CTL Model Checking Partially Specified Systems

Eshita Zaman[1], Christopher Johannsen[2], Andrew S. Miner[2], Gianfranco Ciardo[2], and Samik Basu[2]

[1] Department of Computer Science, Utah Valley University, Orem, UT, USA
eshita.zaman@uvu.edu
[2] Department of Computer Science, Iowa State University, Ames, IA, USA
{cgjohann, asminer, ciardo, sbasu}@iastate.edu

Abstract. The behavioral specifications of a system are often partial at early stages of development due to pending design decisions. We view such specifications as being parameterized by these decisions and consider the problem of identifying a possible concretization, i.e., taking a subset of these decisions, to ensure that the system conforms to the desired requirements. We capture such a partially specified system as a partially-labeled Kripke structure (`plKS`), where certain propositions labeling the states of the Kripke structure correspond to pending design decisions, thus may be *unknown*. We then reduce the verification problem to model checking different `plKS` instances, each corresponding to a specific set of design decisions taken, inducing a semi-lattice on the instances. Central to our solution strategy is the effective and efficient exploration of this semi-lattice and the application of model checking techniques to `plKS`s with 3-valued semantics of temporal properties, to take into account unknown state labels. We also address the problem of identifying an optimal instance of a partially-specified system that satisfies a desired property, where the cost of design decisions drive the optimality criterion. We use a prototype implementation of our strategy to validate its viability on a multi-objective path planning problem.

Keywords: Partially specified systems · Design space exploration · 3-valued logic · CTL model checking.

1 Introduction

Model checking is a well-established technique to automatically verify desired system properties by examining all possible system behaviors. Model checkers [8, 17,21] take as input a system design expressed in a high-level formalism, such as a Petri net, and properties of interest expressed as formulas in some temporal logic, and decide whether each property holds in the system. However, in practice, modern software systems are frequently developed and analyzed under partial knowledge. Whether due to incomplete specifications, evolving requirements, unavailable components, or deferred implementation details, engineers often lack a fully defined model at design time. Yet, critical correctness properties such as safety, liveness, or access control must still be verified to ensure dependable

system behavior. This challenge arises in diverse contexts, including software product lines, security auditing, runtime verification, and autonomous systems.

Driving Problem. We consider a system design where a subset of behavioral aspects is partially specified due to pending design decisions. In this context, the challenge is to decide whether there exists a combination of design decisions that results in a system design satisfying the desired properties. As there may be multiple sets of design decisions which yield the same conformance, we aim to identify an optimal set of design decisions achieving conformance based on a cost assigned to each decision. This leads to two core verification tasks: (i) feasibility analysis: determining the existence of satisfying instantiations, and (ii) optimization: identifying a cost-optimal instantiation. Design decisions may be related, as one decision may require another decision to be taken, or certain decisions may be mutually exclusive.

As a concrete example, consider the planning problem of an autonomous rover that can navigate a terrain with obstacles while collecting samples from specific locations. The rover is powered by a battery with a given capacity and each rover movement consumes battery energy. The property of interest is whether the rover can reach a specified destination, collect a specified minimal number of samples, and maintain the battery level above a certain amount of energy. Here, the minimum number of samples and battery level drive design decisions. One can develop a partial design of the rover behavior where different choices of design decisions result in different instances of the rover behavior. For instance, deciding to maintain the battery level above 5 vs. 10 units results in different rover behaviors.

Solution Strategy. We view the partial behavioral system design as a parameterized system, where the parameters capture the pending design decisions. We model such a system using Petri nets and show that the semantics of the model can be captured using *partially-labeled Kripke structures*, where the states of the Kripke structures are labeled with some propositions whose valuations are associated with the design decisions yet to be considered and are, therefore, *unknown*. Model checking such a Kripke structure against temporal properties expressed in CTL results in true, false, or unknown answers following the 3-valued logical semantics for CTL. Our objective is to identify an optimal set of changes to the valuations of unknown-propositions (true or false) so that the resulting (possibly still partially-labeled) Kripke structure satisfies the CTL property. We show that the possible changes in the valuations of unknown-propositions (corresponding to pending design decisions) induce a semi-lattice over partially-labeled Kripke structures, which forms the solution space for the problem. We present an algorithm to efficiently explore this solution space to compute the partially-labeled Kripke structure and a corresponding set of optimal design decisions that result in satisfaction of a desired property. Key to our efficient exploration is that we compute the state-space of a given partially-labeled Kripke structure (corresponding to given unknown design decisions) once and examine the property conformance corresponding to different instantiations of design decisions in the

context of the generated state-space, thus avoiding repeated generation of the state-space for different configurations of design decisions.

Organization. Sect. 2 discusses modeling formalisms and necessary background on 3-valued model checking; Sect. 3 formalizes an interpretation of 3-valued models as design spaces and provides an algorithm to find an optimal design; Sect. 4 applies our formalism and algorithm on a case study involving an autonomous rover; Sect. 5 describes related work; Sect. 6 discusses impacts and future work.

2 Background

Petri nets (PNs) are a well-known high-level formalism to model distributed and concurrent systems. PNs can represent different classes of models such as Kripke structures, labeled transition systems, etc. [24]. Even with just a few places, a PN can represent a model with an enormous state space, which makes it an excellent choice for compactly modeling partially specified systems.

Definition 1. A Petri net $(\mathcal{P}, \mathcal{T}, \mathbf{D}^-, \mathbf{D}^+, \mu_{init})$ is a finite bipartite marked multigraph where $\mathcal{P} = \{p_1, ..., p_P\}$ and $\mathcal{T} = \{t_1, ..., t_T\}$ are disjoint sets of *places* and *transitions*; $\mathbf{D}^-, \mathbf{D}^+ \in \mathbb{N}^{\mathcal{P}\times\mathcal{T}}$ describe the cardinality of the *input* and *output* arcs; and $\mu_{init} \in \mathbb{N}^{\mathcal{P}}$ is the initial *marking* (an assignment of *tokens* to the places). If the PN is in marking μ (a vector where μ_i is the number of tokens in p_i), transition $t_j \in \mathcal{T}$ is *enabled* iff $\forall p_i \in \mathcal{P}, \mu_i \geq \mathbf{D}^-[i,j]$, in which case it may *fire*, leading to marking μ' s.t. $\forall p_i \in \mathcal{P}, \mu'_i = \mu_i - \mathbf{D}^-[i,j] + \mathbf{D}^+[i,j]$. □

Marking μ' is *reachable* from marking μ if there is a sequence σ of (enabled) transition firings that leads from μ to μ', we write $\mu \xrightarrow{\sigma} \mu'$, or simply $\mu \xrightarrow{*} \mu'$ if the specific σ is not important. The reachability set $\mathcal{S}_{reach}$ of the PN is defined as $\mathcal{S}_{reach} = \{\mu : \mu_{init} \xrightarrow{*} \mu\}$. We consider PNs whose $\mathcal{S}_{reach}$ is finite, in which case their semantics are captured by a Kripke structure formally defined as follows.

Definition 2. A Kripke structure $M = (\mathcal{S}, \mathcal{S}_{init}, \mathcal{N}, \mathcal{A}, L)$ describes a finite set of states $\mathcal{S}$, a set of initial states $\mathcal{S}_{init} \subseteq \mathcal{S}$, a left-total next-state relation over states $\mathcal{N} \subseteq \mathcal{S} \times \mathcal{S}$, a finite set of atomic propositions $\mathcal{A}$, and a labeling function $L : \mathcal{S} \times \mathcal{A} \to \{\mathbb{T}, \mathbb{F}\}$ specifying which atomic propositions hold in each state. □

In the above definition, the relation between a PN and its corresponding M_{KS} is $\mathcal{S} = \mathcal{S}_{reach}$, $\mathcal{S}_{init} = \{\mu_{init}\}$, $\mathcal{N} = \{(\mu, \mu') : \mu \in \mathcal{S}, \exists t \in \mathcal{T}, \mu \xrightarrow{t} \mu'\}$[3], while $\mathcal{A}$ and L are defined using comparisons of arithmetic expressions on the number of tokens in places and integer constants. For example, if $a \equiv p_2 > 3$ is an atomic proposition, then $L(\mu, a) = \mathbb{T}$ iff $\mu_2 > 3$.

[3] An arbitrary PN may not satisfy the requirement that $\mathcal{N}(\mu) \neq \emptyset$ for all reachable markings μ, i.e., it may have *dead* markings. This situation can be detected, so we assume it does not occur or is managed by adding a self-loop on dead markings.

2.1 Kripke Structures in 3-valued Logic

As discussed in Sect. 1, we consider partially specified systems, which result from system specifications with pending design decisions. We incorporate such pending decisions in the PN representation of the system. If a design decision related to the enabling of a PN transition in the system is yet to be taken, then whether or not the transition can be fired remains *unknown*—concretizing such design decision (partially) concretizes the behavioral evolution of the system. In other words, the presence of pending design decisions may result in systems whose conformance to requirements is unknown and some pending design decisions may need to be concretized to ensure the conformance. The pending decisions can be viewed as propositions with unknown valuation in the Kripke structure semantics of the PN models. Consider, for instance, design decision d and proposition a labeling a Kripke structure state defined as follows:

$$a \equiv d \wedge (p_2 > 3),$$

then the valuation of a evaluates to $p_2 > 3$ when the decision is taken (d is $\mathbb{T}$) and evaluates to $\mathbb{U}$ if decision for selecting d is yet to be taken. In the above, we have followed Kleene's logical framework for 3-valued logic [20] to define the semantics of the standard logical operators $\neg$, $\wedge$, and $\vee$ on this domain:

	$\neg$
$\mathbb{T}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{T}$
$\mathbb{U}$	$\mathbb{U}$

$\wedge$	$\mathbb{T}$	$\mathbb{F}$	$\mathbb{U}$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{F}$	$\mathbb{U}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{U}$	$\mathbb{U}$	$\mathbb{F}$	$\mathbb{U}$

$\vee$	$\mathbb{T}$	$\mathbb{F}$	$\mathbb{U}$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$
$\mathbb{F}$	$\mathbb{T}$	$\mathbb{F}$	$\mathbb{U}$
$\mathbb{U}$	$\mathbb{T}$	$\mathbb{U}$	$\mathbb{U}$

We refer to a Kripke structure with such 3-valued labeling as a partially-labeled Kripke structure.

Definition 3. A partially-labeled Kripke structure (`plKS`) $M = (\mathcal{S}, \mathcal{S}_{init}, \mathcal{N}, \mathcal{A}, L)$ describes a finite set of states $\mathcal{S}$, a set of initial states $\mathcal{S}_{init} \subseteq \mathcal{S}$, a nondeterministic next-state function over states $\mathcal{N} : \mathcal{S} \times \mathcal{S}$, a finite set of atomic propositions $\mathcal{A}$, and a labeling function $L : \mathcal{S} \times \mathcal{A} \to \{\mathbb{T}, \mathbb{F}, \mathbb{U}\}$. □

Bruns and Godefroid call such structures "partial Kripke structures" [3, 4], as they can be used to describe system behavior with partial specification in terms of state-labels. Similar consideration of partial behavioral specification also forms the basis for modal transition systems [15, 18], used to describe system families in product lines [16, 22]. Chechik et al. [6] further extend this setup by considering general multi-valued logic (with domain $\mathcal{D}$ of interpretation beyond $\{\mathbb{T}, \mathbb{F}, \mathbb{U}\}$). The key to automatic verification of Kripke structures with such logic against temporal properties is that the semantics of temporal properties implies a partition of size $|\mathcal{D}|$. We use partially labeled Kripke structure as a way to capture the high-level description of systems with pending design decisions. The following presents a brief overview of the temporal logic CTL and the semantics of CTL in the context of partially labeled Kripke structures.

$$
\begin{aligned}
\forall p \in \mathcal{A},\ [\![p, M]\!] &= \langle \{s : L(s,p) = \mathbb{T}\}, \{s : L(s,p) = \mathbb{F}\} \rangle \\
[\![\neg\varphi, M]\!] &= \overline{[\![\varphi, M]\!]} \\
[\![\varphi_1 \wedge \varphi_2, M]\!] &= [\![\varphi_1, M]\!] \cap [\![\varphi_2, M]\!] \\
[\![\mathtt{EX}(\varphi), M]\!] &= \langle \{s : \exists \sigma_s, \sigma_s[1] \in [\![\varphi, M]\!]_{\mathbb{T}}\}, \{s : \forall \sigma_s, \sigma_s[1] \in [\![\varphi, M]\!]_{\mathbb{F}}\} \rangle \\
[\![\mathtt{AF}(\varphi), M]\!] &= \langle \{s : \forall \sigma_s, \exists i \geq 0, \sigma_s[i] \in [\![\varphi, M]\!]_{\mathbb{T}}\}, \{s : \exists \sigma_s, \forall i \geq 0, \sigma_s[i] \in [\![\varphi, M]\!]_{\mathbb{F}}\} \rangle \\
[\![\mathtt{E}(\varphi_1\, \mathtt{U}\, \varphi_2), M]\!] &= \langle \{s : \exists \sigma_s, \exists i \geq 0, \sigma_s[i] \in [\![\varphi_2, M]\!]_{\mathbb{T}} \wedge \forall j < i, \sigma_s[j] \in [\![\varphi_1, M]\!]_{\mathbb{T}}\}, \\
&\qquad \{s : \forall \sigma_s, \forall i \geq 0, \sigma_s[i] \in [\![\varphi_2, M]\!]_{\mathbb{F}} \vee \exists j < i, \sigma_s[j] \in [\![\varphi_1, M]\!]_{\mathbb{F}}\} \rangle
\end{aligned}
$$

Fig. 1. CTL semantics on `plKS` M. $\sigma_{s_0} = s_0 \to s_1 \to \cdots$ denotes an infinite path starting at state s_0, and $\sigma_{s_0}[i]$ the i^{th} state in it, for $i \in \mathbb{N}$.

2.2 CTL Semantics for `plKS`s

We focus on the temporal logic CTL [9], which describes the branching behavior of a system. The syntax of a CTL formula is given by

$$\varphi \ \to\ a \ \mid\ \neg\varphi \ \mid\ \varphi \wedge \varphi \ \mid\ \mathtt{EX}(\varphi) \ \mid\ \mathtt{AF}(\varphi) \ \mid\ \mathtt{E}(\varphi\, \mathtt{U}\, \varphi),$$

where $a \in \mathcal{A}$. Informally, state s satisfies $\mathtt{EX}(\varphi)$ if it has at least one next state satisfying φ, $\mathtt{AF}(\varphi)$ if all evolutions starting from s reach some state satisfying φ, and $\mathtt{E}(\varphi_1\, \mathtt{U}\, \varphi_2)$ if there is an evolution from s leading to a state satisfying φ_2 and all states before that satisfy φ_1. The above syntax includes an adequate set of boolean and temporal operators in the sense that other CTL operators such as `AX`, `EF`, `AG`, `EF`, and `AU` can be expressed using `EX`, `AF`, and `EU` (see [9] for details).

The semantics of CTL interpreted on a Kripke structure with 3-valued logic was presented in [5, 6]; we recall the semantic function for completeness' sake. The semantics of a CTL property φ in the context of a `plKS` returns a 3-way partition of the state space $\mathcal{S}$: the set of states satisfying φ, not satisfying φ, and undetermined to satisfy φ. We represent this partition as the pair of disjoint sets $\langle \mathcal{S}_T, \mathcal{S}_F \rangle$, respectively satisfying and not satisfying φ, so that the set of states where φ is undetermined is implicitly given by $\mathcal{S} \setminus (\mathcal{S}_T \cup \mathcal{S}_F)$. The basic set operations over such a pair satisfy:

$$
\begin{aligned}
&\overline{\langle \mathcal{S}_T, \mathcal{S}_F \rangle} = \langle \mathcal{S}_F, \mathcal{S}_T \rangle; \\
&\langle \mathcal{S}_T, \mathcal{S}_F \rangle \cup \langle \mathcal{S}'_T, \mathcal{S}'_F \rangle = \langle \mathcal{S}_T \cup \mathcal{S}'_T, \mathcal{S}_F \cap \mathcal{S}'_F \rangle; \\
&\langle \mathcal{S}_T, \mathcal{S}_F \rangle \cap \langle \mathcal{S}'_T, \mathcal{S}'_F \rangle = \langle \mathcal{S}_T \cap \mathcal{S}'_T, \mathcal{S}_F \cup \mathcal{S}'_F \rangle.
\end{aligned}
$$

The semantic function $[\![\ ]\!]$ takes as input CTL formula φ and `plKS` M, and returns $\langle \mathcal{S}_T, \mathcal{S}_F \rangle$, where $\mathcal{S}_T$ is the set of states of M that satisfy φ and $\mathcal{S}_F$ is the set of states of M that satisfy $\neg\varphi$. Figure 1 shows the formal definition of $[\![\ ]\!]$.

For any `plKS` M and CTL formula φ, we say that $M \models \varphi$ iff each initial state satisfies φ, $M \not\models \varphi$ iff some initial state does not satisfy φ, and $M \not\models\!\!\!\models \varphi$ otherwise, i.e., no initial state does not satisfy φ but for at least one initial state it is not determined whether it satisfies φ.

3 Exploring Concretizations of a plKS

Now that we have a formalism to describe systems with potentially unknown properties, we move on to defining some types of analysis we can perform on such systems. In particular, we present the concept of *concretizing* unknown values (i.e., changing unknown values to true or false) to obtain a more "concrete" system. Recall that the unknown values of state propositions are induced by the design decisions yet to be taken and this allows us to interpret these systems as *design spaces*. We define an algorithm to search a design space and find a minimum-cost concretized system that satisfies some CTL formula φ.

3.1 A Semi-lattice of plKSs

Consider a plKS $M_L = (\mathcal{S}, \mathcal{S}_{init}, \mathcal{N}, \mathcal{A}, L)$ and let $\mathcal{U}_L = \{(s,a) : L(s,a) = \mathbb{U}\}$ be the set of state and atomic proposition pairs for which the labeling in M is unknown; also, let $|\mathcal{U}_L| = m$.

Labeling function L' is a *concretization* of L iff $L(s,a) = \mathbb{T} \Rightarrow L'(s,a) = \mathbb{T}$ and $L(s,a) = \mathbb{F} \Rightarrow L'(s,a) = \mathbb{F}$, implying that $\mathcal{U}_{L'} \subseteq \mathcal{U}_L$. Let $\mathcal{L}$ be the set of the 3^m possible concretizations of L. Note that $M_{L'}$ is an ordinary Kripke structure for any of the 2^m full concretizations L' of L, i.e., whenever $L' : \mathcal{S} \times \mathcal{A} \to \{\mathbb{T}, \mathbb{F}\}$.

We define the "join" binary operator $\oplus$ on $\{\mathbb{T}, \mathbb{F}, \mathbb{U}\}$ as

$$\mathbb{T}\oplus\mathbb{U}=\mathbb{U}\oplus\mathbb{T}=\mathbb{T}\oplus\mathbb{F}=\mathbb{F}\oplus\mathbb{T}=\mathbb{F}\oplus\mathbb{U}=\mathbb{U}\oplus\mathbb{F}=\mathbb{U}\oplus\mathbb{U}=\mathbb{U}$$

$$\mathbb{T} \oplus \mathbb{T} = \mathbb{T} \qquad \mathbb{F} \oplus \mathbb{F} = \mathbb{F},$$

and extend it to $\mathcal{L}$ as follows:

$$\forall L,L' \in \mathcal{L}, (s,a) \in \mathcal{S} \times \mathcal{A} : (L \oplus L')(s,a) = L(s,a) \oplus L'(s,a).$$

If we let $L \leq L' \Leftrightarrow L \oplus L' = L$, then $(\mathcal{L}, \leq, \oplus)$ is a meet-semi-lattice with least element L, so that "strictly greater" means "having $\mathbb{T}$ or $\mathbb{F}$ instead of $\mathbb{U}$ in some positions". This induces an analogous meet-semi-lattice over the set of plKSs $\{M_L : L \in \mathcal{L}\}$; see Figure 2 for an example.

We can try to find a concretization L such that $M_L \models \varphi$ by searching the semi-lattice for such an L. If no such L exists we know that no concretization of M satisfies φ. Observe that, if there exists an L such that $M_L \not\models \varphi$, then no further concretization of L needs to be explored, as $\forall L', L \leq L' \Rightarrow M_{L'} \not\models \varphi$.

This setting provides fine concretizations of the plKS. In practice, however, a design decision likely changes many values of L from $\mathbb{U}$ to $\mathbb{T}$ or $\mathbb{F}$ at once, thus only a subset of $\mathcal{L}$ may be actually realizable. A design decision could thus map an element $L \in \mathcal{L}$ to another $L' \in \mathcal{L}$, subject to $L < L'$ (with at least one but possibly many $\mathbb{U}$ entries changed to $\mathbb{T}$ or $\mathbb{F}$). A concretization L with m unknowns can in principle be further concretized in $3^m - 1$ possible ways (the number of L' satisfying $L < L'$). We instead assume a (relatively) small set of design decisions, each of which, if applicable to concretization L, changes it into L', by changing a specific subset of $\mathbb{U}$ values of L into $\mathbb{T}$ or $\mathbb{F}$, in a fixed way.

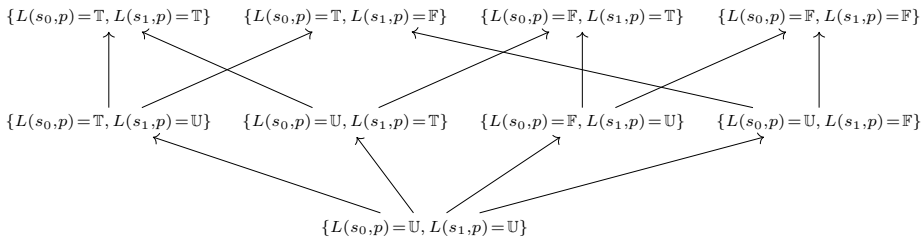

Fig. 2. A meet-semi-lattice $(\mathcal{L}, \leq, \oplus)$ with $m = 2$, $\mathcal{S} = \{s_0, s_1\}$, $\mathcal{A} = \{p\}$, and $L(s_0, p) = L(s_1, p) = \mathbb{U}$ is a directed acyclic graph whose nodes are the elements of $\mathcal{L}$ organized into $m + 1$ levels, with level m at the top and level 0 at the bottom, according to the number $n \in \{0, ..., m\}$ of $\mathbb{U}$ values that have been changed to $\mathbb{T}$ or $\mathbb{F}$.

3.2 Concretizing `plKS`: Exploring the Design Space

We consider `plKS`s where the uncertainty can stem from whether a given atomic proposition holds in a given state or whether a transition is possible between a given pair of states. In an actual design process, however, the available design decisions normally present themselves at a high level, so that taking one such design decision simultaneously concretizes many uncertain aspects in the partially-labeled Kripke structure. Thus, an underlying `plKS` M with m unknowns has 3^m potential concretizations, but the number of possible concretizations of the high-level model defining M is **at most** $\prod_{i=1}^{n}(1 + D_i)$, if there are n possible design decisions, and the i-th one can take one of D_i values (or remain untaken).

In the following paragraphs, we proceed with the presentation of `plKS`s in the context of design decisions, where a design decision corresponds to a possible concretization of the labeling function of a `plKS`. In other words, the labeling function of a `plKS` is parameterized with respect to the concretization induced by the specific design decisions.

Definition 4. A *structured concretizable* `plKS` is a tuple $(\mathcal{S}, \mathcal{S}_{init}, \mathcal{N}, \mathcal{A}, \mathcal{D}, L)$, where $\mathcal{S}$, $\mathcal{S}_{init}$, $\mathcal{N}$ and $\mathcal{A}$ are defined as before (see Definition 3), and

- $\mathcal{D} = \{\mathbb{T}, \mathbb{U}\}^{\{\delta_1, ..., \delta_Q\}}$ is the *design decision space*: concretization $\mathbf{d} = [d_1, ..., d_Q] \in \mathcal{D}$ indicates which design decisions have been taken, i.e., $d_h = \mathbb{T}$ iff design decision δ_h has been taken. Each decision δ_h is described by a 5-tuple $(\mathcal{U}_h, \mathcal{T}_h^-, \mathcal{F}_h^-, \mathcal{T}_h^+, \mathcal{F}_h^+)$ of subsets of $\mathcal{S} \times \mathcal{A}$, satisfying the following conditions:

$$\mathcal{T}_h^+ \cup \mathcal{F}_h^+ \subseteq \mathcal{U}_h \qquad \mathcal{T}_h^+ \cap \mathcal{F}_h^+ = \emptyset,$$

  i.e., $\mathcal{T}_h^+$ and $\mathcal{F}_h^+$ are disjoint subsets of $\mathcal{U}_h$ and

$$\mathcal{T}_h^- \cap \mathcal{F}_h^- = \emptyset \qquad \mathcal{T}_h^- \cap \mathcal{U}_h = \emptyset \qquad \mathcal{F}_h^- \cap \mathcal{U}_h = \emptyset,$$

  i.e., $\mathcal{T}_h^-$, $\mathcal{F}_h^-$, and $\mathcal{U}_h$ have no common elements. This 5-tuple describes when δ_h can be taken, and its effect (details are given below).

– $L : \mathcal{D} \times \mathcal{S} \times \mathcal{A} \to \{\mathbb{T}, \mathbb{F}, \mathbb{U}\}$ is a labeling function specifying whether atomic proposition a holds in state s for concretization $\mathbf{d}$. We let $L_{\mathbf{d}}(s, a)$ denote the function L with arguments $\mathbf{d} \in \mathcal{D}, s \in \mathcal{S}$ and $a \in \mathcal{A}$. □

A design decision δ_h of the form $(\mathcal{U}_h, \mathcal{T}_h^-, \mathcal{F}_h^-, \mathcal{T}_h^+, \mathcal{F}_h^+)$ is *enabled*, thus can be taken, in concretization $\mathbf{d}=[d_1,...,d_Q]$, with $d_h=\mathbb{U}$, if $\mathcal{U}_h \subseteq \{(s,a) : L_{\mathbf{d}}(s,a)=\mathbb{U}\}$, $\mathcal{T}_h^- \subseteq \{(s,a) : L_{\mathbf{d}}(s,a)=\mathbb{T}\}$, and $\mathcal{F}_h^- \subseteq \{(s,a) : L_{\mathbf{d}}(s,a)=\mathbb{F}\}$, i.e., if the labeling of each element of $\mathcal{U}_h$, $\mathcal{T}_h^-$, and $\mathcal{F}_h^-$ is, respectively, $\mathbb{U}$, $\mathbb{T}$, and $\mathbb{F}$ for concretization $\mathbf{d}$; then, $\mathcal{T}_h^+$ and $\mathcal{F}_h^+$ are the state-proposition pairs whose labeling changes from $\mathbb{U}$ to, respectively, $\mathbb{T}$ or $\mathbb{F}$ if the decision δ_h is taken in concretization $\mathbf{d}$.

The initial concretization where no design decision has been taken is then $\mathbf{d}_{init} = [\mathbb{U}, ..., \mathbb{U}] \in \mathcal{D}$. The initial mapping of states-proposition pairs to $\mathbb{T}$, $\mathbb{F}$ and $\mathbb{U}$ is captured by the function parameterized with $\mathbf{d}_{init}$, i.e., $L_{\mathbf{d}_{init}} = L_{orig}$, where $L_{orig}(s, a)$ is $\mathbb{T}$ if initially we know a holds in state s, is $\mathbb{F}$ if initially we know a does not hold in state s, and is $\mathbb{U}$ otherwise.

Taking enabled design decision δ_h in $\mathbf{d}$ leads to concretization $\mathbf{d}'$, equal to $\mathbf{d}$ except that d_h is $\mathbb{T}$ instead of $\mathbb{U}$, and results in the new labeling function $L_{\mathbf{d}'}$ satisfying:

$$\begin{array}{l}\forall (s,a) \in \mathcal{T}_h^+,\ L_{\mathbf{d}'}(s,a) = \mathbb{T} \\ \forall (s,a) \in \mathcal{F}_h^+,\ L_{\mathbf{d}'}(s,a) = \mathbb{F} \\ \forall (s,a) \in \mathcal{S}\times\mathcal{A} \setminus (\mathcal{T}_h^+ \cup \mathcal{F}_h^+),\ L_{\mathbf{d}'}(s,a) = L_{\mathbf{d}}(s,a).\end{array}$$

We can then define $\mathcal{D}_{reach} \subseteq \mathcal{D}$ as the set of concretizations reachable from the initial concretization $\mathbf{d}_{init}$ by taking any sequence of enabled design decisions. Generally, this means that $\mathbf{d}'$ could be reachable from $\mathbf{d}$ by taking a certain subset of design decisions in a particular order, while taking the same subset in a different order might not be possible. If this is undesirable or unrealistic, one can prevent it by appropriately setting the values of $\mathcal{T}_h^-$, $\mathcal{F}_h^-$, and $\mathcal{U}_h$ for each or some of the design decisions δ_h as these three components constrain when δ_h can be taken; in other words, having already taken a design decision δ_g may "disable" design decision δ_h. To guarantee an optimal search, it is sufficient to assume that once a design decision δ_h becomes disabled because of a previously-taken decision δ_g, it cannot become enabled due to taking yet another decision δ_l. This is implied by the fact that each design decision monotonically changes $\mathbb{U}$'s into $\mathbb{T}$'s or $\mathbb{F}$'s, but never changes $\mathbb{T}$'s or $\mathbb{F}$'s.

Given two concretizations $\mathbf{d}, \mathbf{d}' \in \mathcal{D}$, we write $\mathbf{d} \leq \mathbf{d}'$ if $\mathbf{d}'$ is *more concrete* than $\mathbf{d}$, i.e., if it is obtained from $\mathbf{d}$ by taking zero or more design decisions: $\forall h \in [1, Q], d_h = \mathbb{T} \Rightarrow d'_h = \mathbb{T}$. Note that this order on concretizations is consistent with the order on the corresponding labeling functions defined by these concretizations: for any $\mathbf{d}'$ reached from $\mathbf{d}$ by taking a sequence of design decisions, if $L_{\mathbf{d}}(s,a) \in \{\mathbb{T}, \mathbb{F}\}$, then $L_{\mathbf{d}'}(s,a) = L_{\mathbf{d}}(s,a)$, thus $L_{\mathbf{d}} \leq L_{\mathbf{d}'}$.

Proceeding further, we formulate the problem of identifying which design decisions can be taken so that (partially specified) systems conform to requirements in terms of satisfaction of temporal properties by concretizable `plKS`.

Problem 1. Given a structured concretizable plKS $(\mathcal{S}, \mathcal{S}_{init}, \mathcal{N}, \mathcal{A}, \mathcal{D}, L)$ and a CTL formula φ, find $\mathbf{d} \in \mathcal{D}$ reachable from $\mathbf{d}_{init}$ such that $M = (\mathcal{S}, \mathcal{S}_{init}, \mathcal{N}, \mathcal{A}, L_{\mathbf{d}})$ satisfies φ, i.e., $M \models \varphi$. Alternatively, determine that no such $\mathbf{d}$ exists. □

3.3 Assigning Cost and Finding a Minimal Cost Concretization

Assume we have a *cost function* $C : \{\delta_1, ..., \delta_Q\} \to [0,\infty)$ quantifying the cost of taking each (enabled) decision δ_h, and that the cost of taking a subset of design decisions $\mathcal{D}' \subseteq \mathcal{D}$ (in any order that allows all of them to be applied) is simply the sum of the individual cost of each design decision taken in $\mathcal{D}'$. Then, we can define the cost of a concretization $\mathbf{d} \in \mathcal{D}$ as $C(\mathbf{d}) = \sum_{h \in \{1,...,Q\,:\, d_h = \mathbb{T}\}} C(\delta_h)$.

The optimization problem corresponding to decision Problem 1, therefore, involves identifying a $\mathbf{d} \in \mathcal{D}$ of minimal cost such that plKS M satisfies φ.

To find such a minimal cost concretization, we search the lattice of concretizations, starting from the initial one where no decisions has been taken, until we find a concretization that satisfies the target formula. We use a best-first search strategy where, at each step of the search process, the algorithm selects the most promising concretization, with the lowest cost, from a priority queue. The use of a priority queue allows the greedy search to explore the solution space in a systematic and structured manner in order of increasing cost.

Figure 5 illustrates an example design space and the order in which concretizations are considered. For each $\mathbf{d}$ in the queue, we check the property using $L_{\mathbf{d}}$. If the property evaluates to $\mathbb{T}$, then we have found a concretization with minimum cost, and we can stop the search. If the property evaluates to $\mathbb{F}$, then we do not explore further from $\mathbf{d}$, since the property will evaluate to $\mathbb{F}$ for $L_{\mathbf{d}'}$ whenever $\mathbf{d} \leq \mathbf{d}'$; in fact, $\mathbf{d}$ is a minimal set of decisions for which the property is not satisfied. However, $\mathbf{d}$ may be reachable by applying the same set of decisions in different order. One way to detect and ignore these situations is to enumerate all possible supersets of concretization $\mathbf{d}$ as $\mathbf{d}'$ and remove those from the queue to verify the property (but the time and memory to do this is exponential in the number of decisions). Alternatively, one can store a table of all the minimal concretizations already explored for which the property evaluates to $\mathbb{F}$ and, when extracting the next candidate concretization $\mathbf{d}$ from the priority queue, search if it is a (not necessarily strict) superset of an entry in the current table; only if it is not, this concretization is model-checked and, if the property evaluates to $\mathbb{U}$, each new concretization obtained from $\mathbf{d}$ by independently taking one of the enabled design decisions in $\mathbf{d}$, is added to the priority queue (this is the approach we use in our prototype).

4 Case Study

To demonstrate the feasibility of our approach, we implemented a C++ prototype in the model-checking tool SMART [7], and used it to run preliminary experiments on a multi-objective path-finding problem for an autonomous rover moving from a *start* base station to an *end* base station. Along the path, the rover

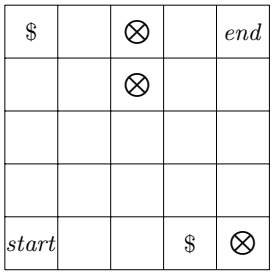

Fig. 3. Example grid: $ is a sampling location, ⊗ is a forbidden location.

collects samples present in certain locations for future analysis, while avoiding certain forbidden locations (Figure 3):

- The rover can move over an area described as an $N \times N$ grid of square cells, thus the position of the rover is given as a cell $[i, j]$, for $0 \leq i, j \leq N-1$.
- The rover has a B_{max} kWh battery, initially fully charged, and consumes 1 kWh to move from cell $[i, j]$ to any of the four adjacent cells, $[i-1, j]$, $[i+1, j]$, $[i, j-1]$, or $[i, j+1]$, subject to not exiting the grid and not entering a forbidden cell. For safety, the battery must not fall below B_{min}.
- A set $\mathcal{F} = \{f_1, ..., f_F\}$ describes F *forbidden cells*, e.g., cells to be avoided due to difficult terrain.
- A set of cells $\mathcal{G} = \{g_1, ..., g_G\}$ describes G *locations of interest* for sample collection. The rover collects a sample the first time it visits a location in $\mathcal{G}$, and must collect at least S_{min} samples.
- The rover starts in cell $[0, 0]$ and must end in cell $[N-1, N-1]$.

We can specify a property defining "success" using propositions to describe the position and battery level of the rover, as well as the number of samples it has collected. This property is satisfiable, if there exists a sequence of moves that realizes success.

We use S_{min} and B_{min} to define a set of design decisions for our analysis. For example, the decisions $S_{min} = s$ for $s \in \{1, 2\}$ and $B_{min} = b$ for $b \in \{1, 2, 4\}$ corresponds to a total of $Q = 2 + 3 = 5$ decisions; of course, the first two, for the value of S_{min}, are mutually exclusive (taking one of them disables the other one), as are the last three, for the value of B_{min}. Later on, we will illustrate how choosing a specific value for B_{min} ($B_{min} = 4$) can affect the labeling of a state in the model, changing it from $\mathbb{U}$ to either $\mathbb{T}$ or $\mathbb{F}$.

4.1 Petri Net Model of the Rover

The rover can successfully complete iff the initial state satisfies the CTL formula

$$\varphi \equiv \mathtt{E}\ (\mathit{battery} \geq B_{min})\ \mathtt{U}\ (\mathit{position} = [N-1, N-1] \wedge\ \mathit{samples} \geq S_{min}),$$

where B_{min} kWh is the lowest level of battery charge we are willing to accept (below that, there is an excessive risk that the rover fails to reach its destination)

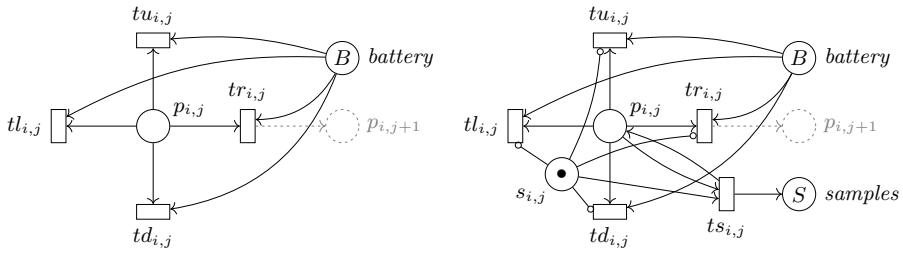

Fig. 4. Our PN model for an individual cell at a generic location (i, j). The left figure is for cells that are not sampling locations, the right figure is for cells that are sampling locations. Initially, place *battery* contains $B = B_{max}$ tokens and place *samples* contains $S = 0$ tokens (of course, places *battery* and *samples* are global, shared by all cells). For a particular (i, j), any of the transitions $tl_{i,j}$, $tu_{i,j}$, $tr_{i,j}$, or $td_{i,j}$ may be absent, if it would correspond to moving outside the grid or into a forbidden location.

and S_{min} is the minimum number of samples we are willing to consider for the mission to be declared a success.

Atomic propositions related to the position of the rover are specified as $position = [i, j]$, which holds whenever the rover is in cell $[i, j]$. Atomic propositions related to the battery level and the number of samples involve design decisions that correspond to choosing the thresholds B_{min} and S_{min}.

For a given battery level, atomic proposition $(battery \geq B_{min})$ is initially unknown, but becomes true or false in each state as soon as we decide a value for B_{min}. Similarly, for a given number of collected samples, atomic proposition $(samples \geq S_{min})$ is initially unknown, but becomes true or false in each state as soon as we decide a value for S_{min}.

Figure 4 shows a single (non-forbidden) cell in our PN model describing the system, depending on whether it is a sampling location (right) or not (left); the cell models are connected together in a grid. Each cell is modeled as a place $p_{i,j}$, and a token in it signifies that the rover is currently in the cell. Transitions $tl_{i,j}$, $tr_{i,j}$, $tu_{i,j}$, and $td_{i,j}$ respectively move the rover left to location $(i, j-1)$, right to location $(i, j+1)$, up to location $(i+1, j)$, or down to location $(i-1, j)$, except that no movement is possible to forbidden cells or cells outside the $N \times N$ grid. The battery level is modeled using a global place *battery*, initialized with B_{max} tokens, and a token (corresponding to 1 kWh) is removed from this place each time a cell is entered (a transition $tl_{i,j}$, $tr_{i,j}$, $tu_{i,j}$, or $td_{i,j}$ fires).

Sampling location cells have a place $s_{i,j}$, where a token signifies that a sample has not yet been collected from this location. Global place *samples* counts the number of collected samples. Transition $ts_{i,j}$ collects a sample the first time

the rover reaches cell (i, j). Inhibitor arcs[4] from place $s_{i,j}$ to the movement transitions $tl_{i,j}$, $tr_{i,j}$, $tu_{i,j}$, and $td_{i,j}$ force the rover to collect a sample before moving. Collecting a sample depletes the battery by a negligible amount.

The system state at any given time can be described as (p, b, s) where p is the current position of the rover (determined by which place $p_{i,j}$ contains a token), b is the current battery level of the rover (tokens in place *battery*), and s is the sample information: which positions have been sampled (place $s_{i,j}$ is empty) and the number of collected samples (tokens in place *samples*). In this model, only the *number* of collected samples is important; the sample locations are needed only to ensure that samples are collected from different locations.

The design decisions are the different valuations of minimum battery level (B_{min}) and minimum number of samples (S_{min}). Then, taking the design decision δ_h, say, corresponding to $B_{min} = 4$, determines that, in all states with battery level ≥ 4, atomic proposition $a \equiv battery \geq B_{min}$ is true. In this case, $\delta_h = (\mathcal{U}_h, \mathcal{T}_h^-, \mathcal{F}_h^-, \mathcal{T}_h^+, \mathcal{F}_h^+)$, where

$$\begin{aligned}
&\mathcal{U}_h = \{((p, b, s), a)\} && \text{(proposition } a \text{ is initially unknown in all states)}\\
&\mathcal{T}_h^- = \mathcal{F}_h^- = \emptyset && \text{(no other dependency affects the enabling of } \delta_h)\\
&\mathcal{T}_h^+ = \{((p, b, s), a) \mid b \geq 4\} && \text{(proposition } a \text{ becomes true in states with } b \geq 4)\\
&\mathcal{F}_h^+ = \{((p, b, s), a) \mid b < 4\} && \text{(proposition } b \text{ becomes false in states with } b < 4).
\end{aligned}$$

When we take an enabled design decision δ_h (we decide a particular value for B_{min}, in this case), the labeling for a subset of state-proposition pairs in $\mathcal{U}_h$ changes to $\mathbb{T}$, if they are in $\mathcal{T}_h^+$, or $\mathbb{F}$, if they are in $\mathcal{F}_h^+$. This results in a concretization with additional $\mathbb{T}$ or $\mathbb{F}$ valuations, and these values cannot be reverted back to $\mathbb{U}$ by taking further enabled decisions, ensuring the mutual exclusion of the design decisions regarding B_{min}.

Figure 5 presents the lattice of concretizations $\mathbf{d}$ for the example grid in Figure 3. The first two components of each node in Figure 5 refer to $S_{min} = 1$ and $S_{min} = 2$, the number of samples collected, and the last three components refer to $B_{min} = 1$, $B_{min} = 2$, and $B_{min} = 4$, the remaining battery level. Design decisions regarding S_{min} and B_{min} are independent of each other and choices for the value of B_{min}(also for S_{min}) are mutually exclusive. For example, let the design decisions for B_{min} be represented as b_1, b_2, and b_4. One could define $enabled(b_1) \equiv \neg taken(b_2) \wedge \neg taken(b_4)$ to state that decision b_1 can only be taken if decision b_2 and decision b_4 have not been taken; $enabled(b_2)$ and $enabled(b_4)$ are defined similarly. The same reasoning applies to possible design choices for S_{min} as well. The order in which the concretization is explored is presented as annotations of the corresponding concretization in Figure 5. It is important to remember that our objective is not to maximize the number of samples collected for a given system configuration (minimum battery level). Instead, we are looking for a set of design decisions (at least S_{min} samples will be collected maintaining

[4] An inhibitor arc from a place p to a transition t disables t whenever p is not empty; for a PN with a finite state space like ours, it is just "syntactic sugar", i.e., its effect could be achieved using an additional place and an ordinary input and output arc.

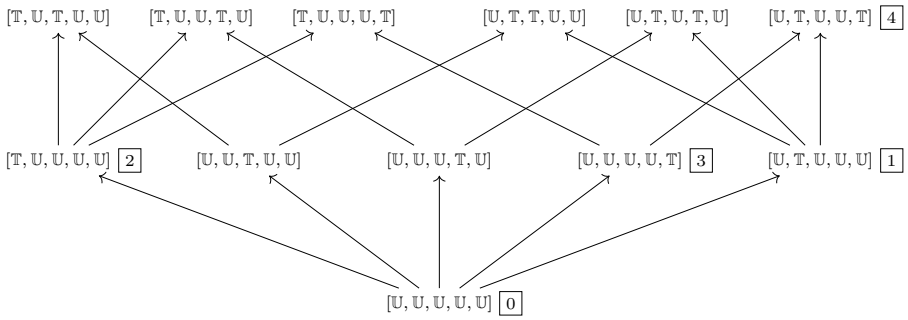

Fig. 5. Evaluation lattice of the decision vector for our example.

Table 1. Results for the autonomous rover for two combinations of N, G, F, B_{max}. The forbidden cell locations are randomly generated, and affect the number of reachable states in the model; thus we show four sample models for each combination.

Parameters				**Kripke Structure**			**Concretizations**			**Solution Found**		
N	G	F	B_{max}	**States**	**Edges**	**Time**	$\mathcal{D}_{srch}$	$\mathcal{D}_{rch}$	**Time**	S_{min}	B_{min}	**Cost**
15	5	5	40	31,855	105,595	4.97 s	13	36	0.46 s	4	4	341
				23,884	78,097	3.75 s	14	36	0.37 s	3	5	341
				47,270	159,877	8.29 s	11	36	0.63 s	4	5	316
				30,067	99,782	4.78 s	14	36	0.46 s	3	5	341
20	5	10	40	22,048	72,994	6.36 s	33	36	1.02 s	3	1	516
				25,418	85,169	7.39 s	29	36	1.04 s	5	1	471
				18,277	60,906	5.14 s	33	36	0.85 s	3	1	516
				26,077	88,865	7.51 s	33	36	1.21 s	3	1	516

at least B_{min} battery) for which the property is satisfied in the system within a budget constraint. The output of our algorithm is not a path in M_{L_d} that proves the satisfaction of the property, but a set of design decisions such that by taking those decisions the system conforms to the property, implying that a path exists for which the property is satisfied (this path could then be obtained as a witness for the existential CTL property).

The optimal concretization for which the rover successfully completes the mission is $\mathbf{d} = [\mathbb{U}, \mathbb{T}, \mathbb{U}, \mathbb{U}, \mathbb{T}]$. Suppose the cost of taking the decisions is given as $C(S_{min} = 1) = 4$, $C(S_{min} = 2) = 2$, $C(B_{min} = 1) = 16$, $C(B_{min} = 2) = 8$, and $C(B_{min} = 4) = 4$. The total cost for the mission success is then $C(\mathbf{d}) = 2{+}4 = 6$.

4.2 Experimental Evaluation

Experimental Setup: We ran experiments using $N \times N$ grids, for grid sizes $N \in \{15, 20, 30, 35\}$, with different values for G (the number of sample locations), F (the number of forbidden cells), and B_{max} (the starting battery level).

We used mutually exclusive decisions to determine the values for the minimum battery level $B_{min} \in \{1, 4, 6, 8, 10\}$ and the number of samples taken $S_{min} \in \{1, 2, ..., G\}$. The costs assigned to design decisions for B_{min} are 200,

150, 100, 50, and 25 in increasing order of remaining battery level. Lowering B_{min} increases the risk of failing to complete the mission (collect sample and reach the destination), thus the cost is higher for lower values of B_{min}. Higher values for S_{min} correspond instead to higher reward from a completed mission; hence the cost is lower cost for higher valuations of S_{min}. When $S_{min} = g$, we assign a cost

$$100(G - (1 + 1/2 + \cdots + 1/g))$$

for $g \in \{1, 2, ..., G\}$ (i.e., the cost decreases approximately logarithmically with increase in the valuation of S_{min}). This captures a "law of diminishing returns": as the number of collected samples g increases, the additional value of the last collected sample grows as $O(1/g)$.

Summary of Results: Table 1 and Table 2 summarize our experimental results; for each PN, we show the size (number of states and edges) of the underlying Kripke structure, and the time required to generate the KS model; the total number of concretizations searched and the total time required to check them; and the design decisions S_{min}, and B_{min} that led to a minimum cost solution. All experiments were run on an Apple Macintosh M1 Pro laptop with 16 GB of RAM, under Ventura 13.3.1.

For Table 1, we randomly assigned F forbidden cells, and repeated the experiment multiple times for a specific choice of N, G, F, and B_{max}. For Table 2, given specific values of N and G, we ran experiments by varying F and B_{max}. In this setup, we first run experiments by assigning forbidden cells for the smallest values of F and randomly add forbidden cells for larger values of F. For instance, for $N = 20$ and $G = 10$, we start with $F = 10$, thus randomly select 10 forbidden cells; then, for $F = 20$, we use the same grid with the previously selected 10 forbidden cells and randomly add 10 more forbidden cells. Thus, the model with $N = 20$, $G = 10$, and $F = 10$ differs from the one with $N = 20$, $G = 10$, and $F = 20$ only in the location of the 10 additional forbidden cells.

The number $\mathcal{D}_{srch}$ gives an indication of how much computation time our approach would save, as compared to running a model checker on a suite of PN models with varying S_{min} and B_{min} values "hard coded" into the models: we would need to generate the Kripke structure for *every* PN, while our approach only builds it once. It took several hours for the model generation and computation times for our largest Kripke structure ($N = 35$, $F = 40$, $B_{max} = 80$). For example, consider configuration $N = 35$, $G = 15$, $F = 30$, and $B_{max} = 80$: our approach took $(22{,}874 + 2{,}367) = 25{,}241$ seconds (about 7 hours) to explore 28 different concretizations, while verifying each PN separately for a particular set of design decisions would have instead taken $28 \times 22{,}874 + 2{,}367 = 642{,}839$ seconds (about 178.5 hours).

The number of reachable states in the model is (loosely) upper-bounded by

$$(N^2 - F) \cdot 2^G \cdot (1 + B_{max}/2) \tag{1}$$

because the rover can potentially be in any of the non-forbidden cells, each of the G sampling locations could have been visited or not, and any cell that can

Table 2. Results for the autonomous rover, for various model instances.

Parameters				Kripke Structure			Concretizations			Solution Found		
N	G	F	B_{max}	**States**	**Edges**	**Time**	$\mathcal{D}_{srch}$	$\mathcal{D}_{rch}$	**Time**	S_{min}	B_{min}	**Cost**
20	10	10	40	89,687	293,159	29 s	59	66	10 s	5	1	971
		20	40	87,083	277,910	26 s	59	66	7 s	5	1	971
30	15	20	60	2,422,351	8,113,490	1600 s	83	96	529 s	7	1	1,440
		30	60	2,263,154	7,490,314	1463 s	83	96	491 s	7	1	1,440
35	15	30	80	25,939,995	89,170,887	22,874 s	28	96	2,367 s	8	5	1,253
		40	80	25,694,819	87,550,513	22,740 s	28	96	2,334 s	8	5	1,253

be visited, will be reachable either in even- or odd-length paths from $[0, 0]$. As shown in Table 1 and 2, the actual number of reachable states can be much smaller. For example, when $N = 20$, $G = 10$, $F = 20$, and $B_{max} = 40$, Eq. (1) gives a bound of 8,171,520 states, but only 87,083 states are reachable.

Complexity analysis. The worst-case complexity of the algorithm is the number N of nodes of the semi-lattice of concretizations that need to be explored (potentially $N = 2^n$, if there are n decisions and they are all independent, possibly many fewer if some of the n decisions are mutually exclusive), times the complexity of CTL model checking for the concretization corresponding to the node (the complexity of CTL model checking is linear in the size of the Kripke structure M and in the size of the CTL formula φ, i.e., $\mathcal{O}(|M| \cdot |\varphi|)$ where $|M| = |\mathcal{S}| + |\mathcal{N}|$). If $\mathcal{D}_{search} \subseteq \mathcal{D}_{reach}$ is the set of concretizations that must be explored, model checking a CTL formula in a partially specified system M with n design decisions has a time complexity in $\mathcal{O}(|\mathcal{D}_{search}| \cdot |M| \cdot |\varphi|)$ where $|\mathcal{D}_{search}| \leq |\mathcal{D}_{reach}| \leq 2^n$.

5 Related Work

Chechik et al. [13] discussed an approach for dealing with "uncertainty" in the software development process. They define uncertainty as having multiple possible modeling choices instead of just one. These uncertainties can arise from problem-domain ambiguities, various design alternatives, or differing interpretations of requirements by multiple stakeholders. They define a partial model as the "union" of potential (low-level) model designs and verify if the property of interest is satisfied in any, all, or none of these models. Based on the result, they apply algorithms to refine the models. The key difference between their approach and ours is that they propose a bottom-up approach to find a "unified" system design that satisfies the property, while we propose a top-down approach where the input is a high-level model defining M, while the (likely quite large) set of potential designs is an output. In our approach, only a subset of these potential designs may need to be considered, as they are explored in increasing cost order. Notably, [13] does not incorporate the notion of cost in their approach to decide a particular system design.

Design Space Exploration (DSE) is a related field with techniques similar to the ones presented. Specific to Software Product Lines (SPLs), [2] uses con-

straint programming to perform automated SPL exploration. FORMULA [19] is a DSE tool that pre-processes then encodes an exploration query into an SMT-query via a user-defined domain-specific language. DESERT [25] is similar to FORMULA, but uses BDDs and a system hierarchy to guide their search. [23] instead uses a pseudo-Boolean encoding to optimize component-based software system designs. Importantly, none of these approaches consider temporal logic properties as system constraints.

For temporal logic-based DSE, FuseIC3 [11] modifies the IC3 algorithm to perform model checking on sets of potential models. Similarly, D^3 [12] is a pre-processing technique that prunes an input space of potential models by leveraging the structural relationships between them. As a case study, [14] investigates the design space of a NASA automated ATC using model checking and contract-based design. None of these techniques consider CTL model checking. [10] solves a similar problem by performing a model checking of "feature CTL" (fCTL). The authors annotate temporal operators with propositional formulas specifying which configurations that CTL expression must hold for. [1] also does model checking of fCTL formulas, instead using SAT-based algorithms. None of these techniques consider cost, thus optimization.

Pecheur and Raimondi [26] introduced the formalism Mixed Transition System (MTS) that combines the definition of a Kripke structure and a Labeled Transition System and proposed Action-Restricted CTL (ARCTL) to describe the behaviors of MTS. A typical reachability property in ARCTL is expressed as $\mathtt{E}_\alpha\mathtt{F}\varphi$: there exists a path where the action label of each edge in the path satisfies α and the path leads to a state satisfying φ. The authors presented a reduction of ARCTL model checking over MTSs to CTL model checking over KSs using a "post-projection" that pushes action labels from transitions to their target states. In contrast, our framework maintains classical CTL syntax, eliminating the need to transform formulas or extend the logic. Instead, it handles incompleteness at the model level by introducing *unknown* design decisions, and evaluates properties over sets of concretizations that extend the partial model.

6 Conclusion

We presented a formalism and algorithmic framework for verifying systems with incomplete specifications, where incompleteness arises from pending design decisions during development. These unresolved decisions are captured as atomic propositions with unknown values in a partially labeled Kripke structure. Our approach identifies concrete decisions that lead to satisfaction of CTL-specified requirements. While CTL is interpreted over Kripke structures, we use Petri nets as a high-level modeling formalism to compactly generate large state spaces and to encode how design choices concretize unknowns in the underlying structure.

Future work includes developing incremental verification strategies that exploit lattice-based relationships between labelings, with the goal of reusing results when extending partial assignments. We also plan to investigate cost-guided heuristics that not only prioritize low-cost decisions but also estimate the likelihood of those decisions leading to successful completions.

References

1. Ben-David, S., Sterin, B., Atlee, J.M., Beidu, S.: Symbolic model checking of product-line requirements using sat-based methods. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. vol. 1, pp. 189–199. IEEE (2015)
2. Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated reasoning on feature models. In: International Conference on Advanced Information Systems Engineering. pp. 491–503. Springer (2005)
3. Bruns, G., Godefroid, P.: Model checking partial state spaces with 3-valued temporal logics. In: Proceedings of the 11th International Conference on Computer Aided Verification. pp. 274–287. CAV '99, Springer-Verlag, London, UK (1999), http://dl.acm.org/citation.cfm?id=647768.733795
4. Bruns, G., Godefroid, P.: Generalized model checking: Reasoning about partial state spaces. In: Concurrency Theory. pp. 168–182. CONCUR, Springer-Verlag, Berlin, Heidelberg (2000), http://dl.acm.org/citation.cfm?id=646735.701611
5. Bruns, G., Godefroid, P.: Model checking with multi-valued logics. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) Automata, Languages and Programming. pp. 281–293. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
6. Chechik, M., Devereux, B., Easterbrook, S., Gurfinkel, A.: Multi-valued symbolic model-checking **12**(4) (2003), https://doi.org/10.1145/990010.990011
7. Ciardo, G., Miner, A.S., Wan, M.: Advanced features in SMART: the Stochastic Model checking Analyzer for Reliability and Timing. ACM SIGMETRICS Perf. Eval. Rev. **36**(4), 58–63 (2009)
8. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV Version 2: an open source tool for symbolic model checking. In: Proc. CAV. LNCS 2404, Springer (July 2002)
9. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Proc. IBM Workshop on Logics of Programs. pp. 52–71. LNCS 131, Springer (1981)
10. Classen, A., Heymans, P., Schobbens, P.Y., Legay, A.: Symbolic model checking of software product lines. In: Proceedings of the 33rd International Conference on Software Engineering. pp. 321–330 (2011)
11. Dureja, R., Rozier, K.Y.: FuseIC3: An algorithm for checking large design spaces. In: 2017 Formal Methods in Computer Aided Design (FMCAD). pp. 164–171. IEEE (2017)
12. Dureja, R., Rozier, K.Y.: More scalable LTL model checking via discovering design-space dependencies (D3). In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 309–327. Springer (2018)
13. Famelis, M., Salay, R., Chechik, M.: Partial models: Towards modeling and reasoning with uncertainty. In: 2012 34th International Conference on Software Engineering (ICSE). pp. 573–583 (2012). https://doi.org/10.1109/ICSE.2012.6227159
14. Gario, M., Cimatti, A., Mattarei, C., Tonetta, S., Rozier, K.Y.: Model checking at scale: Automated air traffic control design space exploration. In: Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II 28. pp. 3–22. Springer (2016)
15. Godefroid, P., Huth, M., Jagadeesan, R.: Abstraction-based model checking using modal transition systems. In: Larsen, K.G., Nielsen, M. (eds.) Concurrency Theory (CONCUR). pp. 426–440. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)

16. Gruler, A., Leucker, M., Scheidemann, K.: Modeling and model checking software product lines. In: Barthe, G., de Boer, F.S. (eds.) Formal Methods for Open Object-Based Distributed Systems. pp. 113–131. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
17. Holzmann, G.J.: The SPIN Model Checker. Addison-Wesley (2003)
18. Huth, M., Jagadeesan, R., Schmidt, D.A.: Modal transition systems: A foundation for three-valued program analysis. In: European Symposium on Programming Languages and Systems. pp. 155–169. ESOP '01, Springer-Verlag, London, UK, UK (2001), http://dl.acm.org/citation.cfm?id=645395.651926
19. Kang, E., Jackson, E., Schulte, W.: An approach for effective design space exploration. In: Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems: 16th Monterey Workshop 2010, Redmond, WA, USA, March 31-April 2, 2010, Revised Selected Papers 16. pp. 33–54. Springer (2011)
20. Kleene, S.C.: Introduction to Metamathematics. North Holland (1987)
21. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM: Probabilistic Symbolic Model Checker. In: Proc. Computer Performance Evaluation / TOOLS. pp. 200–204. Springer (Apr 2002)
22. Larsen, K.G., Nyman, U., Wkasowski, A.: On modal refinement and consistency. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007 – Concurrency Theory. pp. 105–119. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
23. Manolios, P., Vroon, D., Subramanian, G.: Automating component-based system assembly. In: Proceedings of the 2007 international symposium on Software testing and analysis. pp. 61–72 (2007)
24. Murata, T.: Petri nets: properties, analysis and applications. Proc. of the IEEE **77**(4), 541–579 (Apr 1989)
25. Neema, S., Sztipanovits, J., Karsai, G., Butts, K.: Constraint-based design-space exploration and model synthesis. In: International Workshop on Embedded Software. pp. 290–305. Springer (2003)
26. Pecheur, C., Raimondi, F.: Symbolic model checking of logics with actions. In: International Workshop on Model Checking and Artificial Intelligence. pp. 113–128. Springer (2006)