bar{v}} \wedge \mathrm{FAIR})$, where SYS-STEPS are the *actions* (steps) made by the system, $\bar{v}$ is the stuttering index, which contains flexible variables of the system, INIT is a predicate which holds initially, $x_1, \ldots, x_n$ are the internal variables and FAIR stands for the fairness requirements of the system.

Structuring of Specifications VSE-II provides operators to structure state-based specifications. We only focus on the **combine**-operator which models the concurrent execution of components. Concurrency is modeled by considering all possible *interleavings* of actions of the combined systems. Basically a behavior σ, which represents a sequence of states of the specified system, is a behavior of the combined system if and only if it is a behavior of every component of the system. However, in order to model the concurrent execution of, say $\mathcal{S}_1$ and $\mathcal{S}_2$, by conjunction, we have to allow environment steps in the (local) specifications of $\mathcal{S}_1$ and $\mathcal{S}_2$. In [2] environment steps are modeled by stuttering. This technique only works for communication by input-output variables, not in connection with shared variables. A more general approach [9, 6] is to associate a "color" with each component and to mark each step in a behavior by the color of the component which has done the step.

2.1.2 *Structured Deduction*

Structuring specifications as described above supports readability and makes it easier to edit specifications in that the user might use local notions. However, the system exploits structure beyond this purely syntactical level. Components of a combined system can be viewed as systems in their own right where certain parts can be observed from outside while most of the inner structure, including the flow of control and local program variables are hidden.

In particular we can prove properties of a combined system in a modular way. This means that we attach local *lemma bases* to components where local proofs are conducted and stored. Exchange of information between lemma bases is on demand. This approach has two main advantages: First, the given structure of the specification is used to reduce the search space in the sense that large parts of the overall system are not visible and second, storing of proofs local to certain lemma bases and making the export and import of information (between lemma bases) explicit supports the *revision* process.

2.1.3 *The Gasburner in VSE-II*

The development graph of the gasburner in VSE-II is shown in Figure 1. It consists of the abstract data type definitions `natural` and `definition` and of the temporal logic specifications `gasburner` and `safety`. The central part of the specification is the realization of the gasburner:

```
TLSPEC gasburner
 PURPOSE  "Specification of the gasburner. "
 USING definition
 DATA OUT x, y, t : nat
      OUT state : state_t
 ACTIONS
 phi_1 ::= state = leaking AND state' = non_leaking AND
           x' = 0 AND UNCHANGED(y, t)
```

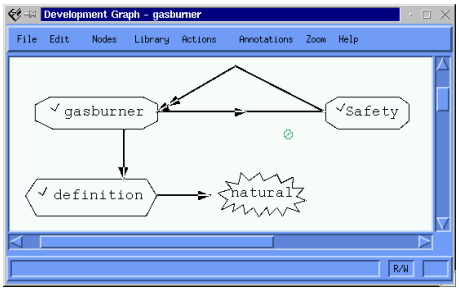


Figure 1: Development graph of the gasburner specification

```
 phi_2 ::= state = non_leaking AND state' = leaking AND
           x >= 30 * c AND x' <= c AND x' = 0 AND UNCHANGED(y, t)
 phi_1_star ::= state = leaking AND state' = non_leaking AND
                NOT x + 1 <= c AND x' = 0 AND UNCHANGED(y, t)
 psi_1 ::= state = leaking AND  state' = leaking AND
           x + 1 <= c AND y' = y + 1 AND
           x' = x + 1 AND t' = t + 1
 psi_2 ::= state = non_leaking AND state' = non_leaking AND
           x' = x + 1 AND y' = y + 1 AND UNCHANGED(t)
 SPEC INITIAL x = 0 AND y = 0 AND t = 0 AND state = leaking
      TRANSITIONS [phi_1, phi_2, phi_1_star, psi_1, psi_2]
                  {x, y, t, state}
      FAIRNESS WF(phi_1_star) {x, y, t, state},
               WF(psi_1) {x, y, t, state},
               WF(psi_2) {x, y, t, state}
 SATISFIES Safety
TLSPECEND
```

The specification uses symbolic values to express time bounds (`c`). This specification technique results in the following properties: The model has discrete time and different resolutions of the system can be expressed by interpreting hindsight the symbolic limits of the incrementation of the variables according to their constraints. In this way we are able to approximate a continuous behavior arbitrarily precise.

2.2 Hybrid Automata

Hybrid Systems [3] are real-time systems that are embedded in analog environments. They contain discrete and continuous components and interact with the physical world through sensors and actuators. Since they typically operate in safety-critical situations, the development of rigorous analysis techniques is of high importance.

A common model for hybrid systems can be found in *hybrid automata*. Briefly, such hybrid automata are finite graphs whose nodes correspond to global states. Such global states represent general observational situations, as, for instance, "the heater is on" or the "the heater is off". During these global states some continuous activity takes place. For instance, the temperature might rise or fall continuously according to a dynamical law until a transition from one node to another occurs.

These transitions are usually guarded with some constraint formula that is required to hold if the transition is to be taken. Moreover, they are annotated with a general assignment that is responsible for the discrete action to be performed by taking the transition.

Similarly, nodes have some attached constraint formula that describes an invariant for this very node, i.e., some property that has to be true while the system resides within this node. The dynamics of the system's behavior, on the other hand, is given by a description of how the data changes with time.

In what follows, we formally describe what a hybrid automaton is, i.e., we formally define their syntax as well as their semantics.

2.2.1 *Syntax*

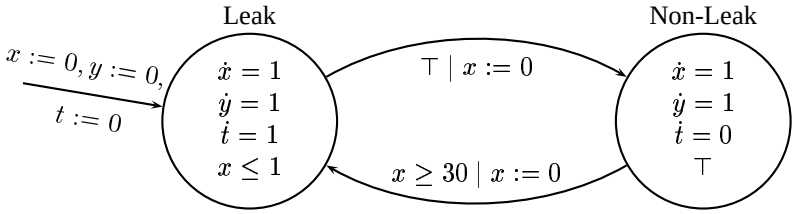
We assume a set CT of *Constraint Terms* over a fixed variable set X as the smallest set containing X, real-valued constants, and, moreover, which is closed under addition, subtraction, and multiplication with real-valued constants.

The set of CF of *Constraint Formulas* (over the variable set X) is defined as the smallest set that is closed under conjunction and contains $\top$ (truth) and $\bot$ (falsity) as well as all atoms of the form $t_1 > t_2$, $t_1 \geq t_2$, $t_1 < t_2$, $t_1 \leq t_2$, and $t_1 = t_2$, where t_1 and t_2 are constraint terms taken from CT.

Such constraint terms and formulas are a necessary prerequisite for the formal description of the hybrid systems themselves.

Definition 2.1 (Hybrid Systems) Hybrid Systems *are tuples of the form* $(X, \mathcal{L}, \mathcal{E}, \mathit{dif}, \mathit{inv}, \mathit{guard}, \mathit{act})$*, where* X *is a finite set of real-valued* data variables*,* $\mathcal{L}$ *is a finite set of* locations*, i.e., nodes of a graph,* $\mathcal{E} \subseteq \mathcal{L} \times \mathcal{L}$ *is a finite (multi)set of* transitions*,* $\mathit{dif}\colon \mathcal{L} \times X \mapsto$ *CT is a mapping that associates with each location and each data variable a constraint term, representing the change of the data variable within this location over time,* $\mathit{inv}\colon \mathcal{L} \mapsto$ *CF is a mapping that associates with each location a constraint formula, representing the location invariant,* $\mathit{guard}\colon \mathcal{E} \mapsto$ *CF is a mapping that associates with each edge a constraint formula (with free variables taken from* X*), representing the enabling condition for that transition, and* $\mathit{act}\colon \mathcal{E} \times X \mapsto$ *CT is a mapping that represents the discrete action to be peformed by taking the transition.*

As already noted earlier, we illustrate hybrid systems as graphs. As an example consider again the "Leaking Gas Burner" from above.



The above hybrid system describes the following behavior. It starts at location Leak with data variables all set to 0. Within the locations the value of the data variables increase by 1 every second (so they all represent *clocks*). There is one exception, however, the clock t has slope 0 in location Non-Leak, therefore it is actually stopped there.

The system leaves location Leak latest after one second and resets x to 0. Similarly, it remains within Non-Leak for at least 30 seconds and reenters Leak after having reset x to 0 again.

The clock x, therefore, acts as a control variable, y accumulates overall time and t counts leakage time (the amount of time the system resides within Leak).

2.2.2 *Semantics*

We define a *state* of a hybrid system as a pair (L, ϕ) where $L \in \mathcal{L}$ is a location and $\phi\colon X \mapsto \mathbb{R}$ is a valuation of the data variables. ϕ naturally extends to (constraint) terms and (constraint) formulas.

Given two states $\sigma = (L, \phi)$ and $\sigma' = (L', \phi')$ we say that σ' is *transition-reachable* from σ – denoted by $\sigma \overset{\text{tr}}{\mapsto} \sigma'$ – if there exists a transition $T = (L, L') \in \mathcal{E}$ with source L and target L', and both $\phi(\mathit{guard}(T))$ and $\phi'(x) = \phi(\mathit{act}(T, x))$ for each $x \in X$. In other words, σ' is transition-reachable from σ (via transition T) if the corresponding guard is true and the valuation of the data variables in σ' is changed according to the discrete transition action.

We call σ' *timely-reachable* from σ with delay δ – denoted by $\sigma \overset{\delta}{\mapsto} \sigma'$, where δ is a non-negative real number – if $L = L'$ and for each $x \in X$ there exists a differentiable function $f_x\colon [0, \delta] \mapsto \mathbb{R}$, with the first derivative $\dot{f}_x\colon (0, \delta) \mapsto \mathbb{R}$, such that (1) $f_x(0) = \phi(x)$ and $f_x(\delta) = \phi'(x)$ and (2) for all $\epsilon \in \mathbb{R}$ with $0 < \epsilon < \delta$: both $\mathit{inv}(L)[x_1/f_{x_1}(\epsilon), \ldots, x_n/f_{x_n}(\epsilon)]$ and $\dot{f}_x(\epsilon) = \mathit{dif}(L, x)[x_1/f_{x_1}(\epsilon), \ldots, x_n/f_{x_n}(\epsilon)]$ are true.

σ' is *timely-reachable* from σ – denoted by $\sigma \overset{\star}{\mapsto} \sigma'$ – if it is timely-reachable from σ with delay δ for some δ.

σ' is said to be *reachable* from σ with respect to $\mathcal{H}$, $\sigma \underset{\mathcal{H}}{\mapsto} \sigma'$, if $(\sigma, \sigma') \in (\overset{\star}{\mapsto} \cup \overset{\text{tr}}{\mapsto})^*$.

A *run* ρ of $\mathcal{H}$ with initial state $\sigma_0 = (L_0, \phi_0)$ is a sequence of states represented as $\rho = \sigma_0 \overset{t_0}{\underset{f_0}{\mapsto}} \sigma_1 \overset{t_1}{\underset{f_1}{\mapsto}} \sigma_2 \overset{t_2}{\underset{f_2}{\mapsto}} \sigma_3 \overset{t_3}{\underset{f_3}{\mapsto}} \cdots$ where $t_i \in \mathbb{R}^{\geq 0}$ and $f_i\colon [0, t_i] \mapsto (X \mapsto \mathbb{R})$, such that $f_i(0) = \phi_i$, $\mathit{inv}(L_i)[X/f_i(t)(X)]$ holds for all $0 \leq t \leq t_i$, $(L_i, f_i(t_i)) \overset{\text{tr}}{\mapsto} \sigma_{i+1}$ and for all $0 \leq t' \leq t' + \delta \leq t_i : (L_i, f_i(t')) \overset{\delta}{\mapsto} (L_i, f_i(t' + \delta))$.

Given such a run ρ, we call the state σ_0 the *starting state* of ρ and denote it by $\mathrm{start}(\rho)$. The set of all runs of a hybrid system $\mathcal{H}$ with initial state σ is denoted by $\mathrm{runs}(\mathcal{H}, \sigma)$.

A *position* π of a run $\rho = \sigma_0 \overset{t_0}{\underset{f_0}{\mapsto}} \sigma_1 \overset{t_1}{\underset{f_1}{\mapsto}} \sigma_2 \overset{t_2}{\underset{f_2}{\mapsto}} \sigma_3 \overset{t_3}{\underset{f_3}{\mapsto}} \cdots$ is a pair $\pi = (i, r) \in \mathbb{N} \times \mathbb{R}$ such that $0 \leq r \leq t_i$. We denote the set of positions of a run ρ as $\mathrm{pos}(\rho)$. Positions are ordered lexicographically, i.e., $(i, r) < (j, s)$ if and only

if $i < j$ or ($i = j$ and $r < s$). Also, $(i,r) \leq (j,s)$ if and only if $(i,r) < (j,s)$ or ($i = j$ and $r = s$). By $\rho(\pi)$ with $\pi = (i,r)$ we denote the state $(L_i, f_i(r))$.

Given a run $\rho = (L_0,\phi_0) \mapsto_{f_0}^{t_0} (L_1,\phi_1) \mapsto_{f_1}^{t_1} \cdots$, an $i \geq 0$, and a t with $0 \leq t \leq t_i$, we define the *suffix* of ρ, starting at $(L_i, f_i(t))$ as $(L_i, f_i(t)) \mapsto_{f_i'}^{t_i - t} (L_{i+1},\phi_{i+1}) \mapsto_{f_{i+1}}^{t_{i+1}} (L_{i+2},\phi_{i+2}) \mapsto_{f_{i+2}}^{t_{i+2}} \cdots$, where $f_i'(t') = f_i(t+t')$. By suf($\rho\prime, \rho$) we indicate that $\rho\prime$ is a suffix of ρ.

Within hybrid systems (automata) data variables change both continuously and discretely. The change is given by some rational number where the underlying base unit is in general neglected. Nevertheless, we usually have certain base units in mind when we specify systems., However, there might be a need for switching the base unit, for example from seconds to milliseconds. The following describes how such a change of the base units has to be performed formally.

Definition 2.2 (Granularity Change) *Given a Constraint Term $\mathcal{T}$ we define the* granularity change *of $\mathcal{T}$ by Δ as*

$$\Delta \star x = x \text{ for } x \in X \qquad \Delta \star (i * x) = i * x \text{ for } x \in X, i \in \mathbb{N}$$
$$\Delta \star i = \Delta * i \text{ for } i \in \mathbb{N} \qquad \Delta \star (t_1 + t_2) = \Delta \star t_1 + \Delta \star t_2$$

For Constraint Formulas $\mathcal{F}$ "$\star$" distributes in an natural way and leaves formulas as $(state = l)$ unchanged. For $\mathcal{H} = (X, \mathcal{L}, \mathcal{E}, dif, inv, guard, act)$, some given hybrid automaton, we define $\Delta \star \mathcal{H} = (X, \mathcal{L}, \mathcal{E}, dif, inv', guard', act')$ where $inv'(l) = \Delta \star inv(l)$, $guard'(e) = \Delta \star guard(e)$, and $act'(e)(x) = \Delta \star act(e)(x)$.We moreover extend the notion of granularity change to runs as follows: Given $\rho = (l_0,\phi_0) \mapsto_{f_0}^{t_0} (l_1,\phi_1) \mapsto_{f_1}^{t_1} (l_2,\phi_2) \cdots$. Then $\Delta \star \rho$ is defined as $(l_0, \Delta\star\phi_0) \mapsto_{\Delta\star f_0}^{\Delta * t_0} (l_1, \Delta\star\phi_1) \mapsto_{\Delta\star f_1}^{\Delta * t_1} (l_2, \Delta\star\phi_2) \cdots$, where $(\Delta\star f_i)(\Delta * t) = \Delta\star f_i(t)$ and $(\Delta\star\phi_i)(x) = \Delta \times \phi_i(x)$. For PSL formulas $\mathcal{F}$ we finally have $\Delta\star(P_1 \wedge P_2) = \Delta\star P_1 \wedge \Delta\star P_2$, $\Delta\star(\neg P) = \neg(\Delta\star P)$ and $\Delta\star(\Box P) = \Box(\Delta\star P)$.

2.2.3 *Property Specifications*

In what follows we describe the syntax and the semantics of the (linear temporal logic) property specification language (PSL) we utilize in this work. The general scenario shown in Figure 2 indicates that PSL and VSE-SL in a sense correspond.

Syntax and Semantics of PSL Any constraint formula $P \in \mathcal{CF}$ is a formula of PSL. If P and Q are formulas of PSL, then so are $P \wedge Q$ and $P \vee Q$. Also, if P is a formula of PSL, then so are $\Box P$ and $\Diamond P$. The expression $state = l$ is a PSL formula, where $l \in \mathcal{L}$ and $state$ is a special variable not occurring in X.

$(\mathcal{H},\sigma) \models P$	**iff**	$(\mathcal{H},\rho) \models P$ for all $\rho \in$ runs$(\mathcal{H},\sigma)$
$(\mathcal{H},\rho) \models state = l$	**iff**	start$(\rho) = (l,\phi)$ for some ϕ
$(\mathcal{H},\rho) \models t_1 \circ t_2$	**iff**	$\phi(t_1) \circ \phi(t_2)$ where $t_1, t_2 \in \mathcal{CT}$ and start$(\rho) = (l,\phi)$
$(\mathcal{H},\rho) \models P \wedge Q$	**iff**	$(\mathcal{H},\rho) \models P$ and $(\mathcal{H},\rho) \models Q$
$(\mathcal{H},\rho) \models P \vee Q$	**iff**	$(\mathcal{H},\rho) \models P$ or $(\mathcal{H},\rho) \models Q$
$(\mathcal{H},\rho) \models \Box P$	**iff**	$(\mathcal{H},\bar\rho) \models P$ for every $\bar\rho$ with suf$(\bar\rho,\rho)$
$(\mathcal{H},\rho) \models \Diamond P$	**iff**	$(\mathcal{H},\bar\rho) \models P$ for some $\bar\rho$ with suf$(\bar\rho,\rho)$

3 INTEGRATING HYBRID AUTOMATA IN VSE-II

Our aim is to embed linear hybrid automata into the VSE-II framework. Figure 2 illustrates the situation. Starting point is the specification of the behavior of a system with special interest on realtime using a hybrid automaton `Hybrid-AutSpec` as indicated in Figure 2. Properties that should be satisfied by this automaton are specified in `Hybrid-PropSpec`. The idea is to translate the hybrid automaton specification into a VSE-SL specification. From this translated specification, we want to prove that the properties described in `VSE-PropSpec` hold. A part of these properties is created by translating the properties in `Hybrid-propSpec` into VSE-SL properties. The proof of the properties can be performed in the VSE-II tool. The proof could equally be done with the help of a tool supporting hybrid automata. In that case we can guarantee (see Theorem 3.4) that the translated VSE-SL specification also fulfills the property. An advantage of such a method is the introduction of a fully automatic method for the handling of realtime systems in VSE-II. As can be seen in Figure 2 we have to define the satisfies and the translation relation between the given specifications. Satisfies in both cases means model inclusion semantically. In the VSE-II setting this is expressed by a satisfies link in the real specification as can be seen for example in Figure 1. In the hybrid automaton setting the satisfies link indicates that the hybrid automaton fulfills the specified property. How this is implemented is not our main concern at this point, though.

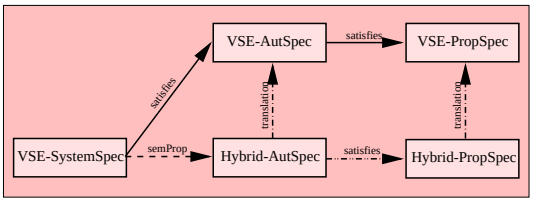


Figure 2: Relation: VSE- Hybrid Automata- specification

3.1 Definition of the Translation Function

The translation function is divided into three cases according to the system we are dealing with. The system is a hybrid automaton together with a definition of an initial state and a description of properties of the automaton. The translation has to be defined for all of these three parts. We begin by defining the translation of properties of a hybrid automaton.

3.1.1 *Translation Function*

Since the hybrid systems as introduced in 2.2 did not have an explicit initial location, we have to extend the definition in that way. We thus assume a set $init$ of pairs of type (CF, L) each representing an admissible initial location with a predefined variable valuation. The definition of the translation is divided into the translation of $init$, the translation of the continuous actions and the translation of the discrete actions.

We use the symbol π to name the translation function. As can be seen in the next definitions we overloaded π in order to make the definitions more readable. It will always be clear from the context which π is meant.

As for contraint terms and constraint formulas we define π as: $\pi(\top) \doteq$ TRUE and $\pi(\bot) \doteq$ FALSE, $\pi(t_1 := t_2) \doteq \pi(t_1) = \pi(t_2)$, and $\pi(e) \doteq c * e$, where c is a new constant. In all other cases π simply distributes over the operators and function symbols.

For instance: $\pi(x \leq 1) \doteq x \leq c$ and $\pi(3 * x + 2 \leq y) \doteq 3 * x + 2 * c \leq y$

Definition 3.1 *Let $init$ be a one element set consisting of the pair (f, l), let $x, x_1, \ldots, x_n$ data variables and l, l_1, l_2 locations.*

$$\pi(\{(f,l)\}) \doteq \pi(f) \wedge state = l$$

$$\phi \equiv \bigvee_{(l_1,l_2)\in\mathcal{E}} \Big(state = l_1 \wedge state' = l_2 \wedge \pi(guard((l_1,l_2)) \wedge \bigwedge_{x\in X} (x' = \pi(act((l_1,l_2),x))) \wedge (\pi(inv(l_2)))[x_1/x_1', \ldots, x_n/x_n'] \Big)$$

$$\phi^* \equiv \bigvee_{(l_1,l_2)\in\mathcal{E}} \Big(state = l_1 \wedge state' = l_2 \wedge \pi(guard((l_1,l_2)) \wedge \neg(\pi(inv(l_1)))[x_1/x_1 + dif(l_1,x_1), \ldots, x_n/x_n + dif(l_1,x_n)] \wedge \bigwedge_{x\in X} (x' = \pi(act((l_1,l_2),x))) \wedge (\pi(inv(l_2)))[x_1/x_1', \ldots, x_n/x_n'] \Big)$$

$$\psi \equiv \bigvee_{l\in L} \Big(state = l \wedge state' = l \wedge \bigwedge_{x\in X} (x' = x + dif(l,x)) \wedge (\pi(inv(l)))[x_1/x_1 + dif(l,x_1), \ldots, x_n/x_n + dif(l,x_n)] \Big)$$

$$stuttering \equiv \bigwedge_{x\in X} x' = x$$

Now the translation of a hybrid automaton including an initial condition is given by: $\pi(\mathcal{H}, init) \doteq \pi(init) \wedge \Box(\phi \vee \psi \vee stuttering)$, a formula in VSE-SL normal form.

Translation of Locations In the translation definition, the locations of the hybrid system are used as enabling conditions for the VSE-SL actions. Furthermore they indicate the change of a location. These enabling conditions are needed because of the translation of certain fairness constraints that are implicitly assumed in an hybrid system.

Treatment of Fairness The treatment of fairness is divided into two cases. First, all the translations of continuous actions from a hybrid system have to be fair. This is because time is assumed not to stop in a hybrid automaton. Every disjunctive part (action) of ψ is assumed to be executed in a fair manner. Therefore we impose weak fairness on every disjunct of ψ.

The second case is concerned with the handling of discrete actions. In this situation we may not impose fairness on every discrete action in the VSE-SL translation since not every discrete action in a hybrid system is executed fairly. A hybrid system may perpetually stay in one of its locations – provided it does not violate the corresponding invariant – without taking a discrete action although the guard is satisfied infinitly many times. The $non\_leaking$ location in the gasburner example is of this kind. We are not allowed to require fairness of the discrete action in such situations. On the other hand we are also not allowed to omit the fairness constraints on the discrete actions in general.

One possible solution to this problem would be to collect all the actions of a system in one action and to require weak fairness for this very action. The solution we suggest, however, distinguishes between those actions that have to be executed in a fair manner from those that don't. This distinction is already part of the translation (see Definition 3.1).

The translation of a hybrid automaton is represented by the following formula: $\pi(\mathcal{H}, init) \doteq \pi(init) \wedge \Box(\phi \vee \phi^* \vee \psi \vee stuttering) \wedge FAIR(\phi^*) \wedge FAIR(\psi)$ where $FAIR(\ldots)$ stands for the application of weak fairness on all the disjunctive parts of the argument formula.

3.2 Main Theorem

After the description of the translation function we present the main theorem of our approach which states the correctness of the translation function π and the discretization step. One of the outcomes of this theorem is that the diagram shown in Figure 2 commutes. Before stating the theorem, however, we first give some preliminaries needed for the proof. These cover the discretization technique as well as some interesting properties of the granularity change for hybrid automata.

3.2.1 *Discretization*

Here we show how we discretize hybrid automata. The discretization step results in a parameterized (δ) hybrid automaton such that the final outcome is an *exact* approximation.

Given a hybrid automaton $\mathcal{H}$ with $\mathcal{H} = (X, \mathcal{L}, \mathcal{E}, dif, inv, guard, act)$ and $X = \{x_1, \ldots, x_n\}$ we define $\Gamma(\mathcal{H}) = (X, \mathcal{L}, \mathcal{E} \cup \mathcal{E}', dif', inv', guard \cup guard', act \cup act')$, where $\mathcal{E}' = \bigcup_{l \in \mathcal{L}} l \times l$, $dif'(l)(x) = 0$ for all $l \in \mathcal{L}$, $x \in X$, $inv'(l) = \top$ for all $l \in \mathcal{L}$, $act'((l,l))(x) = x + dif(l)(x) * \delta$ for all $(l,l) \in \mathcal{E}'$, $x \in X$ and $guard'((l,l)) = inv(l)[x_1/act'(l)(x_1), \ldots, x_n/act'(l)(x_n)]$ The result of $\Gamma(\mathcal{H})$ increases the data variables by multiplies of δ. However our aim is to have an increase by multiples of 1. We achieve this by multiplying each constant with $\Delta = \frac{1}{\delta}$ which is illustrated in section 3.2.2

The following lemma is needed in Theorem 3.4. It says that a hybrid automaton and the Γ-transformed automaton satisfy the same properties.

Lemma 3.2

$(\mathcal{H}, \sigma) \models \Box P$ iff $(\Gamma(\mathcal{H}), \sigma) \models \Box P$ for every δ and P contains no further temporal operator
$(\mathcal{H}, \sigma) \models \Diamond P$ iff $(\Gamma(\mathcal{H}), \sigma) \models \Diamond P$ for some δ and P contains no further temporal operator

Proof: The proof follows immediately from the fact that $\sigma_1 \underset{\mathcal{H}}{\mapsto} \sigma_2$ iff $\sigma_1 \underset{\Gamma(\mathcal{H})}{\mapsto} \sigma_2$ for some $\delta \in \mathbb{Q}$ which in turn can be proved by induction on $\#(\sigma \underset{\mathcal{H}}{\mapsto} \sigma')$, the number of steps of $\mathcal{H}$. □

3.2.2 *Granularity Change*

Before stating the main theorem we first proof a theorem which tells us about the correlation between an hybrid automaton $\mathcal{H}$ and its granularity changed version with respect to a PSL property P.

Theorem 3.3 $\mathcal{H}, \rho \models \mathcal{F}$ *iff* $\Delta \star \mathcal{H}, \Delta \star \rho \models \Delta \star \mathcal{F}$

Proof: By induction on the structure of $\mathcal{F}$. Here we only consider the case $\mathcal{F} = \Box \mathcal{F}_1$.

$$
\begin{array}{lll}
(\mathcal{H},\rho) \models \Box\mathcal{F}_1 & \text{iff} & (\mathcal{H},\rho') \models \mathcal{F}_1 \text{ for all } \rho' \text{ with } \mathrm{suf}(\rho',\rho) \text{ [by definition]} \\
& \text{iff} & (\Delta \star \mathcal{H}, \Delta \star \rho') \models \Delta \star \mathcal{F}_1 \text{ for all } \rho' \text{ with } \mathrm{suf}(\rho',\rho) \text{ [by hypothesis]} \\
& \text{iff} & (\Delta \star \mathcal{H}, \rho') \models \Delta \star \mathcal{F}_1 \text{ for all } \rho' \text{ with } \mathrm{suf}(\rho', \Delta \star \rho) \\
& & \quad \text{[since } \{\rho_1 \mid \mathrm{suf}(\rho_1, \Delta \star \rho)\} = \{\Delta \star \rho_2 \mid \mathrm{suf}(\rho_2,\rho)\}\text{]} \\
& \text{iff} & (\Delta \star \mathcal{H}, \Delta \star \rho) \models \Box(\Delta \star \mathcal{F}_1) \text{ [by definition]} \\
& \text{iff} & (\Delta \star \mathcal{H}, \Delta \star \rho) \models \Delta \star (\Box\mathcal{F}_1) \text{ [by definition]}
\end{array}
$$

and this completes the proof. □

Now we come to the main theorem which states the connection between the discretization, the granularity change and the translation function π.

Theorem 3.4 (Main Theorem) *Let $\mathcal{H}$ be a hybrid automaton, π the translation function and P a PSL formula with no nested temporal operators. Then $(\mathcal{H},\rho) \models P$ iff $(\pi(\mathcal{H}),\rho) \models \pi(P)$.*

Proof: According to the definition of Γ and Lemma 3.2 we perform a discretization step which results in a discrete hybrid automaton $\Gamma(\mathcal{H})$ such that $\Gamma(\mathcal{H})$ satisfies the same PSL formulas as $\mathcal{H}$ does. Note that $\Gamma(\mathcal{H})$ is an automaton that is parameterized with δ. In the next step we perform a granularity change according to Theorem 3.3 with $\Delta = \frac{1}{\delta}$. This results in an automaton $\mathcal{H}' = \Delta \star \Gamma(\mathcal{H})$ that satisfies $\Delta \star P$.

The purpose of $\mathcal{H}'$ was to come as close as possible to $\pi(\mathcal{H})$ such that merely a simple transformation step remains to be performed in order to achieve $\pi(\mathcal{H})$. $\Delta \star P$ is identical to $\pi(P)$ anyway (modulo parameter renaming). □

Summarizing: We have presented a method to integrate hybrid automata into VSE-II. To achieve this we developed a technique to exactly discretize a hybrid automaton and to change the granularity of a hybrid automaton. Moreover, we showed that the right part of the diagram in Figure 2 commutes.

References

[1] Martín Abadi and Leslie Lamport. The Existence of Refinement Mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.

[2] Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, May 1995.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[4] Dieter Hutter, Bruno Langenstein, Claus Sengler, Jörg H. Siekmann, Werner Stephan, and Andreas Wolpers. Deduction in the Verification Support Environment (VSE). In Marie-Claude Gaudel and James Woodcock, editors, *Proceedings Formal Methods Europe 1996: Industrial Benefits and Advances in Formal Methods*. SPRINGER, 1996.

[5] Dieter Hutter, Bruno Langenstein, Claus Sengler, Jörg H. Siekmann, Werner Stephan, and Andreas Wolpers. Verification support environment (vse). *High Integrity Systems*, 1(6):523–530, 1996.

[6] Dieter Hutter, Heiko Mantel, Georg Rock, Werner Stephan, Andreas Wolpers, Michael Balser, Wolfgang Reif, Gerhard Schellhorn, and Kurt Stenzel. VSE: Controlling the Complexity in Formal Software Development. In D. Hutter, W. Stephan, P. Traverso, and M. Ullmann, editors, *Proceedings Current Trends in Applied Formal Methods, FM-Trends 98*, pages 351–358, Boppard, Germany, 1999. Springer-Verlag, LNCS 1641.

[7] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3), 1994.

[8] Georg Rock, Werner Stephan, and Andreas Wolpers. Tool support for the compositional development of distributed systems. In *Tagungsband 7. GI/ITG-Fachgespräch Formale Beschreibungstechniken für verteilte Systeme*, number 315 in GMD Studien. GMD, 1997.

[9] Georg Rock, Werner Stephan, and Andreas Wolpers. Modular Reasoning about Structured TLA Specifications. In R. Berghammer and Y. Lakhnech, editors, *Tool Support for System Specification, Development and Verification*, Advances in Computing Science, pages 217–229. Springer, WienNewYork, 1999.