A Report on QR-Based Testing

Harald Brandl and **Gordon Fraser** and **Franz Wotawa** *
Institute for Software Technology
Graz University of Technology
8010 Graz, Austria
{brandl,fraser,wotawa}@ist.tugraz.at

Abstract

As reactive and embedded systems continuously interact with their environment, it is important to test as many as possible interactions. The use of qualitative models of the environment and hardware has the potential to provide test cases that might not be considered with traditional testing methods. We present an approach that derives abstract test cases from such models using qualitative reasoning, which is a well known artificial intelligence technique to represent and reason about physical behavior. For this purpose we introduce the underlying concepts of qualitative reasoning, show the test case generation process, and provide the results of a case study.

Introduction

This paper extends the ideas proposed by Franz Wotawa (Wotawa 2007) and presents first results for test case generation from QR-models.

The growing demand for smarter products with increased functionality leads to a steady increase of the complexity of systems and software. As an example, current cars typically have more than 40 control units with many sensors and actuators on board; this number will further increase. Verification and validation are therefore very important, and lead to many issues related to the automation of test case generation and execution. In the context of reactive and embedded systems, the difficulty of automated testing is further increased because there is a high degree of interaction with the surrounding environment. Because of this, correctness cannot be guaranteed solely by testing the implemented functionality, but also requires testing of the reactions to general stimuli originating from the environment. Although the behavior upon certain wrong inputs is sometimes explicitly specified in the case of reactive and embedded systems, it is unlikely that all important cases are considered at design time. Consequently, systems might fail in some cases when interacting with the physical world.

In order to overcome this problem it is necessary to consider the behavior of the physical world and its interaction with the system. Qualitative reasoning (QR), which originates from the field of Artificial Intelligence (AI), provides us with means for deriving all possible behaviors. QR was originally developed to model commonsense reasoning within the physical world, e.g., to allow for generating answers to questions like "If I throw a stone upwards, what will happen?" A QR simulator will not only provide us with one answer like the stone will move up to a certain point and fall down afterwards, but with all possible answers, including that the stone might go up and up if thrown fast enough.

In general, model based testing techniques use some kind of formal specification describing a system's expected behavior in order to determine whether an implemented system is correct. The specification is used to derive test cases and also serves as oracle, which decides if an executed test case detected an error. For more details on test based on formal specifications we refer to existing surveys on the topic; e.g., (Hierons et al. 2008). The semantic model of most of these formal languages can be interpreted as a kind of transition system representing a system's time variant behavior. Unlike traditional formal specifications, QR techniques allow to describe a system in a declarative manner by a set of qualitative differential equations (QDEs). From this compact representation a QR engine derives a transition system which represents all possible behaviors that can evolve over time starting from an initial scenario.

QR modeling tools like Garp3 (Bredeweg et al. 2006) are well suited when specifying physical systems on an abstract behavioral level. Especially when there is already a mathematical description of the system available (set of differential equations), the mapping can be done in a straightforward way. Therefore, a focus of the presented approach lies on control systems. The continuous domain of control systems (or integer domain for digital controllers) can be transferred via qualitative abstraction and simulation to a transition system (TS).

In this paper, we briefly introduce the basic concepts of QR. As a running example we use a system that sends GPS data to a base station via GSM on a regular basis (Figure 1); such

*Authors are listed in alphabetical order

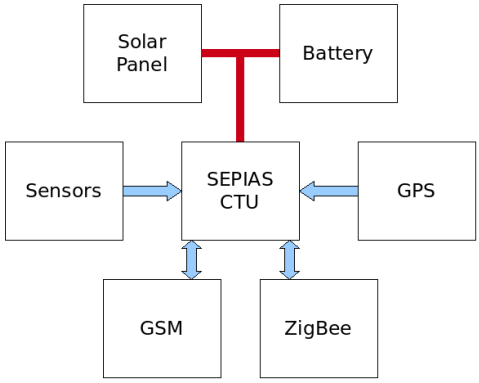


Figure 1: Block Diagram of a container tracking unit.

Figure 2: The implemented prototype of the container tracking unit.

systems can for example be found in the logistics domain. To increase mobility the system has a solar panel which charges a battery. The system monitors the state of the battery and the solar panel and acquires other environmental parameters like the temperature via its sensors. Once new data has been collected, the system decides upon the next actions or if it should go to a power saving mode. A near field communication module allows data exchange with other devices within range.

The running example will be used to introduce the basic concepts of generating test cases from QR models, and as a case study to evaluate the feasibility of the presented approach. The possible behaviors inferable from QR models are represented as labeled transition systems (LTS). Manually specified test purposes are used to derive test cases from the LTS. For example, from the requirement that the system should not run out of battery we create a test purpose that expresses that we do not allow a battery to completely discharge. Traditional LTS conformance testing techniques are adapted to be applicable to QR based models, allowing derivation of test cases from the synchronous product of the system LTS and the test purpose. As the obtained test cases are abstract they have to be refined in order to be executable on an implementation under test (IUT). This is realized by applying abstraction/refinement relations to the information exchanged between a test case and an IUT.

The main contributions of this paper are as follows:

- Models that are close to hardware are used for test case generation.
- The underlying models can include the system hardware, the part of the software which interacts with the environment, and the environment itself.
- The use of environmental models ensures that all possible behaviors are covered, while during manual test case generation some details may be missed.
- We adapt the input-output conformance testing theory (ioco)(Jard and Jeron 2004) to transition systems derived from QR models.

The remainder of the paper is organized as follows. First, we introduce the basic concepts of QR and the modeling of our running example. Then we present our approach of test case generation from QR models in detail and present the first results of our case study. Finally, we discuss related research and conclude the paper.

QR Modeling with Garp3

Garp3 provides every means to build and inspect QR models. A detailed description of the functions can be found in the user manual (Bouwer, Liem, and Bredeweg 2005), and there is an elaborate user guide (Bredeweg et al. 2005) for building QR models.

A Garp3 model consists of a set of model fragments, which are the basic units that describe behavior. Two main types of fragments can be distinguished: static and process model fragments. A third fragment type called agent can be used to model exogenous influences on the system. Static fragments represent behavior that is invariant with regard to time, such as proportional relations between quantities, like, for example, “the amount of water in a vessel is proportional to the water level”. A dynamic fragment introduces changes via influences between quantities, for example “a positive flow rate into a vessel will increase the amount of liquid and hence the liquid level over time”.

Usually, systems consist of several model fragments that are activated when certain boundary conditions are met. Garp3 uses colors to represent information like, e.g., the identification of conditional elements in a model. Garp3 adds the consequences of a model fragment as new facts to the knowledge base, unless they contradict existing facts. Model fragments enable the designer to partition the system domain into qualitative equivalence classes that capture certain behavior. During simulation the set of fragments collected in a library changes between being active and inactive as the system evolves over time.

Within a model fragment the main modeling primitives are entities, quantities, proportionalities, and a set of ordinal re-

lations. In dynamic model fragments there are additional influences. Entities are the components of the system that have certain properties expressed through associated quantities. For example, the entity battery has the quantities "voltage", "current", and "charge".

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 of the effected quantity.

Ordinal relations called inequalities provide means to constrain possible behavior. Influences cause dynamic changes of the system and provide means for integration. For instance, I+ (Q1,Q2) means that the value of Q2 determines the change of direction of Q1. If Q1 is positive Q2 increases, if Q1 is zero Q2 does not change, and if Q2 is negative Q1 decreases. The graphical notation used in Garp3 states relations with arrows between quantities.

The initial state of the system is captured with scenarios. This initial state and the model fragments serve as input to the simulation engine. Simulation is used to generate the behavior of a QR system. The simulation engine derives everything that does not contradict the boundary conditions, i.e, inequalities between quantities. QR models can only describe systems with continuously changing quantities, as stated by the continuity law (Forbus 1984). In general, model creation is an iterative process: One has to find the right level of abstraction and check if the simulation output satisfies the requirements. If there are discrepancies, the model has to be adapted and simulated again.

We use Garp3 models to describe hybrid systems, i.e., systems that combine discrete and continuous behavior, in a qualitative fashion. Control modes of hybrid systems can be captured with sets of model fragments that define a certain control mode when applied. Whenever a model fragment is activated or deactivated during simulation the system switches to a different mode. Garp3 models represent a restricted form of hybrid systems, i.e., they require that changes of variables between control modes follow the continuity law.

Example Model

Our example of a container tracking unit (see Figure 1) consists of a solar panel, a battery, and the controller board drawing some load current. In addition, there are external quantities such as temperature and light conditions.

Figure 3 shows the model fragment for the solar panel. It is a static fragment that defines the main proportionalities (P-arrows) and correspondences (arrows with V for value-, and

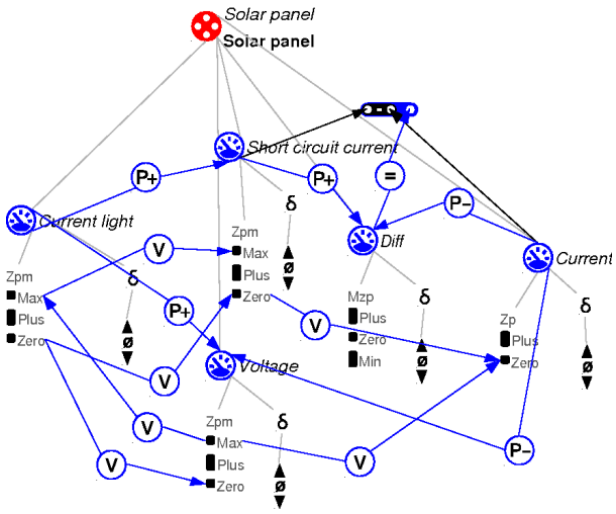


Figure 3: Solar Panel.

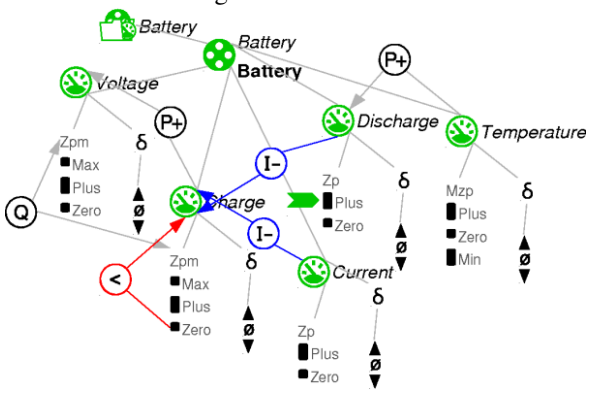


Figure 4: Battery.

Q for quantity space-correspondences) between its quantities. A second fragment of the solar panel not shown here is a conditional one that becomes active when the current drawn from the panel is greater than the generated current. If this condition is met, the fragment introduces a negative feedback (P-) from the calculated *Diff* quantity to the drawing current. This limits the maximum current that the panel can deliver due to the current light conditions. A further property the model reflects is that the voltage decreases with increasing load current (P-). In the case of a short circuit the voltage changes to zero while the maximum load current is provided.

Certain corresponding boundary values are stated with directed or undirected *value correspondences*. A directed value correspondence (V) from *Current light = zero* to *Voltage = zero* indicates that the voltage is zero if the current light is zero. This restricts the state space to allowed behavior. The model fragment has two interface quantities *Current light* and *Current* that can be referenced by other model fragments or exogenous quantities.

The model fragment in Figure 4 depicts the discharging behavior of a battery. The fragment is activated if the charge level of the battery is greater than zero. Furthermore, a discharge rate greater than zero is defined (the arrow on the plus value) which is directly proportional to the temperature. The discharge rate and the load current decrease the battery's charge level as stated by the negative influences. The correspondence (Q arrow) and proportionality (P arrow) between charge and voltage causes *Voltage* to take on the same value as *Charge*.

Figure 5 shows the compound model fragment of our system comprising the solar panel, the battery, and the container tracking unit represented as current drawing load. The fragment is active when the voltage of the solar panel is greater than the battery's voltage, causing the flow of a charge current from the panel to the battery. The arrow on the solar panel's current quantity requires the charge current to be greater than zero. Note that the system software has to capture the behavior of the physical world as far as the specification is concerned.

Test Case Generation

When testing reactive systems it is important that the system model includes the behavior of the environment. To see why this is so, consider an analogy from control theory: The combination of the control process with the controller forms the whole system. These two components influence each other, and many important conclusions like stability of the closed loop system cannot be drawn when looking at them separately. This also applies to testing, where both the system state and the environmental conditions have to be considered. This section shows a method for deriving test cases from QR models and a technique to minimize them.

Deriving Test Models from Garp3 Models

We classify systems based on how they interact with the environment. The first type comprises conventional control systems interacting in both directions with sensors that perceive the environment and actuators that change it. The second type are systems that only perceive their environment without changing it. The gathered information is used to adapt the internal behavior, which is not directly observable from the outside world. In this context the term environment comprises anything but the system software. As software is not well suited for modeling with Garp3 we draw this line between software and the physical world.

Garp3 is suitable to model systems that can be completely specified by their observable external behavior. If there are additional requirements on the internal system state (e.g., "the CPU has to operate in low power mode"), it is necessary to monitor the execution of the software. We propose to annotate the system software with assertions that can check whether certain conditions are fulfilled. A test case can only lead to a pass verdict if there was no assertion violation.

During test case execution the system software may influence the environment and hence any trace in the environmental model that serves as test case. Therefore, we define interaction points as a subset of the quantities in the Garp3 model. The quantities Q are partitioned into three sets, viewed from the system side: input, output, and hidden. Input quantities are refined to concrete sensor inputs for the system, and system outputs are abstracted to output quantities influencing the environment. Consequently, the set of interaction points is the set of all but the hidden quantities.

The simulation output of a QR model is a state space representation of all possible behaviors that may evolve over time, starting from an initial scenario. We define this output as a QR transition system (QR TS) $M = (S, T, s_0, Q, qs, QS, v, \delta)$, where S is the set of states, T is the transition relation $T \subseteq S \times S$, and $s_0 \in S$ is the initial state.

Every quantity in the set of quantities Q of the simulation output has an associated quantity space. QS denotes the domain of quantity spaces, and the function $qs : Q \rightarrow QS$ maps each quantity to its associated quantity space. Each state in the state space binds all quantities to a distinct value and delta. The value v for quantities in Q and states in S is defined as $v : S \times Q \rightarrow qs(Q)$, and the delta δ is defined as $\delta : S \times Q \rightarrow \{min, zero, plus\}$. The $delta$ of a value stands for its direction of change over time, $\delta \sim \frac{\partial value}{\partial t}$.

If there is a transition between two states, then either a value or a delta for some quantity changes. The continuity rule proposed by De Kleer and Brown (Kleer and Brown 1984) states that a qualitative value as a function of time cannot jump over values in its domain but has to change continuously. As all state transitions computed by the QR engine adhere to this rule, it follows that on every transition from a state to a successor state at least one quantity changes its value or direction; otherwise, the two states are qualitatively equivalent.

Test Purposes for QR Models

In order to generate test cases from QR TS derived from QR models, we adapt techniques from LTS testing. An LTS extends a transition system with a set of labels P and a function L mapping transitions to labels: $L : T \rightarrow 2^P$, where P is a set of atomic propositions. Garp3 is able to enumerate the complete state space because of qualitative abstraction as is possible for LTS, but in contrast to LTS, QR TS comprise states with simultaneously changing inputs and outputs. Consequently, the behavior of interest is not only specified via the occurrence of some state sequences. In addition we are interested in the relations between quantities.

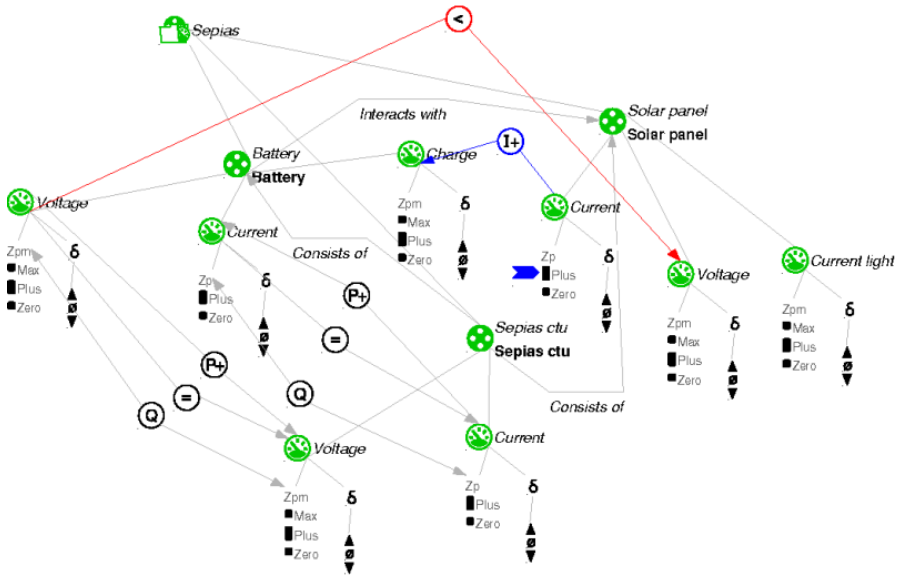


Figure 5: Container Tracking Unit.

Therefore, we define properties on the quantities, and use these properties to define test purposes. The idea is to formally specify both test purpose and test model as LTS, and then to derive test cases by computing the synchronous product of the specification and a test purpose, as initially proposed by Jard and Jeron(Jard and Jeron 2004). A test purpose describes some aspect of the specification that is of interest for testing. It is defined as a regular expression over symbols that represent properties of model quantities. A property set represents the conjunction of several properties. To remove redundancy one can define global properties, which can optionally be added to property sets. The set of all such property sets represents the set P of possible transition labels.

As an example, consider the property of a battery that its charge is greater than zero: $battery : charge > zero$, where $battery$ is the entity name and $charge$ the name of the quantity. This property denotes a value relation where *zero* is a value in the quantity's space. A second type of relation considers the δ of quantities using the operator dx. As an example, the property $battery : charge\ dx = min$ states that the battery charge decreases.

The regular expression that completely specifies the test purpose consists of the defined symbols and operators allowed in regular expressions. The equivalent deterministic automaton accepts all symbol sequences that lead to an accept state. This automaton represents an LTS with labels corresponding to properties of the test purpose. Suppose we are interested in the the cyclic occurrence of a property a, e.g., for three times and thereafter a path leading to property b. The regular expression $([\hat{}a] * a)\{3\}. * b$ describes such a test purpose. Although theoretically possible, our current implementation does not make use of reject states, which are used in LTS testing to consider only parts of the state space. As QR models have discrete, commonly only small value domains the models' state spaces usually are not very big, and reject states might not be necessary at all. Avoiding reject states in test purposes has the effect that test case execution cannot be inconclusive.

Once a test purpose is defined, we use the properties defined in this test purpose as symbolic labels for the transitions of our transition system. We annotate all transitions in the QR TS with labels, where the properties represented by the labels of a transition have to be satisfied in the target state of the transition. This augmentation of the QR TS is necessary for computing the synchronous product with a test purpose. Algorithm 1 describes the annotation process. The LTS is created simply by adding labels to the existing QR TS. The conversion from QR TS to LTS is done by iterating through the set of states. For each outgoing transition all symbols of a given test purpose are considered. Each symbol represents a set of conjunctively combined properties. If all properties in the set are satisfied, the current symbol is added as transition label. In addition, a property set can have a $global$ flag, which requires that all global properties have to be satisfied in addition to the properties of the set. Consequently, if the global properties are not satisfied and the $global$ flag is set, the algorithm rejects the currently considered symbol for the current transition. If the global properties are satisfied, the reserved symbol y is added as a transition label. The algorithm always terminates because the number of states in a QR transition system is finite (The domains of the QR variables are finite).

As an example, Figure 6 shows a QR TS consisting of three QR states. There are three symbols a, b, and c denoting three different properties on the state variables. The QR TS is la-

Algorithm 1 QRSTATEGRAPH2LTS()

```
1: for all states in QR TS do
2:     for all outgoing edges do
3:         for all symbols of the test purpose do
4:             get property set PS corresponding to
               current symbol
5:             get state S pointed to by current outgoing
               edge
6:             if S satisfies global_properties then
7:                 add label 'y' to current edge
8:             else if PS has global flag set to true
               then
9:                 continue with next symbol
10:            end if
11:            if S satisfies PS then
12:                add current symbol to current edge
13:            end if
14:        end for
15:    end for
16: end for
```

beled using these properties. As state $s1$ satisfies property a but not properties b and c, the transition from $s0$ to $s1$ is only labeled with a. State $s2$ satisfies all properties, therefore the transition from $s0$ to $s2$ is labeled with all symbols. If there is more than one symbol in a label, this is interpreted as the disjunction of the represented properties. Finally, $s3$ satisfies a and b and the transition from $s0$ to $s3$ is labeled accordingly.

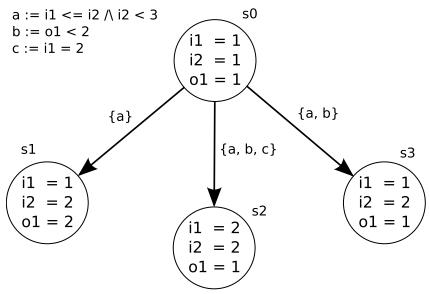


Figure 6: Labeled QR TS.

Test Case Generation with Test Purposes

With the converted LTS and a given test purpose we have two LTS with the same alphabet, and hence the synchronous product (Jard and Jeron 2004) can be computed; this is done with a Depth First Search (DFS) algorithm. Starting from the initial states of both LTS we match common labels on all outgoing edges between the current states both LTS are in. For every match found we get a state tuple by following both edges to states. If this tuple is a new state in the product LTS we add it with an edge. In addition the new state is pushed onto a stack. The algorithm terminates when the stack gets empty. In worst case this happens after all state tuples $Q^{M_1} \times Q^{M_2}$ have been visited. On average this algorithm performs better than the simple approach of considering all state tuples.

The synchronous product is a new LTS, to which Tarjan's algorithm as a framework (Thierry Jeron 2004) for determining the set of states leading to an accepting state is applied. It computes the set of strongly connected components while updating reachability information for the visited states. A state can reach an accepting state if itself or another state in the same SCC can reach an accepting state. A strongly connected component is defined as a subset of graph states, inside which every pair of states can reach each other via transitions to states inside the set. A directed graph with possible cycles partitioned in its SCCs is a directed acyclic graph (DAG) with the set of SCCs as nodes. The computed subgraph is called Complete Test Graph (CTG). For each state of the CTG, quantities can be mapped to actual values with the v and δ functions (see Section), to serve as test data and expected output.

Although input and output information is not contained in labels but in states, it is possible to adapt existing conformance relations to QR models. As an example, in order to determine the conformance of a system to a QR model we adapt the input/output conformance relation (ioco) (Jard and Jeron 2004):

$$i \; ioco \; s \leftrightarrow \forall \sigma \in traces(s) : out(i \; \textit{after} \; \sigma) \subseteq out(s \; \textit{after} \; \sigma)$$

where i is an IUT, s is the specification (the LTS derived from the QR model), $traces(s) = \{\sigma = \langle t_0, t_1, \ldots \rangle \mid t_0 = s_0 \wedge \forall i \geq 0 : t_i \in S \wedge (t_i, t_{i+1}) \in T\}$, and the set $s \; \textit{after} \; \sigma = \{s' \in S \mid \sigma = \langle t_0, ..., t_n \rangle \wedge t_i \in S, \; (t_n, s') \in T\}$. In this definition, $out(s)$ describes the state $s \in S$ with its output quantities $output(Q) \subset Q$, and their values and deltas:

$$out(s) =_{def} \bigcup \{(q, v(s,q), \delta(s,q)) \mid q \in output(Q)\}$$

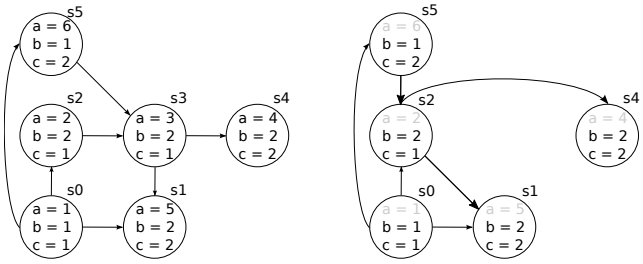
This relation considers all traces of the implementation that are also contained in the specification (QR TS). The values and deltas of all quantities of the implementation after a trace σ have to be a subset of the ones contained in the specification after the same trace. During test case execution when the implementation changes to a state not defined in the specification, meaning that after some trace the conformance relation is violated, we get a fail verdict.

State Space Minimization

As hidden quantities cannot be observed during test case execution they are not relevant in the context of test case generation. When we find two connected states that only differ in their non-relevant quantities we can merge them and update the unconnected edges of the removed state. Two states s_1 and s_2 of a TS $M = (S, T, s_0)$ are equivalent, if the following condition holds, where $rel(Q) \subset Q$ denotes the set of non-hidden quantities:

$$(s_1, s_2) \in T \wedge \forall q \in rel(Q) : \\ v(s_1, q) = v(s_2, q) \wedge \delta(s_1, q) = \delta(s_2, q)$$

Figure 7(a) shows an example state space with three states and three quantities. Assume that quantity a is hidden and thus not relevant. Consequently, states $s2$ and $s3$ are equivalent and can be merged. Figure 7(b) shows the result of this merge, with the updated transitions. Now a problem of nondeterminism arises in state $s2$, as the successor states $s1$ and $s4$ are equivalent but cannot be merged because there is no transition between them.



(a) Original TS. (b) Minimized TS.

Figure 7: State Space Minimization.

If all hidden quantities are constant the minimized CTG remains deterministic. This is because constant quantities cannot discriminate two states. Otherwise the CTG in terms of relevant quantities may become non-deterministic. At the end of the minimization we convert the possibly nondeterministic CTG to its equivalent deterministic CTG using the standard finite automaton technique.

Dealing with Controllability

Test cases have to be controllable. This means that on every node in the CTG where a decision between different inputs for the implementation is possible every branch taken leads to a new test case. As most systems like ours are not controlled by their environment, the implementation can be non-deterministic. Consequently, test cases are not linear sequences but transition systems that can handle alternative outputs. The set of input quantities splits a state's outgoing transitions into partitions where the input quantities all have same values and deltas. The implementation has the possibility to react with the according output quantity assignments in that partition. In the example of Figure 6 the set of input quantities $\{i1, i2\}$ splits the outgoing transitions of state $s0$ into the partitions $\{(s0, s2)\}$ and $\{(s0, s1), (s0, s3)\}$. For the second partition the implementation side decides which branch is taken next. The states of a test case have only outgoing transitions of one of its partitions which ensures controllability.

To achieve controllability we extract test cases from the CTG as follows: For every uncovered transition in the CTG we create a complete path by searching backwards to the start state and forward to an accept state. Then we traverse the path and add all parts of the CTG that are reachable considering the implementation's nondeterministic outputs to the test case TS. This approach returns test cases until all transitions in the CTG are covered.

Test Case Execution

A test case is a QR TS. All information needed for its execution is stored in its states. A QR state comprises a set of quantities with their current values and deltas. This abstract information has to be mapped to concrete quantity values, e.g., $voltage$ with $value = plus$ and $delta = positive$ is mapped to a time variant function like $voltage = start(voltage) + k \cdot t$. When the state is entered the concrete voltage value is updated regularly on some time step Δt as long as all state quantities fulfill the state's conditions. The $start(voltage)$ denotes the starting value of voltage when the state is entered. In this way we proceed with all input quantities and simultaneously observe the conditions of the output quantities. The behavior of the output quantities decides in which state of a branch we are in. If there is no matching state the implementation fails the test case. A state is left when it gets inconsistent with the observed output quantities. The current values of the output quantities have to be transferred to the successor state so that the according functions have initial values to compute further output values. When an accepting state is reached we get a pass verdict. The next section contains an example test case to illustrate this.

Demonstration and Results

For demonstration purposes we simplify the model of our system by replacing the solar panel with an exogenous sine quantity, which emulates a charging current changing with light conditions. With the load also emulated as a sine changing current, our model reduces to the battery model fragment shown in Figure 4. The temperature is defined as constant exogenous quantity. Simulation of the QR model results in a QR TS with 70 states and 228 transitions.

Assume we are interested in the behavior that leads to an empty battery. For this, we define the two property symbols *a* for $battery : charge > zero$ and *b* for $battery : charge = zero \wedge battery : charge\ dx = zero$. The regular expression describing the test purpose is *a*b*. This test purpose is fulfilled by paths containing any number of states where the battery charge is greater than zero (*a*), followed by a state where the battery charge is zero and the battery is not charging (*b*).

Next we define input- and output-quantities for our QR TS. The battery's current drawn by the system is an output. The system acquires the battery's voltage as indicator for the actual charge level as input. These two quantities are relevant for test case generation. From the test case view inputs and outputs of the specification are reversed. In a first step we augment the QR TS with labels and subsequently compute

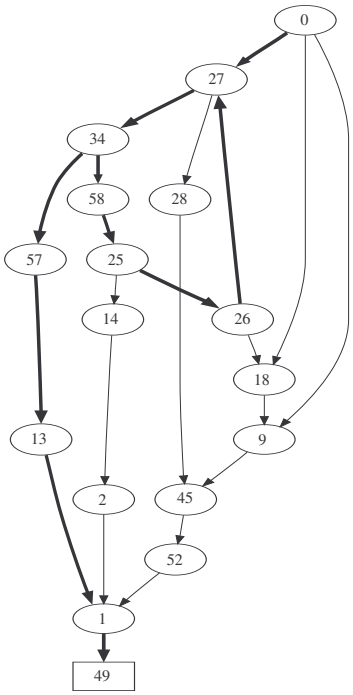


Figure 8: Test Case.

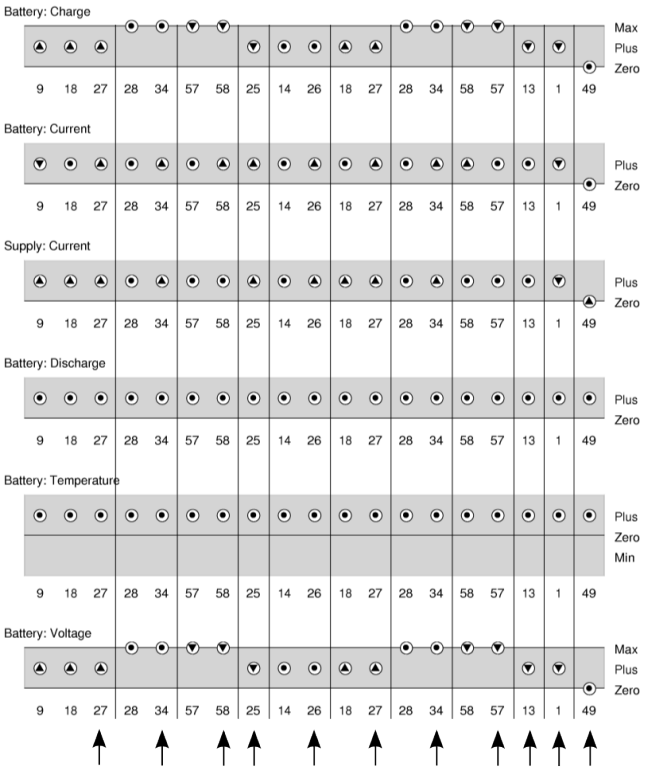


Figure 9: Value History: Test Case leading to an empty Battery.

the synchronous product with the test purpose. After that Tarjan's algorithm for computing reachability information is applied to the product LTS which reduces search space for later test case extraction to 59 states and 207 transitions. We minimize the obtained graph with regard to relevant quantities and ensure determinism. Now we have the CTG ready for test case extraction comprising 26 states and 72 transitions. As the state space is not big the computation time for all this steps is about 2 seconds.

The simplified model results in 16 test cases which cover 71 transitions in the CTG. One transition cannot be covered because when chosen during resolving a controllability conflict it never leads to an accepting state. In total, the 16 test cases cover 31% of the transitions of the QR TS.

Figure 8 shows an example test case for this test purpose. States are annotated with their state IDs as used in the QR TS. The test case has cyclic behavior in states {27, 34, 58, 25, 26}. Figure 9 shows one possible execution sequence through the test case leading to an empty battery. The vertical separation lines denote sets of outgoing transitions starting with the initial state on the left side. The arrows mark which transition has been taken during the execution.

Table 1: Characteristics of example test purposes.

Test Purpose	CTG	# TCs	tr coverage
battery is empty	55	55	82%
battery is full	53	38	65%
	Σ	93	90%

Table 1 depicts some characteristics for two test purposes considering $Battery : Charge$ and $Supply : Current$ as input and $Battery : Current$ as output, applied to the full example model. The CTG column specifies the number of states in the CTG, and $\#TC$ the number of test cases extracted from the CTG. The last column denotes the transition coverage for the test suites measured on the QR TS.

Related research

Tretmans (Tretmans 1996) described test case generation for Labeled Transition Systems (LTS). The paper focused on Input-Output-LTS (IOLTS) and introduced conformance relations for them. The proposed testing theory also deals with states where quiescence is allowed. Jard and Jeron (Jard and Jeron 2004) presented a tool for automatic conformance test generation from formal specifications. They used IOLTS as formal models and defined the *ioco* conformance relation for weak input enabled systems. Test cases are generated using defined test purposes.

Auguston et al. (Auguston, Michael, and Shing 2005) introduced the use of attributed event grammars for generating test-cases from environment models for reactive systems. In the paper the authors use the grammar for representing an event-based model. Possible execution traces of the model

form the test-cases. Insofar the underlying idea for test-case generation as described in this paper is very similar, but can be distinguished with respect to the underlying modeling language. Whereas Auguston et al. are using attributed event grammars, in this paper we are proposing the use of qualitative models for test-case generation.

Conclusions

In this paper we introduced an automated test case generation approach which relies on qualitative models. Qualitative models are well suited for modeling in the embedded systems domain. Especially when such systems strongly interact with their physical envirnoment this approach is a good choice.

Qualitative models represent all possible physical behaviors of systems and their environments, and can be used to find test cases which might not be considered when only using a system's specification. Similar to previous approaches we make use of test purposes in order to generate tests.

The following steps lead from a QR model to a test suite: (1) simulation of the QR model → QR TS, (2) conversion of the TS into a LTS according to a test purpose, (3) product of the system LTS with the test purpose, (4) minimization and ensuring determinism → CTG, and finally (5) extraction of controllable test cases from the CTG.

The first results indicate that useful test cases can be automatically generated from QR models. However, a more in-depth analysis is still required. In general, the approach is well suited when a physical model is available, e.g., in the embedded systems area. We are currently in the process of evaluating the approach on models derived from Matlab Simulink models via qualitative abstraction. First experiments indicate a sound test case execution, and resulting test cases exercise the interactions of a system under test with its environment. Future work will include application of the presented methods to larger models. This represents new challenges for the QR simulation tools, as for example in the case of weakly constrained Garp3 models. Here, the simulation output may become quite big (several thousand states) with many transitions, which leads to very long computation times (several days) and memory problems with current versions of Garp3.

Acknowledgements This work has been supported by the FIT-IT research project *Self Properties in Autonomous Systems(SEPIAS)* which is funded by BMVIT and the FFG, and the EU project ICT-216679, Model-based Generation of Tests for Dependable Embedded Systems (MOGENTES). The research herein is partially conducted within the competence network Softnet Austria (www.soft-net.at) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

References

Auguston, M.; Michael, J. B.; and Shing, M.-T. 2005. Environment behavior models for scenario generation and testing automation. In *International Workshop on Advances in Model Based Testing (A-MOST 2005)*. St. Louis, Missouri, USA: ACM.

Bouwer, A.; Liem, J.; and Bredeweg, B. 2005. *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). Project no. 004074. Project deliverable D4.2.1.

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

Bredeweg, B.; Bouwer, A.; Jellema, J.; Bertels, D.; Linnebank, F. F.; and Liem, J. 2006. Garp3 - a new workbench for qualitative reasoning and modelling. In *Proceedings of 20th International Workshop on Qualitative Reasoning (QR-06)*, 21–28.

Forbus, K. D. 1984. Qualitative process theory. *Artif. Intell.* 24(1-3):85–168.

Hierons, R. M.; Bogdanov, K.; Bowen, J. P.; Cleaveland, R.; Derrick, J.; Dick, J.; Gheorghe, M.; Harman, M.; Kapoor, K.; Krause, P.; Luettgen, G.; Simons, A. J. H.; Vilkomir, S.; Woodward, M. R.; and Zedan, H. 2008. Using formal specifications to support testing. *ACM Computing Surveys*.

Jard, C., and Jeron, T. 2004. TGV: theory, principles and algorithms. *International Journal on Software Tools for Technology Transfer (STTT)* 7(4):297–315.

Kleer, J. D., and Brown, J. S. 1984. A qualitative physics based on confluences. *Artif. Intell.* 24(1-3):7–83.

Thierry Jeron, P. M. 2004. Test generation derived from model-checking. *Computer Aided Verification: 11th International Conference, CAV'99. Trento, Italy, July 1999. Proceedings* 1633/1999(4):682.

Tretmans, J. 1996. Test generation with inputs, outputs, and quiescence. In *TACAS '96: Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, 127–146. Springer-Verlag.

Wotawa, F. 2007. Generating test-cases from qualitative knowledge – preliminary report. In *Proceedings of the 21st Annual Workshop on Qualitative Reasoning*.