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

Paul Maier and Martin Sachenbacher
Technische Universität München
Department of Informatics, Boltzmannstraße 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-
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 \( \dot{u}_{\text{fill}} = -\text{fR} + u_{\text{lvl}} \) (where \( \text{fR} \) 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 \( \Delta t = 2s \)), 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, 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 \textit{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-

---

**2 INDUSTRIAL FILLING STATION EXAMPLE**

---

**Figure 1: Filling station.**

---

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 \textit{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.
cations notEmpty and empty of m-s-f, s-on refers to stuck-on and nom. to nominal. Furthermore, $dz$ 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 hard-ware 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 $A = \langle \Sigma, P_\Theta, \Pi, C, P_T \rangle$, 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 un-marked. A marked location represents an active execution branch.

- $P_\Theta$ is a probability distribution over subsets $\Theta_i \subseteq \Sigma$, denoting the probability that $\Theta_i$ is the set of start locations.

- $\Pi = \Pi_D \cup \Pi_{Obs} \cup \Pi_{Cmd}$ is a set of dependent, observable and commandable variables, all having finite domains. $C[\Pi]$ denotes the set of finite domain constraints over $\Pi$.

- $C : \Sigma \rightarrow C[\Pi]$ associates with each location $l_i \in \Sigma$ a finite domain constraint $C(l_i)$.

- $P_T(l_i)$, for each $l_i \in \Sigma_p$, is a probability distribution over a set of transition functions $T(l_i) : \Sigma_p^{(t)} \times C[\Pi]^{(t)} \rightarrow 2^{\Sigma^{(t+1)}}$. 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 $m^{(t)} \subseteq \Sigma$. A sequence of such markings $\theta = \{m^{(0)}, m^{(1)}, \ldots, m^{(t+N)}\}$ 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 $\ddot{u} = Au + b$, describes the time-continuous evolution of a vector of variables $u = [u_1, \ldots, u_n]^T$ as a set of equations over $u$ and their...
A HyPHCA is a tuple \( HA = \langle \Sigma, P_\theta, \Pi, U, C, F, P_T \rangle \) where

- \( U = U \cup \widehat{U} \cup U' \) 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.
- \( C : \Sigma \rightarrow T[\Pi \cup U \cup U'] \) is a function associating locations with constraints over discrete and/or real-valued variables. \( T[\Pi \cup U \cup U'] \) denotes the set of constraints over \( \Pi \cup U \cup U' \).
- \( F : \Sigma \rightarrow F[U \cup \widehat{U}] \) is a function associating locations with constraints over real-valued variables and their derivatives in the form of differential equations. \( F[U \cup \widehat{U}] \) denotes the set of these differential equations.
- \( P_T \) is a probability distribution over a set of transition functions \( T(l_i) : \Sigma \times C[\Pi \cup U \cup U'] \rightarrow 2^\Sigma \) for locations \( l_i \in \Sigma \). Each transition function \( T(l_i) \) maps a primitive location marked at time \( t \) to the set of locations to be marked at the next time instant, given the location’s guard is entailed.
- \( \Sigma, P_\theta \) and \( \Pi \) are analog to the PHCA definition.

**Definition 3.** A HyPHCA is a tuple \( HA = \langle \Sigma, P_\theta, \Pi, U, C, F, P_T \rangle \) where

\[
\begin{align*}
\dot{U} &= \{\dot{u}_1, \ldots, \dot{u}_n\}^T, \\
P_\theta &= \{b_1, \ldots, b_n\}^T
\end{align*}
\]

is a vector of constants and \( A \) the \( n \times n \)-matrix of coefficients for the equation set.

**Definition 3.** A HyPHCA is a tuple \( HA = \langle \Sigma, P_\theta, \Pi, U, C, F, P_T \rangle \) where

- \( U = U \cup \widehat{U} \cup U' \) 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.
- \( C : \Sigma \rightarrow T[\Pi \cup U \cup U'] \) is a function associating locations with constraints over discrete and/or real-valued variables. \( T[\Pi \cup U \cup U'] \) denotes the set of constraints over \( \Pi \cup U \cup U' \).
- \( F : \Sigma \rightarrow F[U \cup \widehat{U}] \) is a function associating locations with constraints over real-valued variables and their derivatives in the form of linear ordinary differential equations. \( F[U \cup \widehat{U}] \) denotes the set of these differential equations.
- \( P_T \) is a probability distribution over a set of transition functions \( T(l_i) : \Sigma \times C[\Pi \cup U \cup U'] \rightarrow 2^\Sigma \) for locations \( l_i \in \Sigma \). Each transition function \( T(l_i) \) maps a primitive location marked at time \( t \) to the set of locations to be marked at the next time instant, given the location’s guard is entailed.
- \( \Sigma, P_\theta \) and \( \Pi \) are analog to the PHCA definition.

**Definition 4.** (HyPHCA state, HyPHCA trajectory)

The state of a HyPHCA at time \( t \) is a tuple \( S(t_i) = (S(t_i), m(t_i)) \), where \( S(t_i) \in \mathbb{R}^{|U|} \) 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 \( \Delta : \mathbb{R} \rightarrow \mathbb{R}^{|U|} \times \mathcal{M} \), mapping time points (real-valued) to HyPHCA states, is called a HyPHCA trajectory function. A finite sequence \( \theta_{HA} = \Delta(t_i) \), resulting from evaluating \( \Delta \) 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 (dPHCA). A dPHCA is a tuple \( A_{dPHCA} = (\Sigma, P_\theta, \Pi, \Pi_T, C, F_d, P_T, \Delta) \) parameterized with fixed-length time interval \( \Delta t \) where \( \Pi_T = \Pi_U \cup \Pi_U' \) is analogous to \( U \) of a HyPHCA, except that derivatives are omitted and variables have finite domains now. \( F_d : \Sigma \rightarrow F_d[\Pi_U \cup \Pi_U'] \) is the discrete flow, a function associating locations with constraints encoding Markov chains over the discrete flow variables of the location. \( C \) is defined as for HyPHCAs with real-valued variable sets \( U \) and \( U' \) replaced by \( \Pi_U \) and \( \Pi_U' \). The rest is analog to the PHCA definition. The state of \( A_{dPHCA}(\Delta t) \) at time \( t \) is a tuple \( S(t) = (S(t), m(t)) \), where \( S(t) \) is an assignment of values to discretized continuous variables \( x \in U \) at time \( t \) and \( m(t) \) a marking analogous to PHCA states.

A function \( \Delta_{df} : \{t_i\} \rightarrow \mathbb{N} \times \mathcal{M} \), mapping the infinite set of real-valued time points \( \{t_i\} := \{t_i \mid i \in \mathbb{N} : \Delta t = t_i - t_{i+1}\} \) to dPHCA states, is called a dPHCA trajectory function. Evaluating \( \Delta_{df} \) for a finite subset of \( \{t_i\} \) yields a finite sequence of dPHCA states \( \theta = \{S(t), S(t^{(i+1)}), \ldots, S(t^{(i+n)})\} \), called a dPHCA trajectory.

In order to bridge the gap from dPHCAs to PHCAs, we define clocked PHCAs as dPHCAs (also parameterized with \( \Delta t \)) with discrete flows and discrete flow variables omitted. A clocked PHCA trajectory is consequently a function \( \Delta_{cl} : \{t_i\} \rightarrow \mathcal{M} \) 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 \( \Delta_{phca} : \mathbb{N} \rightarrow \mathcal{M} \) maps natural numbers to markings, rather than real-valued time points.

To avoid confusion when referring to trajectories, we write \( \theta_x \) with \( x = A, A_{dPHCA}(\Delta t), A_{dPHCA}(\Delta t), HA \) for PHCA, clocked PHCA, dPHCA and HyPHCA trajectories.

**4 FROM HYBRID TO ABSTRACT DISCRETE MODELS**

We will see that discrete flow constraints encode special case PHCAs and that a dPHCA can thus be turned into an equivalent clocked PHCA. The discrete flows then form sub-PHCAs embedded into composite locations. So intuitively, a discrete abstraction of a HyPHCA is obtained by abstracting continuous flows to discrete flows of a dPHCA, then converting the dPHCA 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 \( F_d(l) \) 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 \( F_d(l) \) is marked at \( t_i \), \( F_d(l) \) 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 dPHCAs. 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 dPHCAs, if time intervals are abstracted, like PHCAs encode HMMs. This can be done by showing that an arbitrary dPHCA has an equivalent clocked PHCA (and thus a PHCA, after time abstraction).

Proposition 5. (Equivalence dPHCA, clocked PHCA) Let \(\langle \alpha_i, \epsilon_i \rangle\) be an arbitrary sequence of observations and commands. Then for an arbitrary dPHCA \(A_{df}(\Delta t)\) exists an equivalent clocked PHCA \(A_{cl}(\Delta t)\), such that

\[
P(\theta_A(\Delta t)) | (\alpha_i, \epsilon_i) = P(\theta_{A_{cl}}(\Delta t)) | (\alpha_i, \epsilon_i)
\]

The function \(\rho : D_{df} \times M_{df} \rightarrow M_{cl}\) maps (sequences of) dPHCA states to (sequences of) clocked PHCA states, specifically discrete flow variable assignments to markings of sub-PHCAs which encode the discrete flow. \(P(\rho(\theta_A(\Delta t))) | (\alpha_i, \epsilon_i))\) and \(P(\theta_{A_{cl}}(\Delta t)) | (\alpha_i, \epsilon_i))\) are the probabilities of a clocked and discrete flow PHCA trajectory occurring, respectively, given the sequence \(\langle \alpha_i, \epsilon_i \rangle\).

We now describe the conversion of HyPHCAs to dPHCAs (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 \(HA = \langle \Sigma, P_H, \Pi, U, C, F, P_T \rangle\) be a HyPHCA. The set \(T\) denotes all transitions \(T\) defined through \(P_T\), source\((T)\) and guard\((T)\) are a transition’s source, destination set and guard constraint, respectively. \(G_{R[U]} = \{G_\lambda\}\) is a set of disjoint grid cells (also called quantization cells) partitioning the continuous state space of \(HA: \bigcup G_\lambda = \mathbb{R}^{|U|}\). Let \(A_{df}(\Delta t) = \langle \Sigma, P_H, \Pi_{df}, C, F_{df}, P_T, \Delta t \rangle\) be the dPHCA with discrete flow constraints generated from HA. In the following, we refer to elements of the respective automaton, like \(\Sigma, HA, C, \Pi, P_H\) and \(A, P_T\) (where we abbreviate \(A_{df}(\Delta t)\) 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 straightforward: \(A, C = HA, C, P_H = HA, P_H, A, \Pi = HA, \Pi\) and \(A, P_T = HA, P_T\). The discretized counterparts to \(U, \Pi, \Pi_U, \Pi_{U^r}\) form \(\Pi_{df}\) (discrete versions of the derivatives \(U\) are not needed and thus omitted): \(\Pi_{df} = \Pi_U \cup \Pi_{U^r}\). 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: \(\text{HA, C}[HA, \Pi U U'] = \text{HA, C}[HA, \Pi U U']\). The finite domain constraints of \(A_{df}(\Delta t)\) are accordingly \(A_{df}[A, \Pi U U'] = \text{HA, C}[HA, \Pi U U']\). The function \(\text{conv}\) maps simple arithmetic constraints such as \(u \leq 1\) or \(u_1 \geq u_2\) to corresponding finite domain variables.

The finite domains of discretized variables \(\Pi_{df}, \Pi_{U^r}\) are derived from the quantization \(G_{R[U]}\). The grid cells \(G_\lambda \in G_{R[U]}\) 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 \(U_{df}\) and \(U'_{df}\). That is, the values of, e.g., a variable \(x_u \in U_{df}\) 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 \(\mathcal{F}_{df}(L)\). 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 \(A_{Markov}^{\Delta t}\), encoded in \(\mathcal{F}_{df}(L)\). It has only primitive locations, corresponding to grid cells of \(G_{R[U]}\), and represents a Markov chain that conservatively approximates the continuous evolution. The discrete flow constraint encodes \(A_{Markov}^{\Delta t}\) by directly relating variables \(x_{u} \in A, U_{df}\) for two time points \(t_i\) and \(t_{i+1}\). \(\mathcal{F}_{df}(L)\) is added to the corresponding location \(L \in A, \Sigma\). If \(\mathcal{F}_{df}(L)\) conflicts with transition guards determining variable values for \(t_{i+1}\) via \(x_{u'} \in A, U_{df}\), the guard takes precedence over \(\mathcal{F}_{df}(L)\) (see, e.g., figure 3).

![Figure 3: HyPHCA (above) is converted to a dPHCA (below).](image-url)

### 4.2 Discrete Abstraction of Continuous Flow

To conservatively estimate transition probabilities of \(A_{Markov}^{\Delta t}\) 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 \(A_{Markov}^{\Delta t}\) are associated with quantization cells within the first partition element in \([t_i, t_{i+1}]\) and destination locations with the last. Let \(G_{start, t_i}\) be the quantization cell of start location \(L_{start}\) and \(G_{start, t_{i+1}}\) the cells of all possible destination locations \(L_{\lambda}\) (with \(\lambda\) 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 $u_{lvl} = -\mathbb{R} * u_{lvl}$ starting from the marked grid cell $G_{start,t_i}$. Right: the derived PHCA $A_{\Delta t}^{Markov}$.

4.3 dPHCA as Conservative Abstraction

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

**Definition 6.** (Set of abstracted PHHyCA trajectories) Let $\theta_{df}(\Delta t)$ be a trajectory of $A_{df}(\Delta t)$ with corresponding timepoint sequence $\langle t_i \rangle$, then $\chi(\theta_{df}(\Delta t)) := \{ \Delta \forall t_i : (S_{ih}(t_i), m(t_i)) \in \Delta(\langle t_i \rangle) \wedge (\hat{S}_{ih}(t_i), \hat{m}(t_i)) \in \theta_{df}(\Delta t) \}^1$ is the set of all HyPHCA trajectories contained in $\theta_{df}(\Delta t)$.

**Proposition 7.** Let $G : D_{U_i} \to G_{R(\in V)}$ be a function that maps assignments to discretized continuous variables $\Pi_{U_i}$ to grid cells $G_\in 

Let $f_{HA}(\Delta(t_i)) \leq P(\theta_{df}(\Delta t)) | (\langle o_i, c_i \rangle)$ be 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 $S(t_{i+1})$, 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 $S(t_{i+1})$. This trajectory is the goal achieving commands can be easily derived. We frame this problem as a discrete constraint optimization problem (DOP) $R = (X, D, C)$ (Schie, 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 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 $D = \{ D_1, ..., D_n \}$. For all time points $t_i, i = 0..N$, it consists of $\Pi_U \subseteq X$ encoding PHCA variables, auxiliary variables (needed to, e.g., encode hierarchical structure) and the solution variables of the COP, a set of binary variables $Y = \{ X_1, X_2 \} \subseteq X$ representing location markings of $A$. $C = \{ C_1, ..., C_n \}$ is a set of constraints $(S_i, F_i)$ with scope $S_i = \{ X_{j_1}, ..., X_{j_m} \} \subseteq X$ and a constraint function $F_i : D_{j_1} \times \cdots \times D_{j_m} \rightarrow [0, 1]$ mapping partial assignments of variables in $S_i$ to a probability value in $[0, 1]$. For all time steps $t_i, i = 0..N$, hard 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. 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 $O_l(t_i) = \text{FALSE} \land X_{d_l}(t_i) = \text{MARKED} \iff X_{d_l}(t_i) = \text{ACTIVE}$. The auxiliary variables $O_l(t)$ with domain \{TRUE, FALSE\} and $X_{d_l}(t)$ with domain \{ACTIVE, INACTIVE\} indicate an override of a discrete flow and its activation, respectively. The discrete flow itself is a function mapping discrete flow variables in $\Pi_{l,i}^{(t_i)}$ and $\Pi_{l,i+1}^{(t_i+1)}$ to transition probabilities. Again for each location and time point we encode these functions as soft constraints, extending the scope by $X_{d_l}(t_i)$. If the flow is active ($X_{d_l}(t_i) = \text{ACTIVE}$) 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 $X_{d_l}(t)$ and one consistency variable for the location’s behavior constraint $C(l)$. 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 $O(2|T| + 2|\Sigma|)$ auxiliary variables. The encoding of discrete flow constraints for dfPHCAs adds the two variables $O_l(t)$ and $X_{d_l}(t)$ for each discrete flow, yielding $O(2|T| + 4|\Sigma|)$. 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/

### 6 RESULTS

We created COP instances with different discretizations for $u_{vl}$ (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 $\approx 920$ 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: 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 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 abtracted too coarsely (d5). We assume spurious solutions to be the culprit: The coarser the abstraction, the more probable become evolutions of $u_{vl}$ 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 $u_{vl}$ 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 2: Runtime (mean time in sec.) for all scenarios and discretizations for $u_{vl}$.**

<table>
<thead>
<tr>
<th>Toolbar config</th>
<th>Discretization</th>
<th>Online Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Scen 1</td>
</tr>
<tr>
<td>default</td>
<td>d2</td>
<td>0.016s</td>
</tr>
<tr>
<td></td>
<td>d5</td>
<td>0.007s</td>
</tr>
<tr>
<td>params</td>
<td>d10</td>
<td>0.014s</td>
</tr>
<tr>
<td></td>
<td>d25</td>
<td>0.030s</td>
</tr>
<tr>
<td>with</td>
<td>d2</td>
<td>0.118s</td>
</tr>
<tr>
<td></td>
<td>d5</td>
<td>0.122s</td>
</tr>
<tr>
<td>tree</td>
<td>d10</td>
<td>0.138s</td>
</tr>
<tr>
<td>decomp.</td>
<td>d25</td>
<td>0.138s</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Time step</th>
<th>Present</th>
<th>Future</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( t_{-7} )</td>
<td>( t_{-6} )</td>
<td>( t_{-5} )</td>
</tr>
<tr>
<td>sensor.out</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>silo.motor</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>silo.repair</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>silo.waitRefill</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>silo.refill</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>silo.location</td>
<td>wait</td>
<td>fill</td>
<td>wait</td>
</tr>
</tbody>
</table>

Legend: nom. → nominal; m-s-f.ne → motor-switch-fault.notEmpty; m-s-f.e → motor-switch-fault.empty

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 addition, in other settings, accurate model-based monitoring and control can only be achieved by considering both hybrid hardware and software behavior.
Figure 5: The inferred system trajectories (black filled circles and arrows) for the sensor-fault scenario as trellis diagram for d10 for $u_{lv1}$ (left) and for d5 (right). Grey shaded arrows show possible transitions (probability $> 0$).

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

$G_{\lambda,t_i}$ 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.

$\Sigma$ The set of all locations of a *PHCA.

$lv1$ 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.

$m_i(t)$ PHCA state/marking.

ODE Ordinary differential equation.

PHCA Probabilistic hierarchical constraint automata.

$\Pi_U$ 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 $R = (X, D, C)$, $\Pi_{U(t_i)}$ is the set of variables in $X$ representing the discretized flow at time $t_i$.

$R$ A constraint optimization problem.

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

$\theta_x$ For $x = A, A_{\text{cl}}(\Delta t), A_{\Delta t}(\Delta t), H_A$ a trajectory of a PHCA, clocked PHCA, dfPHCA or HyPHCA.

$T$ The set of all transitions of a *PHCA.

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

$u_{lv1}$ Real-valued variable representing the fill level of the silo.

REFERENCES


