Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 253 (2009) 53–69

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Conformance Testing of Hybrid Systems with Qualitative Reasoning Models

Bernhard K. Aichernig, Harald Brandl, Franz Wotawa[1]

Abstract

Embedded systems are of growing importance in industry. For example, in a today's vehicle a huge number of embedded and communicating systems can be found. Exhaustive testing of such systems is a requirement, because changes after delivery and use are expensive and sometimes even impossible. In this paper we propose the use of qualitative models, which are an abstraction of quantitative physical models, for test case generation and test execution. In particular, we show how Simulink models from which control programs are automatically extracted can be tested with respect to qualitative models. Since Simulink models are heavily used in industry, the approach is of practical interest.

Keywords: conformance testing, hybrid systems, qualitative reasoning, qrioconf, Garp3

1 Introduction

In industry and especially in the automotive industry Simulink is often used to implement control programs for various purposes. One reason is that those models can be directly converted into C code, which runs on the vehicle's electronic control units (ECUs). As a consequence Simulink models have to be tested thoroughly. This holds in particular for safety critical systems. In order to meet the safety and quality criteria of such models automated test case generation and more specifically model-based testing is of specific interest but has hardly been explored. In order to fill this gap we present an approach that makes use of qualitative models for model-based test cases generation.

Qualitative models represent basically cause-effect relationships and constraints on model variables. They can be seen as an abstraction of the usually implemented quantitative differential equation models when using Simulink or other modeling languages. Hence, Simulink models are a refinement of qualitative models. This is in contrast to the use of other means for representing models in this domain like

[1] Authors are listed in alphabetical order.

[2] Research herein was funded by the EU project ICT-216679, Model-based Generation of Tests for Dependable Embedded Systems (MOGENTES).

doi:10.1016/j.entcs.2009.09.051

hybrid automata, which shares basically the same abstraction level with Simulink models.

In order to allow for using qualitative models for model-based testing we have to specify the equality relation between the specification and the implementation. For this purpose we introduce a new conformance relation, which is motivated by Tretmans input-output conformance relation [21]. Using this relation a test case obtained from a qualitative model can be used to stimulate the corresponding Simulink model and to compare the outcomes. A test case generated from a qualitative model in our case is nothing else than a graph representing potential changes of states where a state comprises the variables and their qualitative values.

In this paper we focus less on test case generation but more on test execution and conformance checking. The test case generation part can be found elsewhere [4,3,5]. To prove the applicability of our approach we further present a case study. In this case study the conformance relation is used to show that mutants of a Simulink model can be detected using the automatically generated test cases.

Next, Section 2 motivates our work by a running example of a hybrid system. Section 3 introduces to the AI-technique of qualitative reasoning and to the corresponding qualitative models. In Section 4, we formalize the conformance relation between qualitative models and hybrid implementations. Then, in Section 5 we discuss our test case generation and execution technique, including the mapping between the abstract and the concrete domains. Section 6 covers experiments of the case study. Finally, in Section 8 we relate to our work, draw the conclusions and give an outlook to future work.

2 Hybrid Systems

Hybrid systems describe both the continuous and the discrete behavior of systems. In the embedded systems domain it is often the case that the interaction of the system with the environment is continuous. In this context we denote interactions as continuous even though they are quantized from the reals to the integers. Continuous actions can be described with differential equations that define the behavior of variables as continuous, differentiable functions. Testing the continuous behavior of systems means to check if after applying specified inputs to the implementation the observed outputs occur with the right value at the right time. In contrast discrete actions represent certain events in a system like the update of a variable or the occurrence of an interrupt.

In the following we use a water tank with two outlets as a running example, see Figure 1. The outlet at the bottom can be controlled continuously with a valve v between the states *open* and *closed*. The hybrid system comprises four modes $S0 - S3$ that we describe with the partially defined differential equations:

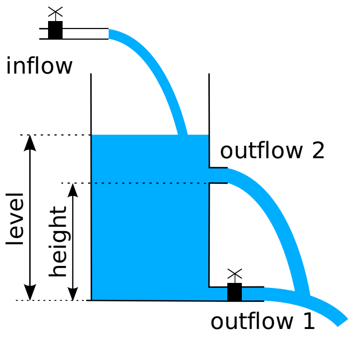


Fig. 1. Water Tank with two Outlets.

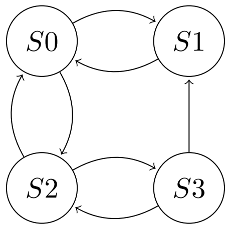


Fig. 2. Hybrid Automaton of the Water Tank.

$$\dot{x} = \begin{cases} i - k1\sqrt{x} - k2\sqrt{x-h} & \text{if } x \geq h \wedge v = \text{open}; \quad S0 \\ i - k2\sqrt{x-h} & \text{if } x \geq h \wedge v = \text{closed}; \quad S1 \\ i - k1\sqrt{x} & \text{if } x < h \wedge v = \text{open}; \quad S2 \\ i & \text{otherwise.} \quad S3 \end{cases} \tag{1}$$

where x denotes the amount of water in the tank. The water level is the amount of water in the tank divided by the tank's base area and h is the height of the upper outlet. The constants *k1* and *k2* are the cross sections of the two outlets. In our example we use the cross section *k1* of the bottom outlet as variable which depends on the position of the valve.

As there are several representations for discrete event systems the same is true for hybrid systems. Hybrid formalisms combine existing ones for discrete systems, e.g., finite state machines or process algebras, with the theory of differential equations. This extends finite state to infinite state systems. In [10] Henzinger introduces the theory of hybrid automata. A hybrid automaton consists of a finite set of states linked via a transition relation, a jump condition which assigns to each edge a predicate, events corresponding to an edge, a set of real numbered variables, and three properties which must be satisfied in each state: *init* is a predicate which defines the initial condition of the variables, the *invariant* predicate, and the *flow conditions* stating the differential behavior of variables. The work deals with a formal definition and classification of hybrid automata. Further the author discusses the symbolic analysis of hybrid automata with model checking techniques. Figure 2 depicts the hybrid automaton of our example. The labels in the four control modes denote the according invariant and differential equation defined in (1). The edges of the automaton have no conditions. The automaton moves between states as the state invariants change.

In order to verify such systems, model checkers apply abstraction techniques which partition the state space into a finite set of infinite regions. These abstraction schemes are based on differential inclusion, i.e. finding bounds within which the real value is located. For instance HyTech [11] is such a model checker for hybrid automata. However, this approach imposes some restrictions on the form of the underlying differential equations. Hence, as an alternative, we propose the use of Qualitative Reasoning (QR), an AI technique, to model and analyze the continuous part of hybrid systems. It enables us to reason with a user defined level of abstraction about systems in a time abstract fashion. The restriction to continuous behavior is due to the fact that QR relies on the theory of Qualitative Differential Equations (QDEs) [15]. Solving ordinary differential equations (ODE) is a challenge. A standard approach is to lift the ODEs to other domains, e.g., via Laplace transformation, where the calculation is easier. In this work ODEs are abstracted to the domain of QDEs, where QR resolves the relational, declarative information of QDEs by cause-effect reasoning. Obviously, the price to pay is a loss of precision of the derived solutions. However, as we will show, for testing purposes the technique is adequate.

In [4] we use QR models, built manually by requirement engineers, as abstract test models for test case generation. This enables us to check if the behavior of the continuous variables of a hybrid system is a refinement of an according QR model regarding a conformance relation. In section 4 we define *qrioconf* as conformance relation between QR models.

In the following we give a brief introduction to QR and Garp3 and present a qualitative model of our example.

3 Qualitative Reasoning

Qualitative Reasoning (QR) is an AI technique for reasoning about physical systems with incomplete knowledge. Systems are specified in a declarative manner by stating relations between variables. A reasoning engine derives all possible behaviors that do not contradict these relations. Relations are constraints in the language of Qualitative Differential Equations (QDEs). Common QR tools are *QSIM* [15] and *Garp3* [6].

When modeling systems using QR one abstracts the domain of continuous, real valued variables of a system to the domain of points and intervals. A point, called landmark, is a boundary between intervals. An abstract domain is called *quantity space (QS)*. For instance when we consider the temperature of water from a qualitative point of view we can map the temperature to the domain: below 0°C water is frozen, at 0°C it melts, above 0°C it is liquid, and around 100°C, depending on the pressure it starts boiling. Above 100°C it changes to steam. The qualitative domain consists of three intervals and two points, i.e. 0°C and 100°C. In QR time is abstracted to a sequence of temporally ordered states of behavior. A state consists of a complete valuation of all system variables, called *quantities*. The value of a quantity is a pair consisting of an abstract value $qVal \in QS$ from its quantity space and the direction of change δ. The direction of change δ is an abstraction of the first derivation $\frac{\partial}{\partial t}$ where δ is either increasing, decreasing, or steady. The behavior of a QR model is the set of all sequences of states starting from an initial state. A QR state is left if one of the quantities changes its behavior, i.e. there is either a change of $qVal$ or δ.

3.1 Garp3 Models

The tool Garp3 supports the building and inspection of QR models. Its foundation is Qualitative Process Theory [9]. A detailed description of the functions can be found in the user manual [2], and there is an elaborate user guide [7] for building QR models.

Garp3 creates an *envisionment*, i.e. a state graph containing all possible future valuations of quantities, regarding a QR model and an initial scenario. The initial scenario defines the initial values of a system. The state graph is deterministic and in the following we refer to it as QR transition system (QR TS).

A Garp3 model consists of a set of model fragments, that capture certain modes of a system and are activated when required conditions are met. In our example, the effect of the valve on the outflow at the bottom is common in all four modes. Therefore, only two model fragments were needed. The further mode distinctions in the hybrid automaton result from the fact that a closed valve causes no outflow. Figure 3 shows a model fragment which comprises mode *S0* and *S1* of the hybrid automaton in Figure 2. The model is created by applying *Qualitative Abstractions* [16] on the equations, given in (1).

A model fragment is a graph representation of qualitative relations (edges) between a set of quantities (nodes). Quantities in a Garp3 model are associated with an entity. In our example the entity *Tank* has the quantities *Inflow*, *Outflow*, *Valve*, *Level*, *Diff a*, *Diff b*, and *Diff c*. The model contains three difference relations, i.e. *Inflow* − *Outflow* = *Diff a*, *Level* − *Level.Set* = *Diff b*, and *Diff a* − *Diff b* = *Diff c*. We use them to calculate the effective flow rate *Diff c* that influences the water level. The Landmark *Level.Set* represents the level of the upper outlet. Proportionalities establish a mathematical relation between two quantities in the form of a monotonic increasing or decreasing function. The notation P+ (Q1;Q2) expresses that a change of Q2 causes a change of Q1 in the same direction. A proportionality with a minus sign states that a change of the cause quantity induces a change in the opposite direction on the affected quantity. For instance we have a direct proportionality between quantity *Inflow* and *Diff a*, i.e. P+ (Diff a;Inflow). Due to qualitative abstraction intermediate quantities that are chained via proportionalities can be removed if they are not needed for computation. In our example the water level *Diff b* above the upper outlet is proportional to a flow rate. Hence we can directly use *Diff b* as a flow rate.

Influences cause dynamic changes in a system and provide a means for integration. For instance, I+ (Q1,Q2) states that the value of Q2 determines the direction of change of Q1. If Q2 is positive Q1 increases, if Q2 is zero Q1 stays steady, and if Q2 is negative Q1 decreases. We have a positive influence of quantity *Diff c* on the water level. A further possibility to constrain qualitative behavior are *inequalities*. Inequalities are ordinal relations over quantity values and δs. In our example the inequality *Level* > *Level.Set* is a condition which must be satisfied s.t. the model fragment is considered by the reasoning process. As last qualitative constraint our model fragment contains two *value correspondences*. The directed correspondence from *Valve.Zero* to *Outflow.Zero* is an implication stating that if *Valve* is zero then also *Outflow* is zero. The other possibility that the *Outflow* gets zero is when there is no water in the tank.

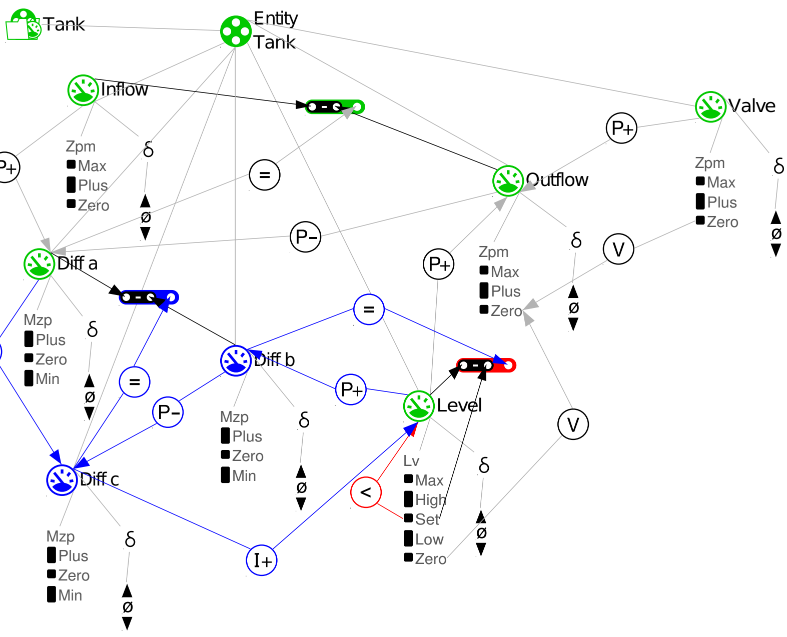


Fig. 3. Tank Model Fragment for Mode S0 and S1.

In the following we present a conformance relation between QR models.

4 Conformance Relation for QR-Models

Garp3 takes as input a system model and an initial scenario and produces as output a QR TS, defined in [4]. For the purpose of testing we use the QR TS as test model and we partition the set of quantities Q into disjoint sets of inputs L_I, outputs L_U, and hidden quantities. We denote the set of observable quantities $L = L_I \cup L_U$.

We refer to the set of state sequences in a QR TS, starting from the initial state, as *Traces(QR TS)*. After a minimization step, described in [5], which ignores hidden quantities we get a QR TS that reflects the behavior of the original QR TS projected on inputs and outputs, i.e. *Traces(QR TS)* $\downarrow L$. By combining inputs and outputs to vectors we can represent states in the QR TS as $(\overrightarrow{L_I}, \overrightarrow{L_U})$ pairs. The behavior of a QR TS may be infinite if the QR TS contains cycles.

An initial version of a conformance relation between QR models can be found in [5]. It is based on the input/output conformance relations by Tretmans in [21]. QR models as abstractions of continuous systems are strongly responsive, i.e. for every input exists an observable output, hence there is no need to deal with quiescence as considered in the *ioco* relation. Input and output are coupled and traces can be seen as two synchronous vector streams. Hence we modify the *ioconf* relation to *qrioconf*:

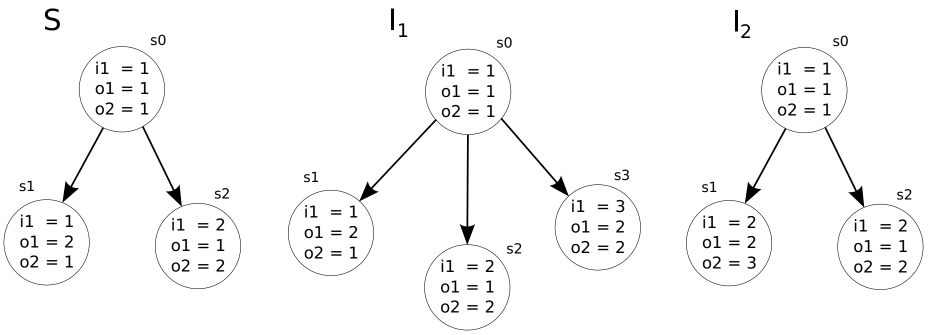


Fig. 4. Conformance between QR transition systems: I_1 qrioconf S and I_2 ~~qrioconf~~ S

Definition 4.1

$$i \ \mathbf{qrioconf} \ s =_{df} \forall \sigma \in Traces(s) \downarrow L_I \ : \quad \mathbf{out}^{\sim}(i \ \mathbf{after}^{\sim} \ \sigma) \subseteq \mathbf{out}^{\sim}(s \ \mathbf{after}^{\sim} \ \sigma)$$

where i is an IUT, s is the specification (QR TS), $Traces(s) = \{\langle t_0, t_1, \dots\rangle \mid t_0 = s_0 \wedge \forall i \geq 0 : t_i \in S \wedge (t_i, t_{i+1}) \in T\}$. For $\sigma \in Traces(s) \downarrow L_I$ the set s $\mathbf{after}^{\sim}$ $\sigma = \{t_n \in S \mid \sigma = \langle \overrightarrow{L}_{I0}, ..., \overrightarrow{L}_{In}\rangle \wedge \ \overrightarrow{L}_{Ii} \in S \downarrow L_I \wedge \mu = \langle t_0, \dots, t_n\rangle \wedge t_0 = s_0 \wedge t_i \in S \wedge (t_i, t_{i+1}) \in T \wedge \sigma = \mu \downarrow L_I\}$. In this definition, S is the set of states, s_0 is the initial state, T is the transition relation, $out(s)$ describes the state $s \in S$ with its output quantities L_U, and their values and deltas:

$$\mathbf{out}^{\sim}(s) =_{df} \{(q, v(s,q), \delta(s,q)) \mid s \in S \wedge q \in L_U\}$$

$$\mathbf{out}^{\sim}(S) =_{df} \bigcup_{s \in S} out(s)$$

The function $v(s,q)$ valuates a quantity q in a state s to an qualitative domain value $qVal \in QS$. The function $\delta(s,q)$ maps a quantity q in a state s to $\{min,\ zero,\ plus\}$ respectively.

Figure 4 shows three QR transition systems where we use integers as landmark values. In the following we refer to the values of inputs and outputs within a state in a vectorized form, $[i_1, \dots, i_n]^T$ and $[o_1, \dots, o_m]^T$ respectively. The states s_0 of I_1 and I_2 are *qrioconf* to S since I_1 $\mathbf{after}^{\sim}$ $\langle[1]\rangle = I_2$ $\mathbf{after}^{\sim}$ $\langle[1]\rangle = \{s_0\}$ and the outputs $\mathbf{out}^{\sim}(s_0) = \{[1,1]^T\}$ are allowed in S after the same input trace $\langle[1]\rangle$. The outputs in I_1 after the input traces $\langle[1],[1]\rangle$ and $\langle[1],[2]\rangle$ are both allowed in the specification S. For the third input trace $\langle[1],[3]\rangle$ the last input $i_1 = 3$ is not specified in S. Due to implementation freedom the behavior following unspecified inputs is not considered by the conformance relation. For I_2 we get I_2 $\mathbf{after}^{\sim}$ $\langle[1],[2]\rangle = \{s_1, s_2\}$. However, the outputs $\mathbf{out}^{\sim}(\{s_1, s_2\})$ of I_2 are not a subset of $\mathbf{out}^{\sim}(S$ $\mathbf{after}^{\sim}$ $\langle[1],[2]\rangle)$. Hence I_2 is not *qrioconf* S.

5 Test Case Generation and Test Execution

We generate test cases according to test purposes [4], similar to the testing tool *TGV* [13], and certain coverage criteria [3]. Starting from a QR model the main steps to a test case are:

- Simulation of the QR model yielding a QR TS (using Garp3).
- We annotate the unlabeled QR TS with symbolic labels that correspond to properties of the quantities, resulting in a QR Labeled Transition System (QR LTS). A regular expression over these labels expresses a test purpose which specifies the behavior of interest.
- The product of the QR LTS with the test purpose yields the Complete Test Graph (CTG).
- We minimize the CTG due to input and output quantities, hidden quantities are ignored, leading to nondeterminism.
- After determinization we use the obtained CTG as test case.

Test cases have to be controllable, i.e. the tester has no choice between different inputs that can be applied to the implementation or between applying an input or observing outputs. In previous work we used this form of input controllability which leads to several test cases per CTG. In Section 5.2 we discuss why an adaptive input selection is preferable, in our case. Since inputs and outputs are coupled and occur alternately we only have to deal with the selection of inputs.

For offline test case generation we need envisionment simulation, i.e. the simulation of all possible futures starting from an initial scenario. This can lead to quite big state spaces. In order to deal with the state space explosion problem the simulation engine can be constraint to produce fewer possible futures. Another option is to state additional constraints between quantities that reduce ambiguities. For instance in a two-tank-system the information that one tank is higher than the other can be expressed by a constraint between landmark values: $Tank1.Height > Tank2.Height$.

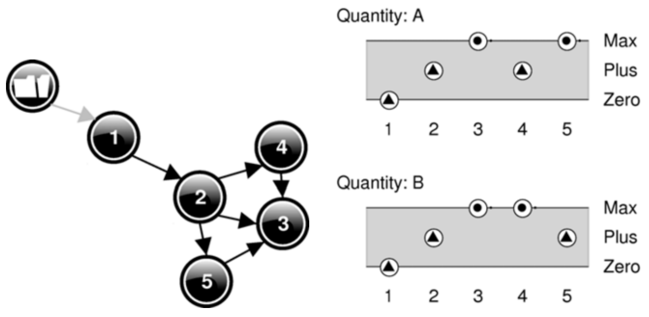
In order to establish a link between the abstract behavior of a QR TS and the continuous behavior of a hybrid system we have to map between the domains accordingly. Such a link is called a *Galois Connection* and we describe the mapping functions in the following section.

5.1 Mapping between Abstract and Concrete Data

An inherent effect of qualitative abstraction, or abstraction in general, is the increment of possible behavior. In contrast to the exact solution of an ODE's *Initial Value Problem* (IVP) the solution of a QDE and an initial scenario, let us call it *Initial Scenario Problem* (ISP), in general, is not unique. This over-approximation ensures, that if there exists a solution to an IVP then it will we contained in the solution of the according ISP.

The approach also allows specifications to be approximations rather than abstractions. Consider a QR model *M1* that is a direct abstraction of an underlying continuous system *M0*. Then we can remove constraints (relations) from *M1* and the resulting model will still be consistent with all behaviors that can occur in the models below it, i.e. *M0* and *M1*.

QR state spaces abstract the timing behavior of physical systems by all possible interleavings of quantity valuations. Figure 5 depicts the behavior of two unrelated, increasing quantities which start from point *zero*. The simulation predicts three possibilities: in state sequence $\{1, 2, 5, 3\}$ quantity A reaches the maximum before quantity B, in state sequence $\{1, 2, 4, 3\}$ quantity B reaches the maximum before quantity A, and in state sequence $\{1, 2, 3\}$ both quantities increase in-phase to the maximum.



(a) State Graph. (b) Value History.

Fig. 5. Interleaving of two increasing Quantities.

$$\begin{tikzcd}
L_I \arrow[r] \arrow[d, "\alpha"] & L_U \\
I_t \arrow[r] & O_t \arrow[u, "\beta"]
\end{tikzcd}$$

Fig. 6. Mapping between Abstract and Concrete Data.

Qualitative test cases are abstract in two dimensions: time and magnitude. In order to execute them on a real implementation we have to map between abstract and concrete domains. This is commonly presented in a commuting diagram, see Figure 6, where α is a refinement function and β is an abstraction function. It states that after mapping an abstract input $i_s \in L_I$ to a concrete input $i_t \in I_t$, applying it to an implementation, and then mapping the observed output $o_t \in O_t$ to an abstract, qualitative output $o_s \in L_U$ must be the same as applying the abstract input i_s to the specification. The subscript s denotes state dependency and subscript t stands for time dependency. In the following we denote $(o_t(t), \frac{\partial}{\partial t} o_t(t)) \in (\mathbb{R}, \mathbb{R})$ as concrete output and $i_s \in L_I = QS \times \delta$ as abstract (qualitative) input. For test case execution we have to map the abstract quantity spaces of each observable quantity $q \in L$ to a concrete domain *CD*, i.e. a partition of real valued intervals. For instance quantity space QS_{level} of a quantity *level* with $QS_{level} = \{zero, low, medium, high, max\}$ corresponds to a concrete domain $CD_{level} = \{[0, 0], (0, 5), [5, 5], (5, 10), [10, 10]\}$. We can map between an abstract value $qVal \in QS$ and a real valued interval $c_i \in CD$ by the functions $interval : L \times QS \to CD$, and $quality : L \times CD \to QS$.

For qualitative inputs L_I we refine the infinite set of increasing and decreasing functions to linear functions I_t by applying the refinement function α. For calculating the slope we assume a certain time interval TI within which the linear function is a refinement of the abstract (QS, δ) input pair. Whenever we enter a new QR state we continue the piecewise linear function starting from the last concrete value of the previous state. Let us consider the following example. We want to map the abstract input $i_s = (A, plus)$ to a concrete linear function. Every abstract value matches a concrete real valued interval by definition, e.g., $A =_{df} [3.7, 5.8]$. Assume that the last value $cVal = i_t(t)$ is 4.1 and the time interval is one second. Then the slope is $(5.8 - 4.1)/1 = 1.7$. By multiplying the slope with the time step $deltaT$ we

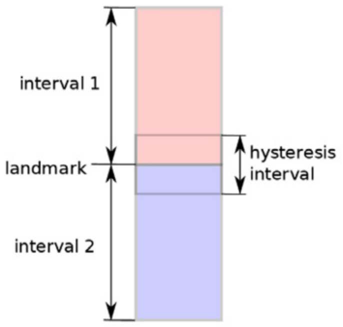


Fig. 7. Landmarks as Real Valued Intervals.

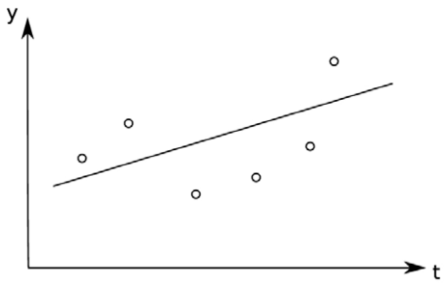


Fig. 8. Slope of observed Values.

get the increment value *deltaY* that updates the function at each time step. The choice of *deltaT* must be fine-grained enough to detect the fastest signal changes of interest.

In order to decide the output conformance of an implementation we have to abstract the observed, concrete outputs O_t to qualitative outputs $L_U = QS \times \delta$ by applying the abstraction function β. Since outputs are pairs we have to deal with the two cases $\beta(o_t)$ and $\beta(\frac{\partial}{\partial t} o_t)$. We start with abstracting a concrete value $cVal = o_t(t)$ to an abstract value $qVal = o_s(s)$. It is not possible that a physical system will provide a constant real value over time. Due to sensor resolution, noise, and drifts constant values will follow some distribution. In order to prevent that small oscillations of *cVals* around landmark values cause flipping between intervals, we have to blur the sharp separation due to landmark values. Therefore, we extend landmarks to intervals themselves, see Figure 7, and call them *hysteresis intervals.* The term *hysteresis* refers to the fact that the abstraction function is dependent on the state, i.e. the last mapped value. According to the last *qVal*, denoted $qVal_{n-1}$, we can decide which new *qVal* is the right abstraction of the current concrete value. Hence, the type of the value abstraction function is: $\mathbb{R} \times QS \rightarrow QS$.

For example a quantity space $\{zero,\ plus,\ max\}$ can be mapped to the concrete domain $CD =_{df} [0, 3.14] \pm 0.01$ where 0.01 is the tolerance around the landmark values. The corresponding overlapping intervals are: $\{[0, 0.02], [0.01, 3.13], [3.12, 3.14]\}$. Note, the special treatment of boundary values (0, 3.14) as they cannot be expanded symmetrically: an overlap in the size of the tolerance (0.01) with their neighboring interval is generated (by setting their interval size to twice the tolerance value, here 0.02). Consequently for a quantity $q \in L_U$ with a concrete domain CD_q, we can map observed *cVals* to abstract *qVals* by the following function:

$$qVal_n = \beta(cVal, qVal_{n-1}) = \begin{cases} qVal_{n-1} & \text{if } cVal \in c_i; \\ quality(q, c_j) & \text{if } \exists c_j : cVal \in c_j; \\ undef & \text{if } \neg\exists c_j : cVal \in c_j, \end{cases}$$

where $c_i = interval(q, qVal_{n-1})$ and $c_j \in CD_q \backslash \{c_i\}$.

We compute the slope of the measured quantities with a regression line of the last n samples, see Figure 8. The length of the delay line determines the degree of noise suppression. Depending on a defined boundary slope bs we abstract a gradient $\frac{\partial}{\partial t}$ to a qualitative δ. In order to improve the abstraction we use methods of curve sketching. Therefore we compute the first three derivations *dt, ddt, dddt* and map accordingly:

$$\delta_n = \beta(\frac{\partial}{\partial t}o_t, \delta_{n-1}) = \begin{cases} \mathit{min} & \text{if } dt < -bs; \\ \mathit{plus} & \text{if } dt > bs; \\ \mathit{zero} & \text{if } \neg(-bs < ddt < bs \wedge -bs < dddt < bs); \\ \delta_{n-1} & \text{otherwise.} \end{cases}$$

If *ddt* is the first non-zero derivation we can conclude the occurrence of an extremum. If *dddt* is the first non-zero derivation we have encountered a saddle point. By considering higher order derivations we get a better assurance that the first derivation is zero not only because of the choice of the boundary slope *bs*. In the case that the magnitude of all derivations is below the boundary slope we cannot conclude anything. For such points we assume the δ of the previous point (δ_{n-1}).

The introduction of tolerances during abstraction is necessary because it is impossible to detect infinitesimal small changes. The tolerances have to be reflected by the system requirements, e.g. a defined boundary slope must be accurate enough to detect the smallest relevant slopes of system quantities. We consider the tolerances and the choice of the time step *deltaT* as part of the system specification and hence *qrioconf* is preserved. It can be seen analogous to the choice of the *quiescence* time interval in the *ioco* testing theory.

5.2 Test Case Execution

In behavior traces of a QR TS inputs and outputs are coupled. In order to stay within the specified domain of the QR TS inputs cannot be selected independently of outputs between state transitions. Otherwise the probability increases that an input trajectory leads out of the specification and to inconclusive verdicts. Hence when we leave a state because of an inconsistency with one of the inputs or outputs we filter the set of successor states due to states that are consistent again. By removing states that are output-consistent but not consistent with the inputs the filtering also resolves input conflicts. Because the Complete Test Graph (CTG) contains no controllability conflicts we can extract only one test case from it which is the CTG itself. In order to test different system behaviors one has to write according test purposes and simulate the system with different initial scenarios.

Due to the nondeterministic behavior of the implementation obtained test cases are transition systems with possible loops. The implementation can decide which branch in the test case is taken next and how many iterations a loop is run through before taking a transition which exits the loop. A test case has two types of final states: *accept* and *inconclusive* states. A third type, which is not represented explicitly, is the *fail* state. It can be reached from any state that is not final by completing the inputs with unexpected outputs. In an accept state the behavior of interest has been observed and from an inconclusive state the test purpose can never be satisfied.

Algorithm 1 describes the abstract test case execution. At first we initialize the

Algorithm 1 execute()

```
 1: $\forall$ q $\in$ I : initialize(q.cVal)
 2: s := initialState
 3: **while** s $\in$ S $\wedge$ s $\neq$ inconclusive $\wedge$ s $\neq$ accept **do**
 4:     **for all** q $\in$ I **do**
 5:         qVal, $\delta$ := v(s, q), $\delta$(s, q)
 6:         $c_i$ := interval(q, qVal)
 7:         **if** $\delta$ = zero **then**
 8:             deltaY := 0.0
 9:         **else if** $\delta$ = plus **then**
10:             deltaY := $c_i$.max - q.cVal
11:         **else**
12:             deltaY := $c_i$.min - q.cVal
13:         **end if**
14:         q.deltaY := deltaY $\cdot$ deltaT / TI
15:     **end for**
16:     **repeat**
17:         **for all** q $\in$ I **do**
18:             q.cVal := q.cVal + q.deltaY
19:             send(q.cVal)
20:         **end for**
21:         **for all** q $\in$ O **do**
22:             q.cVal := receive(q)
23:         **end for**
24:     **until** getSuccessor(s) $\notin$ S $\wedge$ inputInvariant(s) $\wedge$ outputInvariant(s)
25:     s := getSuccessor(s)
26: **end while**
27: **if** s = accept **then**
28:     verdict := pass
29: **else if** s = inconclusive **then**
30:     verdict := inconclusive
31: **else**
32:     verdict := fail
33: **end if**
```

inputs for the implementation. To each concrete input we assign the mean value of the interval corresponding to the abstract initial value. Starting from the initial state we branch through the states of the test case until we reach a final state. For each visited state we first map the abstract inputs, i.e. (QS, δ) pairs, to concrete time-dependent functions. This is realized in the *for loop* from line 4 to 15.

The *repeat-until loop* from line 16 to line 24 sends inputs to the implementation and observes outputs from the implementation. We stay in the current state as long as there is no valid successor and the applied inputs and outputs are consistent with the current state. The boolean functions *inputInvariant(s)* and *outputInvariant(s)* check the consistency between abstract and concrete trajectories. We are looking for a path through the test case by taking transitions as soon as they become possible. The function *getSuccessor(s)* checks the set of outgoing transitions for valid successors. It returns states that are consistent with the current inputs and outputs with the preference of states that are not inconclusive. This enables us to find longer traces that may hit an accept state. Depending on the reached final state the test case execution yields the verdicts: (1) *pass* for an accept state, (2) *fail* if execution leads to a state that is not reachable from the current state, (3) *inconclusive* for an inconclusive state. If a test case has cycles and an execution sequence does not reach a final state after some time passed or an upper bound of states visited the tester can end the execution with an inconclusive verdict.

6 Experimental Evaluation

We model our example in Simulink which we consider as our implementation under test, see Figure 9. In the implementation we define the height of the upper outlet

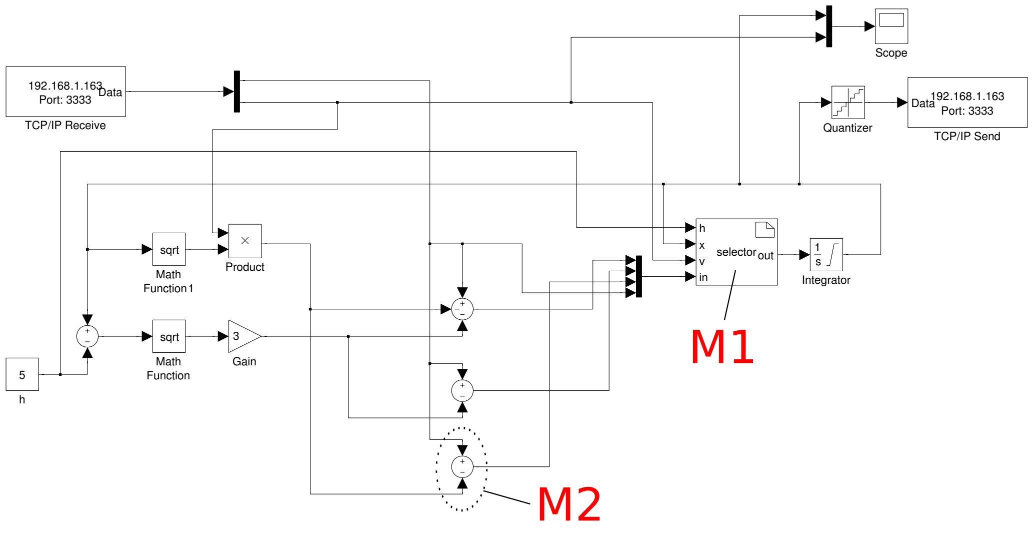


Fig. 9. Simulink Model of the Water Tank.

with the constant $h = 5m$. The *Gain* block represents the outflow friction of the upper outlet and is set to 3. Depending on the invariants the *selector* block switches between the partially defined differential equations accordingly. Further the model contains a scope to visualize the simulation result. We use TCP/IP blocks to communicate with our external test application. This provides a generic interface for writing different kinds of test adapters.

In order to evaluate the fault detection capability of our approach we generate two mutants by introducing mutation *M1* and *M2* in the Simulink model, see Figure 9. Let us assume for mutation *M1* that a developing engineer has swapped the invariants of modes *S0* and *S1* of the partially defined differential equations in (1). In mutation *M2* the plus and the minus sign in the difference node are exchanged.

We define a test scenario with the following initial values: tank level is set to *zero*, the inflow rate changes sinusoidal starting from *zero*, and the valve position is steady at *plus*. We map the input quantities *inflow* and *valve* with the qualitative domain $\{zero,\ plus,\ max\}$ to the concrete domains $[0, 5] \pm 0.01$ and $[0, 3] \pm 0.01$ respectively. The output quantity *level* with $\{zero,\ low,\ set,\ high,\ max\}$ corresponds to the real valued intervals $[0, 5, 10] \pm 0.1$.

The following test purpose expresses that the inflow rate has to reach its maximum twice. Therefor we define two properties: $c =_{df} inflow = max$ and $d =_{df}$

inflow $\neq$ *max*. The regular expression $(d^+c^+)\{2\}$ over the property symbols specifies the test purpose which accepts sequences of states containing two maxima of the inflow rate. For calculating the slope of the input we define a state interval of one second. Further we define a time step of one millisecond.

In a second scenario we set the initial tank level to a height of $8m$ which is located in the qualitative interval *high*. In addition we set the inflow rate to the steady abstract value *plus* and change the valve position sinusoidal starting from *zero*. The concrete interval for the valve position is $[0, 5] \pm 0.01$. As test purpose we specify that the valve position has to reach its maximum twice. The test purpose reads similar to the one of the first test case. Figure 10 shows the input and output trajectories of the executed test cases *TC1* and *TC2* on the implementation *I*, the mutant *M1*, and the mutant *M2*. For our example both generated test cases were able to distinguish the correct implementation from a mutant.

7 Related Work

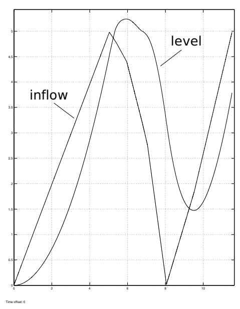
In addition to hybrid automata, other existing modeling approaches have been extended to include continuous behavior: for example, hybrid action systems [20] are an extension to classical action systems [1]. Another example is *hioco* [22], an extension of Tretmans' *ioco* testing theory for labeled transition systems. In addition to the traditional definition of *ioco* the set of trajectories in the implementation must be included in the set of allowed trajectories in the specification. An abstract test case generation algorithm is presented which is not directly implementable.

Reactis [19] is a common test generation tool for Simulink models. It produces test cases regarding certain coverage criteria on the Simulink model. Test cases can be used to ensure conformance of a source code implementation or to validate the behavior of the model itself.

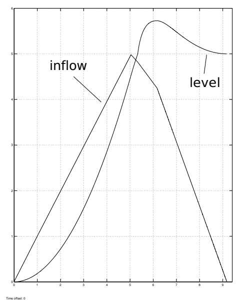
The work in [8] deals with randomized test case generation for hybrid systems. Based on the notation of hybrid automata the approach refers to states as (x, q) tuples where $x \in \mathbb{R}^n$ is a valuation of the continuous variables and q is a set of discrete variables which index the system mode. The idea is to explore the state space by building Rapidly Exploring Random Trees (RRTs) [17]. The RRT algorithm has been used in robotics for path planning by computing control signals for trajectories in high dimensional spaces. For testing, the RRT algorithm is used to find counter examples, i.e. input sequences that drive the system into states that are not in a defined specification set. The authors in [18] extend the random exploration technique with coverage information. Less explored regions are preferred in the exploration process. The usability and performance of RRTs depends on finding appropriate metrics in a system's state space.

The selection of test cases for hybrid systems is addressed in [14]. The idea is that a test case with certain parameters has many test cases with slight changed parameters in its neighborhood which all show the same qualitative behavior. The work deals with the generation of test cases based on coverage metrics.

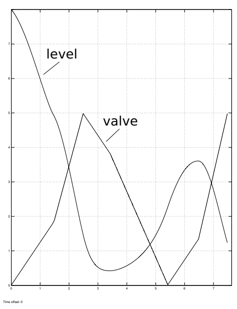
The work in [12] deals with the modeling of hybrid systems using interval arith-



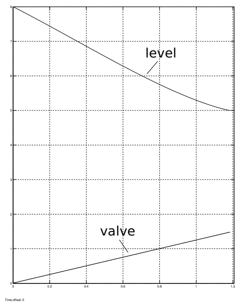
(a) The Implementation passes TC1.



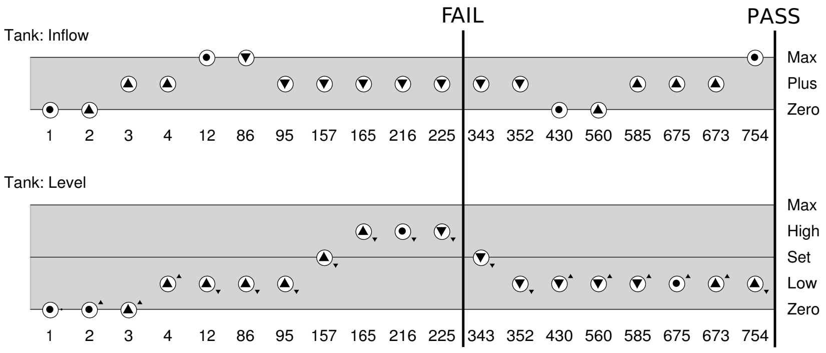
(b) The Mutant *M1* fails TC1.



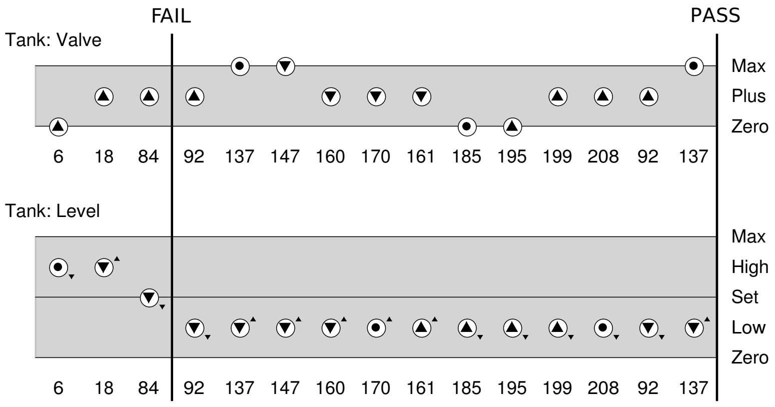
(c) The Implementation passes TC2.



(d) The Mutant *M2* fails TC2.



(e) Value History of TC1 executed on I and M1.



(f) Value History of TC2 executed on I and M2.

Fig. 10. Execution of TC1 and TC2 on I, M1, and M2.

metic constraints. Interval arithmetic provides a means to deal with rounding errors where the real value of a variable is located somewhere within an interval. Systems are specified in the *CLP(F)* language which can state constraints over real numbers and analytic constraints over differentiable functions. The underlying constraint solver calculates, similar to *QSIM* [15], an over approximation of the solution of a system of ODEs. Due to over approximation the solver returns a set of solution intervals. If there is a correct solution to a query it will be in one of the returned intervals. On the other hand not all solutions in the returned set may contain actual solutions. The CLP(F) system solves analytic constraints by using power series to approximate analytic functions. It is also possible to handle non-linear ODEs. A drawback of the approach is the high ressource consumption with increasing modeling time. This is because of an increase of constraints over Taylor coefficients in the according power series.

8 Conclusions

We presented an approach to model and simulate the behavior of the continuous variables of hybrid systems with QR methodologies. We used abstract, qualitative models as test models to check whether a given concrete implementation is a valid refinement under the *qrioconf* relation. This provides a means, e.g., to validate the correctness of environmental simulations. We use *Matlab Simulink* as implementation language and generate offline test cases due to defined test purposes or according to coverage criteria. We demonstrated our approach on a small example and were able to detect faults introduced in the implementation of the example. To our knowledge, our work is the first in applying QR-techniques to the testing of hybrid systems.

The advantage of our approach is threefold: (1) the abstraction is directly supported by means of qualitative modeling. However, (2) we still keep the ability to solve the differential (flow) equations. (3) Items (1–2) are supported by off-the-shelf tools. The complete qualitative simulation of even small systems can lead to quite big state spaces where Garp3 reaches its limits. In future work we will evaluate the tool *QSIM* [15] for computing the qualitative behavior of hybrid systems. We will work on the combination of timed events with qualitative behavior to describe systems more accurately. Another promising approach for future work is the evaluation of semi-qualitative simulation for online testing. In semi-qualitative simulation concrete numerical information, which is available during online testing, is used to exclude abstract behavior which does not apply to the current context.

References

[1] Back, R. J. R. and F. Kurki-Suonio, *Distributed cooperation with action systems*, ACM Trans. Program. Lang. Syst. **10** (1988), pp. 513–554.

[2] Bouwer, A., J. Liem and B. Bredeweg, “User Manual for Single-User Version of QR Workbench,” Naturnet-Redime, STREP project co-funded by the European Commission within the Sixth Framework Programme (2002-2006) (2005), project no. 004074. Project deliverable D4.2.1.

[3] Brandl, H., G. Fraser and F. Wotawa, *Coverage-based testing using qualitative reasoning models*, in: *Proceedings of the Twentieth International Conference on Software Engineering & Knowledge Engineering (SEKE'2008)* (2008), pp. 393–398.

[4] Brandl, H., G. Fraser and F. Wotawa, *QR-model based testing*, in: *AST'08: Proceedings of the 3rd international workshop on automation of software test* (2008), pp. 20–17.

[5] Brandl, H., G. Fraser and F. Wotawa, *A report on QR-based testing*, in: *22nd International Workshop on Qualitative Reasoning*, 2008, pp. 1–9.

[6] Bredeweg, B., A. Bouwer, J. Jellema, D. Bertels, F. F. Linnebank and J. Liem, *Garp3 - a new workbench for qualitative reasoning and modelling*, in: *Proceedings of 20th International Workshop on Qualitative Reasoning (QR-06)*, Hannover, New Hampshire, USA, 2006, pp. 21–28.

[7] Bredeweg, B., J. Liem, A. Bouwer and P. Salles, "Curriculum for learning about QR modelling," Naturnet-Redime, STREP project co-funded by the European Commission within the Sixth Framework Programme (2002-2006) (2005), project no. 004074. Project deliverable D6.9.1.

[8] Esposito, J., *Randomized test case generation for hybrid systems: metric selection*, System Theory, 2004. Proceedings of the Thirty-Sixth Southeastern Symposium on System Theory (2004), pp. 236–240.

[9] Forbus, K. D., *Qualitative process theory*, Artif. Intell. **24** (1984), pp. 85–168.

[10] Henzinger, T. A., *The theory of hybrid automata*, Technical Report UCB/ERL M96/28, EECS Department, University of California, Berkeley (1996).
URL http://www.eecs.berkeley.edu/Pubs/TechRpts/1996/3019.html

[11] Henzinger, T. A., P. Ho and H. Wong-Toi, *Hytech: A model checker for hybrid systems*, Software Tools for Technology Transfer **1** (1997), pp. 460–463.

[12] Hickey, T. J. and D. K. Wittenberg, *Rigorous modeling of hybrid systems using interval arithmetic constraints*, in: R. Alur and G. J. Pappas, editors, *HSCC*, Lecture Notes in Computer Science **2993** (2004), pp. 402–416.

[13] Jard, C. and T. Jeron, *TGV: theory, principles and algorithms*, International Journal on Software Tools for Technology Transfer (STTT) **7** (2004), pp. 297–315.

[14] Julius, A. A., G. E. Fainekos, M. Anand, I. Lee and G. J. Pappas, *Robust test generation and coverage for hybrid systems*, in: A. Bemporad, A. Bicchi and G. C. Buttazzo, editors, *HSCC*, LNCS **4416** (2007), pp. 329–342.

[15] Kuipers, B., *Qualitative simulation*, Artificial Intelligence **26** (1986), pp. 289–338, reprinted in Qualitative Reasoning about Physical Systems, ed. Daniel Weld and J. De Kleer, Morgan Kaufmann, 1990, 236-260.
URL citeseer.ist.psu.edu/kuipers01qualitative.html

[16] Kuipers, B., "Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge," MIT Press, 1994.

[17] LaValle, S. M. and J. J. K. Jr., *Randomized kinodynamic planning*, in: *International Conference on Robotics and Automation*, 1999, pp. 473–479.

[18] Nahhal, T. and T. Dang, *Test coverage for continuous and hybrid systems*, in: W. Damm and H. Hermanns, editors, *Computer Aided Verification*, LNCS **4590** (2007), pp. 449–462.

[19] Reactive Systems, I., *Model-based testing and validation with reactis*, Technical report, Reactive Systems, Inc. (2003).

[20] Rönkkö, M., A. P. Ravn and K. Sere, *Hybrid action systems*, Theor. Comput. Sci. **290** (2003), pp. 937–973.

[21] Tretmans, J., *Conformance testing with labelled transition systems: Implementation relations and test generation*, Computer Networks and ISDN Systems **29** (1996), pp. 49–79.

[22] van Osch, M., *Hybrid input-output conformance and test generation*, in: K. Havelund, M. Núñez, G. Rosu and B. Wolff, editors, *FATES/RV*, LNCS **4262** (2006), pp. 70–84.