Diagnostic Hypothesis Enumeration vs. Probabilistic Inference for Hierarchical Automata Models

Paul Maier [1], **Dominik Jain** [1], **Martin Sachenbacher** [1]

[1] *Technische Universität München, Department of Informatics, Boltzmannstraße 3, 85748 Garching, Germany*
{*maierpa,jain,sachenba*}*@in.tum.de*

ABSTRACT

AI problems in engineering domains often lie at the intersection of model-based diagnosis and probabilistic reasoning. A typical example is the plan assessment problem studied in this paper, which comprises the identification of possible faults and the computation of remaining success probabilities based on a model. In addition, solutions to such problems need to be tailored towards the needs of engineers, and thus use high-level, expressive modeling formalisms such as probabilistic hierarchical constraint automata (PHCA).

This work introduces a translation from PHCA models to lower-level Bayesian nets, which enables to compare model-based diagnosis approaches with a wide array of probabilistic reasoning methods. Using a state-of-the-art probabilistic solver, we compare this approach to an alternative model-based diagnosis approach that translates the PHCA models to lower-level logic models and computes solutions by enumerating most likely hypotheses. Experimental results on realistic problem instances demonstrate that the probabilistic reasoning approach is a promising alternative to the model-based diagnosis approach.

INTRODUCTION

In engineering domains, one is increasingly confronted with AI problems that lie at the intersection of the fields of model-based diagnosis and probabilistic reasoning. Furthermore, both model-based diagnosis and probabilistic reasoning follow a trend towards rich and expressive formalisms, auto-generating low-level representations such as Bayesian networks (BNs), which can then be fed to off-the-shelf tools. This has three advantages: 1) It spares users the tedious effort of hand crafting low-level models and re-implementing algorithms, 2) the algorithms implemented by said tools seek to exploit problem structure in a most general fashion and 3) they are typically supported by large communities. Probabilistic hierarchical constraint automata (PHCA) (Williams, Chung, and Gupta, 2001) are a first-order model-based diagnosis formalism specifically tailored to compactly represent complex hard- and software behavior, addressing the needs of engineers. Following the approach of using off-the-shelf tools, (Mikaelian, Williams, and Sachenbacher, 2005) provides an automatic translation for constraint optimization, which is common in model-based diagnosis. However, there is still a gap between first-order modeling on the model-based diagnosis side and the methods and frameworks on the probabilistic reasoning side.

This work aims to provide a possibility to compare solution approaches from the model-based diagnosis side, typically logic-based, with approaches from the probabilistic reasoning side. We contribute a translation of first-order PHCA models to statistical relational models (a first-order generalisation of graphical models such as Bayesian nets). This opens up the opportunity to choose among a wide range of probabilistic reasoning off-the-shelf tools. To our knowledge, it is the first such translation from PHCAs to a first-order probabilistic reasoning framework. We evaluate our approach within the context of a novel problem called *plan assessment* (Maier et al., 2010). We translate different PHCAs and feed the resulting BNs to the publicly available probabilistic reasoning tool Ace 2.0 (Darwiche, 2003) in order to solve this problem and compare the results with an existing model-based diagnosis approach for this problem (Maier et al., 2010), which builds on the afore mentioned PHCA translation for constraint optimization.

The problem of plan assessment arises when technical systems, in particular manufacturing systems, shall be equipped with the necessary autonomous behavior to automatically cope with contingencies. For example, in a manufacturing scenario, the AI controller of a factory plant needs to take decisions depending on the success or failure of the production of individual products. The manufacturing of products is automatically scheduled in advance, specifying which actions to perform where and when. During production, model-based diagnosis computes most likely diagnostic hy-

potheses for given observations (Kurien and Nayak, 2000). For informed decisions, however, the controller additionally needs the probability with which the production of a particular item will succeed given the observations. This is the classical probabilistic reasoning problem of computing posterior marginals. This problem of computing most likely hypotheses and, at the same time, probabilities for certain model states is a variant of the plan assessment problem introduced in (Maier et al., 2010).

The remainder of this section discusses some related work and details the example scenario used throughout the paper. The next three sections formally define plan assessment, recap the model-based diagnosis solution and describe our novel translation along with the probabilistic reasoning solution based upon it. The final sections discuss the evaluation results and conclude the paper.

Related work While there has been work on combining model-based diagnosis and probabilistic reasoning (Abreu, Zoeteweij, and Van Gemund, 2009; Knox and Mengshoel, 2009), a general bridge between the two areas, such as our translation, which allows the comparison of model-based diagnosis and probabilistic reasoning solutions, has not yet been developed. The field of probabilistic model checking (Rutten et al., 2004), which is close to plan assessment, can answer queries such as "What is the probability that the system reaches state s?", based on a model of the system. However, plan assessment is ultimately geared towards supporting robust online control, which, unlike probabilistic model checking, includes generating diagnoses and focussing computation using available observations.

Example: metal machining and assembly Within the aforementioned manufacturing scenario, products (toy mazes) are first being machined and then assembled. Two different faults can flaw products. The cutter of machining stations might break, damaging the alloy base plate (see Fig. 1d). This damage later on triggers an observable alarm at the assembly station. That same alarm might also be triggered if the assembly station is miscalibrated (the second fault). The question is: Given the alarm, how are the manufacturing goals affected, and what faults could have caused the alarm? We modeled several slight variations of this scenario. Fig. 1a shows a PHCA that models an assembly and two machining stations, as well as three products which are to be produced. Sub-PHCA model the stations' (nominal and known fault) behavior as well as the production processes for each scheduled product. Stations, or generally components, and products are connected via shared variables. The connections are called product-component-links and are derived from the schedule. Fig. 1b and 1c show detailed versions of the two machining stations.

PHCA-BASED PLAN ASSESSMENT

Plan assessment extends the maximum probability diagnosis problem (Sachenbacher and Williams, 2004) towards additionally computing success probabilities for given goals. The problem arises when three aspects come together: 1) A rigidly designed system with some remaining uncertainties that would be hard to eliminate executes 2) pre-planned operations to 3) achieve defined goals. These aspects are reflected in three formal elements: 1) A system model, in our case a PHCA M_{PHCA}, 2) an operation sequence $\mathcal{S}$ of operation steps, and 3) a set of goals $\{\mathcal{G}_i\}$.

System Model PHCA models define automata states, called *locations*, and transitions between them. The transitions may be guarded and probabilistic, locations may be composed of sub-locations. PHCAs are formally defined as follows (Mikaelian, Williams, and Sachenbacher, 2005):

Definition. A PHCA is a tuple $(\Sigma, P_\Xi, \Pi, O, \mathit{Cmd}, \mathcal{C}, P_T)$:

- $\Sigma = \Sigma_p \uplus \Sigma_c$ is a set of locations, partitioned into *primitive locations* Σ_p and *composite locations* Σ_c, where a composite location represents a PHCA whose elements are subsets of the elements of the containing PHCA. A location may be marked or unmarked. A marked location represents an active execution branch. A *marking* m^t (at time t during execution) is given as a subset of Σ.
- $P_\Xi(\Xi_i)$ denotes the probability that $\Xi_i \subseteq \Sigma$ is the set of start locations (initial state).
- $\Pi = O \uplus \mathit{Cmd} \uplus \mathit{Dep}$ is a set of variables with finite domains. O is the set of observation variables, *Cmd* is the set of action or command variables, and *Dep* is a set of dependent hidden variables which are connected to other variables via constraints.
- $\mathcal{C}$ is the set of finite domain constraints defined over Π, which comprises behavior constraints of locations and guard constraints of transitions. A location's behavior constraint serves to define the observations that are consistent with the location; a transition's guard defines the conditions under which the transition can be taken (usually depending on commands).
- $P_T[l_i]$ (defined for each $l_i \in \Sigma_p$) is a probability distribution over the subset of the set of transitions $\mathcal{T}$, which contains transitions leading away from l_i whose guards (elements of $\mathcal{C}$) are satisfied given the current state.

Given a PHCA model M_{PHCA} of a technical system, the behavior of the system over time is estimated by generating sequences of *location markings* $\theta = (m^{t_0}, m^{t_1}, \ldots, m^{t_N})$. We denote by $\mathit{St}(M_{\mathrm{PHCA}})$ the set of all such fixed-length sequences, called *trajectories*.

Operation Sequences Pre-planned operations are typically given as a sequence $\mathcal{S}$ of operation steps. Steps can be anything from simple (sets of) commands to more complicated operations, such as scheduled allocations of machines, products and actions to perform. We consider scenarios where $\mathcal{S}$ is synthesized automatically (using, e.g., a planner or a scheduler),

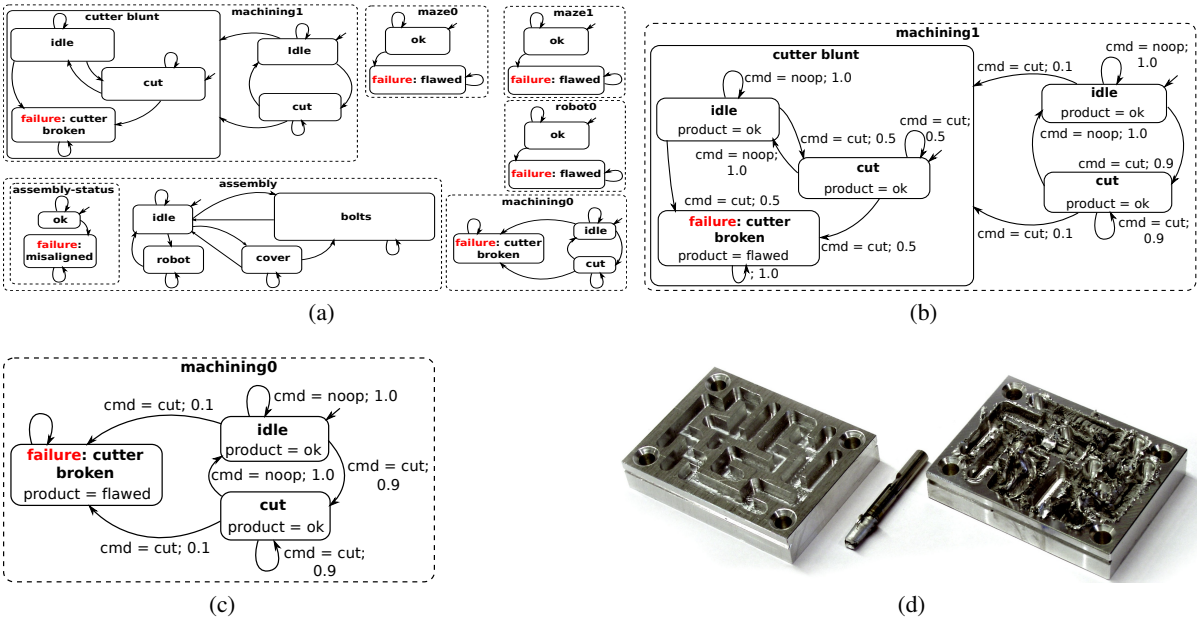

Figure 1: (1a) Overview of a typical PHCA. It models a factory scenario with three individual products being manufactured. (1c and 1b) Detail views of sub-automata machining0, with simple probabilistic behavior, and machining1, showing a more complex hierarchical machining fault: the cutter goes blunt before breaking. (1d) A product damaged during machining.

and later executed on the system. This is captured by an *execution adaptation function* $\mathcal{E}_{\mathcal{S}}$, which adapts a PHCA model M^N_{PHCA} unfolded for N time steps (reproducing the PHCA's components for each time step t, yielding Σ^t, Π^t, $\mathcal{T}^t$ and $\mathcal{C}^t$). It sets all command variables as given by $\mathcal{S}$, removing the transitions whose guards become unsatisfiable as a result, and modifies the constraints $\mathcal{C}^t$. The resulting model $M^{\mathcal{S}}_{\text{PHCA}} = \mathcal{E}_{\mathcal{S}}(M^N_{\text{PHCA}})$ therefore no longer contains the command variables *Cmd* and will typically feature a reduced number of possible trajectories. Markov properties of M_{PHCA} are unaffected.

Each sequence of observations $\mathbf{o}^{0:t}$ $(t \leq N)$ and each model $M^{\mathcal{S}}_{\text{PHCA}}$ defines a joint distribution $P(\theta, \mathbf{O} = \mathbf{o}^{0:t})$:

$$P(\theta, \mathbf{O} = \mathbf{o}^{0:t}) \quad = \quad P_{\Xi}(m^0) \prod_{u \in \{0..t\}} P(\mathbf{O}^u \mid m^u) \prod_{\tau \in \mathcal{T}[\theta]} P(\tau) \qquad (1)$$

where $\mathcal{T}[\theta]$ is the multiset of all transitions as implied by θ, in which a transition τ from location l_i to l_j may occur multiple times; the transition probability is computed as $P(\tau) = P_T[l_i](\tau)$. The above distribution corresponds to the PHCA hidden Markov model semantics (Mikaelian, Williams, and Sachenbacher, 2005; Williams, Chung, and Gupta, 2001).

Goals A goal is a tuple (l, t) of a location l that should be marked at time t. $\mathcal{G}_i := \{(m^{t_0}, \ldots, m^t, \ldots, m^{t_N}) \mid l \in m^t\}$ denotes the set of goal-achieving trajectories, i.e. which lead to the marking required by the goal. As goal for a product we use (Ok, t) (see product model in Fig. 1a), following the assumption that the product is produced successfully after all scheduled operations if no station was faulty while working the product. t is the discrete time point in the schedule after the last operation.

Definition. Let M_{PHCA} be a PHCA model, $\mathcal{S}$ a sequence of N operation steps with execution adaptation function $\mathcal{E}_{\mathcal{S}}$ and $\{\mathcal{G}_i\}$ a set of goals. Let $M^{\mathcal{S}}_{\text{PHCA}} = \mathcal{E}_{\mathcal{S}}(M^N_{\text{PHCA}})$. The **plan assessment problem** is, given observations $\mathbf{o}^{0:t}$, to compute the most probable diagnosis as a trajectory in $St(M^{\mathcal{S}}_{\text{PHCA}})$ as well as $P(\mathcal{G}_i \mid \mathbf{o}^{0:t})$ for each i.

Solving Plan Assessment

Model representation always strongly influences problem solving efficiency. Often, Markov modelling is used to exploit the Markov property during inference. However, compared to more expressive non-Markov modeling this can lead to bigger representations. This size-vs-expressiveness trade-off impacts even more when automatically generating such representations from first-order models. Originally, PHCA semantics were defined in terms of hidden Markov models (HMM) (Williams, Chung, and Gupta, 2001), which allows complex Markov modeling at the cost of potentially larger models. This is problematic for existing off-the-shelf HMM solvers: On the one hand, a naive automated flattening of hierarchical models would generate prohibitively large HMMs. Automatically decomposing PHCAs into explicit, tractable HMM representations, on the other hand, is only possible in a small number of special cases where components are not connected. Our example is not such a case: The assembly and manufacturing stations seem independent of each other; however, they are connected via product models and their scheduled actions.

In addition, automatic compilation typically incurs an overhead compared to custom tailored translation, which further increases the size of explicit HMMs. Consequently, it is not usually advisable to encode the translation result as explicit HMMs: We would easily obtain hidden variables with domain sizes of $\approx 2^{100}$, thus requiring a $2^{100} \times 2^{100}$ transition matrix!

For these reasons, we apply a different approach. Using a translation Υ we translate, in an offline step, M_{PHCA} to low-level representations M that are still expressive enough to represent structure and are sufficiently general to allow the use of off-the-shelf solvers that do not depend on the Markov property but exploit model structure in a general fashion. We define execution adaptation functions $\mathcal{E}_{\mathcal{S}}$ on these representations, which allows to reuse the translation for different operation sequences. We now describe two solutions for plan assessment that we evaluate in this work.

HYPOTHESIS ENUMERATION

Recent work developed a model-based diagnosis inspired solution that first computes the most likely trajectories as the best solutions to a constraint optimization problem (COP) and then computes $P(\mathcal{G}_i \mid \mathbf{o}^{0:t})$ by summing over goal-achieving trajectories among these, normalizing over all (Maier et al., 2010). The most probable trajectory is chosen as most probable diagnosis. The work exploits an automatic translation of a PHCA to a COP $\mathcal{R} = (X, D, C)$ (Mikaelian, Williams, and Sachenbacher, 2005). We term this translation Υ_{COP}, which maps a model M_{PHCA} to variables X, their finite domains D and local objective functions $c \in C$, called soft constraints. Soft constraints map partial variable assignments to $[0, 1]$. Execution adaptation functions modify $\mathcal{R}$ by adding soft constraints. We refer to (Mikaelian, Williams, and Sachenbacher, 2005) for the details.

Plan assessment is solved for a translated and adapted model $\mathcal{E}_{\mathcal{S}}(\Upsilon_{\text{COP}}(M_{\text{PHCA}}))$ as follows. Each COP solution of the translation corresponds to a trajectory θ of M_{PHCA}. A COP solution is an assignment to all variables in X. $P(\mathcal{G}_i \mid \mathbf{o}^{0:t})$ can be computed as $P(\mathcal{G}_i \mid \mathbf{o}^{0:t}) = \sum_{\theta \in \mathcal{G}_i} P(\theta \mid \mathbf{o}^{0:t}) = \frac{\sum_{\theta \in \mathcal{G}_i} P(\theta, \mathbf{o}^{0:t})}{\sum_{\theta \in \Theta} P(\theta, \mathbf{o}^{0:t})}$, i.e. $P(\mathcal{G}_i \mid \mathbf{o}^{0:t})$ can be computed based on probabilities $P(\theta, \mathbf{o}^{0:t})$, which can be retrieved from objective values of the COP solutions. The computation is exact if all trajectories with non-zero probability $\Theta \subseteq St(M_{\text{PHCA}})$ are enumerated. This can become intractable quickly, therefore we approximate $P(\mathcal{G}_i \mid \mathbf{o}^{0:t})$ by resorting to the subset $\Theta(k) \subseteq \Theta$ of the k best (most probable) trajectories and the set of goal-achieving trajectories among them, $\mathcal{G}_i(k) \subseteq \Theta(k)$.

The key motivation for this approach is that strong off-the-shelf constraint optimization solvers can be exploited. While (Maier et al., 2010) used the Toolbar solver, in this work, we use the more recent, award-winning Toulbar2 solver[1], which implements many state-of-the-art techniques (including, for example, soft-constraint consistency (Cooper et al., 2008)).

[1] https://mulcyber.toulouse.inra.fr/projects/toulbar2 (02/2011)

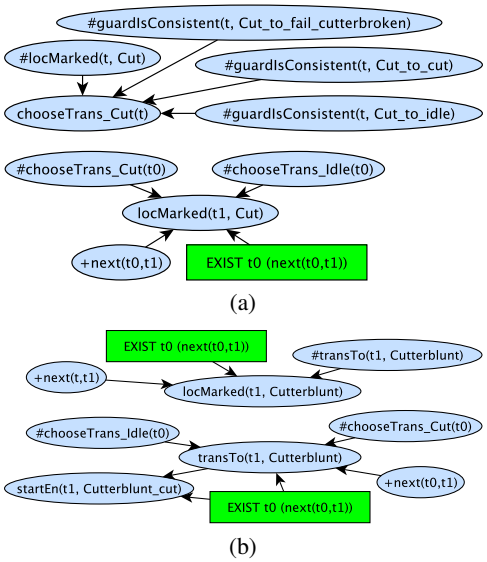

Figure 2: Figures 1c and 1b translated to BLN fragments, showing excerpts of location marking and probabilistic transition choice (2a) as well as composite and full target marking (2b).

For k-best enumeration, we replaced Toulbar2's core branch-and-bound procedure with a k-best branch-and-bound algorithm (Dechter and Flerova, 2011).

PROBABILISTIC INFERENCE

From a probabilistic reasoning point of view, plan assessment can be seen as a combination of computing most likely a posteriori hypotheses (MAP) and marginals. MAP and the computation of marginals are long time standard problems in probabilistic reasoning with existing off-the-shelf solutions. Bayesian networks (BNs) are a common representation for probabilistic models. We now describe a new translation of PHCA models to abstract, generalized Bayesian networks, which can in turn be automatically instantiated to BNs. This opens up the possibility of comparing the COP-based approach from the previous section with solutions from the probabilistic reasoning community. Once translated, $P(\mathcal{G}_i \mid \mathbf{o}^{0:t})$ can be computed as the posterior marginal of a goal variable (BN node).

We translate PHCAs into Bayesian logic networks (BLNs) (Jain, Waldherr, and Beetz, 2009). A BLN is a combination of Bayesian networks with first-order logic. These types of models are also known as statistical relational models. A BLN can be seen as a template for the construction of Bayesian networks, much like a first-order logic model can be seen as template for propositional models. Given a set of entities, a BLN may be instantiated to either a BN or a mixed network (MN) that explicitly represents logical constraints on the distribution (Mateescu and Dechter, 2008). The instantiation is called grounding, and the resulting BNs or MNs are called ground models. Probabilistic inference is often performed in these. However, the strength of relational models is a very compact representation of repeated substructures in ground models. This is exploited in recent research, which de-

velops inference algorithms that work directly on the relational models (Poole, 2003). So besides the added expressivity and flexibility offered by a first-order formalism, the possibility of exploiting first-order encodings is one of our main reasons for seeking a relational translation rather than a direct encoding to BNs.

Key elements of a BLN $M_{\mathcal{B}}$ are abstract random variables (ARVs), typed entities, fragments and first-order logical (FOL) formulas. ARVs are parametrized random variables that correspond to either predicates or non-boolean functions. They encode, for example, partial states such as a station being faulty, relations such as stations working on specific products or a probabilistically chosen transition. The arguments of an ARV refer to abstract, typed entities, which allows, in particular, quantification over time. A fragment associates an ARV with a conditional probability table (CPT); the set of fragments (ellipses in Fig. 2) collectively defines, for a set of ground instances of ARVs, a probability distribution. The applicability of a fragment may be restricted by (mutually exclusive) first-order logic preconditions (boxes in Fig. 2 as well as ellises with special symbol '+'). Finally, hard logical formulas may be specified in the model, restricting the set of possible worlds. When instantiating a BLN to a BN, a grounding (X, D, G, P) is created. It consists of random variables X, their corresponding set of domains D, a graph G connecting variables according to parent-child condition relations defined via the fragments, and the set of CPTs P.

For our novel translation, we adapted the COP encoding Υ_{COP} of PHCAs. This encoding is defined in terms of formal higher-order rules for structure, probabilistic behavior and consistency with observations and commands (Mikaelian, Williams, and Sachenbacher, 2005). The translation to BLNs largely follows these rules, however often exploits first-order modeling features of BLNs. The translation function Υ_{BLN} takes as input a model M_{PHCA} and creates a BLN $M_{\mathcal{B}} = (\mathcal{D}, \mathcal{F}, \mathcal{L})$ and a knowledge base DB. $M_{\mathcal{B}}$ consists of the fundamental declarations $\mathcal{D}$, the set of fragments $\mathcal{F}$ and the set of FOL formulas $\mathcal{L}$. The knowledge base defines existing objects or entities for the FOL formulas and fragments as well as known facts about relations among these entities. When the BLN is grounded, DB is extended with further evidence. Execution adaptation functions $\mathcal{E}_{\mathcal{S}}$ for BLNs add formulas to $\mathcal{L}$ and facts to DB. $\mathcal{D}$ contains, among other things, the entity types and predicate signatures for ARVs. Most of $\mathcal{D}$ is model-independent, i.e. stays the same for arbitrary M_{PHCA}. Formulas in $\mathcal{L}$ are, in particular, used to define behavior and transition guard consistency predicates in terms of formulas over assignments of PHCA variables O and *Cmd*. An example of a concrete behavior constraint is `behaviorIsConsistent(t,Cutter_broken) <=> var_PRODUCT(t,Faulty)`. It specifies that the behavior of location "broken" (which is part of composite location "cutter") is consistent if and only if the product being processed will be broken in the next time step. In addition, $\mathcal{L}$ contains the general **behavior consistency** rule: `locMarked(t,l) => behaviorIsConsistent(t,l).` It says that for all points in time, a location's behavior must be consistent if it is marked. Note that this rule suffices to connect locations and their behavior, the $behaviorIsConsistent()$ predicate will not appear among the fragments.

Next, we look at target marking, i.e. the marking of locations that are enabled start locations or that are transitioned to. In general, the predicate $\mathrm{locMarked}(t, l)$ (abbreviated as $lm(t, l)$) encodes a location l being marked at time t. We first treat the marking of composite locations. The **composite target marking** rule marks a composite location if it is transitioned to or if it is an enabled start location:

$$\forall t \in \{1..N\}.\ \forall l_c \in \Sigma_c.$$
$$transTo(t, l_c) \vee startEnabled(t, l_c) \Rightarrow lm(t, l_c)$$

Note that the marking for $t = 0$ is treated separately. The $transTo$ predicate is defined as follows,

$$\forall t_0 \in \{0..N-1\}.\ \forall t_1 \in \{1..N\}.\ \forall l_c \in \Sigma_c.$$
$$next(t_0, t_1) \Rightarrow (transTo(t_1, l_c) \Leftrightarrow$$
$$\exists l \in parents(l_c).\ target(chooseTrans(t_0, l)) = l_c),$$

where the function $target$ maps transitions to their target locations and $parents$ maps a location to the set of locations connected to it via transitions.

Using the location *Cutterblunt* as an example, we show how the composite marking rule is translated into fragments. Generally, each predicate in a rule corresponds to one fragment. The fragment is created only if the translator cannot determine, e.g. from the model structure, that its predicate is always True or False. Predicates are partially instantiated, removing all quantification except over time. In case of *Cutterblunt*, fragments for partially instanciated predicates $transTo(t, \textit{Cutterblunt})$ and $lm(t, \textit{Cutterblunt})$ are created. No fragment is created for $startEnabled(t, \textit{Cutterblunt})$ because *Cutterblunt* is not a start location and the predicate is thus always False. The following table shows the CPT template for $lm(t, \textit{Cutterblunt})$:

$transTo(t, \textit{Cutterblunt})$	T	F
$lm(t, \textit{Cutterblunt}) = T$	1	0.5
$lm(t, \textit{Cutterblunt}) = F$	0	0.5

The CPT encodes that *Cutterblunt* is marked if it is being transitioned to. If not, the CPT doesn't influence the marking. See Fig. 2b for the partial fragment network.

The visual BLN coding uses '+' to declare a special precondition. The child fragment of such a parent only applies if the parent evaluates to True. E.g., $lm(t_1, \textit{Cutterblunt})$ only applies if $next(t, t_1)$ is True. '#' declares a reference to an existing fragment elsewhere in the BLN, allowing cleaner arrangements.

The second rule that influences the marking of composite locations is the **hierarchical marking/unmarking** rule, which ensures that a composite location is marked iff at least one of its sub-locations (which are given by function sub) is marked:

$$\forall t \in \{0..N\}.\ \forall l_c \in \Sigma_c.\ lm(t, l_c) \Leftrightarrow \exists l \in sub(l_c).\ lm(t, l)$$

This rule can be directly translated using logical formulas that we can add to $\mathcal{L}$.

The **primitive target marking** rule marks primitive locations if they are either transitioned to or if they are enabled starting locations:

$\forall t_0 \in \{0..N-1\}.\ \forall t_1 \in \{1..N\}.\ \forall l_p \in \Sigma_p.\ \exists l \in parents(l_p).$
$next(t_0, t_1) \Rightarrow (target(chooseTrans(t_0, l)) = l_p \vee startEnabled(t_1, l_p) \Leftrightarrow lm(t_1, l_p))$

The **full target marking** rule ensures that all start sub-locations of a composite location are enabled iff this composite location is the target of a chosen transition or is itself enabled:

$\forall t \in \{1..N\}.\ \forall l_c \in \Sigma_c.\ transTo(t, l_c) \vee startEnabled(t, l_c) \Leftrightarrow \forall l \in sub(l_c).\ startEnabled(t, l)$

These two rules are handled analogously to the composite marking rule.

The central rule for probabilistic behavior is **probabilistic transition choice**. Given a primitive location, exactly one of its outgoing transitions may be chosen (according to transition probabilities defined in the model) iff the location is marked and the chosen transition's guard is consistent:

$\forall t \in \{0..N\}.\ \forall l_p \in \Sigma_p.\ \exists \tau \in outgoing(l_p) \cup \{NoTransition\}.\ lm(t, l_p) \wedge guardIsConsistent(t, \tau) \Leftrightarrow chooseTrans(t, l_p) = \tau \wedge \tau \neq NoTransition$

In the formula, function $chooseTrans(t, l_p)$ maps time and location to an admissible outgoing transition. The translation creates BLN functions of time only, eliminating quantification over l_p (see, e.g., Fig. 2a and 2b). Their CPTs define the following probability function:

$$Pr(T^t_{l_p} = \tau \mid L^t_p, \mathbf{G}^t) = \begin{cases} p_\tau & \text{if (a)} \\ 1 & \text{if (b)} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where $T^t_{l_p}$ is a random variable for choosing among l_p's outgoing transitions, L^t_p encodes $lm(t, l_p)$ and $\mathbf{G}^t$ is a vector of random variables encoding $guardIsConsistent(t, \tau)$ for each outgoing transition. Condition (a) is $lm(t, l_p) \wedge guardIsConsistent(t, \tau)$ and (b)[2] is $(\neg lm(t, l_p) \vee (\neg\exists \tau' \in outgoing(l_p).\ guardIsConsistent(t, \tau'))) \wedge \tau = NoTransition$. Other cases, e.g. $chooseTrans()$ returning an inconsistent transition, are ill-conditioned and hence yield probability 0. Note that this fragment also encodes **guard consistency**, which is more compact than having a separate logical rule as for behavior consistency.

We now address the issue of incorporating the constraints added by $\mathcal{E}_\mathcal{S}$. In our implementation, $\mathcal{S}$ is a schedule, i.e. a sequence of tuples $\langle (p, c, t, a) \rangle_j$, each defining for time t two entities p and c being connected and an action a to perform. In our example, a is a command to be executed on station c, which works product p. We call these tuples *product-component links*. We can exploit the flexibility of the BLN framework to encode these: For any tuple (p, c, t, a), logical formulas are added to $\mathcal{L}$ that enforce equality of variables $\{X^t_p\}$ and $\{X^t_c\}$. Those variables and formulas encode the connection between p and c for time t. The common domain of two variables X^t_p and X^t_c encodes influences, e.g. *Faulty* for a station c inflicting damage on a product p or a faulty p causing unusual observations in c. *Ok* encodes no (harmful) influence. An example is the following rule in BLN code: `var_MAZE0_WORKER(T1,x) <=> var_MACHINING1_PRODUCT(T1,x).` The `var_...` predicates encode variable assignments, e.g. x being assigned to PHCA variable Worker of the Maze0 product at time $T1$. Commands given as a in tuples in $\mathcal{S}$ are simply added as facts to DB.

We have not looked in detail at rules for the initial time point t_0, which encode which locations are initially marked/unmarked. Special t_0 rules encode the marking and unmarking as given by the start distribution $P_\Xi(\Xi_i)$ and handle special conditions for hierarchical marking, e.g., that at t_0 locations cannot be transitioned to. For more details we refer to (Mikaelian, Williams, and Sachenbacher, 2005).

Translation Correctness

We say that the translation is correct iff the BN given by translating M_{PHCA} to a BLN and grounding it, encodes the same distribution over variables L^t_i (that encode location markings at time points t) as PHCA distribution $P(\theta, \mathbf{O} = \mathbf{o}^{0:t})$. Variable L^t_i being True corresponds to location l_i being marked, i.e. $l_i \in m^t$.

Theorem. *Let $M_\mathcal{B} = (\mathcal{D}, \mathcal{F}, \mathcal{L})$ be a BLN generated with the above described translation process from a PHCA model: $M_\mathcal{B} = \Upsilon_{\text{BLN}}(M_{\text{PHCA}})$. Let $\mathcal{E}_{\text{BLN}}$, $\mathcal{E}_{\text{PHCA}}$ be the execution adaptation functions for an arbitrary operation sequence, BN (X, D, G, P) the grounding of $\mathcal{E}_{\text{BLN}}(M_\mathcal{B})$, and $\theta = (m^{t_0}, m^{t_1}, \ldots, m^{t_N})$ an arbitrary trajectory of the adapted PHCA $\mathcal{E}_{\text{PHCA}}(M_{\text{PHCA}})$. Then*

$$P_\Xi(m^0) \prod_{u \in \{0..t\}} P(\mathbf{O}^u \mid m^u) \prod_{\tau \in \mathcal{T}[\theta]} P(\tau) \quad = \quad (3)$$

$$P(\mathbf{L}^{t_0} = \mathbf{m}^{t_0}, \ldots, \mathbf{L}^{t_N} = \mathbf{m}^{t_N}, \mathbf{O}_{\text{BN}} = \mathbf{o}^{0:t} | A = \mathbf{a})$$

$\mathbf{L}^{t_j}$ are vectors of location marking variables $L^{t_j}_i$ in the BN for each time point t_j, and $\mathbf{O}_{\text{BN}}$ is a vector of observation variables $O^{t_j}_r$ for each time point t_j (r ranges over indices of observation variables for a given time point). A is a set of auxiliary variables used to represent additional logical constraints. $\mathbf{m}^t$ is boolean vector that encodes a marking in terms of L^t_i assignments for time t.

Solving Plan Assessment with Probabilistic Reasoning Tools and Methods

Translating and executing a given PHCA, $\mathcal{E}_\mathcal{S}(\Upsilon_{\text{BLN}}(M_{\text{PHCA}}))$, yields a BLN as starting point for possible probabilistic reasoning solutions for plan assessment. In our experiments, we grounded BLN to an auxiliary BN and used the state-of-the-art inference tool Ace 2.0[3], which compiles a given BN into an arithmetic circuit (AC) (Darwiche, 2003). Ace exploits local structure given by, e.g., determinism in

[2] (b) means "don't care" if the location isn't marked or no consistent,outgoing transition exists. Also, $chooseTrans()$ is expected to return $NoTransition$.

[3] http://reasoning.cs.ucla.edu/ace/ (03/2011)

Table 1: The size of PHCAs, COP and BN translations. PHCA models fm1 and sm are taken from (Maier et al., 2010) and (Mikaelian, Williams, and Sachenbacher, 2005), respectively.

instance	N	phca size	# var	# con	# nodes
fm1	6	11/6/27	643	670	1106
fm2	9	15/8/33	1202	1251	2122
fm3	9	17/8/33	1305	1311	2292
fm2(long $\mathcal{S}$)	19	15/8/33	2482	2601	4444
fm3(long $\mathcal{S}$)	33	18/8/35	4748	4892	8878
sm	8	8/4/22	640	661	1080

the model, to achieve very compact ACs. Once an AC is given, marginals, and thus $P(\mathcal{G}_i \mid \mathbf{o}^{0:t})$, can be computed online in time linear in the size of the AC. The most probable trajectory (as a diagnosis) can be computed as the most probable a posteriori hypothesis using the algorithm AceMAP (Huang, Chavira, and Darwiche, 2006). The algorithm based on Ace is not yet part of the public distribution, therefore we focus on computing $P(\mathcal{G}_i \mid \mathbf{o}^{0:t})$ in this work. The Ace compilation is considered a (potentially expensive) offline step, the evaluation the (quick) online step. The compilation is done after $\mathcal{E}_\mathcal{S}$ has been applied. Certain applications might require to apply $\mathcal{E}_\mathcal{S}$ online, which in turn requires a non-trivial modification of $\mathcal{E}_\mathcal{S}$ such that it can be applied to the Ace compilation.

EVALUATION

Experiments We ran experiments on six different problem instances, five plan assessment instances based on factory models and one diagnosis instance based on a satellite camera model from (Mikaelian, Williams, and Sachenbacher, 2005). The factory models 2 and 3 are variations of the model shown in Fig. 1a, factory model 1 is taken from (Maier et al., 2010). All models contain one assembly and one or two machining stations. Factory models 1, 2 and 3 model one, two and three products respectively. There are some other, minor differences regarding, e.g., the sensors. Factory models 2 and 3 are used for two scenarios each, one with a short and one with a longer operation sequence $\mathcal{S}$. Finally, the diagnosis instance simulates diagnosing hardware or software faults in a satellite camera module (Mikaelian, Williams, and Sachenbacher, 2005). **Tab. 1** lists the size statistics for the instances: the number of time steps N, the PHCA size (primitive loc./composite loc./no. transitions), the number of variables and constraints in the COP translation, and the number of nodes in the BN model obtained through the BLN translation. In the following we denote the six instances by fm1, fm2, fm3, fm2(long $\mathcal{S}$), fm3(long $\mathcal{S}$) and sm, where we abbreviate "factory model i" with "fm$i$" and "satellite model" with "sm". We used a virtual machine with 2GB of memory, one core of an intel Core 2 Duo and Ubuntu Linux. For all scenarios, we computed the exact results (i.e. $k = k_{\max}$ for Toulbar2) for $P(\mathcal{G}_i \mid \mathbf{o}^{0:t})$ using both Toulbar2 and Ace 2.0. (For the diagnosis instance we defined the goal as a certain location of interest being marked). We used default options for both tools, except when it was obvious how to obtain better results. For Toulbar2, the results table shows search time (seconds), memory usage (MB) and expanded search tree nodes. For Ace, it shows compilation + evaluation time (both in seconds) and memory usage during compilation (in MB).

Table 2: Measurements for Toulbar2 and Ace solving the instances shown in Tab. 1. * Results obtained with Ace 3.0.

instance	Toulbar2	Ace
fm1	0.01 / 8 / 186	0.37 + 0.09 / 10
fm2	0.05 / 10 / 1878	0.93 + 0.18 / 90
fm3	0.32 / 10 / 5464	0.36 + 0.07 / 5
fm2(long $\mathcal{S}$)	0.65 / 19 / 11320	1128.04 + ERR / $\approx$ 900
fm3(long $\mathcal{S}$)	3.79 / 53 / 10180	(1.36 + 0.17 / –)*
sm	0.01 / 9 / 176	0.87 + 0.03 / 187

We additionally ran Toulbar2 with different settings for k to evaluate approximation quality. Fig. 3 shows graphs of these results. On the left, the absolute error $|P(\mathcal{G}_i \mid \mathbf{o}^{0:t}) - P^k(\mathcal{G}_i(k) \mid \mathbf{o}^{0:t})|$ is plotted against k for all instances ($k \in \{2, \ldots, k_{\max}\}$ on a log scale). $k_{\max}$ is the number of trajectories with probability > 0. On the right, we plot the error and the number of trajectories relative to the maximum error and $k_{\max}$, respectively, against the number of expanded nodes.

Results and Discussion When taking into account that Ace compilation must potentially be done online (as explained in the previous section), Toulbar2 outperforms Ace for most of our instances in runtime (e.g., fm2(long $\mathcal{S}$)) or memory usage (e.g., sm). Still, Ace performs well for four instances out of six, yielding exact results within 2 seconds, even including compilation. The two bigger instances with long $\mathcal{S}$ failed due to precision loss (indicated by "ERR"). With the not yet publicly available Ace 3.0, results for fm3(long $\mathcal{S}$) could be obtained, but not for fm2(long $\mathcal{S}$) within the 2GB memory limit. Interestingly, fm2 and fm2(long $\mathcal{S}$) are much more costly for Ace (compilation) than the *bigger* instances fm3 and fm3(long $\mathcal{S}$). Toulbar2 and Ace 2.0 differed slightly ($\triangle < 0.0001$) in their exact results, most likely due to precision loss or Toulbar2 using a lower bound to cut off solutions. Considering diagnosis, the good runtime results of Ace indicate that AceMAP could be competitive with Toulbar2. However, dedicated experiments need to confirm that.

For the approximations computed by Toulbar2, we observe that the error decreases non-monotonically with increased k or resource investment (search nodes). However, in our examples Toulbar2 was quick enough to enumerate *all* of the relatively few trajectories with non-zero probability (1710 in the biggest instance fm3), allowing quick exact computation.

CONCLUSION

In this work, we compared two approaches to solve a problem at the intersection of model-based diagnosis and probabilistic reasoning. This problem, called *plan assessment*, involves computing diagnoses for a technical system and the probabilities with which this system's pre-planned operations achieve their goals. To enable comparison of model-based diagnosis approaches with a wide range of probabilistic reason-

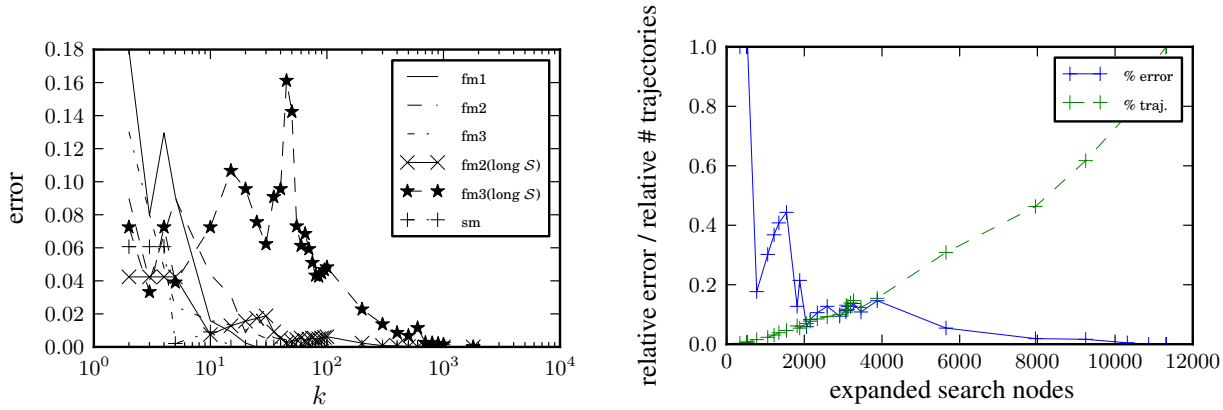

Figure 3: Results for running Toulbar2 with increasing k. (Left) k vs. the absolute error for all instances. (Right) For factory model 2 with long $\mathcal{S}$: expanded nodes vs. relative error in % and relative number of trajectories in %.

ing methods, we formalized and described the first automatic translation from the first-order model-based diagnosis framework probabilistic hierarchical constraint automata (PHCA) to a first-order probabilistic reasoning framework, Bayesian logic networks (BLNs). On six realistic instances of plan assessment we compared a), an existing solution based on generating most probable hypotheses as the best solutions of a constraint optimization problem, with b), a novel solution based on our translation, which employs the Ace 2.0 probabilistic reasoning solver. The results demonstrate that probabilistic reasoning tools such as Ace provide a strong alternative for solving plan assessment. Future work will fully exploit our novel translation by evaluating more probabilistic reasoning solutions, such as the more recent Ace 3.0, lifted inference and sampling methods.

REFERENCES

(Abreu, Zoeteweij, and Van Gemund, 2009) Abreu, Rui, Peter Zoeteweij, and Arjan J. C. Van Gemund (2009). A new Bayesian Approach to Multiple Intermittent Fault Diagnosis. In *Proc. IJCAI-2009*, pp. 653–658, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

(Cooper et al., 2008) Cooper, M., S. De Givry, M. Sanchez, T. Schiex, and M. Zytnicki (2008). Virtual Arc Consistency for Weighted CSP. In *Proc. AAAI*, pp. 253–258. AAAI Press.

(Darwiche, 2003) Darwiche, Adnan (2003). A Differential Approach to Inference in Bayesian Networks. *Journal of the ACM* 50(3): 280–305.

(Dechter and Flerova, 2011) Dechter, Rina and Natalia Flerova (2011). Heuristic search for m best solutions with applications to graphical models. In *Proc. Soft-2011 (a workshop of CP 2011)*.

(Huang, Chavira, and Darwiche, 2006) Huang, Jinbo, Mark Chavira, and Adnan Darwiche (2006). Solving MAP Exactly by Searching on Compiled Arithmetic Circuits. In *Proc. AAAI-2006*, pp. 143–148. AAAI, AAAI Press.

(Jain, Waldherr, and Beetz, 2009) Jain, Dominik, Stefan Waldherr, and Michael Beetz (2009). Bayesian Logic Networks. Technical report, Technische Universität München.

(Knox and Mengshoel, 2009) Knox, Bradley and Ole Mengshoel (2009). Diagnosis and Reconfiguration using Bayesian Networks: An Electrical Power System Case Study. In *Workshop Proc. SAS*.

(Kurien and Nayak, 2000) Kurien, James and P. Pandurang Nayak (2000). Back to the Future for Consistency-Based Trajectory Tracking. In *Proc. AAAI*, pp. 370–377. AAAI Press.

(Maier et al., 2010) Maier, Paul, Martin Sachenbacher, Thomas Rühr, and Lukas Kuhn (2010). Automated Plan Assessment in Cognitive Manufacturing. *Adv. Eng. Informat.* 24(3): 241–376.

(Mateescu and Dechter, 2008) Mateescu, R. and R. Dechter (2008). Mixed Deterministic and Probabilistic Networks. *Annals of Mathematics and Artificial Intelligence* 54(1): 3–51.

(Mikaelian, Williams, and Sachenbacher, 2005) Mikaelian, Tsoline, C. Brian Williams, and Martin Sachenbacher (2005). Model-Based Monitoring and Diagnosis of Systems with Software-Extended Behavior. In *Proc. AAAI*, Pittsburgh, USA. AAAI Press.

(Poole, 2003) Poole, David (2003). First-Order Probabilistic Inference. In *Proc. IJCAI-2003*, pp. 985–991.

(Rutten et al., 2004) Rutten, J., M. Kwiatkowska, G. Norman, and D. Parker (2004). *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*, Vol. 23 of *CRM Monograph Series*. American Mathematical Society.

(Sachenbacher and Williams, 2004) Sachenbacher, Martin and Brian Williams (2004). Diagnosis as Semiring-based Constraint Optimization. In *Proc. ECAI-2004*, Valencia, Spain.

(Williams, Chung, and Gupta, 2001) Williams, Brian C., Seung Chung, and Vineet Gupta (2001). Mode Estimation of Model-Based Programs: Monitoring Systems with Complex Behavior. In *Proc. IJCAI-2001*, pp. 579–590.