Self-Monitoring and Control for Embedded Systems using Hybrid Constraint Automata

Paul Maier
Technische Universität München
Boltzmannstraße 3, 85748 Garching
+49.89.289.19595
maierpa@in.tum.de

Martin Sachenbacher
Technische Universität München
Boltzmannstraße 3, 85748 Garching
+49.89.289.19556
sachenba@in.tum.de

Summary

Many of today's mechatronic systems – such as automobiles, automated factories or chemical plants – are a complex mixture of hardware components and embedded control software, showing both continuous (vehicle dynamics, robot motion) and discrete (software) behavior. The problems of estimating the internal discrete/continuous state and automatically devising control actions as intelligent reaction are at the heart of self-monitoring and self-control capabilities for such systems. In this paper, we address these problems with a new integrated approach, which combines concepts, techniques and formalisms from AI (constraint optimization, hidden markov model reasoning), fault diagnosis in hybrid systems (stochastic abstraction of continuous behavior), and hybrid systems verification (hybrid automata, reachability analysis). Preliminary experiments with an industrial filling station scenario show promising results, but also indicate current limitations.

Keywords

Maximal 5 Keywords

1 Introduction

Many complex systems today – such as automobiles, automated factories or chemical plants – consist of hardware components whose functionality is extended or controlled by embedded software. Model-based diagnosis and planning algorithms using a discrete Hidden Markov Model (HMM) of the system's internal behavior have been proposed to address the problems of self-monitoring under partial observations and intelligent self-control to compensate for faults and other contingencies in such systems [WICE03]. Specifically, [WCG01] introduced Probabilistic Hierarchical Constraint Automata (PHCA) as a compact means of HMM encoding, which allows to conveniently model uncertain hardware behavior as well as complex software behavior. In previous work [MWS05], we have introduced an approach to efficiently compute best diagnoses and plans for systems modeled as PHCAs. It is based on encoding PHCA as soft constraints and then using a decomposition-based constraint optimization algorithm to compute best, i.e. most probable, system trajectories over a given time horizon of N steps.

Figure 1: Filling station.

However, many real-world components, like the silo of a filling station shown in figure 1, involve not only discrete behavior but also continuous dynamics; failures often manifest themselves as a subtle combination of the system's continuous dynamics, and its evolution through discrete behavior modes.

Hybrid systems have long been at the center of interest in model-based verification and increasingly gain attention in areas such as model-predictive control, model-based diagnosis and reconfiguration. Henzinger introduced the formalism of hybrid automata as a modeling framework for hybrid systems [H96], which is nowadays a widely accepted standard not only in hybrid systems verification. Recent advances in modeling concurrent stochastic hybrid systems have been published by Alur et al. [AGLS06, BSA04].

In this paper, we propose an extension of the PHCA formalism to Hybrid PHCAs (HyPHCAs), which allow modeling of continuous behavior as linear Ordinary Differential Equations (ODEs). Since HyPHCAs allow an infinite number of system trajectories, the main challenge is then to make computation of best trajectories on HyPHCAs tractable. We address this problem with an abstraction-based approach that combines concepts, techniques and formalisms from AI (constraint

optimization, hidden markov model reasoning), fault diagnosis in hybrid systems (stochastic abstraction of continuous behavior), and hybrid systems verification (hybrid automata, reachability analysis).

Model-based diagnosis/monitoring of hybrid systems is also addressed by the works of Lunze et al. [LN01, BKLS+06] and Williams et al. [HW02]. Lunze and co-workers introduced a method which abstracts continuous system models with stochastic automata, which encode Markov chains. The stochastic automaton formalism is similar to PHCA, but doesn't allow for complex hierarchical structures. Therefore they are less suited for creating models during the design phase of a technical system. Williams et al. introduced Probabilistic Hybrid Automata and describe a hybrid tracking algorithm, which combines discrete tracking using hidden markov models with continuous tracking using extended Kalman Filters.

The key difference between these existing approaches and our work is that we avoid specialized algorithms fitted to the modeling formalism (HyPHCAs in our case). Instead, we employ the general framework of constraint optimization [MRS06], and can therefore use existing, highly optimized off-the-shelf constraint solvers [BHGL+04] to solve the problems of monitoring and intelligent control. To take advantage of specific model features, we plan to develop formalism or model specific heuristics to, e.g., exploit the often refined model structure due to design. This makes our approach very flexible and it is a lot easier to incorporate new developments such as, e.g., Quantified Constraint Optimization [BLV08]. Furthermore, by extending PHCAs, a modeling framework explicitly designed for embedded model-based development, we are moving closer to the over-arching ideal of using a single model for system design, system verification and online model-based monitoring/control.

Here, we do not address the problem of hybrid *control* [KH05], we exclusively focus on discrete control inputs (commands). However, this mostly depends on the tools we use, and hence it should be possible to extend our method to hybrid control problems. In the next section, we introduce our motivating example, which we also used for our experiments. Then we introduce HyPHCAs in section 3, describe how we abstract them to receive discrete models in section 4 and show how monitoring and control problems can be solved based on a soft-constraint encoding of the discrete models in section 5. Finally, we present results and conclude with a discussion and future work.

2 Industrial Filling Station Example

As an example we use an industrial filling station employed in teaching [D07]. The station fills a granulate material in small bottles, which are transported to and away from the station on a conveyor belt. A pneumatic arm moves bottles from

the conveyor onto a swivel and back when they are finished. The swivel positions the bottles below a silo, where they are filled by a screw mechanism powered by an electrical motor. A photo sensor (binary signaled) indicates when the silo is empty. We created a simplified model of the filling station (shown in figure 2), which consists only of the silo and the sensor model. The silo fill level, during filling, is continuously modeled as $\dot{u}_{lvl} = -fR * u_{lvl}$ (where fR is the fill rate). This equation, while not realistic, demonstrates that our approach can handle such equations. We experimented with a scenario in which we address the *combined* problem of monitoring and control, and a second scenario which demonstrates how varying degrees of abstraction influence the monitoring quality.

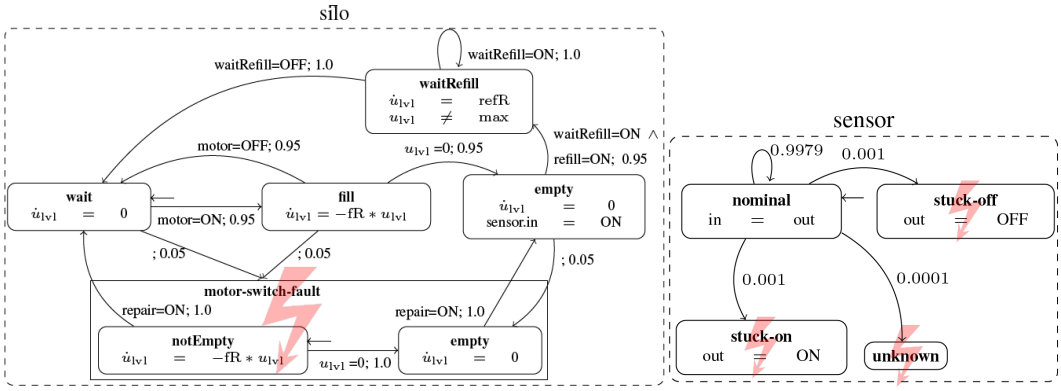

Figure 2: HyPHCA modeling the silo and the silo empty sensor of a filling station. The bolt indicates failure states (e.g., silo.motor-switch-fault, sensor.stuck-on).

In the first scenario (referred to as scenario 1, shown in table 2), which ranges over 10 time steps (duration of a single step $\Delta t = 2s$), the silo receives motor commands to fill two bottles. It has an initial fill level of 50 units. Within the first 7 time steps, the motor switch breaks, causing the motor to continue running and emptying the silo (referred to as motor-switch-fault). At t_0 the sensor indicates an empty silo.

The monitoring problem is to choose among three possible hypotheses explaining the signal: (1) the silo emptied nominally (2) the silo emptied too quickly due to the motor-switch-fault or (3) the sensor is stuck-on. A model which respects the continuous behavior allows a reasoner to detect an inconsistency with the sensor signal: the silo couldn't have emptied nominally, without the motor running. Thus, hypothesis (1) is ruled out. Since the sensor fault is much less likely than the motor fault, the reasoner correctly assumes hypothesis (2) as most probable.

The control problem is to find suitable actions to deal with the fault and reach a given goal stated by a high level control program. In this case the goal is that at t_3 in the future, the silo must have a fill level between 5 and 10 and be in its initial

location **wait** (see table 2). In the following, we describe an approach, combining several well known methods, which at the same time allows to deduce the correct fault hypothesis and the sequence of commands to reach the goal.

The second scenario (referred to as scenario 2, shown in figure 5) is a slight variation of the first, where we know that the motor control doesn't break . Again the sensor indicates an empty silo, but now earlier at t_{-3}. The reasoner, knowing about the continuous behavior, can deduce that even with the motor-switch-fault, the silo couldn't have emptied that quickly, ruling this fault out. Therefore, it correctly assumes that the sensor must be stuck-on, given the model abstraction is not too coarse.

Throughout the remaining text, we will use the following abbreviations: m-s-f refers to the motor-stuck-fault, m-s-f.ne and m-s-f.e refer to the primitive locations notEmpty and empty of m-s-f, dx refers to a partitioning of u_{lvl} with x partition elements (e.g., d10 if we partition u_{lvl} with 10 elements), s-on refers to stuck-on and nom. to nominal.

3 Modelling Hybrid System Behavior with HyPHCAs

Systems with mixed discrete/continuous behavior can be modeled using hybrid automata [H96], capturing continuous system evolution with ODEs over real-valued variables. They however don't support hierarchical structure and probabilistic behavior in order to uniformly model both probabilistic hardware behavior (e.g., likelihood of component failures) and complex software behavior (such as high level control programs). In contrast, PHCA [WCG01] have the required expressivity.

Definition 1. *A PHCA is a tuple* $A = \langle \Sigma, P_\Theta, \Pi, \mathcal{C}, P_T \rangle$*, where:*

- $\Sigma = \Sigma_c \cup \Sigma_p$ *is a set of composite and primitive locations. Each composite location denotes another PHCA. A location may be marked or unmarked. A marked location represents an active execution branch.*
- P_Θ *is a probability distribution over subsets* $\Theta_i \subseteq \Sigma$*, denoting the probability that* Θ_i *is the set of start locations.*
- $\Pi = \Pi_D \cup \Pi_{\mathrm{Obs}} \cup \Pi_{\mathrm{Cmd}}$ *is a set of dependent, observable and commandable variables, all having finite domains.* $\mathcal{C}[\Pi]$ *denotes the set of finite domain constraints over* Π*.*
- $\mathcal{C}$: $\Sigma \rightarrow \mathcal{C}[\Pi]$ *associates with each location* $l_i \in \Sigma$ *a finite domain constraint* $\mathcal{C}(l_i)$*.*
- $P_T(l_i)$*, for each* $l_i \in \Sigma_p$*, is a probability distribution over a set of transition functions* $T(l_i)$: $\Sigma_p^{(t)} \times \mathcal{C}[\Pi]^{(t)} \rightarrow 2^{\Sigma^{(t+1)}}$*. Each transition function*

maps a marked location into a set of locations to be marked at the next time point, provided that the transition's guard constraint is entailed.

Definition 2. *(PHCA state, PHCA trajectory) The state of a PHCA at time t is a set of marked locations called a marking $m^{(t)} \subset \Sigma$. A sequence of such markings $\theta = \{m^{(t)}, m^{(t+1)}, \ldots, m^{(t+N)}\}$ is called a PHCA trajectory.*

In the remainder, we will use the notation D_x to refer to the domain of a variable x, and D_X to refer to the cross product $D_{x1} \times \ldots \times D_{xn}$ of the domains of variables $x1, \ldots, xn \in X$.

PHCA don't allow to model continuous state evolution. Therefore, in style of hybrid automata, we define Hybrid PHCAs (HyPHCAs). They extend PHCAs with linear ODEs, a widely used standard for modeling continuous system evolution. A system of linear ODEs $\dot{\mathbf{u}} = \mathbf{A}\mathbf{u} + \mathbf{b}$, describes the time-continuous evolution of a vector of variables $\mathbf{u} = [u_1, \ldots, u_n]^T$ as a set of equations over $\mathbf{u}$ and their first derivatives $\dot{\mathbf{u}} = [\dot{u}_1, \ldots, \dot{u}_n]^T$. $\mathbf{b} = [b_1, \ldots, b_n]^T$ is a vector of constants and $\mathbf{A}$ the $n \times n$-matrix of coefficients for the equation set.

Definition 3. *A HyPHCA is a tuple $HA = \langle \Sigma, P_\Theta, \Pi, \mathcal{U}, \mathcal{C}, \text{flow}, P_T \rangle$ where*

- *$\mathcal{U} = U \cup \dot{U} \cup U'$ is a set of real-valued variables $U = \{u_1, \ldots, u_n\}$, their first derivatives $\dot{U} = \{\dot{u}_1, \ldots, \dot{u}_n\}$ and a set $U' = \{u_1', \ldots, u_n'\}$ representing values of U right after discrete transitions.*
- *$\mathcal{C}: \Sigma \rightarrow \mathcal{C}[\Pi \cup U \cup U']$ is a function associating locations with constraints over discrete and/or real-valued variables. $\mathcal{C}[\Pi \cup U \cup U']$ denotes the set of constraints over $\Pi \cup U \cup U'$.*
- *$\text{flow}: \Sigma \rightarrow \text{flow}[U \cup \dot{U}]$ is a function associating locations with constraints over real-valued variables and their derivatives in the form of* linear ordinary differential equations. *$\text{flow}[U \cup \dot{U}]$ denotes the set of these differential equations.*
- *P_T is a probability distribution over a set of transition functions $T(l_i): \Sigma_p \times \mathcal{C}[\Pi \cup U \cup U'] \rightarrow 2^\Sigma$ for locations $l_i \in \Sigma$. Each transition function $T(l_i)$ maps a primitive location marked at time to the set of locations to be marked at the next time instant, given the location's guard is entailed.*

Σ, P_Θ and Π are analog to the PHCA definition.

Definition 4. *(HyPHCA state, HyPHCA trajectory) The state of a HyPHCA at time t is a tuple $S^{(t_i)} = (S_U^{(t_i)}, m^{(t_i)})$, where $S_U^{(t_i)} \in \mathbb{R}^{|U|}$ is an assignment to all variables $u \in U$ at time t, called continuous state, and $m^{(t_i)} \in \mathcal{M}$ a marking analogous to the PHCA state marking (with $\mathcal{M} \subseteq 2^\Sigma$ the set of all markings). A function $\Delta: \mathbb{R} \rightarrow \mathbb{R}^{|U|} \times \mathcal{M}$, mapping time points (real-valued) to HyPHCA states, is called a HyPHCA trajectory function. A finite sequence $\theta_{HA} = \Delta(< t_i >)$, re-*

sulting from evaluating Δ on a finite sequence of time points, is called a discrete-time HyPHCA trajectory.

Discrete Flow and Clocked PHCAs

Our discrete approach to simultaneous tracking and control of hybrid system evolution requires to convert a HyPHCA model to a *discrete flow* PHCA (dfPHCA), conservatively abstracting continuous evolution with Markov chains. The evolution of a continuous variable $u \in U$ in between two time points t_i and t_{i+1} is thereby mapped to a discrete, timed transition between the quantized states of u at time t_i and time t_{i+1}. These discrete evolutions are encoded as *discrete flow constraints*. A dfPHCA is a tuple $A(\Delta t) = \langle \Sigma, P_\Theta, \Pi, \Pi_\mathcal{U}, \mathcal{C}, \mathrm{flow_d}, P_T, \Delta t \rangle$ (parameterized with fixed-length time interval Δt) where $\Pi_\mathcal{U} = \Pi_U \cup \Pi_{U'}$ is analogous to $\mathcal{U}$ of a HyPHCA, except that derivatives are omitted and variables have finite domains now. $\mathrm{flow_d}: \Sigma \to \mathrm{flow_d}[\Pi_U \cup \Pi_{U'}]$ is the discrete flow, a function associating locations with constraints encoding *Markov chains* over the discrete flow variables of the location. $\mathcal{C}$ is defined as for HyPHCAs with real-valued variable sets U and U' replaced by Π_U and $\Pi_{U'}$. The rest is analog to the PHCA definition. The state of $A(\Delta t)$ at time t is a tuple $S^{(t_i)} = (S^{(t_i)}_{\Pi_U}, m^{(t_i)})$, where $S^{(t_i)}_{\Pi_U}$ is an assignment of values to discretized continuous variables $x_u \in \Pi_U$ at time t, and $m^{(t_i)}$ a marking analogous to the PHCA state marking. A function $\Delta_{df}: (t_i) \to D_{\Pi_U} \times \mathcal{M}$, mapping the infinite set of real-valued time points $(t_i) := \{t_i | \forall i \in \mathbb{N}: \Delta t = t_i - t_{i+1}\}$ to dfPHCA states, is called a dfPHCA trajectory function. Evaluating Δ_{df} for a finite subset of (t_i) yields a finite sequence of dfPHCA states $\theta = \{S^{(t_i)}, S^{(t_{i+1})}, \ldots, S^{(t_{i+N})}\}$, called a dfPHCA trajectory.

To bridge the gap from dfPHCAs to PHCAs, we define *clocked* PHCAs as dfPHCAs (also parameterized with Δt) with discrete flows and discrete flow variables omitted. A clocked PHCA trajectory is therefore a function $\Delta_{\mathrm{clocked}}: (t_i) \to \mathcal{M}$ mapping to markings only. Clocked PHCAs can be seen as PHCAs with a fixed duration between time points. The key difference is the trajectory semantic. For a PHCA trajectory, only the indices of successive time points are relevant. I.e. a PHCA trajectory is a function $\Delta_{\mathrm{phca}}: \mathbb{N} \to \mathcal{M}$ mapping *natural numbers* to markings, rather than real-valued time points.

To avoid confusion, we write θ_A, $\theta_{A(\Delta t)}$ and θ_{HA} for trajectories of a PHCA A, clocked/discrete flow PHCA $A(\Delta t)$, and HyPHCA HA, respectively.

4 From Hybrid to Abstract Discrete Models

Discrete flow constraints encode special case PHCAs. In theory, a dfPHCA can be turned into an equivalent clocked PHCA by emedding discrete flows as sub-

PHCA into composite locations. So intuitively, a discrete abstraction of a HyPHCA is obtained by converting it to a dfPHCA, then converting the dfPHCA to a clocked PHCA as above and finally abstracting from time intervals, leaving a PHCA. However, certain non-trivial issues with hierarchical execution of PHCAs arise. First of, the PHCA formalism doesn't allow transitions originating from a composite location $l \in \Sigma_c$, they must originate from primitive locations within l. A more demanding problem is the following: Let $\mathrm{flow}_\mathrm{d}(l)$ be a discrete flow we embed as sub-PHCA A_{sub} into location l, rendering it composite. The PHCA marking semantics demand that sub-locations of l can only be marked when l itself is marked. Let further l be marked at t_i and, due to a transition away, not marked at t_{i+1}. Specifically, all locations of A_{sub} are unmarked at t_{i+1}. However, if location l with discrete flow $\mathrm{flow}_\mathrm{d}(l)$ is marked at t_i, $\mathrm{flow}_\mathrm{d}(l)$ should determine the values for variables in its scope at t_{i+1}. But since it is now encoded as sub-PHCA A_{sub}, which determines these values via its *marked* locations, this becomes impossible.

These issues make it hard to define and understand the abstraction of HyPHCAs using PHCAs directly. Therefore, we describe the abstraction using dfPHCAs. It remains for future work to show that, when time intervals are abstracted, an arbitrary dfPHCA has an equivalent PHCA and thus encodes an HMM like a PHCA.

4.1 Converting HyPHCAs to Discrete Flow PHCAs

First, we define further required entities. Let $HA = \langle \Sigma, P_\Theta, \Pi, \mathcal{U}, \mathcal{C}, \mathrm{flow}, P_\mathcal{T} \rangle$ be a HyPHCA. The set $\mathcal{T}$ denotes all transitions T defined through $P_\mathcal{T}$. $\mathrm{source}(T)$, $\mathrm{dest}(T)$ and $\mathrm{guard}(T)$ are a transition's source, destination set and guard constraint, respectively. $G_{\mathbb{R}^{|\mathcal{U}|}} = \{G_i\}$ is a set of disjunct grid cells (also called quantization cells) partitioning the continuous state space of HA: $\bigcup_i G_i = \mathbb{R}^{|\mathcal{U}|}$. Let $A(\Delta t) = \langle \Sigma, P_\Theta, \Pi, \mathcal{C}, P_\mathcal{T} \rangle$ be the dfPHCA with discrete flow constraints generated from HA. In the following, we refer to elements of the respective automatons, like Σ, by $HA.\Sigma$ and $A.\Sigma$, $HA.P_\Theta$ and $A.P_\Theta$, etc., except for those elements which are unique to one or the other formalism (e.g. $\mathcal{U}$).

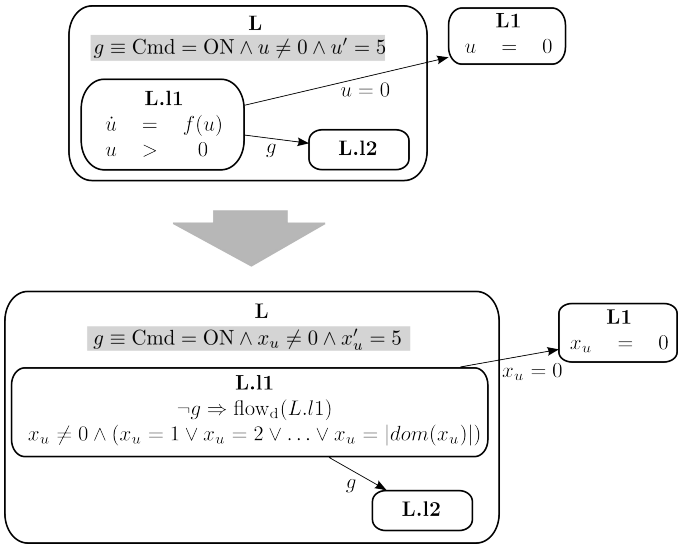

Figure 3: HyPHCA (above) is converted to a dfPHCA (below).

Figure 3 illustrates the HyPHCA to dfPHCA conversion. The conversion of locations, initial probability distributions and probabilistic transitions is straight forward: $A.\Sigma = HA.\Sigma$, $A.P_\Theta = HA.P_\Theta$ and $A.P_\mathcal{T} = HA.P_\mathcal{T}$. The PHCA vari-

ables of A consist of the discrete variables $HA.\Pi$ as well as Π_U and $\Pi_{U'}$, the discretized counterparts to U and U' (discrete versions of $\dot{U}$ are not needed): $A.\Pi = HA.\Pi \cup \Pi_U \cup \Pi_{U'}$. The constraints over finite domain and continuous variables in HA can be split into a set of constraints over discrete variables only and constraints over both finite and continuous variables: $HA.\mathcal{C}[HA.\Pi \cup U \cup U'] = HA.\mathcal{C}[HA.\Pi] \cup HA.\mathcal{C}'[HA.\Pi \cup U \cup U']$. The finite domain constraints of $A(\Delta t)$ are accordingly $A.\mathcal{C}[A.\Pi \cup A.\Pi_U \cup A.\Pi_{U'}] = HA.\mathcal{C}[HA.\Pi] \cup \mathrm{conv}(HA.\mathcal{C}'[HA.\Pi \cup U \cup U'])$. The function conv maps simple arithmetic constraints such as $u \leq 1$ or $u_1 \geq u_2$ to corresponding finite domain constraints.

The finite domains of discretized variables $A.\Pi_U, A.\Pi_{U'}$ are derived from the quantization $G_{\mathbb{R}^{|U|}}$. The grid cells $G_i \in G_{\mathbb{R}^{|U|}}$ can be mapped onto intervals of the variables U and U'. Index sets of these intervals then form the domains of $A.\Pi_U$ and $A.\Pi_{U'}$. That is, the values of, e.g., a variable $x_u \in A.\Pi_U$ represent intervals of corresponding variable $u \in U$.

Now for each primitive location $L \in HA.\Sigma$, its continuous flow $\mathrm{flow}(L)$ is converted to a discrete flow constraint $\mathrm{flow_d}(L)$. It encodes a special clocked PHCA $A_{\Delta t}^{\mathrm{Markov}}$, which has only primitive locations, corresponding to grid cells of $G_{\mathbb{R}^{|U|}}$, and represents a Markov chain that conservatively approximates the continuous flow. Its discrete, unguarded probabilistic transitions represent the evolution of variables $u \in U$ in between two time points t_i and t_{i+1}. $\mathrm{flow_d}(L)$ is added to the corresponding location $L \in A.\Sigma$. If it conflicts with transition guards determining variable values for t_{i+1} via $x_{u'} \in A.\Pi_{U'}$, the guard takes precedence over $\mathrm{flow_d}(L)$ (see, e.g., figure 3).

4.2 Discrete Abstraction of Continuous Flow

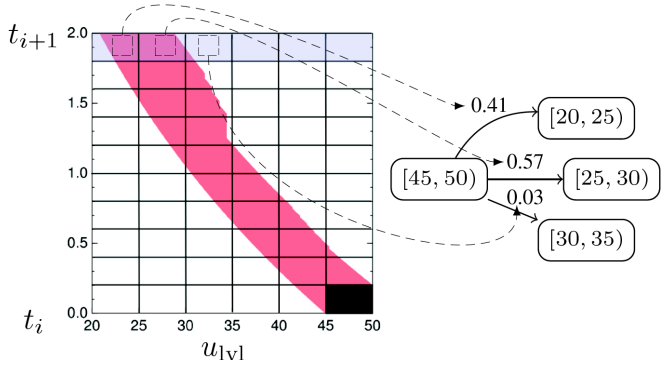

Figure 4: Reachable set R_{start} for $\dot{u}_{\mathrm{lvl}} = -\mathrm{fR} * u_{\mathrm{lvl}}$ starting from the marked grid cell. Right: the derived PHCA $A_{\Delta t}^{\mathrm{Markov}}$.

We can build on a lot of related work in the areas of automated verification and hybrid diagnosis. To conservatively estimate transition probabilities of $A_{\Delta t}^{\mathrm{Markov}}$ we use the geometric abstraction method introduced in [LN01]. The process is illustrated in figure 4: The reachable set R_{start} is computed (red), and from its overlap with gridcells at t_{i+1} (repre-

senting destination locations of $A_{\Delta t}^{\text{Markov}}$) the probabilities are derived. Currently we use PHAVer [F05] to compute R_{start}, but different approaches can be employed: In [ASB07] Stursberg et al. combine Markov chains abstracting continuous behavior with a more advanced reachability analysis. A too coarse state space quantisation can lead to spurious solutions, as scenario 2 demonstrates. Currently, the right number of partitions must be determined empirically. Hofbaur and Rienmüller introduced a method to intelligently quantize the continuous state space based on qualitative properties of piecewise affine systems [HR08]. This might be a useful extension to our approach as it automatically chooses a good number of partition elements, balancing precision of the abstraction against tractability, and reduces the number of spurious solutions.

It remains to show that a dfPHCA, generated as described above from a HyPHCA, is actually a conservative abstraction in terms of the probabilities of system trajectories, expressed with the proposition

Proposition 1. *(Discrete flow PHCAs as conservative abstraction of HyPHCAs) Let* $G: D_{\Pi_U} \rightarrow G_{\mathbb{R}^{|U|}}$ *be a function that maps assignments to discretized continuous variables* Π_U *to grid cells* $G_{x,t_i} \in G_{\mathbb{R}^{|U|}}$. *Let* $A(\Delta t)$ *be a discrete flow PHCA generated as described above from a HyPHCA* HA, $\langle o_i, c_i \rangle$ *a finite sequence of observations* $o_i \in D_{\Pi_{\text{Obs}}}$ *and commands* $c_i \in D_{\Pi_{\text{Cmd}}}$ *and* $\langle t_i \rangle$ *the corresponding sequence of time points. Let* $\theta_{A(\Delta t)}$ *be a trajectory of* $A(\Delta t)$ *consistent with* $\langle o_i, c_i \rangle$ *(i.e.* $P(\theta_{A(\Delta t)} | \langle o_i, c_i \rangle) > 0$*) and* $\chi(\theta_{A(\Delta t)}) := \{\Delta | \forall (S_U^{(t_i)}, m^{(t_i)}) \in \Delta(<t_i>), (\hat{S}_{\Pi_U}^{(t_i)}, \hat{m}^{(t_i)}) \in \theta_{A(\Delta t)} : m^{(t_i)} = \hat{m}^{(t_i)} \wedge S_U^{(t_i)} \in G(\hat{S}_{\Pi_U}^{(t_i)})\}$ *the set of all HyPHCA trajectories contained in* $\theta_{A(\Delta t)}$. *Then the following holds:*

$$\forall \Delta \in \chi(\theta_{A(\Delta t)}): f_{HA}(\Delta(\langle t_i \rangle) | \langle o_i, c_i \rangle) \leq P(\theta_{A(\Delta t)} | \langle o_i, c_i \rangle)$$

$P(\theta_{A(\Delta t)} | \langle o_i, c_i \rangle)$ is the probability of dfPHCA trajectory $\theta_{A(\Delta t)}$ occurring given the sequence $\langle o_i, c_i \rangle$ of observations and commands. Likewise, $f_{HA}(\Delta(\langle t_i \rangle) | \langle o_i, c_i \rangle)$ is the conditional density function of a distribution over discrete-time HyPHCA trajectories.

5 Monitoring and Control as Constraint Optimization

Given a discretized model, partial observations, known commands and a goal state $S^{(t_{i+n})}$, we combine the problems of system monitoring and finding goal achieving commands into a single problem of finding the most probable system trajectory over N time steps which is consistent with the observations and contains $S^{(t_{i+n})}$. From this trajectory the goal achieving commands can be easily derived. We frame this problem as a discrete constraint optimization problem (COP) $\mathcal{R} = (X, D, C)$ [SFV95] with transition probabilities as preferences by translating the discretized model to soft-constraints following our framework in [MWS05].

The translation unfolds a given PHCA A over a time window of N steps as follows: $X = (X_1, \ldots, X_n)$ is a set of variables with corresponding set of finite domains $D = (D_1, \ldots, D_n)$. For all time points $t_i, i = 0..N$, it consists of $\Pi^{(t_i)} \subseteq X$ encoding PHCA variables, auxiliary variables (needed to, e.g., encode hierarchical structure) and the solution variables of the COP, a set of binary variables $Y = (X_{L_1}^{(t_i)}, X_{L_2}^{(t_i)}, \ldots) \subseteq X$ representing location markings of A. $C = (C_1, \ldots, C_n)$ is a set of constraints (S_j, F_j) with scope $S_j = (X_{j1}, \ldots, X_{jm}) \subseteq X$ and a constraint function $F_j : D_{j1} \times \ldots \times D_{jm} \rightarrow [0,1]$ mapping partial assignments of variables in S_j to a probability value in $[0,1]$. For all time steps $t_i, i = 1..N$, hard constraints in C (F_j evaluates to $\{0,1\}$) encode hierarchical structure as well as consistency of observations and commands with locations and transitions, while soft constraints in C encode probabilistic choice of initial locations at t_0 (here, $i = 0$ marks the start of the time window, not the time point corresponding to present) and probabilistic transitions. All assignments to Y form a set ordered by the global probability value in terms of the functions F_j (evaluated on the assignments extended to X). The k assignments with highest probability are the k-best solutions to $\mathcal{R}$, which correspond to the most probable PHCA system trajectories. Their extension to X provides assignments to, e.g., goal achieving commands.

To encode dfPHCAs, we extended the framework with a soft-constraint encoding of discrete flow constraints. This can be done very compactly and is thus a further advantage over a PHCA encoding (as discussed in section 4). A flow constraint is ``active" if and only if its associated location is marked and it is not overridden by a transition guard. We encode this logic with hard constraints for each location and time point, which implement the formula $O_l^{(t_i)} = \mathrm{FALSE} \wedge X_l^{(t_i)} = \mathrm{MARKED} \Leftrightarrow X_{\mathrm{flow_d}(l)}^{(t_i)} = \mathrm{ACTIVE}$. The auxiliary variables $O_l^{(t_i)}$ with domain $\{\mathrm{TRUE}, \mathrm{FALSE}\}$ and $X_{flow_d}(l)^{(t_i)}$ with domain $\{\mathrm{ACTIVE}, \mathrm{INACTIVE}\}$ indicate an override of a discrete flow and its activation, respectively. The discrete flow itself is a function mapping discrete flow variables in $\Pi_U^{(t_i)}$ and $\Pi_U^{(t_{i+1})}$ to transition probabilities. Again for each location and time point we encode these functions as soft constraints, extending their scope with $X_{flow_d}(l)^{(t_i)}$, keeping the former mappings if $X_{flow_d}(l)^{(t_i)} = \mathrm{ACTIVE}$ and mapping to zero otherwise.

All described steps up to now – discretizing, generating Markov chains, encoding as COP $\mathcal{R}$ – can be done offline. Online, we iteratively add observations and known commands to $\mathcal{R}$ and solve the COP to generate the k most likely system trajectories. For this step we employ existing off-the-shelf solvers such as Toulbar2[1].

[1]https://mulcyber.toulouse.inra.fr/projects/toulbar2/

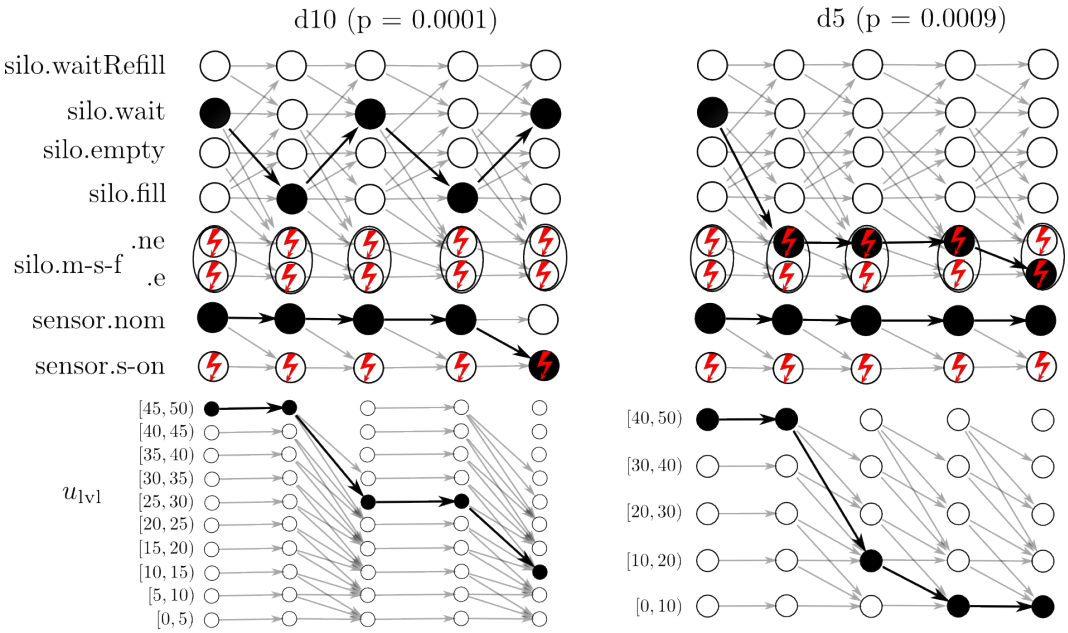

Figure 5: The inferred system trajectories for the sensor-fault scenario as trellis diagram for d10 for u_{lvl} (left) and for d5 (right).

6 Results

We created and solved COP instances with different discretizations for u_{lvl} (d2, d5, d10 and d25) for our example scenarios and some variations. We tried Toulbar2's default and a second, decomposition based configuration. The problem size was for all instances 843 variables and 920 constraints. For scenario 1 with d10 table 2 shows the most probable system trajectory the solver deduced from the given observations and goals as variable assignments in bold face (given assignments in normal font). The generated solution correctly identifies the motor-switch-fault and provides the necessary commands to reach the goal: repair = ON for t_0, refill = ON and waitRefill = ON for t_1 and waitRefill = OFF for t_2. The most probable system trajectories for scenario 2 with d5 and d10 are shown in figure 5 as trellis diagrams depicting discrete transitions of the dfPHCA. Big black arrows and black filled circles mark the found most probable trajectory, grey arrows show possible transitions. In this scenario, the reasoner misses the fault sensor.stuck-on if u_{lvl} is abstracted too

Table 1: Left: Runtime results (mean time in sec.) for example scenarios over different discretizations for u_{lvl}.

Online Runtime					
Toulbar2 config.	Discretisation	Scenarios 1	1.1	1.2	2
default	d2	0.016s	0.026s	0.028s	0.023s
	d5	0.007s	0.013s	0.010s	0.037s
params	d10	0.014s	0.026s	0.016s	0.037s
	d25	0.030s	0.054s	0.032s	0.103s
with	d2	0.126s	0.172s	0.200s	0.250s
	d5	0.118s	0.130s	0.158s	0.252s
tree	d10	0.122s	0.156s	0.164s	0.240s
decomp.	d25	0.138s	0.178s	0.178s	0.327s

coarsely (d5). We assume spurious solutions to be the culprit: The coarser the abstraction, the more probable become evolutions of $\mathfrak{u}_{\text{lvl}}$ which in reality are very unlikely or impossible. This causes the combination of the more likely motor-switch-fault and a spurious evolution of $\mathfrak{u}_{\text{lvl}}$ to become most probable. With a sufficiently fine grained abstraction (d10), the spurious evolution's probability is reduced to near zero, which rules out the incorrect motor-switch-fault and leaves the sensor.stuck-on fault as most probable.

Table 1 shows the average online runtimes for all scenarios. The columns show results for scenario 1, its two variations 1.1 and 1.2, and scenario 2. The variations are nominal behavior (1.1) and diagnose motor-switch-fault only (1.2). As one would expect, a slight increase in runtime can be seen for the more fine grained discretization d25. The variations 1.1 and 1.2 take roughly the same time as scenario 1. Small differences are probably due to the fact that the variations are the same COP with some constraints omitted. E.g., when diagnosing the motor-switch-fault only, the goal is omitted. This makes the problem slightly harder because more future evolutions are possible. Finally, we expected the offline decomposition of the problem to lower online computation effort, but surprisingly, it had a negative effect in our scenario.

The offline steps (discretization, Markov chain generation and soft constraint encoding) for d2, d5, d10 and d25 take 16.5, 39.0, 138.5 and 215.4 seconds. They show that the effort for model abstraction and encoding is considerable, even for a small model. However, runtime is still within manageable bounds. Memory consumption might be a bigger issue (for d25: $\approx$ 300 MB), it remains for future experiments to show the limits of our method. Most of the resources are, however, consumed by the Markov chain generation, i.e. the discrete model part influences offline resource usage only marginally. This is not surprising: Converting the discrete part of a HyPHCA to a discrete PHCA and encoding the final discrete model as soft-constraints is linear in the size of the model, whereas the step of Markov chain generation is exponential in the dimension of the continuous state space.

Resource usage might be reduced by improving reachability analysis, e.g., by optimizing PHAVer parameters. Also, there's room for improvement over our prototype implementation of the Markov chain generation. Finally note that of course components with the same continuous behavior require Markov chain generation only once. Due to the costly reachability analysis the approach might not be suitable for systems with many components showing *different* continuous behavior.

Table 2: The monitoring/control results for our example scenario 1 (discretization with 10 partition elements of $\mathfrak{u}_{\text{lvl}}$). The rows show: Known sensor values (1 row), known commands (4 rows), marked locations for sensor and silo (2 rows) and fill level (1 row).

	Time step										
	Past							Present	Future		
Variable	t_{-7}	t_{-6}	t_{-5}	t_{-4}	t_{-3}	t_{-2}	t_{-1}	t_0	t_1	t_2	t_3
sensor.out	OFF	OFF	OFF	OFF	OFF	OFF	OFF	ON	-	-	-
silo.motor	ON	OFF	ON	OFF	OFF	OFF	OFF	OFF	-	-	-
silo.repair	OFF	OFF	OFF	OFF	OFF	OFF	OFF	**ON**	**OFF**	**OFF**	**OFF**
silo.waitRefill	OFF	OFF	OFF	OFF	OFF	OFF	OFF	**OFF**	**ON**	**OFF**	**OFF**
silo.refill	OFF	OFF	OFF	OFF	OFF	OFF	OFF	**OFF**	**ON**	**OFF**	**OFF**
silo location	**wait**	**fill**	**wait**	**m-s-f.ne**	**m-s-f.ne**	**m-s-f.ne**	**m-s-f.ne**	**m-s-f.e**	**empty**	**waitRefill**	wait (goal)
sensor location	**nom.**	**nom.**	**nom.**	**nom.**	**nom.**	**nom.**	**nom.**	**nom.**	**nom.**	**nom.**	**nom.**
u_{lvl}	$[45,50)$	$[45,50)$	$[25,30)$	$[25,30)$	$[10,15)$	$[5,10)$	$[0,5)$	$[0,5)$	$[0,5)$	$[0,5)$	$[10,15)$ (goal)
Legend:	nom. → nominal; m-s-f.ne → motor-switch-fault.notEmpty; m-s-f.e → motor-switch-fault.empty										

7 Conclusion

Estimating the internal discrete/continuous state, and automatically devising control actions as intelligent reaction to identified failures and contingencies are at the heart of self-monitoring and self-control capabilities for embedded (mixed hardware/software) systems. We introduced HyPHCAs, an extension to PHCAs, as a modeling framework and showed how to combine several methods from AI, fault diagnosis in hybrid systems and hybrid systems verification to track the state and compute reactive actions for mixed discrete/continuous systems modeled as HyPHCAs. In an offline step, the approach abstracts the differential equations of the HyPHCA to Markov chains encoded as PHCAs, embeds them in the discrete part of the HyPHCA, and encodes the discrete abstraction with soft-constraints, such that on-line monitoring and control of the system can be done by solving a discrete constraint optimization problem. Our experimental results demonstrate the feasibility of the approach on a small, but real-world factory scenario. Our next steps are to refine the semantics of HyPHCAs in terms of probability distributions over trajectories, to develop an estimator module which iteratively shifts the time window ([MWS05]) to monitor systems over long time periods and verify our results on larger factory settings such as [BBW]. In this and in other settings, accurate model-based monitoring and control can only be achieved by considering both hybrid hardware and software behavior.

Literature

[WICE03] Williams, B. C.; Ingham, M.; Chung, S. H.; Elliott, P. H.: *Model-Based Programming of Intelligent Embedded Systems and Robotic Space Explorers.* Proceedings of the IEEE: Special Issue on Modeling and Design of Embedded Software, 91(1):212--237, 2003

[WCG01] Williams, B. C.; Chung, S. H.; Gupta, V.: *Mode Estimation of Model-Based Programs: Monitoring Systems With Complex Behavior*. In: Proc. IJCAI-01, pages 579--590, 2001.

[MWS05] Mikaelian, T.; Williams, B. C.; Sachenbacher, M.: *Model-based Monitoring and Diagnosis of Systems with Software-Extended Behavior*. In: Proc. AAAI-05, 2005.

[H96] Henzinger, T.: *The Theory of Hybrid Automata.* In: Proc. LICS-1996, pages 278--292, New Brunswick, New Jersey, 1996.

[AGLS06] Alur, R.; Grosu, R.; Lee, I.; Sokolsky, O.: *Compositional Modeling and Refinement for Hierarchical Hybrid Systems.* Journal of Logic and Algebraic Programming, 68(1-2):105-128, 2006.

[BSA04] Bernadsky, M.; Sharykin, R.; Alur, R.: *Structured Modeling of Concurrent Stochastic Hybrid Systems.* In: Proc. FORMATS/FTRFT-2004, pages 309--324, 2004.

[LN01] Lunze, J.; Nixdorf, B.: *Representation of Hybrid Systems by Means of Stochastic Automata.* Mathematical and Computer Modelling of Dynamical Systems, 7:383--422, 2001.

[BKLS+06] Blanke, M.; Kinnaert, M.; Lunze, J.; Staroswiecki, M.; Schröder, J.: *Chapter 9: Diagnosis and Reconfiguration of Quantized Systems.* In: Diagnosis and Fault Tolerant Control, pages 447-504. Springer, 2nd edition, 2006.

[HW02] Hofbaur, M. W. and Williams, Brian C.: *Mode Estimation of Probabilistic Hybrid Systems.* In: Proc. HSCC-02, pages 253--266, Stanford, California, USA, 2002. Springer.

[MRS06] Meseguer, P.; Rossi, F.; Schiex, T.: *Chapter 9: Soft Constraints.* In: Handbook of Constraint Programming, Elsevier, 2006.

[BHGL+04] Bouveret, S.; Heras, F.; Givry, S. de; Larrosa, J.; Sanchez, M.; Schiex, T.: *Toolbar: a State-of-the-Art Platform for WCSP*, http://www.inra.fr/mia/T/degivry/ToolBar.pdf, 2004.

[BLV08] Benedetti, M.; Lallouet, A.; Vautard, J.: *Quantified Constraint Optimization*. In: Proc. CP-2008, LNCS, pages 463--477, Sydney, Australia, 2008. Springer.

[KH05] Kleissl, W.; Hofbaur, M.: *A Qualitative Model for Hybrid Control.* In: Workshop Proc. QR-05. Volume 19. Graz, Austria, 2005

[D07] Dominka, S.: *Hybride Inbetriebnahme von Produktionsanlagen --- Von der Virtuellen zur Realen Inbetriebnahme (in German)*. PhD thesis, Technische Universität München, 2007.

[F05] Frehse, G.: *PHAVer: Algorithmic Verification of Hybrid Systems Past Hytech.* In: Proc. HSCC-05, pages 258-273, 2005.

[ASB07] Althoff, M.; Stursberg, O.; Buss, M.: *Online Verification of Cognitive Car Decisions*. In: IEEE Intelligent Vehicles Symposium, 2007.

[HR08] Hofbaur, M. W.; Rienmüller, T.: *Qualitative Abstraction of Piecewise Affine Systems*. In: Workshop Proc. QR-08, 2008.

[SFV95] Schiex, T.; Fargier, H.; Verfaillie, G.: *Valued Constraint Satisfaction Problems: Hard and Easy Problems*. In: Proc. IJCAI-1995, 1995.

[BBW] Buss, M.; Beetz, M.; Wollherr, D.: *CoTeSys - Cognition for Technical Systems.* In: Proc. COE Workshop on Human Adaptive Mechatronics (HAM), 2007.

Author

Vita (short text)