# Automated Plan Assessment in Cognitive Manufacturing

Paul Maier*,[a], Martin Sachenbacher[a], Thomas Rühr[a], Lukas Kuhn[b]

[a] *Technische Universität München, Boltzmannstraße 3, 85748 Garching, Germany*
[b] *PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA*

**Abstract**

In a cognitive factory setting, product manufacturing is automatically planned and scheduled, exploiting a knowledge base that describes component capabilities and behaviors of the factory. However, because planning and scheduling are computationally hard, they must typically be done offline using a simplified system model, and are thus unaware of online observations and potential component faults. This leads to a problem: Given behavior models and online observations of possibly faulty behavior, how likely is each manufacturing process plan to still succeed? In this work, we first formalize this problem in the context of probabilistic reasoning as *plan assessment*. Then we contribute a solution which computes plan success probabilities based on most likely system behaviors retrieved from solving a constraint optimization problem. The constraint optimization problem is solved using well-optimized off-the-shelf solvers. Results obtained with a prototype show that our method can guide systems away from plans which rely on suspect components.

*Key words:*
model-based reasoning, diagnosis, probabilistic reasoning, planning, constraint optimization

## 1. Introduction

As the market demands for customized and variant-rich products, the industry struggles to implement production systems that demonstrate the necessary flexibility while maintaining cost efficiency comparable to highly automated mass production. A main cost driver in automated production is the human workforce needed for setup steps, the development of processes, and quality assurance. These high labor costs can only be amortized by very large lot sizes. For small lot sizes as found in prototype and highly customized production, human workers are still unchallenged in flexibility and cost. Therefore, to facilitate the emergence of mass customization at prices only highly automated systems can achieve, levels of flexibility similar to the flexibility of human workers must be reached.

Future technical systems are expected to act robustly under high uncertainty, reliably handle unexpected events, quickly adapt to changing tasks and own capabilities. A key technology for the realization of such systems is automated planning combined with self-diagnosis and self-assessment. These capabilities can allow the system to plan its own actions, and also react to failures and adapt the behavior to changing circumstances. Cognitive architectures try to achieve such capabilities for industrial applications by implementing solutions inspired from human and animal cognitive behavior [1]. Research within the German cluster "Cognition for Technical Systems" (CoTeSys) [2] tries to understand human cognition to make its performance accessible for technical systems.

In a scenario of cognitive manufacturing, a factory generates the manufacturing process plans for numerous individualized products during night for manufacturing the next day. A factory knowledge base, describing component capabilities and behavior, serves as model basis for the factory's intelligent capabilities such as planning. During night enough time is available to generate complex plans. Still,

*Corresponding author. Tel.: ++49 89 289 19595
*Email addresses:* maierpa@in.tum.de (Paul Maier), sachenba@in.tum.de (Martin Sachenbacher), ruehr@in.tum.de (Thomas Rühr), Lukas.Kuhn@parc.com (Lukas Kuhn)

relevant parts of the knowledge have to be selected [3] from the knowledge base, as planning/scheduling on the whole knowledge base would be intractable. The question is: can it be guaranteed that the plan works given the behavioral knowledge of the system?

Planning and scheduling finishes at a deadline the next day (e.g. 8 am). However, partial observations can be made after that deadline, especially during execution of the plans. In the light of this new information, it might become clear that success for certain plans cannot be guaranteed anymore (e.g., if a plan operates a component intensely which has been observed to be prone to failure).

The two problems illustrated above lead to the same problem of evaluating manufacturing process plans in the face of information that was not available or not used for their generation. In the first case, a sparse model without behavior knowledge was used due to problem hardness. In the second case, it's observations which where not available at planning/scheduling time. In both cases, planning and scheduling are complex tasks (even on the sparse model), which prohibit quick reformulation of whole plans, only slight modifications are possible.

In this work, we are interested in the probability of plan success, i.e. that it achieves its goal, or plan failure. We want to provide a criterion upon which an AI decision component or a human operator can decide to a) continue with a plan, b) stop it because it probably won't succeed or c) gather more information. We call the problem of computing this probability the *Plan Assessment Problem*. In this work, we do not address planning and/or scheduling problems. We assume a given cognitive architecture which provides typical AI capabilities such as planning [4]. This work is based on prior work presented in [5, 6]. It combines concepts from these works and extends them by a) a formal definition of the plan assessment problem as a probabilistic reasoning problem within the domain of cognitive manufacturing and b) elaborating ideas on how focussed plan assessment models which model behavior of multiple, different products can be created.

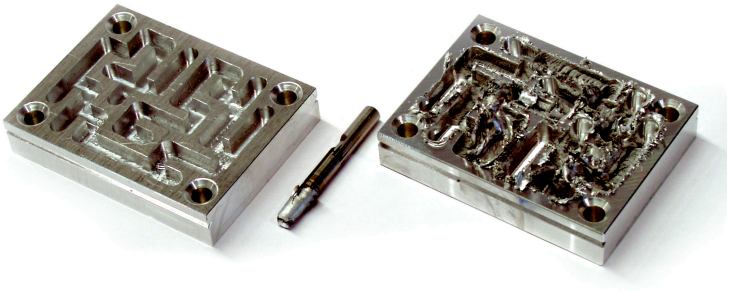
The rest of our article is organized as follows: In the next section, we introduce our example scenario for the CoTeSys cognitive factory. We then precisely analyse all aspects of the plan assessment problem in section 3 and discuss related work in section 4. In section 5 we describe in detail the modeling formalisms used to create planning and plan assessment models. Plan assessment with *hybrid* discrete/continuous models based on hybrid automata is shortly explained in section 6. Then in section 7 we show how belief state approximations can be computed and introduce our approach to estimating the success probability using soft-constraint optimization. Section 8 is concerned with our restricted prototype implementation of plan assessment, followed by the results obtained with it.

## 2. Metal Machining and Assembly Example

Part of the CoTeSys cognitive factory test-bed is a customized and extended Flexible Manufacturing System (FMS) based on the iCim3000 from Festo AG (see figure 1b). The system consists of conveyor transports and three stations: storage, machining (milling and turning), and assembly.

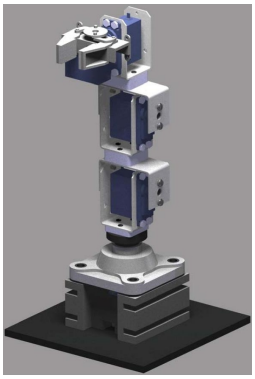
The following scenario will serve as basis for examples throughout the article. In the cognitive factory, a planner creates plans for a toy maze and a toy robot arm. The maze consists of an alloy base plate and an acrylic glass cover fixed by metal pins (see figure 1a), the robot arm (see figure 1c) consists of alloy parts, joints and servos. The robot is configurable regarding the number of joints and their orientation as well as in the choice of a manipulator. A single joint consists of two metal brackets and a servo motor. In the example laid out, some CNC (Computerized Numerical Control) cutting operations are done on each of the brackets, later sets of two brackets are assembled with a servo. Cabling is not considered in this scenario and has to be done manually as a last step. A scheduler assigns the necessary resources. The two product plans, i.e. two sequences of (action, time)-pairs, look like shown in figure 4. A rough visualization of the complete schedule is shown in figure 3.

Errors can be detected in the plant using a vibration sensor at suspicious components. In our situation, the machining station is suspicious, because its cutter can go blunt during operation. A blunt cutter is very likely to break, leading to flawed products (see figure 1a). However, not every vibration means that a component is faulty. Some components generate random vibrations, e.g., the assembly station. Furthermore, with some probability, vibrations in one station can trigger signals in sensors of nearby stations. In our example, the vibrations of the assembly station and conveyor belts can trigger sensor signals of the machining station sensor. The plan assessment must be able to cope



(a)

(b)



(c)

Figure 1: (1a) Effects of cutter deterioration until breakage in machining. (1b) The hardware setup used for experimentation, showing storage, transport, robot and machining components. (1c) The robotic arm product. Images © Prof. Shea TUM PE.

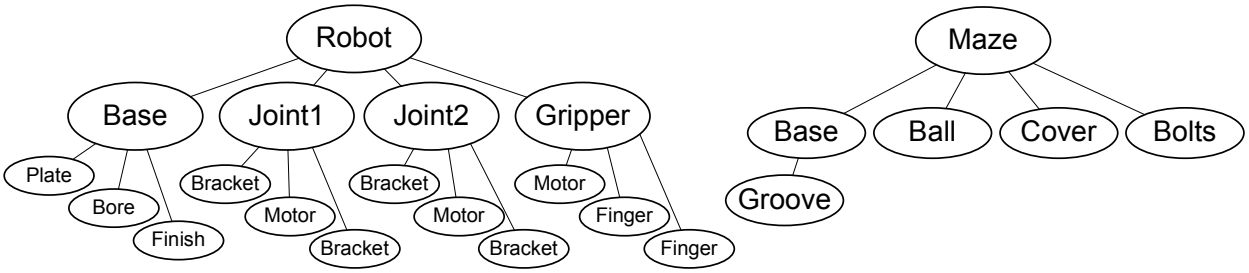



Figure 2: An example product description depicting the graph of a toy maze and a robot.

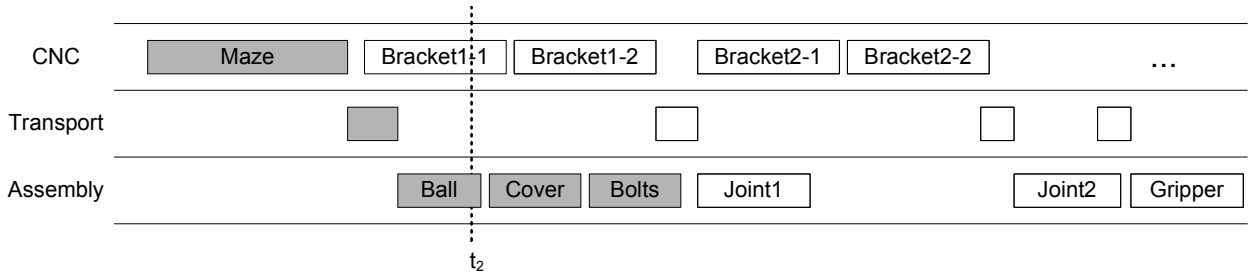



Figure 3: A visualisation of the manufacturing schedule of a maze (dark) and a robot (bright).

```
⟨ ... , (ROBOT–PLACE–CNC CNC MAZE CNC1, 310s),
 (CUT GROOVE1 MAZE CNC1, 320s),
 (ROBOT–PICK–CNC CNC MAZE CNC1, 870s), ... ⟩

⟨ ... , (ROBOT–PLACE–ASSEMBLY ASSY ROBOT–ARM–PART1 ASM1, 510s),
(ASSEMBLE–FROM SERVO1 ROBOT–ARM–PART1 ASSY FEED1 ASM1, 515s),
(ASSEMBLE–FROM ROBOT–ARM–PART2 ROBOT–ARM–PART1 ASSY FEED2 ASM1, 530s), ... ⟩
```

Figure 4: Excerpt of the product plans showing the actions with their parameters and durations.

with these kinds of ambiguities. Our approach can deal with them by finding *most probable explanations* for what happened in the past.

A vibration is detected at $t_2 = 520s$, while the machining station is cutting a bracket for the robot arm and the maze is being assembled (as can be seen from the schedule in figure 3). The question is: Is the vibration an indicator for a blunt cutter, and how does this possibility affect the plans?

## 3. Plan Assessment with Predicted Belief States

Now we detail the plan assessment problem and define necessary entities.

**Plan Assessment Problem** Given a model $M_{\text{assess}}$, the problem of plan assessment is to compute good lower and upper bounds $p_l$ and $p_u$ on the success probability $Pr(\mathcal{G}|o_{0:t}, M_{\text{assess}})$ for each plan $\mathcal{P}$ :

$$p_l \leq Pr(\mathcal{G}|o_{0:t}, M_{\text{assess}}) \leq p_u$$

"Good" means that $p_l$ is as big and $p_u$ as small as possible.

$M_{\text{assess}}$ is the underlying model for plan assessment, which encodes the plant component behavior and possible observations *caused by concurrently executing a set of plans* $\{\mathcal{P}_i\}$. It is a probabilistic model as known from probabilistic reasoning [7] which defines a distribution $Pr(X_{t+1}|X_t)$ over possible state transitions given the current plant state and a distribution $Pr(O_t|X_t)$ over possible observations given the current state. $\mathcal{G}$ represents the event that the plan $\mathcal{P}$ is executed successfully. The probability $Pr(\mathcal{G}|o_{0:t}, M_{\text{assess}})$ is the mentioned probability of plan success. It is conditioned on the observations $o_{0:t}$ obtained up to now, and the plan assessment model.

### *3.1. System Models and Knowledge Base*

The probabilistic model $M_{\text{assess}}$ models the behavior of plant components used by the plans to be assessed, omitting all other components. The modeled behavior can incorporate probabilistic failures as well as continuous behavior, such as wear of cutting tools. The plans are generated based on a planning model $M_{\text{plan}}$. It is focussed on the planning and scheduling task, i.e. it contains descriptions of component and their capabilities, leaving out details like breakdown probabilities and continuous dynamics.

Both models are derived from a comprehensive knowledge base that describes all components, capabilities and behaviors in the plant as well as products, product parts and resources. The three models are connected as shown in figure 5. The planning model is derived from the knowledge base using knowledge selection [3]. The assessment model is created using the generated manufacturing sched-

ule. We will later explain this process in more detail.

In the following, we will refer with $M$ to either an arbitrary model or the complete knowledge base, with $M_{\text{plan}}$ to the planning model and with $M_{\text{assess}}$ to the assessment model.

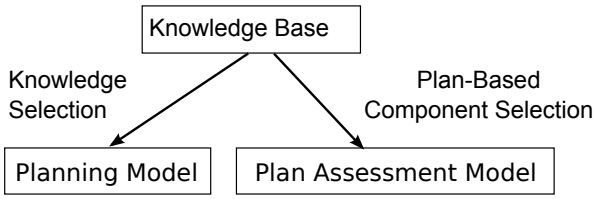



Figure 5: The models for planning and for plan assessment are derived from a common knowledge base.

A system model, be it for planning or plan assessment, is a symbolic description of system components, products, parts etc. which generates a *System State Space*.

**Definition 1. System State Space** Given a system model $M$, $\mathcal{S}(M)$ forms its state space.

If $M$ is a logical model then $\mathcal{S}(M)$ is the set of all possible configurations of true and false facts. I.e. a single state $s$ is a specific configuration of true and false facts. If $M$ uses a set of variables $X$ to describe a system, then $\mathcal{S}(M)$ is the set of all possible assignment vectors for $X$. System states are atomic, i.e. assuming a correct model, a system state fully determines the modeled system.

The term "state" is somewhat ambiguous. We use automata models which also have states, but in our case these states can describe whole sets of system states. In other words, automata states are not atomic. Therefore, we'll talk about automata *locations* and transitions between them, and reserve the term "state" for atomic system states.

### *3.2. Product Models and Plan Goals*

Assessing a manufacturing process plan naturally requires that the factory's state during or after plan execution is checked against the plan's goal. Goal specifications suitable for plan assessment can be derived from product models. Such models are predominant in product development, where recent research focusses, e.g., on model-based collaborative product design [8].

In manufacturing, the goal of plans is to manufacture products according to a given specification, typically a mixture of CAD/CAM (Computer Aided Design/Computer Aided Manufacturing) and CAPP (Computer Aided Process Planning) models. However, these types of models are too detailed for our purposes, e.g. they often contain geometric descriptions of parts. We focus on basic product parts and features, or compositions thereof, in an abstract product description $P = \langle V, E, C \rangle$. $V, E$ are the vertices and edges of a graph and $C$ set of first-order logic constraints. The vertices $V$ describe the basic product parts, features and compositions. The edges $E$ are "part-of" relations. The constraints $C$ enforce orderings on subsets of the edges. The product parts, features and compositions are types defined in the knowledge base $M$. The description is similar to notions introduced in the literature, such as [9], and could also be seen as the Bill of Materials of the product. See figure 2 for examples of product descriptions.

Given an abstract product description $P$ and a model $M_{\text{plan}}$, the goal for a planner is to find an optimal sequence of actions which realize the "part-of" relations in $P$. Part of this task is to identify entities in $M_{\text{plan}}$ which describe concrete parts, features and compositions. The constraints $C$ in $P$ establish precedence relations among actions which guide the planner.

Note that there's a trade-off here between what is specified in the product description and what needs to be planned. Theoretically, the edges of a product description graph could specify directly the necessary actions to achieve a part-of relation. In this case the description would already be the final plan.

We now formalize the notions of plan, plan goal and plan success.

**Definition 2. Plan, Plan Goal, Plan Success** Let $P$ be a product specification.

1. A plan $\mathcal{P}$, generated from $P$, is a sequence of tuples of actions and their associated start times $(a, t) \in A \times \mathbb{T}$. Start times can be real numbers indicating absolute time points or integers indicating time point indices, i.e. $\mathbb{T} = \mathbb{R}$ or $\mathbb{T} = \mathbb{N}$.
2. Associated with a plan $\mathcal{P}$ is the goal $G$ it should achieve. $G$ defines the reference point used for assessing the plan's success probability. $G$ is simply a vector of binary variables which represent relevant product features, and is derived from the product specification $P$.

3. A plan $\mathcal{P}$ is *successful* if it entails $G$, i.e. if $G$ is consistent with all system states possible after the execution of $\mathcal{P}$.

An important question is how the goal $G$ is derived from the product description $P$. A possibility is to introduce a binary variable for each vertex in the graph of $P$. These variables then have to be linked to the plant model used for assessment, $M_{\text{assess}}$. We haven't worked out a complete method yet. However, in section 5.3 we sketch a method to create $M_{\text{assess}}$ from $M$ and a given schedule, which also addresses the problem of representing manufacturing goals.

### *3.3. Plan Assessment with Predicted Belief States*

We evaluate plans by computing bounds on the plan's success probability $Pr(\mathcal{G}|o_{0:t}, M_{\text{assess}})$. This can be done by summing over the probabilities of goal-achieving states, based on the conditional distribution over all possible system states at the future time point $t+n$ the goal is (potentially) achieved. This distribution is the *belief state* at $t+n$, and the problem of computing it is the well known prediction problem in probabilistic reasoning [7]. In fact, we define $Pr(\mathcal{G}|o_{0:t}, M_{\text{assess}})$ as the sum over probabilities of *all* goal-achieving states of the belief state:

**Definition 3. Plan Success Probability** Given a model $M_{\text{assess}}$, observations $o_{0:t}$ and the goal $G$ of a plan $\mathcal{P}$ we define the probability that $G$ will be achieved as

$$Pr(\mathcal{G}|o_{0:t}, M_{\text{assess}}) = \sum_{x_{t+n} \in \mathcal{G}} Pr(x_{t+n}|o_{0:t}, M_{\text{assess}})$$

$\mathcal{G}$ is the set of all goal-achieving states $\mathcal{G} := \{x_{t+n} \in \Theta | x_{t+n} \text{ entails } G\}$. $\Theta$ is the set of all system states, i.e. $\Theta = \mathcal{S}(M_{\text{assess}})$.

In most cases it is infeasible to compute the complete distribution as it requires enumerating all system states. The solution is to use approximation schemes which try to compute good bounds of the success probability based on a reduced set $\Theta^* \subset \mathcal{S}(M_{\text{assess}})$ of system states and their conditional probabilities $Pr(x_{t+n}|o_{0:t}, M_{\text{assess}})$, with $x_{t+n} \in \Theta^*$. Then, an upper bound on $Pr(\mathcal{G}|o_{0:t}, M_{\text{assess}})$ is computed by summing over probabilities of goal-achieving states, and a lower bound by summing over probabilities of goal-violating states.

**Proposition 1. Bounds for** $Pr(\mathcal{G}|o_{0:t}, M_{\text{assess}})$ *Let $M_{\text{assess}}$ be an assessment model and $G$ the goal of a plan $\mathcal{P}$. Let $\Theta^* \subset \mathcal{S}(M_{\text{assess}})$ be a set of system states of $M_{\text{assess}}$ and $Pr(x_{t+n}|o_{0:t}, M_{\text{assess}})$, $x_{t+n} \in \Theta^*$ their probabilities conditioned on given observations. Let further $\mathcal{G}^* = \{x_{t+n} \in \Theta^* | x_{t+n}$ consistent with $G\}$ be the set of states among $\Theta^*$ which entail the plan's goal, and $\overline{\mathcal{G}}^* = \{x_{t+n} \in \Theta^* | x_{t+n}$ not consistent with $G\}$ those which violate it. Then*

$$p_l = \sum_{x_{t+n} \in \mathcal{G}^*} Pr(x_{t+n}|o_{0:t}, M_{\text{assess}})$$

*is a lower bound on $Pr(\mathcal{G}|o_{0:t}, M_{\text{assess}})$ and*

$$p_u = 1 - \sum_{x_{t+n} \in \overline{\mathcal{G}}^*} Pr(x_{t+n}|o_{0:t}, M_{\text{assess}})$$

*an upper bound on $Pr(\mathcal{G}|o_{0:t}, M_{\text{assess}})$.*

Roughly, the bounds are computed following these steps:

*Computing bounds for success probability.* Input: $M_{\text{assess}}$, set of goals $\{G_i\}$ of concurrently executed plans $\{\mathcal{P}_i\}$.

1. Determine the time point $t+n$ when all plans are finished
2. Generate $\Theta^*$ and compute $Pr(x_{t+n}|o_{0:t}, M_{\text{assess}})$ for $x_{t+n} \in \Theta^*$.
3. Compute the lower and upper bounds for each plan goal

$$p_l = \sum_{x_{t+n} \in \mathcal{G}_i^*} Pr(x_{t+n}|o_{0:t}, M_{\text{assess}})$$

$$p_u = 1 - \sum_{x_{t+n} \in \overline{\mathcal{G}}_i^*} Pr(x_{t+n}|o_{0:t}, M_{\text{assess}})$$

To make an actual decision to consider a plan as likely to succeed or fail, $p_l$ must be compared against a threshold $\omega_{\text{success}}$ and $p_u$ against a threshold $\omega_{\text{fail}}$ (which we assume as given externally).

## 4. Related Work

Closest to our work are verification approaches. As a static preprocessing step, computing a plan success probability could possibly be done within the framework of probabilistic model checking [10, 11]. $M_{\text{assess}}$ would be the model, and the success

probability for a specific product could be formulated as a temporal logic formula expressing "At $t + n$, with $p > \omega_{\text{success}}$ the product state is in $\mathcal{G}$". However, we are not aware of approaches to online probabilistic model checking which regard available sensor information (as our method does), or which focus on models not only of plants and controllers but also products. Similarly, verification approaches in control and manufacturing [12, 13] typically focus on modeling plants and controllers only.

In probabilistic verification of model-based programs [14], the problem is to determine the most likely circumstances under which a high-level control program drives the system towards a goal violating state. A plan can be understood as such a high level control program; so in general, this problem is similar to the plan assessment problem. However, our problem differs in that we are interested in the *set* of all goal-achieving system trajectories, from which we derive the plan's success probability, while for the verification problem, only the single most probable goal violating trajectories are interesting.

Early work addressed the problem of *automated manufacturability analysis*, which is to evaluate a process plan for machining a product from stock with respect to, e.g., design tolerances (does plan execution violate tolerances?) or the plan's cost. [15] discusses the Interactive Manufacturing Analysis and Critique System (IMACS), which generates process plans and then checks them against, e.g., design tolerance constraints, to evaluate whether the plans can reach their machining goals. Another work in this domain is [16], which introduces a Petri net based approach to process plan evaluation. The authors estimate costs of process plans by representing the plans along with plant machines and their tools and configurations as special Petri nets. They propose two kinds of Petri net models, each with a special algorithm to compute the costs. Clearly there is some similarity to our plan assessment problem, especially the idea of generating models of machines and plans for plan evaluation. The main difference is that we're interested in dynamic machine behavior (nominal and off-nominal) induced by plan execution, whereas automated manufacturability analysis is concerned with evaluating plans against static constraints, such as design tolerances.

Work on distributed manufacturing [17], carried out within the PABADIS and PABADIS'PROMISE [18] projects, addresses similar problems. Some focus on robustly generating plans for newly occurring product variations, while others address robust plan execution in the face of plant faults. PABADIS and PABADIS'PROMISE also address the related problems of flexible production [19] and scheduling for production [20].

Automated planning is also applied for large scale operations management, e.g. reacting to oil spill disasters. [21] gives an overview of such planning scenarios and discusses, among other things, an oil spill response configuration system. It creates plans based on a spill trajectory forecast and then evaluates how much oil a specific plan is able to remove. It uses a special evaluation model along with the projected oil flow and the plan. The system monitors plan execution and is able to react to new events or goals provided by human operators. The main difference to these approaches is the scope (large scale operations planning vs our scenario of product manufacturing) and the fact that plans are not evaluated against a fixed goal (i.e. how likely they are to succeed), but against optimality criteria such as how much oil is being removed.

McDermott [22] and Beetz's [23] Reactive Plan Language (RPL) chooses a different approach to deal with system failures and uncertainty. It uses a hierarchical task decomposition, breaking down top level goals to a finer granularity recursively. The plan itself is not a sequence of actions but executable code. The language allows reasoning on and transformation of the plans. Heuristic routines attain the sub-goals and cope with failures and unexpected events during the execution. This approach is particularly promising in domains of high uncertainty, where classical planning fails. However, the RPL approach currently neglects explicit diagnosis techniques and relies on the observability of relevant environment states.

Also related to our work are model-based diagnosis approaches as introduced in [24]. The general problem is to identify faulty components in a system given model and current observations. The authors propose a novel iterative diagnosis approach based on truth maintenance systems.

## 5. Modeling for Planning and Plan Assessment

Before we go into detail about computing plan success probabilities and deriving decisions from

them, we explain what kind of planning models $M_{\text{plan}}$ and plan assessment models $M_{\text{assess}}$ we use.

### 5.1. PDDL Planning Models

Classical Planning including the Action Description Language (ADL), typing and conditional post conditions as described in the different publications on the Planning Domain Definition Language (PDDL) [4] is used to generate manufacturing plans. Our planning models are logical system models, based on first-order predicate logic. They lack time, continuous and concurrent behaviour.

**Planning Model** A planning model is a tuple $M = (E, P, A)$, where $E$ is the set of entities and $P$ a set of first-order logic predicates encoding the mentioned relations and connections. $A$ is a set of possible actions $a = (pre(a), add(a), del(a))$.

$pre(a)$ is a set of facts encoding the precondition, and $add(a), del(a)$ encode an action's post conditions as facts that are added (become true) and facts that are removed (become false) after execution. The model defines a state space with possible transitions between states defined by the actions.

Behavioral models of plant components, including possible failures, are left out, since for planning only component capabilities are relevant. It has to be considered that the component capabilities modelled for planning are not the principal capabilities of e.g. a robot, but only the ones that are realized by for example programmed robot arm trajectories. As an example, a robot may be able to assemble two parts in principle, but this capability is only included if there is a program for the robot for that specific task.

The planning domain used for the experiments comprises all aspects of logistics and use of resources necessary for the execution of generated plans on the simulation model using primitive action controllers. Specifically, the products are modelled as a tree of parts that have to be assembled and CNC operations carried out on parts to change their shape. The capabilities of the assembly robots and CNC machines are defined by the types of assemblies they can handle, so that similar machines can have e.g. partly overlapping capabilities and multiple solutions to the manufacturing task can be found. The transportation is fully modelled, including the storage system and the palletized transport system based on connected conveyor belts.

### 5.2. Plan Assessment Models as Hybrid Probabilistic Automata

For plan assessment, behavioral knowledge of the plant or factory and its components is important. We assume that in a rigid factory setting, the nominal behavior of plant components can be specified comprehensively using hierarchical automata encoding discrete transitions between modes of operation (where a mode corresponds to an automaton location). Uncertainty only appears as unlikely (failure) behaviors, and can be encoded using probabilistic transitions to failure modes. Finally, a plant component can exhibit many kinds of behavior which are best modeled continuously, e.g. using ordinary differential equations (ODEs) or piece-wise affine systems (PWAs). Examples for such behavior are wear or temperature levels of cutting tools.

In previous work [5] we introduced Hybrid Probabilistic Hierarchical Constraint Automata (HyPHCA), in style of the well known hybrid automata [25]. They combine the modeling power of PHCA [26], a formalism specifically tailored to embedded systems development and model-based monitoring/tracking of complex, uncertain system behavior, with linear ODEs. Linear ODEs are 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 4. Probabilistic Hierarchical Constraint Automata (PHCA)** A PHCA is a tuple $\langle \Sigma, P_\Xi, \Pi, O, Cmd, \mathcal{C}, P_T \rangle$, where:

- $\Sigma$ is a set of locations, partitioned into primitive locations $\Sigma_p$ and composite locations $\Sigma_c$. Each composite location denotes a hierarchical constraint automaton. A location may be marked or unmarked. A marked location represents an active execution branch.
- $P_\Xi(\Xi_i)$ denotes the probability that $\Xi_i \subseteq \Sigma$ is the set of start locations (initial state). Each composite location $l_i \in \Sigma_c$ may have a set of start locations that are marked when $l_i$ is marked.
- $\Pi$ is a set of variables with finite domains. $\mathcal{C}[\Pi]$ is the set of all finite domain constraints over $\Pi$.

- $O \subseteq \Pi$ is the set of observable variables.
- $Cmd \subseteq \Pi$ is the set of command variables.
- $\mathcal{C} : \Sigma \rightarrow \mathcal{C}[\Pi]$ associates with each location $l_i \in \Sigma$ a finite domain constraint $\mathcal{C}(l_i)$.
- $P_T(l_i)$, for each $l_i \in \Sigma_p$, is a probability distribution over a set of transition functions $T(l_i) : \Sigma_p \times \mathcal{C}[\Pi] \rightarrow 2^{\Sigma}$. Each transition function maps a marked location into a set of locations to be marked at the next time step, provided that the transition's guard constraint is entailed. The set of all transitions is $\mathcal{T}$.

**Definition 5. (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. To avoid ambiguity with "system state" we will refer to markings rather than PHCA states.

HyPHCAs extend PHCAs with continuous variables and linear ODEs:

**Definition 6. Hybrid Probabilistic Hierarchical Constraint Automata (HyPHCA)** 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.
- $\mathcal{C} : \Sigma \rightarrow \mathcal{C}[\Pi \cup U \cup U']$ is a function associating locations with constraints over discrete and/or real-valued variables. $\mathcal{C}[\Pi \cup U \cup U']$ denotes the set of constraints over $\Pi \cup U \cup U'$.
- $\mathcal{F} : \Sigma \rightarrow \mathcal{F}[U \cup \dot{U}]$ 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 $T(l_i) : \Sigma_p \times \mathcal{C}[\Pi \cup U \cup U'] \rightarrow 2^{\Sigma}$ 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 7. (HyPHCA state, HyPHCA trajectory)** The state of a HyPHCA at time $t$ is a tuple $S^{(t)} = (S_U^{(t)}, m^{(t)})$, where $S_U^{(t)} \in \mathbb{R}^{|U|}$ is an assignment to all variables $u \in U$ at time $t$, called continuous state, and $m^{(t)} \in \mathcal{M}$ a marking analogous to PHCA markings (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 $\theta_{HA} = \Delta(\langle t_i \rangle)$, resulting from evaluating $\Delta$ on a finite sequence of time points, is called a discrete-time HyPHCA trajectory.

In both cases, discrete and hybrid modelling, we use discrete time steps of a fixed, predefined length $\triangle t$. Throughout the article we refer to (Hy)PHCA variables at time $t$ with superscript $^{(t)}$, e.g. $\mathsf{Vibration}^{(t)}$. In the context of probability we comply with standard notation, i.e. we refer to random variables at time $t$ or a sequence of them from $b$ to $t$ with subscripts $_t$ and $_{b:t}$, respectively. In both cases $t$ is a short cut for $t_i$, where $i$ is a natural number index of the time point. Since we have a fixed length interval between time steps we can refer to time points using these indexes. Therefore we also shortcut $t_{i+1}$ to $t+1$. The semantic of a single time step is that a marking $m^{(t)}$ holds or is active within $[t, t + \triangle t)$, i.e. just before the next time point.

The prototype PHCA shown in figure 7 illustrates the PHCA definition. The factory components machining and assembly are encoded as top level composite locations. A dashed border indicates that locations may be marked at the same time, which means they can run in parallel. There is a third top level location at the bottom of figure 7 whose behavior constraint encodes that an observed vibration is caused by one of the two components or both. Primitive locations are for example machining.idle and machining.cut, which encode the machining station being idle and working on a piece. An example for an observable variable is Vibration, which encodes whether a vibration has occurred or not. The dependent variables machining.Vibration and assembly.Vibration encode for each component whether it caused a vibration. A command variable is, e.g., machining.cmd. It occurs in the guard constraint for transition idle → cut within composite location machining: machining.cmd = cut. Transition guards have the general form <guard constraint>;<transition

probability>. The guard constraint is a logical constraint over PHCA variables, usually an assignment to command variables. The transition is nondeterministic: Given the guard is satisfied, it is taken with probability 0.9. The remaining possibility (completing the probability distribution) is the transition from idle to the composite location cutter blunt, which has the same guard and is taken with probability 0.1.

In our prototype implementation we didn't include continuous behavior yet. We refer to [5] for an example of hybrid modeling. In this work we introduce an approach to convert a HyPHCA model to a PHCA model by discretizing the continuous model-part. In section 6 we briefly describe this approach.

### *5.3. Plan Assessment Models from Plan Actions and Goals*

The plan assessment model $M_{\text{assess}}$ represents the plant behavior. It specifies each component's behavior as well as interactions among them (through shared variables). But how are plan goals represented? We have seen that plan success is defined in terms of goal consistent system states after plan execution. Therefore the assessment model has to represent the goal for each plan such that system states can be classified as goal entailing or violating for each goal. We now sketch a method to create assessment models from a given knowledge base and a schedule created from a set of plans, which also answers the question of goal representation.

The model $M_{\text{assess}}$ is a hierarchical automaton (using the mentioned HyPHCA framework) composed of sub-automata that represent the plant components. The actions given in the schedule dictate the number and type of sub-automata in the model. They map to given sub-automata (templates) in the knowledge base representing the behavior of plant components. For example, in our schedule we may have one action which puts the ball into the maze and a second action which lets a palette move on the conveyor (palettes on conveyors are stopped by holding them in place with a magnetic bolt). Thus two automata are added to the model $M_{\text{assess}}$: an automaton representing the assembly-behavior and one representing the conveyor-behavior. To model the fact that the same component might work on parts from different products, each component incorporates a simple buffer automaton. This buffer automaton indicates whether the component is empty or working on a part from a specific product.

A product's manufacturing progress is itself modeled as an automaton, which is aligned (through e.g. shared variables) with the according plant-component-automata. Component failure then automatically leads to a location in the progress automaton representing a faulty product. Depending on how detailed the progress is represented, individual product parts and their possible faults might be identified as well as forbidden product states. In the simplest case the progress automaton has only three locations, *not-processed*, *finished* and *faulty*. The model $M_{\text{assess}}$ is now composed in such a way that the *finished*-location for a specific product is only reachable if all actions finished without fault, i.e. if all the automata representing these actions did not run into any fault locations. The goal vector composed of binary variables, is simply tied to the *finished*-location, setting the binary variables to true.

## 6. Plan Assessment With Hybrid Discrete/-Continuous Models

In our example scenario, it might be beneficial to model the temperature of the cutting tool to prevent overheating. This renders $M_{\text{assess}}$ a *hybrid* discrete/continuous model and leads to the problem of computing a hybrid belief state: A distribution over states $\langle s, u \rangle$, where $s$ is a vector of discrete variable value assignments, and $u$ a vector of reals, i.e. assignments to the continuous variables of the model. This is the well known problem of hybrid estimation. In our case $u$ is the temperature of the cutting tool, which we denote as $u_{\text{temp}}$ .

A variety of solutions exist: a $k$-best hybrid estimation approach which tracks the $k$ most probable hybrid trajectories of a complex system [27], particle filter approaches [28] and methods which automatically discretize the continuous state space [29].

In [5] we proposed a discretization approach which integrates the discretization method from [29] with model-based diagnosis and soft-constraint reasoning [30]. This approach differs from others in that it separates algorithms from modeling formalism, thereby deepening the separation of concerns and leveraging the power of existing soft-constraint solvers[1]. The core idea is to convert the parts of

[1] We use toolbar `https://mulcyber.toulouse.`

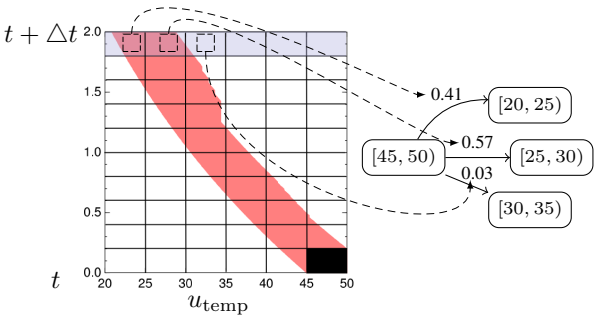



Figure 6: Illustration of the reachable set $R_{\text{start}}$ for $\dot{u}_{\text{temp}} = -k(u_{\text{temp}} - u_{ambient})$ starting from the marked grid cell $G_{\text{start},t_i}$. Right: the derived PHCA $\mathcal{F}_d(L)$.

a HyPHCA system model $M_{\text{assess}}$ described with linear ODEs to Markov chains (which are actually a restricted type of PHCA) and reinsert these into $M_{\text{assess}}$, rendering it a completely discrete model. In our example, the cooling of the cutting tool can be modeled as $\dot{u}_{\text{temp}} = -k(u_{\text{temp}} - u_{ambient})$, which is the standard cooling equation. In the next paragraphs we explain how this continuous model is abstracted to a discrete Markov chain.

For each primitive location $L \in M_{\text{assess}}$ the evolution of its continuous variables in between two time points, encoded in its continuous flow $\mathcal{F}(L)$, is mapped to discrete, unguarded probabilistic transitions between locations of a special PHCA $\mathcal{F}_d(L)$. It has only primitive locations, corresponding to cells of a quantization of the continuous state space, and represents a Markov chain that conservatively approximates the continuous evolution.

To conservatively estimate transition probabilities of $\mathcal{F}_d(L)$ we use the geometric abstraction method introduced in [29]. We recap this method shortly. The quantized state space is combined with a partition of the time interval $[t, t + \triangle t]$. Start locations of transitions of $\mathcal{F}_d(L)$ are associated with quantization cells within the first partition element in $[t, t + \triangle t]$ and destination locations with the last. Let now $G_{\text{start},t}$ be the quantization cell of start location $L_{\text{start}}$ and $G_{\lambda,t+\triangle t}$ the cells of all possible destination locations $L_\lambda$ (with $\lambda$ indexing cells and locations). The *reachable set* $R_{\text{start}}$ is computed, which is as small as possible yet guaranteed to include all continuous states reachable from $G_{\text{start},t}$ within $[t, t+\triangle t]$. Now the probabilities for the transitions $L_{\text{start}}$ to destination locations $L_\lambda$ are computed as

$$Pr(L_\lambda | T = t+\triangle t, L_{\text{start}}) = \frac{V(G_{\lambda,t+\triangle t} \cap R_{\text{start}})}{V(\bigcup_x G_{x,t+\triangle t} \cap R_{\text{start}})},$$

where $V()$ measures the volume of the given set and $T$ is the random variable for the time passed. The process is illustrated in figure 6.

Currently we use PHAVer [31] 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 [32]. In [32] they combine Markov chains abstracting continuous behavior with a more advanced reachability analysis.

A too coarse state space quantization can lead to spurious solutions. 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 [33]. 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.

*Conservative Abstraction of HyPHCAs*

The discretized flow $\mathcal{F}_d(L)$ together with the discrete part of the original HyPHCA now forms a discrete model which we call discrete flow PHCA (dfPHCA). We define dfPHCA states just like HyPHCA states as $(\hat{S}^{(t)}_{\Pi_U}, \hat{m}^{(t)})$, with the single difference that instead of assignments to continuous variables $U$ we have assignments to their discrete counterparts $\Pi_U$. The hat $\hat{}$ is used to differentiate HyPHCA states $(S^{(t)}_U, m^{(t)})$ from dfPHCA states $(\hat{S}^{(t)}_{\Pi_U}, \hat{m}^{(t)})$. dfPHCA are a step in between HyPHCA and PHCA, and it can be argued that for each dfPHCA there exists an equivalent PHCA. We refer to [5] for the specifics.

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

`inra.fr/projects/toolbar/` (15.12.2009) and toulbar2 `https://mulcyber.toulouse.inra.fr/projects/toulbar2/` (15.12.2009)

**Definition 8.** (Set of abstracted HyPHCA trajectories) Let $G : D_{\Pi_U} \rightarrow G_{\mathbb{R}^{|U|}}$ be a function that maps assignments to discretized continuous variables $\Pi_U$ to grid cells $G_\lambda \in G_{\mathbb{R}^{|U|}}$, with $D_{\Pi_U} = D_{x1} \times \ldots \times D_{xn}$ the combined domain of discretized variables $x1, \ldots, xn \in \Pi_U$. Let $\theta_{A_{\text{df}}}$ be a trajectory of $A_{\text{df}}$ with corresponding time point sequence $\langle t_i \rangle$, generated from a HyPHCA $HA$. Then $\chi(\theta_{A_{\text{df}}}) := \{\Delta \text{ is a trajectory of HA } | \forall t_i : (S_U^{(t)}, m^{(t)}) \in \Delta(\langle t_i \rangle) \wedge (\hat{S}_{\Pi_U}^{(t)}, \hat{m}^{(t)}) \in \theta_{A_{\text{df}}} \Rightarrow m^{(t)} = \hat{m}^{(t)} \wedge S_U^{(t)} \in G(\hat{S}_{\Pi_U}^{(t)})\}$ is the set of all HyPHCA trajectories contained in $\theta_{A_{\text{df}}}$.

**Proposition 2.** *Let $\langle o_i, c_i \rangle$ be an arbitrary finite sequence of observations and commands and $\langle t_i \rangle$ the corresponding sequence of time points. Then, for a trajectory $\theta_{A_{\text{df}}}$ consistent with $\langle o_i, c_i \rangle$ (i.e. $Pr(\theta_{A_{\text{df}}} | \langle o_i, c_i \rangle) > 0$), the following holds:*

$$\forall \Delta \in \chi(\theta_{A_{\text{df}}}) :$$

$$f_{HA}(\Delta(\langle t_i \rangle) | \langle o_i, c_i \rangle) \leq Pr(\theta_{A_{\text{df}}} | \langle o_i, c_i \rangle)$$

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

## 7. Belief State Approximation for Bounding Plan Success Probability

In section 3.3 we have seen how bounds for the plan success probability $Pr(\mathcal{G} | o_{0:t}, M_{\text{assess}})$ can be computed based on an approximated belief state. Now we look at methods to compute this approximated belief state.

### *7.1. General Belief State Approximation*

PHCAs and dfPHCAs are probabilistic models, i.e. they encode a distribution $Pr(X_{t+1} | X_t)$ for the possible state transitions given the current state and a distribution $Pr(O_t | X_t)$ over possible observations given the current state. A well known approach to compute the belief state based on these kind of models is filtering, whereby the distribution over system states, conditioned on observations $o_{0:t}$, is computed iteratively given the last belief state and the observation for the current time point. This computation can be done approximately using sampling techniques or with a $k$-best approach. The widely employed particle filtering (see, e.g. [28]) is an instance of the former, while a $k$-best belief update is described in, e.g., [27].

The $k$-best enumeration approach is generally more appropriate in settings with a dominant, nominal behavior and many unlikely off-nominal behaviors. In this case, the distribution over states or sequences is peaked at the nominal behavior. If in contrast many different behaviors with small probabilities are possible without clear domination, e.g. when many different executions of a plan can lead to the plan's goal, $k$-best is a bad choice: Suppose that a plan yields 100 possible executions. 10 of them are goal-violating, the other 90 goal-achieving. Suppose that the success probability $Pr(\mathcal{G} | o_{0:t}, M_{\text{assess}})$ is 0.8, uniformly distributed among the 90 goal-achieving behaviors. This yields a probability of $\approx 0.008$ per goal-achieving behavior. Likewise, the failure probability (0.2) is distributed uniformly among the 10 goal-violating behaviors, yielding 0.02 per goal-violating behavior. Clearly, $k$-best fails in this case: the goal-violating behaviors are enumerated first, yielding a lower bound of 0 for $Pr(\mathcal{G} | o_{0:t}, M_{\text{assess}})$ as long as $k < 11$, and then only slowly increases.

The factory settings we look at are rigid enough to assume a dominant, nominal behavior. Consequently we assume that a $k$-best approximation serves us better than a sampling approach.

Approximation means that unlikely states are pruned, focussing on the more likely states. A problem with that within the context of model-based diagnosis can be *delayed symptoms*, as shown in [34]: A currently very unlikely state is pruned, but it is the only state consistent with observations which become available only later (delayed). The result is that the model becomes inconsistent as soon as the delayed observations become available, because the only state(s) consistent with them where too unlikely to be kept. As solution the authors propose to enumerate *sequences* of states instead of single states. The length of the sequence $N$ must be chosen such that they include the delays.

In our prototype implementation we use such a time window approach which generates sequences of the length $N$ (where $N$ is chosen as a parameter). The end of the time window coincides with time point $t + n$, and the belief state at that time point can be computed by marginalizing over the subsets of sequences which end in the same state. We combine it with $k$-best approximation, enumerating the $k$ most probable state sequences within the time window. This is an extension of the *Most Probable Explanation Problem*. Here another advantage becomes apparent, namely that a sequence

of states can provide diagnostic information by explaining what happened in the past (i.e. within the time window). In our example, state sequences leading to a failure of the plan explain that this happens because the cutter first blunts and then breaks. After deciding that a plan is likely to fail, this information might indicate, e.g, what can be done to prevent it (e.g., not using this machining station).

Enumerating the most probable trajectories dominates the complexity of our chosen approach, which is essentially exponential in the number of factory components (assembly, machining, etc.) plus the number of products scheduled times the number of time steps considered. The various translation and model generation processes are linear in these quantities and thus negligible.

### *7.2. Computing Most Probable State Sequences Using Constraint Optimization*

A PHCA encodes possible system evolutions, and sequences of states are consequently represented by markings of the PHCA's locations. Previous work [35] introduced an encoding of PHCA as soft constraints [30], and casted the problem of most probable sequences of PHCA markings as a soft constraint optimization problem (COP) $\mathcal{R} = (X, D, C)$. Its solutions directly correspond to sequences of markings within a window of $N$ time steps.

Executing a PHCA, given a marking $m^{(t)}$, means to identify possible target locations to be marked at $t+1$, probabilistically choose transitions and check consistency of observations and commands with transition guards as well as behavior of the targets. Also, it involves checking for interdependences encoded in behavior PHCA constraints, e.g., that a vibration occurs if and only if a vibration occurs in machining or in assembly. Finally, targets have to be marked correctly regarding, among other things, the hierarchical structure of a PHCA and initial marking.

These execution semantics are encoded as COP constraints for single time points and for transitions between time points. The COP consists of $N$ copies of these constraints, corresponding to the $N$ time steps of the time window. This unfolds the model over $N$ time 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 $t = 0..N$, it consists of $\Pi^{(t)} \subseteq X$ encoding PHCA variables, auxiliary variables (needed to, e.g., encode hierarchical structure) and a set of binary variables $Y = \{X_{L_1}^{(t_i)}, X_{L_2}^{(t_i)}, \ldots\} \subseteq X$ representing location markings. $Y$ is the set of solution variables of $\mathcal{R}$. $C = \{C_1, \ldots, C_n\}$ is a set of constraints $(S_i, F_i)$ with scope $S_i = \{X_{i1}, \ldots, X_{im}\} \subseteq X$ and a constraint function $F_i : D_{i1} \times \cdots \times D_{im} \rightarrow [0, 1]$ mapping partial assignments of variables in $S_i$ to a probability value in $[0, 1]$. For all time steps $t = 0..N$, hard constraints in $C$ ($F_i$ evaluates to $\{0, 1\}$ only) 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 $t = 0$ and probabilistic transitions. $\mathcal{R}$ then consists of $O(N(|\mathcal{T}| + |\Sigma| + |\Pi|))$ variables and $O(N(|\Sigma| + |\mathcal{T}|))$ constraints. The $k$-best solutions to $\mathcal{R}$ are assignments to $Y$ which, extended to all variables $X$, maximize the global probability value in terms of the functions $F_i$ (i.e. the best solution has the largest probability value, the second best the second largest etc.). These assignments correspond to the most probable PHCA system trajectories and their extension to $X$ provides assignments to, e.g., goal-achieving commands.

Technically, the enumeration of the $k$ most likely sequences is done by translating the generated COP into the weighted CSP format as used by various solvers. We used a modified version of the soft constraint solver toolbar [36] that implements mini-bucket elimination to generate a search heuristic for the problem. The heuristic is used by a subsequent A* search to enumerate the $k$ best solutions. This approach is described in detail in [37]. Note that up to using an external solver such as toolbar, all steps (e.g. COP encoding, generating the assessment model) can be done offline. Only the final solving step is done online.

## 8. Prototype Implementation

We built a simplified model of the manufacturing system (see figure 7) which consists only of the machining and the assembly station and allows to track system behavior over time, including unlikely component faults. In particular, the machining station can transition to a "cutter blunt" composite location, where vibrations are caused during operation due to the blunt cutter. The assembly station model contains a composite location which models occasional vibrations. A sensor can detect these

vibrations, yielding binary signals "vibration occurred" and "no vibration occurred".

*8.1. Restricted Plan Assessment Scenario*

Three major restrictions apply to our prototype implementation. First, products and manufacturing plans are simplified: the robot arm is reduced to a single alloy part being cut, the schedule is reduced to six sequential steps and the maze only consists of the base plate and the cover (no pins). The simplified plan now has actions which cut the maze, assemble the maze and cut parts of the robot arm: $\mathcal{P} = \langle(\mathsf{cut}, 0), (\mathsf{assemble}, 1), (\mathsf{cut}, 2), (\mathsf{cut}, 3), (\mathsf{cut}, 4), (\mathsf{cut}, 5)\rangle$ (Note that times are already mapped to indices of time points).

Second, the assessment model cannot differentiate among different products, i.e. it only "sees" a merged product consisting of the maze and the robot arm part. Third, the manufacturing goal is a single feature added as binary variable to the model. The feature is present in all cases except when the cutter breaks. In this case, the feature is absent and the plan considered failed.

*8.2. Computing Success Probability Using Most Probable Trajectories*

For our prototype we implemented the constraint-based $k$-best enumeration of most probable state sequences or trajectories as described in section 7.2. Rather than first computing an approximate belief state at $t + n$, we sum directly over *goal-achieving/goal-violating trajectories* to compute an approximate success probability and bounds of the success probability. Goal-achieving/violating trajectories are simply defined through the last system state in the trajectory: if it is goal-achieving, the trajectory is goal-achieving, otherwise it is goal-violating. Summing over trajectories achieves the same result as summing over states of a belief state, because the belief state can in fact be computed from a distribution over trajectories by summing over trajectories leading to the same state. It is easy to see that, instead of first summing over trajectories leading to the same state and then summing over goal-achieving states, one can directly sum over goal-achieving trajectories.

In the following, we omit the model $M_{\text{assess}}$ as condition in the probabilities for brevity. First, observe that

$$Pr(\mathcal{G}|o_{0:t}, M_{\text{assess}}) = \sum_{\theta\in\mathcal{G}} Pr(\theta|o_{0:t}) = \sum_{\theta\in\mathcal{G}} \frac{Pr(\theta, o_{0:t})}{Pr(o_{0:t})} = \sum_{\theta\in\mathcal{G}} \frac{Pr(\theta, o_{0:t})}{\sum_{\theta\in\Theta} Pr(\theta, o_{0:t})} = \frac{\sum_{\theta\in\mathcal{G}} Pr(\theta, o_{0:t})}{\sum_{\theta\in\Theta} Pr(\theta, o_{0:t})}.$$

Here, $\Theta$ is not the set of all system states but of all trajectories within the chosen time window, and likewise $\mathcal{G} \subseteq \Theta$ is the set of all goal-achieving trajectories. Let now $\Theta(k)$ be the set of $k$-best trajectories enumerated by our external solver toolbar, and $\mathcal{G}(k) \subseteq \Theta(k)$ the subset of goal-achieving trajectories. We see that we receive simple bounds of $Pr(\mathcal{G}|o_{0:t}, M_{\text{assess}})$ by replacing $\mathcal{G}$ with $\mathcal{G}(k)$ in the above equations and approximating the denominator with 1 (with $\overline{\mathcal{G}}(k) \subseteq \Theta(k)$ the set of goal-violating trajectories among the $k$-best):

$$\begin{aligned} p_l &= \sum_{\theta\in\mathcal{G}(k)} Pr(\theta, o_{0:t}) \\ p_u &= 1 - \sum_{\theta\in\overline{\mathcal{G}}(k)} Pr(\theta, o_{0:t}) \end{aligned}$$

The $k$-best trajectories can also be used to approximate the denominator, which then yields this approximate success probability:

$$Pr^k(\mathcal{G}(k)|o_{0:t}, M_{\text{assess}}) = \frac{\sum_{\theta\in\mathcal{G}(k)} Pr(\theta, o_{0:t})}{\sum_{\theta\in\Theta(k)} Pr(\theta, o_{0:t})}$$

The idea behind this is to make as much use of the $k$-best trajectories as possible. For our small prototype the approximation worked quite well. However, as we argue in [6], the error of the above approximation is very hard to determine. It depends non-monotonically on $k$, which makes it hard to choose a $k$ which minimizes the error of the above approximation.

*8.3. Decision Procedure using Plan Assessment*

Plans are advanced until they are finished or new observations are available. In the latter case currently executed plans are evaluated using the procedure in figure 8. The procedure takes as input a plan $\mathcal{P}_i$, it's goal $G_i$ and the assessment model as COP. It first computes the $k$-best solutions to the COP using an external solver (toolbar in our case). This results in the $k$ most probable trajectories, which it then uses to approximate the success probability of plan $\mathcal{P}_i$. Finally, it compares the probability against the two thresholds $\omega_{\text{success}}$ and $\omega_{\text{fail}}$. Now it decides among three cases: (1) The probability is above $\omega_{\text{success}}$, i.e. the plan will probably succeed, (2) the probability is below $\omega_{\text{fail}}$,

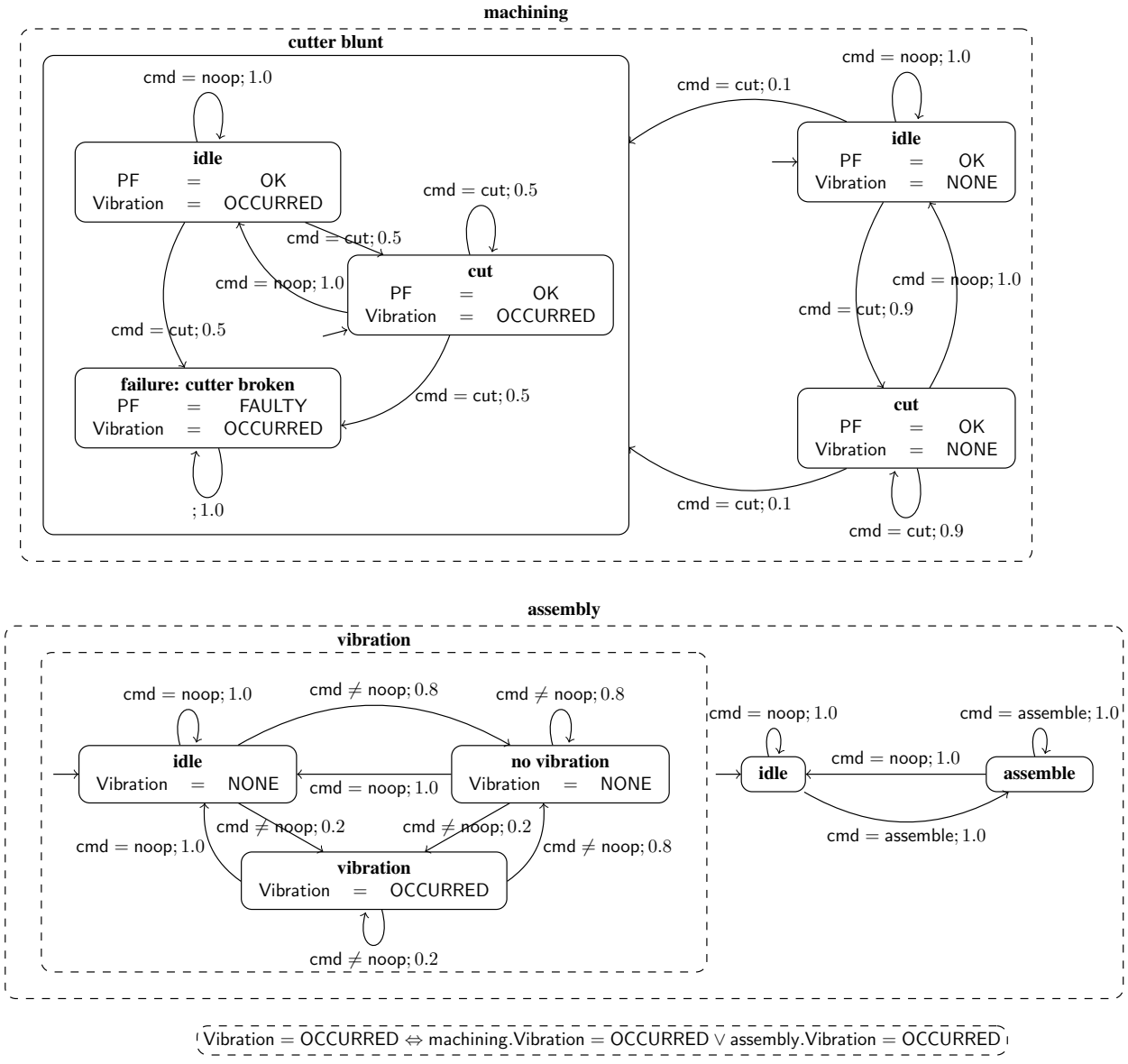



Figure 7: Prototype PHCA of the cognitive factory. The machining and assembly station are parallel running composite locations (indicated by dashed borders). Variables such as Vibration are local. machining.Vibration refers globally to Vibration within composite location machining. Note: "noop" means "no operation".

1: **procedure** EVALUATEPLAN($\mathcal{R} = (X, D, C)$, $o_{0:t}$, $\mathcal{P}_i$, $G_i$)
2: $\mathcal{R}$' $\leftarrow$ add constraints encoding $o_{0:t}$ to $\mathcal{R}$
3: $\Theta(k) \leftarrow k$-best solutions of $\mathcal{R}$' for $Y$
4: $p \leftarrow Pr^k(\mathcal{G}_i(k)|o_{0:t}, M_{\text{assess}})$
5: **if** $p > \omega_{\text{success}}$ **then return**
6: **else if** $p < \omega_{\text{fail}}$ **then**
7: stop execution of $\mathcal{P}_i$
8: REPLAN ($\mathcal{P}_i$,$\Theta(k)$)
9: **else**
10: stop execution of $\mathcal{P}_i$
11: REPLANPERVASIVEDIAGNOSIS($\mathcal{P}_i$,$\Theta(k)$)
12: **end if**
13: **end procedure**

Figure 8: Procedure that uses the success probability of plan $\mathcal{P}_i$ to decide whether to a) continue with it, b) stop it because it probably won't succeed or c) gather more information.

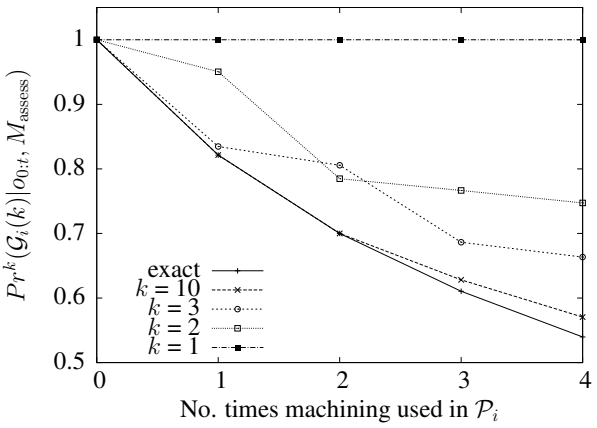



Figure 9: Approximate success probability (y-axis) of plan $\mathcal{P}_i$ against varying usage of the machining station (x-axis) after the observation of a vibration at $t = 2$.

i.e. the plan will probably fail or (3) the probability is in between both thresholds, which means the case cannot be decided. In the first case it simply continues execution. In the second case it adapts the plan to the new situation. This is done by REPLAN($\mathcal{P}_i$, $\Theta(k)$), which modifies the future actions of $\mathcal{P}_i$ taking into account the diagnostic information contained in $\Theta(k)$. The third case indicates that not enough information about the system's current state is available. As a reaction, the procedure REPLANPERVASIVEDIAGNOSIS($\mathcal{P}_i$, $\Theta(k)$) implements a recently developed method called *pervasive diagnosis* [38]. It addresses this problem by augmenting a plan with information gathering actions (we do not detail the procedures REPLAN and REPLANPERVASIVEDIAGNOSIS as they are beyond this article's scope).

## 9. Results

### *9.1. Experiments*

We ran experiments for our prototype example scenario, where we varied the number of cut actions after $t = 2$ in our example plan, yielding different $\mathcal{P}_i$. The time window size $N$ accordingly ranges from 2 to 6. Using our soft-constraint method, we generated COPs with according sizes of 240 to 640 variables and 240 to 670 constraints. Figure 9 shows the success probabilities for the $\mathcal{P}_i$ and $k$. Table 1 shows the runtime in seconds and the peak memory consumption in megabytes for computing success probabilities in the planning scenarios. We also tried different values for the mini-bucket parameter $i$, where higher values mean a higher cpu/memory investment to generate better search heuristics.

As expected, with increasing use of the machining station (more cut operations), $Pr^k(\mathcal{G}(k)|o_{0:t}, M_{\text{assess}})$ decreases. Also, runtime increases for larger time windows. The effect of approximation (choosing lower $k$) is that $Pr^k(\mathcal{G}(k)|o_{0:t}, M_{\text{assess}})$ increasingly deviates from the exact solution. In our example, the approximation tends to be optimistic. In general, however, we think that $Pr^k(\mathcal{G}(k)|o_{0:t}, M_{\text{assess}})$ can be pessimistic, if success trajectories are pruned first when decreasing $k$. Increasing $k$ hardly seems to affect the runtime, especially if the mini-bucket search heuristic is strong (bigger $i$-values). For weaker heuristics the influence increases slightly. Memory consumption is affected much stronger by $k$. Here also, a weaker search heuristic means stronger influence of $k$.

### *9.2. Discussion*

The typical scenarios that we have in mind do not require real-time plan assessment. Rather, "online" means that the assessment is done on, e.g., a separate PC (such as the one we used for experiments). The results show that a) plan assessment is in principle a valid approach to identify harmful influences on plans and b) the runtime/memory consumption is within the limits of typical setups. However, they

| | | No. times machining used in $\mathcal{P}$ (window size $N$, #Variables, #Constraints) | | | | |
|---|---|---|---|---|---|---|
| $k$ | $i$ | 0 (2,239,242) | 1 (3,340,349) | 2 (4,441,456) | 3 (5,542,563) | 4 (6,643,670) |
| 1 | 10 | < 0.1 / 1.8 | 0.1 / 6.8 | 0.1 / 19.0 | (mem) | (mem) |
| | 15 | 0.1 / 1.9 | 0.3 / 4.2 | 0.5 / 7.8 | 0.5 / 16.6 | 0.8 / 32.0 |
| | 20 | 0.1 / 1.9 | 0.5 / 5.2 | 3.7 / 20.1 | 6.5 / 34.5 | 9.5 / 50.7 |
| 2 | 10 | < 0.1 / 2.1 | 0.1 / 11.9 | 0.2 / 38.5 | (mem) | (mem) |
| | 15 | 0.1 / 2.2 | 0.3 / 5.4 | 0.5 / 9.7 | 0.6 / 28.0 | 0.8 / 52.0 |
| | 20 | 0.1 / 2.2 | 0.5 / 6.4 | 3.7 / 21.8 | 6.5 / 37.2 | 9.5 / 55.8 |
| 3 | 10 | < 0.1 / 2.3 (e) | 0.1 / 11.9 | 0.2 / 40.1 | (mem) | (mem) |
| | 15 | 0.1 / 2.4 (e) | 0.3 / 5.4 | 0.5 / 11.4 | 0.6 / 29.9 | 0.9 / 55.5 |
| | 20 | 0.1 / 2.4 (e) | 0.5 / 6.4 | 3.7 / 23.5 | 6.6 / 38.3 | 9.5 / 57.4 |
| 4 | 10 | (e) | 0.1 12.5 | 0.2 / 40.1 | (mem) | (mem) |
| | 15 | (e) | 0.3 / 5.9 | 0.5 / 11.4 | 0.6 / 30.9 | 0.9 / 57.2 |
| | 20 | (e) | 0.5 / 6.9 | 3.7 / 23.5 | 6.6 / 39.3 | 9.5 / 59.1 |
| 5 | 10 | (e) | 0.1 / 13.1 | 0.2 / 40.7 | (mem) | (mem) |
| | 15 | (e) | 0.3 / 6.6 | 0.5 / 12.0 | 0.6 / 33.6 | 0.9 / 59.5 |
| | 20 | (e) | 0.5 / 7.6 | 3.7 / 24.0 | 6.6 / 42.8 | 9.5 / 63.9 |
| 10 | 10 | (e) | 0.1 / 14.0 (e) | 0.2 / 43.4 (e) | (mem) | (mem) |
| | 15 | (e) | 0.3 / 6.7 (e) | 0.5 / 14.7 (e) | 0.6 / 36.2 | 0.9 / 64.8 |
| | 20 | (e) | 0.6 / 7.7 (e) | 3.8 / 26.6 (e) | 6.6 / 45.8 | 9.6 / 68.9 |

Table 1: Runtime in seconds / peak memory consumption in megabytes. (e) indicates that the exact success probability $Pr(\mathcal{G}|o_{0:t}, M_{\text{assess}})$ could be computed with this configuration. (mem) indicates that A* ran out of memory (artificial cutoff at > 1 GB, experiments were run on a Linux computer with a recent dual core 2.2 Ghz CPU with 2 GB RAM).

also indicate that exponential complexity is still a problem of our approach, despite using optimized constraint solving techniques (e.g. tree decomposition, mini-bucket elimination). One problem here is that a number of advanced COP techniques cannot be readily employed since it is non-trivial to adapt them to produce $k$-best solutions. The bottom line is that our approach is currently unlikely to scale up to real application domains.

A number of future research avenues are open to address these issues. One idea is to combine trajectory enumeration with filtering: Choosing a small time window with fixed $N$, one could shift it stepwise to cover the whole schedule, enumerating and summing over trajectories in each step. This would reduce the complexity to being exponential only in the number of components and products (since $N$ would be constant independently of the schedule length). Another idea is to reduce a detailed schedule to only a few important steps, e.g. by combining repetitive actions, to fit it into a small enough time window ($\approx 15$ steps). Finally, plan assessment could be restricted to computing (bounds on) product success probabilities for a fixed time horizon only.

## 10. Conclusion

Within a cognitive factory setting, we described the problem of *plan assessment*, which is to compute bounds on the probability of success of an automatically generated manufacturing process plan based on a predicted belief state of the system. We presented a model-based method to solve this problem which first enumerates the $k$ most probable system trajectories and then computes an approximate success probability based on them. In this method, we enumerate the trajectories by generating the $k$-best solutions to a probabilistic constraint optimization problem. Preliminary results with a prototype implementation for our cognitive factory scenario show that plan assessment can indeed guide the system away from plans which rely on suspect system components.

In the future, we would like to fully implement our approach, using assessment models generated for scheduled manufacturing of multiple individualized products. We also intend to fully develop a method for automatic generation of assessment models from a knowledge base and a given schedule. Furthermore, we plan experiments to compare our approach against other possible solutions of plan

assessment, e.g. particle filters or sampling based methods. Another interesting idea is to exploit diagnostic information from most probable system trajectories to update transition probabilities in behavior models, for instance, to adapt to parameter drifts or wear.

## 11. Acknowledgements

We would like to thank Dominik Jain for helpful discussions and comments, Andreas Artmeier for proof reading and Prof. Shea for providing photographs and graphics concerning the Cognitive Factory. This work is supported by the the Deutsche Forschungsgemeinschaft (DFG) under grant SA 1000/2-1 and by the German excellence cluster "Cognition for Technical Systems" (CoTeSys).

## References

[1] Martin Hülse and Manfred Hild, Informatics for cognitive robots, Advanced Engineering Informatics 24 (1) (2010) 2 – 3.

[2] M. Beetz, M. Buss, D. Wollherr, Cognitive technical systems — what is the role of artificial intelligence?, in: Proc. KI-2007, 2007, pp. 19–42.

[3] E. Brooks, J. Cunningham, Knowledge Selection for Planning in Complex Domains.

[4] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, D. Wilkins, Pddl-the planning domain definition language, The AIPS-98 Planning Competition Comitee.

[5] P. Maier, M. Sachenbacher, Diagnosis and Fault-adaptive Control for Mechatronic Systems using Hybrid Constraint Automata, in: Proc. PHM-2009, San Diego, CA, USA, 2009, available online at `www.phmconference.org`.

[6] P. Maier, M. Sachenbacher, T. Rühr, L. Kuhn, Integrating Model-based Diagnosis and Prognosis in Autonomous Production, in: Proc. PHM-2009, San Diego, CA, USA, 2009, available online at `www.phmconference.org`.

[7] S. Russell, P. Norvig, Artificial Intelligence: a modern approach, 2nd Edition, Prentice Hall Series in Artificial Intelligence, Prentice Hall, 2002, Ch. 15 Probabilistic Reasoning Over Time.

[8] K.-M. Chao, W. Shen, Enabling technologies for collaborative design, Advanced Engineering Informatics 24 (2) (2010) 119 – 120, enabling Technologies for Collaborative Design.

[9] W. van Holland, W. F. Bronsvoort, Assembly features in modeling and planning, Robotics and Computer-Integrated Manufacturing 16 (4) (2000) 277–294.

[10] J. Rutten, M. Kwiatkowska, G. Norman, D. Parker, Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems, Vol. 23 of CRM Monograph Series, American Mathematical Society, 2004.

[11] C. Baier, J.-P. Katoen, Principles of Model Checking, MIT Press, 2008, Ch. 10, pp. 745–900.

[12] M. Heiner, P. Deussen, J. Spranger, A case study in design and verification of manufacturing system control software with Hierarchical Petri nets, International Journal of Advanced Manufacturing Technology 15 (2) (1999) 139–152.

[13] V. Vyatkin, H. Hanisch, Verification of distributed control systems in intelligent manufacturing, Journal of Intelligent Manufacturing 14 (1) (2003) 123–136.

[14] T. Mahtab, G. Sullivan, B. C. Williams, Automated verification of model-based programs under uncertainty, in: Proc. ISDA-2004, 2004.

[15] D. S. Nau, S. K. Gupta, W. C. Regli, Manufacturing - operation planning versus ai planning, in: Integrated Planning Applications: Papers from the 1995 AAAI Spring Symposium, AAAI Press, 1995, pp. 92–101.

[16] D. Kiritsis, K. P. Neuendorf, P. Xirouchakis, Petri net techniques for process planning cost estimation, Advances in Engineering Software 30 (6) (1999) 375–387.

[17] H. Kühnle, Distributed Manufacturing: Paradigm, Concepts, Solutions and Examples, Springer Verlag, 2009.

[18] L. Ferrarini, C. Veber, A. Luder, J. Peschke, A. Kalogeras, J. Gialelis, J. Rode, D. Wunsch, V. Chapurlat, D. e Informazione, Control architecture for reconfigurable manufacturing systems: The PABADIS'PROMISE approach, in: Proc. ETFA-2006, 2006, pp. 545–552.

[19] A. Treytl, G. Pratl, B. Khan, Agents on RFID tags - A chance to increase flexibility in automation, in: Proc. ANIPLA-2006, 2006.

[20] A. Bratukhin, B. Khan, A. Treytl, Resource-oriented scheduling in the distributed production, in: Proc. Industrial Informatics, Vol. 2, 2007.

[21] D. E. Wilkins, M. desJardins, A call for knowledge-based planning, AI Magazine 22 (1) (2001) 99–115.

[22] D. McDermott, A reactive plan language, Tech. rep., Yale University, Computer Science Dept. (1993).

[23] M. Beetz, Concurrent Reactive Plans: Anticipating and forestalling execution failures, Vol. 1772 of Lecture Notes in Artificial Intelligence, Springer Publishers, 2000.

[24] G. M. Coghill, G. Wu, Incremental state based diagnosis, Advanced Engineering Informatics 23 (3) (2009) 309–322.

[25] T. Henzinger, The theory of hybrid automata, in: Proc. LICS-1996, IEEE CompSoc Press, New Brunswick, New Jersey, 1996, pp. 278–292.

[26] B. C. Williams, S. Chung, V. Gupta, Mode estimation of model-based programs: Monitoring systems with complex behavior, in: in: Proceedings of International Joint Conference on Artifical Intelligence 2001, 2001, pp. 579–590.

[27] M. W. Hofbaur, B. C. Williams, Hybrid estimation of complex systems, IEEE Transactions on Systems, Man, and Cybernetics, Part B 34 (5) (2004) 2178–2191.

[28] X. Koutsoukos, J. Kurien, F. Zhao, Monitoring and Diagnosis of Hybrid Systems Using Particle Filtering Methods, in: Proc. MTNS-2002, Citeseer, University of Notre Dame, 2002.

[29] J. Lunze, B. Nixdorf, Representation of hybrid systems by means of stochastic automata, Mathematical and Computer Modelling of Dynamical Systems 7 (2001) 383–422(40).

[30] T. Schiex, H. Fargier, G. Verfaillie, Valued constraint satisfaction problems: Hard and easy problems, in:

C. Mellish (Ed.), Proc. IJCAI-1995, Montreal, 1995, pp. 631–637.
[31] G. Frehse, Phaver: Algorithmic verification of hybrid systems past hytech, in: Proc. HSCC-2005, LNCS, Springer, 2005, pp. 258–273.
[32] M. Althoff, O. Stursberg, M. Buss, Online Verification of Cognitive Car Decisions, in: Proc. IV-2007, 2007, pp. 728–733.
[33] M. W. Hofbaur, T. Rienmüller, Qualitative Abstraction of Piecewise Affine Systems, in: Workshop Proc. QR-2008, 2008, pp. 43–48.
[34] J. Kurien, P. P. Nayak, Back to the future for consistency-based trajectory tracking, in: Proc. AAAI-2000, AAAI Press, 2000, pp. 370–377.
[35] T. Mikaelian, C. B. Williams, M. Sachenbacher, Model-based Monitoring and Diagnosis of Systems with Software-Extended Behavior, in: Proc. AAAI-2005, AAAI Press, Pittsburgh, USA, 2005, pp. 327–333.
[36] S. Bouveret, F. Heras, S. de Givry, J. Larrosa, M. Sanchez, T. Schiex, Toolbar: a state-of-the-art platform for wcsp (2004).
[37] K. Kask, R. Dechter, Mini-bucket heuristics for improved search, in: Proc. UAI-1999, Morgan Kaufmann, San Francisco, CA, 1999, pp. 314–323.
[38] L. Kuhn, B. Price, J. de Kleer, M. B. Do, R. Zhou, Pervasive diagnosis: the integration of diagnostic goals into production plans, in: D. Fox, C. P. Gomes (Eds.), Proc. AAAI-2008, AAAI Press, 2008, pp. 1306–1312.