Compiling AI Engineering Models for Probabilistic Inference⋆

Paul Maier and Dominik Jain and Martin Sachenbacher

Deparment of Informatics, Technische Universität München

Abstract. In engineering domains, AI decision making is often confronted with problems that lie at the intersection of logic-based 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, AI 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 statistical relational models, which enables a wide array of probabilistic reasoning solutions to be leveraged (e.g., by grounding to problem-specific Bayesian networks). We illustrate this approach for the plan assessment problem, and compare it to an alternative logic-based 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 logic-based approach.

Introduction

In engineering domains, AI decision making is often confronted with problems that lie at the intersection of model-based diagnosis and probabilistic reasoning. 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 hypotheses for given observations [1], and thus provides the controller with a global view of the plant. To take the aforementioned decisions, however, it additionally needs a local view indicating, for example, the probability with which the production of a particular item will succeed given the observations. For example, a product might be predicted to be successfully manufactured with 0.35 probability. This is the classical probabilistic reasoning problem of computing posterior marginals. The problem of computing most likely hypotheses and, at the same time, probabilities for certain model states is a variant of the *plan assessment problem* [2].

Model-based diagnosis is a technique that supports AI decision making by identifying faulty components of a technical system, typically based on logical reasoning, while probabilistic reasoning is the classical AI technique for decision support. A key aspect

---

⋆ Preprint submitted to KI 2011.

in applying these methods is modeling. 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) [3] 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, [4] provides an automatic translation for constraint optimization, which is common in model-based diagnosis. This work was extended in [2] towards solving plan assessment. However, there is still a gap between high-level modeling on the model-based diagnosis side and the methods and frameworks on the probabilistic reasoning side.

This work aims to provide engineers with a way of applying the tools of probabilistic reasoning to the novel engineering problem of plan assessment. 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 high-level probabilistic reasoning framework. We evaluate our approach by translating different PHCAs and feeding the resulting BNs to the publicly available probabilistic reasoning tool Ace 2.0 [5, 6] in order to solve the plan assessment problem. We compare the results with the aforementioned model-based diagnosis approach.

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. Closely related is [7], which solves plan assessment for a hand-crafted BLN model. While there has been work on combining model-based diagnosis and probabilistic reasoning [8, 9], 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 [10], 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 triggers an observable alarm at the assembly station later on. 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
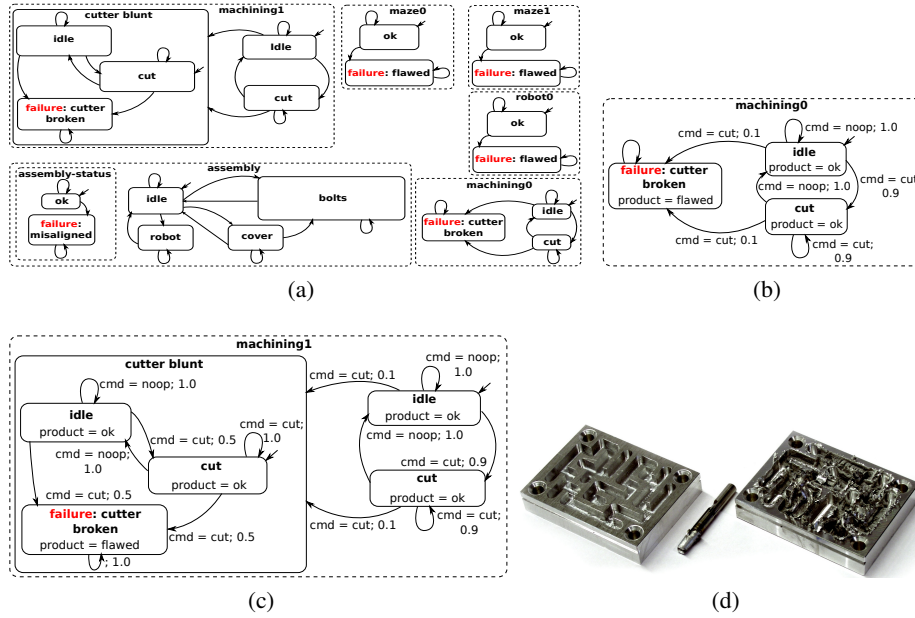
Fig. 1: (1a) Overview of a typical PHCA. It models a factory scenario with three individual products being manufactured. (1b and 1c) Detail views of sub-automata machining0, with simple probabilistic behavior, and machining1, showing a more complex hierarchical machining fault: the cutter goes blunt before breaking. (1d) A product damaged during machining.

manufacturing goals affected, and what are potential faults? 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. Fig. 1c and 1b show detailed versions of the two machining stations.

PHCA-Based Plan Assessment

Plan assessment extends the maximum probability diagnosis problem [11] 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 [4]:

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

- $\Sigma = \Sigma_p \uplus \Sigma_c$ 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).

- $\Pi = O \uplus Cmd \uplus Dep$ 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 $\mathcal{C}[\Pi]$) 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 $St(M_{\mathrm{PHCA}})$ 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), and later executed on the system. This is captured by an *execution adaptation function* $\mathcal{E}_{\mathcal{S}}$, which adapts a PHCA model M_{PHCA}^N 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 $M_{\mathrm{PHCA}}^{\mathcal{S}} = \mathcal{E}_{\mathcal{S}}(M_{\mathrm{PHCA}}^N)$ 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 $M_{\mathrm{PHCA}}^{\mathcal{S}}$ defines a joint distribution $P(\theta, \mathbf{O} = \mathbf{o}^{0:t})$:

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

where $\mathcal{T}[\theta]$ is the multiset of all transitions between primitive locations 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 [4, 3].

Goals. A goal $\mathcal{G}_i$ is a set of goal-achieving trajectories. A practical way of defining $\mathcal{G}_i$ is in terms of designated goal states $m^{t_{\mathcal{G}_i}}$ that trajectories in $\mathcal{G}_i$ should lead to, and the time $t_{\mathcal{G}_i}$ when the goal should be achieved.

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 $M_{\text{PHCA}}^{\mathcal{S}} = \mathcal{E}_{\mathcal{S}}(M_{\text{PHCA}}^{N})$. The **plan assessment problem** is, given observations $\mathbf{o}^{0:t}$, to compute the most probable diagnosis as trajectory in $St(M_{\text{PHCA}}^{\mathcal{S}})$ 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 high-level models. Originally, PHCA semantics were defined in terms of hidden Markov models (HMM) [3], 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 ground models 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 [2]. The work exploits an automatic translation of a PHCA to a COP $\mathcal{R} = (X, D, C)$ [4]. 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 functions modify $\mathcal{R}$ by adding soft constraints. We refer to [4] 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 directly 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 exactly based on probabilities $P(\theta, \mathbf{o}^{0:t})$, which correspond to objective values of the COP solutions.

The key motivation for this approach is that strong off-the-shelf constraint optimization solvers can be exploited to generate a list of all non-zero probability assignments, corresponding to valid trajectories, sorted by probability. The most probable trajectory is then chosen as most probable diagnosis, while the complete list is used as set Θ. While [2] 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 [12]).

Note that Toulbar2 requires another transcoding that involves conversion between probability values and integer costs, resulting in a precision loss. For our examples, however, this loss was negligible.

Probabilistic Inference

From a probabilistic reasoning point of view, plan assessment can be seen as a combination of simultaneously computing most likely explanations (MPE) and marginals. MPE 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) [13]. A BLN is a statistical relational model and can be seen as a template for the construction of Bayesian networks. Given a set of entities, a BLN may be instantiated to either a ground BN or a ground mixed network (MN) that explicitly represents logical constraints on the distribution [14]. Probabilistic inference is often performed in such ground models. However, recent research in statistical relational reasoning tries to (partially) lift the

---

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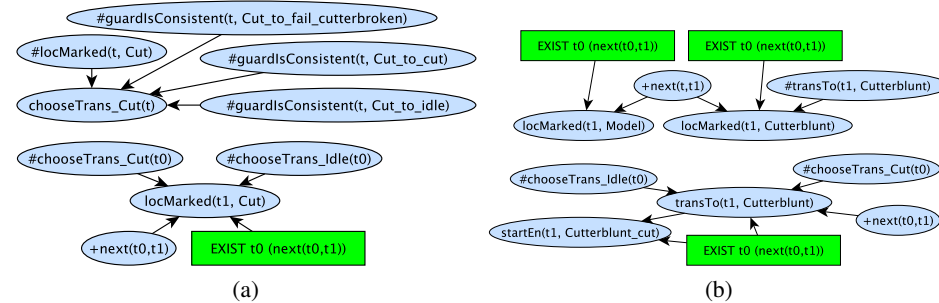

Fig. 2: Figures 1b and 1c translated to BLN fragments, showing excerpts of location marking and probabilistic transition choice (2a) as well as composite and full target marking (2b).

inference problem to the first-order level in order to exploit repeated sub-structures in ground models [15]. 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). 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 that 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 [4]. 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 `behaviorConsistent(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) => behaviorConsistent(t,l).` It says that for all points in time, a location's behavior must be consistent if it is marked.

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:

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

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, one fragment is created for each predicate occuring in a rule, except if the translator can determine, e.g. from the model structure, that a 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 no start location and the predicate 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 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:

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

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:

$$Pr(T_{l_p}^t = \tau \mid L_p^t, \mathbf{G}^t) = \begin{cases} p_\tau & \text{if (a)} \\ 1 & \text{if (b)} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $T_{l_p}^t$ is a random variable for choosing among l_p's outgoing transitions, L_p^t 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) is $(\neg lm(t, l_p) \vee (\neg \exists \tau' \in outgoing(l_p).\ guardIsConsistent(t, \tau'))) \wedge \tau = NoTransition$. 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 an action a to perform on a product p at time t on a station or component c. We call these tuples *product-component links*. We can exploit the flexibility of the BLN framework to encode these: If at time t a product p is worked by station c, logical formulas are added to $\mathcal{L}$ that enforce equality of variables $\{X_p^t\}$ and $\{X_c^t\}$ for products and stations respectively. The common domain of two variables X_p^t and X_c^t encodes influences, e.g. *Faulty* for c inflicting damage on p or a faulty p causing unusual observations in c. *Ok* encodes no (harmful) influence. Commands given 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 [4].

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_i^t (that encode location markings at time points t) as PHCA distribution $P(\theta, \mathbf{O} = \mathbf{o}^{0:t})$. Variable L_i^t 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*

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

$$P(\mathbf{L}^{t_0} = \mathbf{m}^{t_0}, \ldots, \mathbf{L}^{t_N} = \mathbf{m}^{t_N}, \mathbf{O}_{\text{BN}} = \mathbf{o}^{0:t} \mid X_{\text{aux}} = x_{\text{aux}})$$

$\mathbf{L}^{t_j}$ are vectors of location marking variables $L_i^{t_j}$ in the BN for each time point t_j, and $\mathbf{O}_{\text{BN}}$ is a vector of observation variables $O_l^{t_j}$ for each time point t_j (l ranges over indices of observation variables for a given time point). $X_{\text{aux}} = X - \{\mathbf{L}^{t_0}, \ldots, \mathbf{L}^{t_N}, \mathbf{O}_{\text{BN}}\}$

is a set of auxiliarly variables used to represent additional logical constraints. $\mathbf{m}^t$ is boolean vector that encodes a marking in terms of L_i^t assignments for time t.

Solving Plan Assessment with Probabilistic Reasoning Tools and Methods

Translating and executing a given PHCA, $\mathcal{E}_{\mathcal{S}}(\Upsilon_{\mathrm{BLN}}(M_{\mathrm{PHCA}}))$, yields a BLN as starting point for possible probabilistic reasoning solutions for plan assessment. In our experiments, we ground BLN to an auxiliary BN and used the state-of-the-art inference tool Ace 2.0[2], which compiles a given BN into an arithmetic circuit (AC) [5, 6]. Ace exploits local structure given by, e.g., determinism in 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. Note that we focus on computing $P(\mathcal{G}_i \mid \mathbf{o}^{0:t})$ in this work, while, in general, solving plan assessment also requires computing the most probable trajectory (as a diagnosis). This can be done by computing the most probable explanation.

The Ace compilation is considered a (potentially expensive) offline step, the evaluation the (quick) online step. The addition of evidence, i.e. clamping variables to given values, is part of the latter. This step is independent of translation, execution adaptation, grounding, and Ace compilation, which means we can perform all of these steps offline without introducing evidence too early. Ace compilation can only be done offline if the execution adaptation is done offline, because the latter modifies the model. Real-world applications, however, might require execution adaptation to be performed online. Because the size of $\mathcal{S}$ might be too large to allow reasoning over the complete time horizon, receding horizon schemes such as [16] may be applied: the model would be translated for a fixed number N of time steps, then iteratively moved along the operation sequence with potentially much larger length $N_S \gg N$. Consequently, $\mathcal{E}_{\mathcal{S}}$ must be applied for every iteration. It is nontrival to redefine $\mathcal{E}_{\mathcal{S}}$ such that it could be applied online to an Ace compiled model.

A different class of approximate probabilistic reasoning algorithms for tasks such as the computation of marginals is sampling. Sampling is attractive, since it is an anytime approach with a stochastic accuracy guarantee for marginals in terms of the confidence interval [7]. Models with strong determinism, such as ours, pose practical problems owing to the rejection problem. However, schemes such as SampleSearch [17] have recently been proposed to address this problem by exploiting constraint-solving methods to quickly exclude inconsistent samples.

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 [4]. The factory models 2 and 3 are variations of the model shown in 1a, factory model 1 is taken from [2]. 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

---

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

longer operation sequence $\mathcal{S}$. Finally, the diagnosis instance simulates diagnosing hardware or software faults in a satellite camera module [4]. **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 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 for Ace compilation of fm2(long $\mathcal{S}$), where enforcing logical model counting yielded much better results. The results table shows, for Toulbar2, search time (seconds), memory usage (MB) and expanded search tree nodes, and, for Ace, compilation + evaluation time (both in seconds) and memory usage during compilation (in MB).

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.

Conclusion

In this work, we formalized and described the first automatic translation from the high-level model-based diagnosis framework probabilistic hierarchical constraint automata (PHCA) to a high-level probabilistic reasoning framework, Bayesian logic networks (BLNs). It provides engineers with a way of applying a wide range of probabilistic reasoning methods and tools to problems such as the *plan assessment* problem, which occurs in AI decision making in engineering domains. It lies at the intersection of model-based diagnosis and probabilistic reasoning and involves the computation of the

Table 1: (Left) The size of PHCAs, COP and BN translations. (Right) Measurements for Toulbar2 and Ace solving the instances. * Results from Ace 3.0.

instance	N	phca size	# var	# con	# nodes	instance	Toulbar2	Ace
fm1 [2]	6	11/6/27	643	670	1106	fm1 [2]	0.01 / 8 / 186	0.37 + 0.09 / 10
fm2	9	15/8/33	1202	1251	2122	fm2	0.05 / 10 / 1878	0.93 + 0.18 / 90
fm3	9	17/8/33	1305	1311	2292	fm3	0.32 / 10 / 5464	0.36 + 0.07 / 5
fm2(long $\mathcal{S}$)	19	15/8/33	2482	2601	4444	fm2(long $\mathcal{S}$)	0.65 / 19 / 11320	1128.04 + ERR / $\approx$ 900
fm3(long $\mathcal{S}$)	33	18/8/35	4748	4892	8878	fm3(long $\mathcal{S}$)	3.79 / 53 / 10180	(1.36 + 0.17 / –)*
sm [4]	8	8/4/22	640	661	1080	sm [4]	0.01 / 9 / 176	0.87 + 0.03 / 187

probabilities with which pre-planned operations for a technical system achieve their goals. We evaluate our approach by solving six realistic problem instances with a) an existing solution based on generating most probable hypotheses as the best solutions of a constraint optimization problem, generated by an off-the-shelf solver, and 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. In particular, we found combinations of sampling with state-of-the-art constraint solvers to be a promising future direction.

References

1. Kurien, J., Nayak, P.P.: Back to the future for consistency-based trajectory tracking. In: Proc. AAAI, AAAI Press (2000) 370–377
2. Maier, P., Sachenbacher, M., Rühr, T., Kuhn, L.: Automated plan assessment in cognitive manufacturing. Adv. Eng. Informat. (May 2010)
3. Williams, B.C., Chung, S., Gupta, V.: Mode estimation of model-based programs: Monitoring systems with complex behavior. In: Proc. IJCAI. (2001) 579–590
4. Mikaelian, T., Williams, C.B., Sachenbacher, M.: Model-based Monitoring and Diagnosis of Systems with Software-Extended Behavior. In: Proc. AAAI, Pittsburgh, USA, AAAI Press (2005)
5. Chavira, M., Darwiche, A.: Compiling bayesian networks with local structure. In: Proc. IJCAI. (2005) 1306–1312
6. Darwiche, A.: A differential approach to inference in bayesian networks. Journal of the ACM **50**(3) (2003) 280–305
7. Maier, P., Jain, D., Waldherr, S., Sachenbacher, M.: Plan assessment for autonomous manufacturing as bayesian inference. In: Proc. KI. LNAI, Springer (Sep. 2010)
8. Abreu, R., Zoeteweij, P., Van Gemund, A.J.C.: A new bayesian approach to multiple intermittent fault diagnosis. In: Proc. IJCAI, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2009) 653–658
9. Knox, B., Mengshoel, O.: Diagnosis and reconfiguration using bayesian networks: An electrical power system case study. In: Workshop Proc. SAS. (2009)
10. Rutten, J., Kwiatkowska, M., Norman, G., Parker, D.: Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems. Volume 23 of CRM Monograph Series. American Mathematical Society (2004)
11. Sachenbacher, M., Williams, B.: Diagnosis as semiring-based constraint optimization. In: Proc. ECAI-2004, Valencia, Spain (2004)
12. Cooper, M., De Givry, S., Sanchez, M., Schiex, T., Zytnicki, M.: Virtual arc consistency for weighted csp. In: Proc. AAAI, AAAI Press (2008) 253–258
13. Jain, D., Waldherr, S., Beetz, M.: Bayesian Logic Networks. Technical report, Technische Universität München (2009)
14. Mateescu, R., Dechter, R.: Mixed Deterministic and Probabilistic Networks. Annals of Mathematics and Artificial Intelligence **54**(1) (2008) 3–51
15. Poole, D.: First-order probabilistic inference. In: IJCAI. (2003) 985–991
16. Maier, P., Sachenbacher, M.: Receding time horizon self-tracking and assessment for autonomous manufacturing. In: Workshop Proc. Self-X. (Sep. 2010)
17. Gogate, V., Dechter, R.: SampleSearch: A Scheme that Searches for Consistent Samples. In: Proc. AISTATS. (2007)