Diagnosis and Fault-Adaptive Control for Mechatronic Systems using Hybrid Constraint Automata

Paul Maier and Martin Sachenbacher
Technische Universität München
Department of Informatics, Boltzmanstraße 3, 85748 Garching, Germany
{maierpa, sachenba}@in.tum.de

ABSTRACT

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.

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 (Williams *et al.*, 2003). Specifically, (Williams *et al.*, 2001) 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 (Mikaelian *et al.*, 2005), we have introduced an approach to efficiently compute best diagnoses and plans for systems modeled as PHCAs, which is based on encoding PHCA as soft constraints and then using a decomposition-based constraint optimization algorithm to compute best solutions over a given time horizon of N steps.

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 (Henzinger, 1996), 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. (Alur *et al.*, 2006; Bernadsky *et al.*, 2004).

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. (Lunze and Nixdorf, 2001; Blanke *et al.*, 2006) and Williams et al. (Hofbaur and Williams, 2002). 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 track-

Figure 1: Filling station.

ing 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 (Pedro Meseguer *et al.*, 2006), and can therefore use existing, highly optimized off-the-shelf constraint solvers (Bouveret *et al.*, 2004) to solve the problems of monitoring/state estimation and intelligent control. To take advantage of specific model features, we plan to develop formalism or model specific heuristics. For example, the general dynamic programming method *cluster tree elimination* used in constraint optimization (Dechter, 2003) could be guided by a HyPHCA-specific heuristic, taking advantage of 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 (Benedetti *et al.*, 2008). Furthermore, by extending PHCAs, a modeling framework which is explicitly designed for model-based development of embedded systems, we are moving closer to the over-arching ideal of one-model-fits-it-all, i.e. from system design to system verification and online model-based monitoring and control.

We do not address the problem of hybrid *control* (Kleissl and Hofbaur, 2005) in this paper, since we exclusively focus on discrete, finite control inputs (commands). However, this is mostly a question of the tools we use in our framework, 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 the HyPHCA formalism in section 3, describe how we abstract HyPHCAs 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 (Dominka, 2007). 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 $\dot{u}_{\mathrm{lvl}} = -\mathrm{fR} * u_{\mathrm{lvl}}$ (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.

In the first scenario (referred to as scenario 1, shown in table 1), which ranges over 10 time steps (duration of a single step $\triangle t = 2s$), 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, commands in our case, to deal with the fault and reach a given goal stated by a high level control program. In this scenario 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 1). 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 lo-

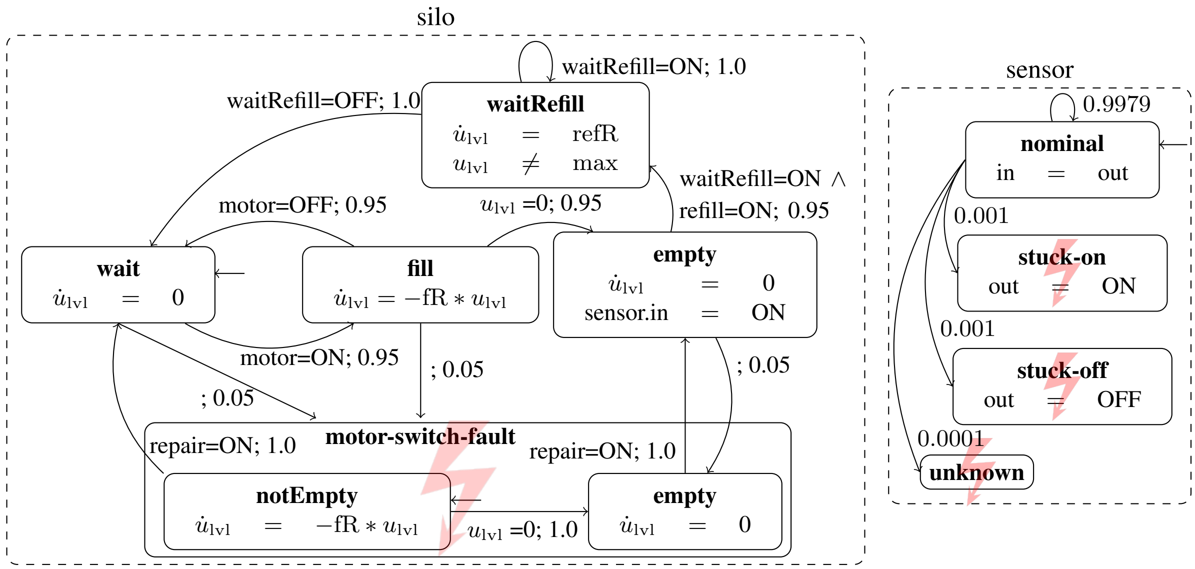

Figure 2: HyPHCA modeling the silo and the silo empty sensor of a filling station. The bolt indicates failure states (e.g., silo.motor-switch-fault, sensor.stuck-on).

cations notEmpty and empty of m-s-f, s-on refers to stuck-on and nom. to nominal. Furthermore, dx refers to a partitioning of u_{lvl} with x partition elements (e.g., d10 if we partition u_{lvl} with 10 elements, yielding a discrete variable lvl with 10 values).

3 MODELLING HYBRID SYSTEM BEHAVIOR WITH HYBRID PHCAS

Systems with mixed discrete/continuous behavior can be modeled using the well known Hybrid Automata (Henzinger, 1996), capturing continuous system evolution with ordinary differential equations (ODEs) over real-valued variables and discrete, commanded switches with guarded transitions. They however don't support hierarchical structure and probabilistic transitions in order to uniformly model both uncertain hardware behavior (e.g., likelihood of component failures) and complex software behavior (such as control programs). In contrast, probabilistic hierarchical constraint automata (PHCA) (Williams *et al.*, 2001) 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.
- $\mathcal{C} : \Sigma \to \mathcal{C}[\Pi]$ 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 $T(l_i) : \Sigma_p^{(t)} \times \mathcal{C}[\Pi]^{(t)} \to 2^{\Sigma^{(t+1)}}$. 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 $x1, \ldots, xn \in X$.

One important set of parameters in PHCA models are the transition probabilities, e.g., failure probabilities. These are typically a) specified by domain experts or b) learned, e.g., through an online learning component as described in (de Kleer *et al.*, 2009). A combination of the two options is possible as well. In our example, the probabilities have simply been chosen following our intuition, but our approach could be extended by a learning component such as (de Kleer *et al.*, 2009).

PHCA don't allow to model continuous state evolution over real-valued variables. Therefore, in style of hybrid automata, we extend PHCAs to so called Hybrid PHCAs (HyPHCAs). We adopt linear ODEs for the HyPHCA formalism, 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 $HA = \langle \Sigma, P_\Theta, \Pi, \mathcal{U}, \mathcal{C}, \mathcal{F}, P_T \rangle$ 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.
- $\mathcal{C} : \Sigma \to \mathcal{C}[\Pi \cup U \cup U']$ 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 $\Pi \cup U \cup U'$.
- $\mathcal{F} : \Sigma \to \mathcal{F}[U \cup \dot{U}]$ is a function associating locations with constraints over real-valued variables and their derivatives in the form of *linear ordinary differential equations*. $\mathcal{F}[U \cup \dot{U}]$ denotes the set of these differential equations.
- P_T is a probability distribution over a set of transition functions $T(l_i) : \Sigma_p \times \mathcal{C}[\Pi \cup U \cup U'] \to 2^\Sigma$ for locations $l_i \in \Sigma$. Each transition function $T(l_i)$ maps a primitive location marked at time t 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 PHCA states (with $\mathcal{M} \subseteq 2^\Sigma$ the set of all markings). A function $\Delta : \mathbb{R} \to \mathbb{R}^{|U|} \times \mathcal{M}$, mapping time points (real-valued) to HyPHCA states, is called a HyPHCA trajectory function. A finite sequence $\theta_{HA} = \Delta(\langle t_i \rangle)$, resulting from evaluating Δ on a finite sequence of time points, is called a discrete-time HyPHCA trajectory.

Discrete Flow and Clocked PHCAs

Our purely discrete approach to simultaneous tracking and control of hybrid system evolution requires a discrete abstraction of the hybrid model. We achieve this by converting a HyPHCA to a *discrete flow* PHCA, conservatively abstracting continuous variables and their evolution over time 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* of a discrete flow PHCA (dfPHCA). A dfPHCA is a tuple $A_{\text{df}}(\triangle t) = \langle \Sigma, P_\Theta, \Pi, \Pi_\mathcal{U}, \mathcal{C}, \mathcal{F}_d, P_T, \triangle t \rangle$ (parameterized with fixed-length time interval $\triangle t$) 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. $\mathcal{F}_d : \Sigma \to \mathcal{F}_d[\Pi_U \cup \Pi_{U'}]$ 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 $A_{\text{df}}(\triangle t)$ at time t is a tuple $S^{(t_i)} = (S_{\Pi_U}^{(t_i)}, m^{(t_i)})$, where $S_{\Pi_U}^{(t_i)}$ 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 PHCA states. A function $\Delta_{\text{df}} : \{t_i\} \to D_{\Pi_U} \times \mathcal{M}$, mapping the infinite set of real-valued time points $\{t_i\} := \{t_i | \forall i \in \mathbb{N} : \triangle t = t_i - t_{i+1}\}$ 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.

In order to bridge the gap from dfPHCAs to PHCAs, we define *clocked* PHCAs as dfPHCAs (also parameterized with $\triangle t$) with discrete flows and discrete flow variables omitted. A clocked PHCA trajectory is consequently a function $\Delta_{\text{cl}} : \{t_i\} \to \mathcal{M}$ mapping to markings only. Clocked PHCAs can be seen as PHCAs with a forced, 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. the PHCA trajectory function $\Delta_{\text{phca}} : \mathbb{N} \to \mathcal{M}$ maps *natural numbers* to markings, rather than real-valued time points.

To avoid confusion when referring to trajectories, we write θ_x with $x = A, A_{\text{cl}}(\triangle t), A_{\text{df}}(\triangle t), HA$ for PHCA, clocked PHCA, dfPHCA and HyPHCA trajectories.

4 FROM HYBRID TO ABSTRACT DISCRETE MODELS

We will see that discrete flow constraints encode special case PHCAs and that a dfPHCA can thus be turned into an equivalent clocked PHCA. The discrete flows then form sub-PHCA embedded into composite locations. So intuitively, a discrete abstraction of a HyPHCA is obtained by abstracting continuous flows to discrete flows of a dfPHCA, then converting the dfPHCA to a clocked PHCA and finally abstracting from time intervals, leaving a PHCA. However, certain non-trivial issues with hierarchical execution of PHCAs arise. One problem is that 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 second, more demanding problem is this: Let's assume we simply embed a discrete flow $\mathcal{F}_d(l)$ as a sub-PHCA A_{sub} into a 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's further assume that l is marked at t_i and that a transition occurs such that l is 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 $\mathcal{F}_d(l)$ is marked at t_i, $\mathcal{F}_d(l)$ 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 clocked PHCAs or PHCAs directly. Therefore, we describe the abstraction using dfPHCAs. Also, discrete flow constraints can be directly encoded as soft-constraints (discussed later in the paper), which yields a very compact encoding. It remains for future work to show that dfPHCAs, if time intervals are abstracted, like PHCAs encode HMMs. This can be done by showing that an arbitrary dfPHCA has an equivalent clocked PHCA (and thus a PHCA, after time abstraction).

Proposition 5. *(Equivalence dfPHCA, clocked PHCA) Let $\langle o_i, c_i\rangle$ be an arbitrary sequence of observations and commands. Then for an arbitrary dfPHCA $A_{\text{df}}(\triangle t)$ exists an equivalent clocked PHCA $A_{\text{cl}}(\triangle t)$, such that*

$$P(\rho(\theta_{A_{\text{df}}(\triangle t)})|\langle o_i, c_i\rangle) = P(\theta_{A_{\text{df}}(\triangle t)}|\langle o_i, c_i\rangle)$$

The function $\rho : D_{\Pi_U} \times \mathcal{M}_{\text{df}} \rightarrow \mathcal{M}_{\text{cl}}$ maps (sequences of) dfPHCA states to (sequences of) clocked PHCA states, specifically discrete flow variable assignments to markings of sub-PHCAs which encode the discrete flow. $P(\rho(\theta_{A_{\text{df}}(\triangle t)})|\langle o_i, c_i\rangle)$ and $P(\theta_{A_{\text{df}}(\triangle t)}|\langle o_i, c_i\rangle)$ are the probabilities of a clocked and discrete flow PHCA trajectory occurring, respectively, given the sequence $\langle o_i, c_i\rangle$.

We now describe the conversion of HyPHCAs to dfPHCAs (illustrated in figure 3) and then in detail how discrete flows are generated from continuous flows.

4.1 Converting HyPHCAs to Discrete Flow PHCAs

First, we define further required entities. Let $HA = \langle \Sigma, P_\Theta, \Pi, \mathcal{U}, \mathcal{C}, \mathcal{F}, P_T\rangle$ be a HyPHCA. The set $\mathcal{T}$ denotes all transitions T defined through P_T. $\text{source}(T)$, $\text{dest}(T)$ and $\text{guard}(T)$ are a transition's source, destination set and guard constraint, respectively. $G_{\mathbb{R}^{|U|}} = \{G_\lambda\}$ is a set of disjunct grid cells (also called quantization cells) partitioning the continuous state space of HA: $\bigcup_\lambda G_\lambda = \mathbb{R}^{|U|}$. Let $A_{\text{df}}(\triangle t) = \langle \Sigma, P_\Theta, \Pi, \Pi_{\mathcal{U}}, \mathcal{C}, \mathcal{F}_d, P_T, \triangle t\rangle$ 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 $A.P_\Theta$ (where we abbreviate $A_{\text{df}}(\triangle t)$ with A), etc., except for those elements which are unique to one or the other formalism (e.g. U).

The conversion of locations, initial probability distributions, discrete variables and probabilistic transitions is straight forward: $A.\Sigma = HA.\Sigma$, $A.P_\Theta = HA.P_\Theta$, $A.\Pi = HA.\Pi$ and $A.P_T = HA.P_T$. The discretized counterparts to $\mathcal{U}$, Π_U and $\Pi_{U'}$ form $\Pi_{\mathcal{U}}$ (discrete versions of the derivatives $\dot{U}$ are not needed and thus omitted): $\Pi_{\mathcal{U}} = \Pi_U \cup \Pi_{U'}$. The constraints over finite domain and continuous variables in HA can be split into a purely discrete set of finite domain constraints 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 $A_{\text{df}}(\triangle t)$ are accordingly $A.\mathcal{C}[A.\Pi \cup A.\Pi_U \cup A.\Pi_{U'}] = HA.\mathcal{C}[HA.\Pi] \cup \text{conv}(HA.\mathcal{C}'[HA.\Pi \cup U \cup U'])$. 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 $\Pi_U, \Pi_{U'}$ are derived from the quantization $G_{\mathbb{R}^{|U|}}$. The grid cells $G_\lambda \in G_{\mathbb{R}^{|U|}}$ can be mapped directly onto intervals of the variables U and U'. Index sets of these intervals then form the domains of the discretized, finite domain variables Π_U and $\Pi_{U'}$. That is, the values of, e.g., a variable $x_u \in \Pi_U$ represent intervals of corresponding variable $u \in U$.

Now for each primitive location $L \in HA.\Sigma$, its continuous flow $\mathcal{F}(L)$ is converted to a discrete flow constraint $\mathcal{F}_d(L)$. The evolution of continuous variables $u \in U$ in between two time points t_i and t_{i+1} is mapped to discrete, unguarded probabilistic transitions between locations of a special clocked PHCA $A^{\text{Markov}}_{\triangle t}$, encoded in $\mathcal{F}_d(L)$. It has only primitive locations, corresponding to grid cells of $G_{\mathbb{R}^{|U|}}$, and represents a Markov chain that conservatively approximates the continuous evolution. The discrete flow constraint encodes $A^{\text{Markov}}_{\triangle t}$ by directly relating variables $x_u \in A.\Pi_U$ for two time points t_i and t_{i+1}. $\mathcal{F}_d(L)$ is added to the corresponding location $L \in A.\Sigma$. If $\mathcal{F}_d(L)$ conflicts with transition guards determining variable values for t_{i+1} via $x'_u \in A.\Pi_{U'}$, the guard takes precedence over $\mathcal{F}_d(L)$ (see, e.g., figure 3).

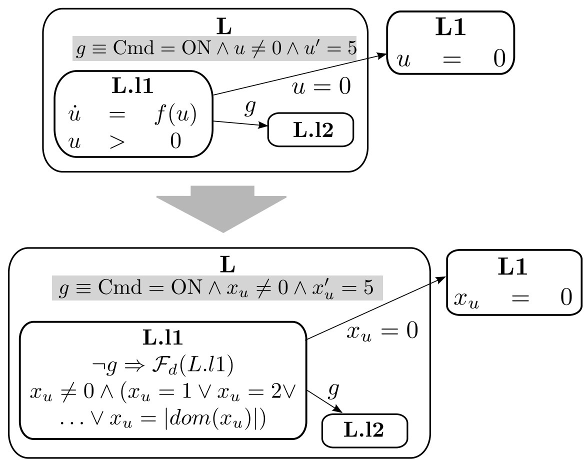

Figure 3: HyPHCA (above) is converted to a dfPHCA (below).

4.2 Discrete Abstraction of Continuous Flow

To conservatively estimate transition probabilities of $A^{\text{Markov}}_{\triangle t}$ we use the geometric abstraction method introduced in (Lunze and Nixdorf, 2001). We recap this method shortly. The quantized state space is combined with a partition of the time interval $[t_i, t_{i+1}]$. Start locations of transitions of $A^{\text{Markov}}_{\triangle t}$ are associated with quantization cells within the first partition element in $[t_i, t_{i+1}]$ and destination locations with the last. Let now G_{start,t_i} be the quantization cell of start location L_{start} and $G_{\lambda,t_{i+1}}$ the cells of all possible destination locations L_λ (with λ indexing cells and locations). The *reachable set* R_{start} is computed, which is as small as possible yet guaranteed to include all continuous states reachable from G_{start,t_i} within $[t_i, t_{i+1}]$. Now the probabilities for the transitions L_{start} to destina-

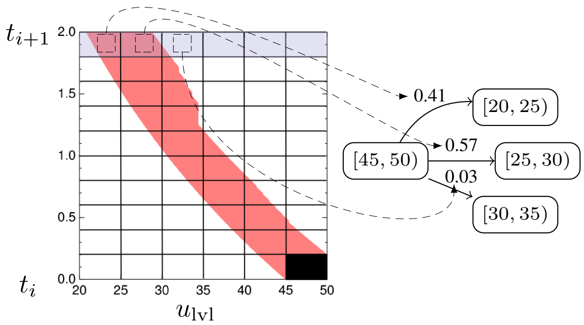

Figure 4: Reachable set R_{start} for $\dot{u}_{\mathrm{lvl}} = -\mathrm{fR} * u_{\mathrm{lvl}}$ starting from the marked grid cell G_{start,t_i}. Right: the derived PHCA $A^{\mathrm{Markov}}_{\triangle t}$.

tion locations L_λ are computed as

$$P(L_\lambda | t_{i+1}, L_{\mathrm{start}}) = \frac{V(G_{\lambda,t_{i+1}} \cap R_{\mathrm{start}})}{V(\bigcup_x G_{x,t_{i+1}} \cap R_{\mathrm{start}})},$$

where $V()$ measures the volume of the given set. The process is illustrated in figure 4.

The volume of the sets (which are polyhedrons) is computed using the vinci tool by (Büeler *et al.*, 2000), cutsets $(G_{\lambda,t_{i+1}} \cap R_{\mathrm{start}})$ are computed using the Parma Polyhedra Library (Bagnara *et al.*, 2002).

Currently we use PHAVer (Frehse, 2005) for reachability analysis, but different approaches can be employed. Regarding abstraction of hybrid models, we can build on a lot of related work in the area of automated verification of model properties. Stursberg et al. address the problem of online verification of properties such as that the planned path of a cognitive vehicle doesn't cross the path of another vehicle (M. Althoff *et al.*, 2007). In (M. Althoff *et al.*, 2007) they 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 (Hofbaur and Rienmüller, 2008). The method 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.

4.3 dfPHCAs as Conservative Abstraction

It remains to show that a dfPHCA $A_{\mathrm{df}}(\triangle t)$, generated as described above from a HyPHCA HA, is a conservative abstraction in terms of the probabilities of system trajectories, or formally:

Definition 6. (Set of abstracted HyPHCA trajectories) Let $\theta_{A_{\mathrm{df}}(\triangle t)}$ be a trajectory of $A_{\mathrm{df}}(\triangle t)$ with corresponding timepoint sequence $\langle t_i \rangle$, then $\chi(\theta_{A_{\mathrm{df}}(\triangle t)}) := \{\Delta | \forall t_i : (S_U^{(t_i)}, m^{(t_i)}) \in \Delta(\langle t_i \rangle) \wedge (\hat{S}_{\Pi_U}^{(t_i)}, \hat{m}^{(t_i)}) \in \theta_{A_{\mathrm{df}}(\triangle t)} \Rightarrow m^{(t_i)} = \hat{m}^{(t_i)} \wedge S_U^{(t_i)} \in G(\hat{S}_{\Pi_U}^{(t_i)})\}$[1] is the set of all HyPHCA trajectories contained in $\theta_{A_{\mathrm{df}}(\triangle t)}$.

Proposition 7. *Let $G : D_{\Pi_U} \to G_{\mathbb{R}^{|U|}}$ be a function that maps assignments to discretized continuous variables Π_U to grid cells $G_\lambda \in G_{\mathbb{R}^{|U|}}$. Let $\langle o_i, c_i \rangle$ an arbitrary finite sequence of observations $o_i \in D_{\Pi_{\mathrm{Obs}}}$ and commands $c_i \in D_{\Pi_{\mathrm{Cmd}}}$ and $\langle t_i \rangle$ the corresponding sequence of time points. Then, for a trajectory $\theta_{A_{\mathrm{df}}(\triangle t)}$ consistent with $\langle o_i, c_i \rangle$ (i.e. $P(\theta_{A_{\mathrm{df}}(\triangle t)} | \langle o_i, c_i \rangle) > 0$), the following holds:*

$$\forall \Delta \in \chi(\theta_{A_{\mathrm{df}}(\triangle t)}) :$$

$$f_{HA}(\Delta(\langle t_i \rangle) | \langle o_i, c_i \rangle) \leq P(\theta_{A_{\mathrm{df}}(\triangle t)} | \langle o_i, c_i \rangle)$$

$f_{HA}(\Delta(\langle t_i \rangle) | \langle o_i, c_i \rangle)$ is the density function of a distribution over discrete-time HyPHCA trajectories, conditioned on the sequence $\langle o_i, c_i \rangle$.

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/diagnosis 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) $\mathcal{R} = (X, D, C)$ (Schiex *et al.*, 1995) with transition probabilities as preferences by translating the discretized model to soft-constraints following our framework in (Mikaelian *et al.*, 2005). The diagnosis part of the problem is an instance of *maximal probability diagnosis* (Sachenbacher and Williams, 2004).

The translation unfolds a given PHCA A over a time window of N steps as follows: $X = \{X_1, ..., X_n\}$ 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 $F_j : D_{j1} \times \cdots \times D_{jm} \to [0, 1]$ 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

[1]The hat ˆ is used to differentiate the HyPHCA state $(S_U^{(t_i)}, m^{(t_i)})$ from the dfPHCA state $(\hat{S}_{\Pi_U}^{(t_i)}, \hat{m}^{(t_i)})$

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. A flow constraint is "active" if and only if its associated location is marked and not overridden by a transition guard. We encode this logic with hard constraints for each location and time step, which implement the formula $O_l^{(t_i)} = \text{FALSE} \wedge X_l^{(t_i)} = \text{MARKED} \Leftrightarrow X_{\mathcal{F}_d(l)}^{(t_i)} = \text{ACTIVE}$. The auxiliary variables $O_l^{(t_i)}$ with domain $\{\text{TRUE}, \text{FALSE}\}$ and $X_{\mathcal{F}_d(l)}^{(t_i)}$ with domain $\{\text{ACTIVE}, \text{INACTIVE}\}$ 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 the scope by $X_{\mathcal{F}_d(l)}^{(t_i)}$. If the flow is active ($X_{\mathcal{F}_d(l)}^{(t_i)} = \text{ACTIVE}$) we keep the former mappings, and map to one if the flow is inactive. In the latter case the discrete flow is not determined, since all possible transitions are allowed.

Of course the soft-constraint encoding of dfPHCAs leads to a certain overhead in terms of auxiliary variables and constraints, which however is linear in the model size. For a single time point, the PHCA encoding creates per location l one marking variable $X_l^{(t_i)}$ and one consistency variable for the location's behavior constraint $\mathcal{C}(l)$. Per transition, two variables encode whether the transition is enabled and whether its guard is satisfied or not (again for a single time point). Thus, the PHCA encoding creates an overhead of $O(2|\mathcal{T}| + 2|\Sigma|)$ auxiliary variables. The encoding of discrete flow constraints for dfPHCAs adds the two variables $O_l^{(t_i)}$ and $X_{\mathcal{F}_d(l)}^{(t_i)}$ for each discrete flow, yielding $O(2|\mathcal{T}| + 4|\Sigma|)$. Note however that this estimate is very conservative as it assumes that every location has a discrete flow. Typically only the components with dedicated continuous behavior will have discrete flows in their abstraction, the dfPHCA.

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[2], which requires another minor (offline) translation step: $\mathcal{R}$ must be translated to a Weighted Constraint Satisfaction Problem (WCSP), a widely used formalism in soft-constraint optimization.

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

Table 2: Runtime (mean time in sec.) for all scenarios and discretizations for u_{lvl} .

Online Runtime					
Toulbar2 config.	Discretisation	Scenario 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

6 RESULTS

We created COP instances with different discretizations for u_{lvl} (d2, d5, d10 and d25) for our example scenarios and some variations, and solved them using Toulbar2. We tried its default and a second, decomposition based configuration. The problem size was for all instances 843 variables and ≈ 920 constraints (the latter number varies with the different variations).

For scenario 1 with d10 table 1 shows the most probable system trajectory the solver deduced from the given observations and goals as variable assignments in bold face. The generated solution correctly identifies the motor-switch-fault and provides the necessary commands to reach the goal: $\text{repair} = \text{ON}$ for t_0, $\text{refill} = \text{ON}$ and $\text{waitRefill} = \text{ON}$ for t_1 and $\text{waitRefill} = \text{OFF}$ 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 trajectory found as most probable solution, grey arrows show possible transitions. It can be seen that in this scenario, the reasoner misses the fault sensor.stuck-on if the continuous variable is abstracted too coarsely (d5). We assume spurious solutions to be the culprit: The coarser the abstraction, the more probable become evolutions of u_{lvl} which in reality are very unlikely or impossible. With too coarse an abstraction (d5), the combination of the more likely motor-switch-fault and a spurious evolution of u_{lvl} with heightened probability becomes 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 2 shows the average online runtimes for all scenarios. The columns show results for scenario 1, its two variations 1.1 and 1.2, and 2. The variations are diagnose motor-switch-fault only (1.1) and nominal behavior (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. We ex-

Table 1: The monitoring/control results for our example scenario 1 (discretization with 10 partition elements of u_{lvl}). The rows show: Known sensor values (1 row), known commands (4 rows), marked locations for sensor and silo (2 rows) and finally the fill level (1 row). Table entries are variable values; bold values are derived automatically by our method.

Variable	Time step										
	Past							Present	Future		
	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}	$[45, 50)$	$[\mathbf{45}, \mathbf{50})$	$[\mathbf{25}, \mathbf{30})$	$[\mathbf{25}, \mathbf{30})$	$[\mathbf{10}, \mathbf{15})$	$[\mathbf{5}, \mathbf{10})$	$[\mathbf{0}, \mathbf{5})$	$[\mathbf{0}, \mathbf{5})$	$[\mathbf{0}, \mathbf{5})$	$[\mathbf{0}, \mathbf{5})$	$[10, 15)$ (goal)
Legend:	nom. → nominal; m-s-f.ne → motor-switch-fault.notEmpty; m-s-f.e → motor-switch-fault.empty										

pected the offline decomposition of the problem to lower online computation effort, but surprisingly, it had a negative effect in our scenario.

The runtimes of the three offline steps discretization, Markov chain generation and soft constraint encoding for d2, d5, d10 and d25 are 16.5, 39.0, 138.5 and 215.4 seconds. They show that the effort for hybrid model abstraction and encoding is considerable, even for such a small model. However, runtime is still within manageable bounds. Memory consumption might be a bigger issue (the offline steps for d25 consumed $\approx$ 300 MB), it remains for future experiments to show the limits of our method. The biggest portion of the resources are consumed by the Markov chain generation, which 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 subspace associated with the abstracted continuous flow.

Scalability of the Approach

Our intuition on the scalability of our approach is that it scales well as long as the number of components showing *different* continuous behavior is comparably small. The most expensive step is the generation of Markov chains to retrieve the discrete flows. Scalability can be improved, if unnecessary generation is avoided, i.e. by sharing the same abstraction among components with the same continuous behavior. Also, the expensive reachability analysis could be improved,e.g., by optimizing PHAVer parameters (or use a better tool). Finally, intelligent state space quantization (Hofbaur and Rienmüller, 2008) would reduce the number of quantization cells, resulting in fewer Markov chain states and thus a much less expensive abstraction step and smaller abstract models.

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 (constraint optimization, hidden markov model reasoning), fault diagnosis in hybrid systems (stochastic abstraction of continuous behavior), and hybrid systems verification (hybrid automata, reachability analysis) 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 online 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 (based upon (Mikaelian *et al.*, 2005)) to monitor systems over long time periods and verify our results on larger factory settings such as (Buss *et al.*, 2007). In this and in other settings, accurate model-based monitoring and control can only be achieved by considering both hybrid hardware and software behavior.

NOMENCLATURE

$A, A_{\text{cl}}(\triangle t), A_{\text{df}}(\triangle t), HA$ A PHCA, clocked PHCA, dfPHCA or HyPHCA.

$A^{\text{Markov}}_{\triangle t}$ Special clocked PHCA encoding a markov chain.

Δ Function mapping time to HyPHCA states.

Δ_x For $x = \text{df}, \text{cl}, \text{phca}$ the function mapping time to dfPHCA, clocked PHCA or PHCA states.

dfPHCA Probabilistic hierarchical constraint automata with discrete flow constraints.

$\mathcal{F}, \mathcal{F}_d$ Functions associating a continuous flow (HyPHCA)/discrete flow(dfPHCA) constraint with a location.

fR Model parameter encoding the fill rate of bottles being filled from the silo.

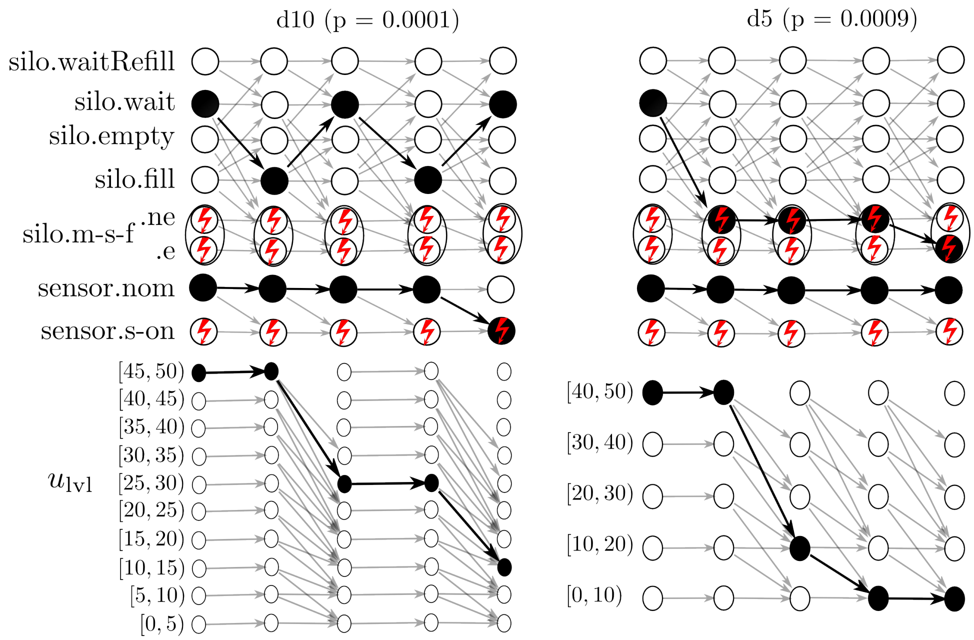

Figure 5: The inferred system trajectories (black filled circles and arrows) for the sensor-fault scenario as trellis diagram for d10 for u_{lvl} (left) and for d5 (right). Grey shaded arrows show possible transitions (probability > 0).

G_λ — A grid cell, a hyper cuboid in the state space $\mathbb{R}^n$ of a continuous flow involving n variables.

G_{λ,t_i} — A grid cell with time added, i.e. a hyper cuboid in the state space $\mathbb{R}^{n+1}$ with an additional dimension for time.

HyPHCA — Hybrid probabilistic hierarchical constraint automata, which support modelling of continuous behavior with linear ordinary differential equations.

Σ — The set of all locations of a *PHCA.

lvl — Finite domain variable representing the discretized fill level of the silo.

m-s-f.e — Primitive location empty within composite location motor-stuck-fault.

m-s-f.ne — Primitive location not-empty within composite location motor-stuck-fault.

m-s-f — Fault/Composite location motor-stuck-fault.

$m^{(t)}$ — PHCA state/marking.

ODE — Ordinary differential equation.

PHCA — Probabilistic hierarchical constraint automata.

Π_U — The set of discrete flow variables of a dfPHCA, generated from real-valued variables U of a HyPHCA. In the context of a constraint optimization problem $\mathcal{R} = (X, D, C)$, $\Pi_U^{(t_i)}$ is the set of variables in X representing the discretized flow at time t_i.

$\mathcal{R}$ — A constraint optimization problem.

$S^{(t_i)}$ — HyPHCA/dfPHCA state.

$S_x^{(t_i)}$ — For $x = U, \Pi_U$ an assignment to real-valued variables U (HyPHCA) or finite domain variables Π_U (dfPHCA).

θ_x — For $x = A, A_{\mathrm{cl}}(\triangle t), A_{\mathrm{df}}(\triangle t), HA$ a trajectory of a PHCA, clocked PHCA, dfPHCA or HyPHCA.

$\mathcal{T}$ — The set of all transitions of a *PHCA.

U — The set of real-valued variables of a HyPHCA.

u_{lvl} — Real-valued variable representing the fill level of the silo.

REFERENCES

(Alur *et al.*, 2006) Rajeev Alur, Radu Grosu, Insup Lee, and Oleg Sokolsky. Compositional modeling and refinement for hierarchical hybrid systems. *Journal of Logic and Algebraic Programming*, 68(1-2):105–128, 2006.

(Bagnara *et al.*, 2002) R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the Parma Polyhedra Library. Quaderno 286, Dipartimento di Matematica, Università di Parma, Italy, 2002.

(Benedetti *et al.*, 2008) Marco Benedetti, Arnaud Lallouet, and Jérémie Vautard. Quantified constraint optimization. In *Proc. CP-2008*, LNCS, pages 463–477, Sydney, Australia, September 2008. Springer.

(Bernadsky *et al.*, 2004) Mikhail Bernadsky, Raman Sharykin, and Rajeev Alur. Structured modeling of concurrent stochastic hybrid systems. In *Proc. FORMATS/FTRFT-2004*, pages 309–324, 2004.

(Blanke *et al.*, 2006) Mogens Blanke, Michel Kinnaert, Jan Lunze, Marcel Staroswiecki, and Jochen Schröder. *Diagnosis and Fault Tolerant Control*, chapter Chapter 9, Diagnosis and Reconfiguration of Quantized Systems, pages 447–504. Springer Verlag, 2nd edition, 2006.

(Bouveret *et al.*, 2004) S. Bouveret, F. Heras, S.de Givry, J. Larrosa, M. Sanchez, and T. Schiex. Toolbar: a state-of-the-art platform for wcsp. http://www.inra.fr/mia/T/degivry/ToolBar.pdf, 2004.

(Büeler *et al.*, 2000) Benno Büeler, Andreas Enge, and Komei Fukuda. *Polytopes — Combinatorics and computation*, chapter Exact volume computation for polytopes: a practical study, pages 131–154. Number 29 in DMV Seminar. Birkhäuser, 2000.

(Buss *et al.*, 2007) Martin Buss, Michael Beetz, and Dirk Wollherr. CoTeSys - Cognition for Technical Systems. In *Proc. COE Workshop on Human Adaptive Mechatronics (HAM)*, 2007.

(de Kleer *et al.*, 2009) J. de Kleer, L. Kuhn, J.J. Liu, R. Price, M. B. Do, and R. Zhou. Continuously Estimating Persistent and Intermittent Failure Probabilities. In *Proc. SAFE Process 2009*, Barcelona, Spain, June 2009.

(Dechter, 2003) Rina Dechter. *Constraint processing*, chapter 13. Morgan Kaufmann Publishers, San Francisco, CA 94104-3205, 2003.

(Dominka, 2007) Sierke Dominka. *Hybride inbetriebnahme von produktionsanlagen — Von der virtuellen zur realen inbetriebnahme*. PhD thesis, Technische Universität München, 2007.

(Frehse, 2005) Goran Frehse. Phaver: algorithmic verification of hybrid systems past hytech. In *Proc. HSCC-05*, pages 258–273, 2005.

(Henzinger, 1996) Thomas Henzinger. The theory of hybrid automata. In *Proc. LICS-1996*, pages 278–292, New Brunswick, New Jersey, 1996.

(Hofbaur and Rienmüller, 2008) Michael W. Hofbaur and Theresa Rienmüller. Qualitative Abstraction of Piecewise Affine Systems. In *Proc. QR-08 Workshop*, 2008.

(Hofbaur and Williams, 2002) Michael W. Hofbaur and Brian C. Williams. Mode estimation of probabilistic hybrid systems. In *In HSCC-02*, pages 253–266, Stanford, California, USA, 2002. Springer Verlag.

(Kleissl and Hofbaur, 2005) Wolfgang Kleissl and Michael Hofbaur. A Qualitative Model for Hybrid Control. In *Proc. QR-05 Workshop*, volume 19, pages 8–16, Graz, Austria, May 2005.

(Lunze and Nixdorf, 2001) Jan Lunze and Bernhard Nixdorf. Representation of hybrid systems by means of stochastic automata. *Mathematical and Computer Modelling of Dynamical Systems*, 7:383–422, December 2001.

(M. Althoff *et al.*, 2007) M. Althoff, O. Stursberg, and M. Buss. Online Verification of Cognitive Car Decisions. In *IEEE Intelligent Vehicles Symposium*, 2007.

(Mikaelian *et al.*, 2005) Tsoline Mikaelian, Brian C. Williams, and Martin Sachenbacher. Model-based Monitoring and Diagnosis of Systems with Software-Extended Behavior. In *Proc. AAAI-05*, 2005.

(Pedro Meseguer *et al.*, 2006) Pedro Meseguer, Francesca Rossi, and Thomas Schiex. *Handbook of constraint programming*, chapter 9. Soft Constraints. Elsevier, 2006.

(Sachenbacher and Williams, 2004) Martin Sachenbacher and Brian Williams. Diagnosis as semiring-based constraint optimization. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-04)*, Valencia, Spain, 2004.

(Schiex *et al.*, 1995) Thomas Schiex, Hélène Fargier, and Gerard Verfaillie. Valued constraint satisfaction problems: hard and easy problems. In *Proc. IJCAI-1995*, 1995.

(Williams *et al.*, 2001) Brian C. Williams, Seung Chung, and Vineet Gupta. Mode estimation of model-based programs: monitoring systems with complex behavior. In *Proc. IJCAI-01*, pages 579–590, 2001.

(Williams *et al.*, 2003) Brian C. Williams, Michel Ingham, Seung H. Chung, and Paul H. Elliott. 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.