Maximum Realizability for Linear Temporal Logic Specifications

Rayna Dimitrova[1]*, Mahsa Ghasemi[2], and Ufuk Topcu[2]

[1]University of Leicester, UK
[2]University of Texas at Austin, USA

Abstract. Automatic synthesis from linear temporal logic (LTL) specifications is widely used in robotic motion planning and control of autonomous systems. A common specification pattern in such applications consists of an LTL formula describing the requirements on the behaviour of the system, together with a set of additional desirable properties. We study the synthesis problem in settings where the overall specification is unrealizable, more precisely, when some of the desirable properties have to be (temporarily) violated in order to satisfy the system's objective. We provide a quantitative semantics of sets of safety specifications, and use it to formalize the "best-effort" satisfaction of such *soft* specifications while satisfying the *hard* LTL specification. We propose an algorithm for synthesizing implementations that are optimal with respect to this quantitative semantics. Our method builds upon the idea of bounded synthesis, and we develop a MaxSAT encoding which allows for maximizing the quantitative satisfaction of the soft specifications. We evaluate our algorithm on scenarios from robotics and power distribution networks.

1 Introduction

Automatic synthesis from temporal logic specifications is increasingly becoming a viable alternative for system design in a number of domains such as control and robotics. The main advantage of synthesis is that it allows the system designer to focus on *what* the system should do, rather than on *how* it should do it. Thus, the main challenge becomes providing the right specification of the system's required behaviour. While significantly easier than developing a system at a lower level, specification design is on its own a difficult and error-prone task. For example, in the case of systems operating in a complex adversarial environment, such as robots, the specification might be over-constrained, and as a result unrealizable, due to failure to account for some of the behaviours of the environment. In other cases, the designer might have several alternative specifications in mind, possibly with some preferences, and wants to know what the best realizable combination of requirements is. For instance, a temporary violation of a safety requirement might be acceptable, if it is necessary to achieve an important goal. In such cases it is desirable that, when the specification is unrealizable, the synthesis

* This work was done while the author was at The University of Texas at Austin.

procedure provides a "best-effort" implementation either according to some user-given criteria, or according to the semantics of the specification language.

The challenges of specification design motivate the need to develop synthesis methods for the *maximum realizability problem*, where the input to the synthesis tool consists of a *hard specification* which *must* be satisfied by the system, and *soft specifications* which describe other desired, possibly prioritized properties.

A key ingredient of the formulation of the maximum realizability problem is a quantitative semantics of the soft requirements. We focus on soft specifications of the form $\square\varphi_1, \ldots, \square\varphi_n$, where each φ_i is a safety LTL formula, and consider a quantitative semantics typically used in the context of robustness. The quantitative semantics accounts for how often each φ_i is satisfied. In particular, we consider truth values corresponding to φ_i being satisfied at every point of an execution, being violated only finitely many times, being both violated and satisfied infinitely often, or being continuously violated from some point on. Based on this semantics, we define the numerical value of a conjunction $\square\varphi_1 \wedge \ldots \wedge \square\varphi_n$ of soft specifications in a given implementation. We propose a method for synthesizing an implementation that maximizes this value.

Our approach to maximum realizability is based on the bounded synthesis technique. Bounded synthesis is able to synthesize implementations by leveraging the power of SAT (or QBF, or SMT) solvers. Since maximum realizability is an optimization problem, we reduce its bounded version to maximum satisfiability (MaxSAT). More precisely, we encode the bounded maximum realizability problem with hard and soft specifications as a partial weighted MaxSAT problem, where hard specifications are captured by hard clauses in the MaxSAT formulation, and the weights of soft clauses encode the quantitative semantics of soft specifications. By adjusting these weights our approach can easily capture different quantitative semantics. Although the formulation encodes the bounded maximum realizability problem (where the maximum size of the implementation is fixed), by providing a bound on the size of the optimal implementation, we establish the completeness of our synthesis method. The existence of such completeness bound is guaranteed by considering quantitative semantics in which the values of soft specifications can be themselves encoded by LTL formulas.

We have applied the proposed synthesis method to examples from two domains where considering combinations of hard and soft specifications is natural and often unavoidable. For example, such a combination of specifications arises in power networks where generators of limited capacity have to power a set of vital and non-vital loads, whose total demand may exceed the capacity of the generators. Another example is robotic navigation, where due to the adversarial nature of the environment in which robots operate, safety requirements might prevent a system from achieving its goal, or a large number of tasks of different nature might not necessarily be consistent when posed together.

Related work. Maximum realizability and several closely related problems have attracted significant attention in recent years. Planning over a finite horizon with prioritized safety requirements was studied in [20], where the goal is to synthesize a least-violating control strategy. A similar problem for infinite-horizon temporal logic planning was studied in [12], which seeks to revise an inconsistent specification, minimizing the cost of revision with respect to costs for atomic propositions provided by the specifier. [14] describes a method for computing optimal plans for co-safe LTL specifications, where optimality is again with respect to the cost of violating each atomic proposition, which is provided by the designer. All of these approaches are developed for the planning setting, where there is no adversarial environment, and thus they are able to reduce the problem to the computation of an optimal path in a graph. The case of probabilistic environments was considered in [15]. In contrast, in our work we seek to maximize the satisfaction of the given specification against the worst-case behaviour of the environment.

The problem setting that is the closest to ours is that of [19]. The authors of [19] study a maximum realizability problem in which the specification is a conjunction of a *must* (or *hard*, in our terms) LTL specification, and a number of weighted *desirable* (or *soft*, in our terms) specifications of the form $\square \varphi$, where φ is an arbitrary LTL formula. When φ is not a safety property it is first strengthened to a safety formula before applying the synthesis procedure, which then weakens the result to a mean-payoff term. Thus, while [19] considers a broader class of soft specifications than we do, when φ is not a safety property there is no clear relationship between $\square \varphi$ and the resulting mean-payoff term. When applied to multiple soft specifications, the method from [19] combines the corresponding mean-payoff terms in a weighted sum, and synthesizes an implementation optimizing the value of this sum. Thus, it is not possible to determine to what extent the individual desirable specifications are satisfied without inspecting the synthesized implementation. In contrast, in our maximum realizability procedure each satisfaction value is characterized as an LTL formula, which is useful for explainability and providing feedback to the designer.

To the best of our knowledge, our work is the first to employ MaxSAT in the context of reactive synthesis. MaxSAT has been used in [11] for preference-based planning. However, since maximum realizability is concerned with reactive systems, it requires a fundamentally different approach than planning.

Two other main research directions related to maximum realizability are *quantitative synthesis* and *specification debugging*. There are two predominant flavours of quantitative synthesis problems studied in the literature. In the first one (cf. [4]), the goal is to generate an implementation that maximizes the value of a mean-payoff objective, while possibly satisfying some ω-regular specification. In the second setting (cf. [1, 18]), the system requirements are formalized in a multi-valued temporal logic. The synthesis methods in these works, however, do not solve directly the corresponding optimization problem, but instead check for the existence of an implementation whose value is in a given set. The optimization problem can then be reduced to a sequence of such queries.

An optimal synthesis problem for an ordered sequence of prioritized ω-regular properties was studied in [2], where the classical fixpoint-based game-solving algorithms are extended to a quantitative setting. The main difference in our work is that we allow for incomparable soft specifications each with a number of prioritized relaxations, for which the equivalent set of preference-ordered combinations would be of size exponential in the number of soft specifications. Our MaxSAT formulation avoids considering explicitly these combinations.

In specification debugging there is a lot of research dedicated to finding good explanations for the unsatisfiability or unrealizability of temporal specifications [6], and more generally to the analysis of specifications [5, 9]. Our approach to maximum realizability can prove useful for specification analysis, since instead of simply providing an optimal value, it computes an optimal relaxation of the given specification in the form of another LTL formula.

2 Maximum Realizability Problem

We first give an overview of linear-time temporal logic (LTL) and the corresponding synthesis problem, which asks to synthesize an implementation, in the form of a transition system, that satisfies an LTL formula given as input.

Then, we proceed by providing a quantitative semantics for a class of LTL formulas, and the definition of the corresponding maximum realizability problem.

2.1 Specifications, Transition Systems, and the Synthesis Problem

Linear-time temporal logic (LTL) is a standard specification language for formalizing requirements on the behaviour of reactive systems. Given a finite set $\mathcal{P}$ of atomic propositions, the set of LTL formulas is generated by the grammar $\varphi := p \mid \mathit{true} \mid \mathit{false} \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2 \mid \varphi_1 \,\mathcal{R}\, \varphi_2$, where $p \in \mathcal{P}$ is an atomic proposition, $\bigcirc$ is the *next* operator, $\mathcal{U}$ is the *until* operator, and $\mathcal{R}$ is the *release* operator. As usual, we define the derived operators *finally*: $\diamondsuit\varphi = \mathit{true}\,\mathcal{U}\,\varphi$ and *globally*: $\square\varphi = \mathit{false}\,\mathcal{R}\,\varphi$. Every LTL formula can be converted to an equivalent one in negation normal form (NNF), where negations appear only in front of atomic propositions. Thus, we consider only formulas in NNF.

Let $\Sigma = 2^{\mathcal{P}}$ be the finite alphabet consisting of the valuations of the propositions $\mathcal{P}$. A letter $\sigma \in \Sigma$ is interpreted as the valuation that assigns value true to all $p \in \sigma$ and false to all $p \in \mathcal{P} \setminus \sigma$. LTL formulas are interpreted over infinite words $w \in \Sigma^\omega$. If a word $w \in \Sigma^\omega$ satisfies an LTL formula φ, we write $w \models \varphi$. The definition of the semantics of LTL can be found for instance in [3]. We denote with $|\varphi|$ the length of φ, and with $\mathsf{subf}(\varphi)$ the set of its subformulas.

A *safety LTL formula* φ is an LTL formula such that for each $w \in \Sigma^\omega$ with $w \not\models \varphi$ there exists $u \in \Sigma^*$ such that for all $v \in \Sigma^\omega$ it holds that $u \cdot v \not\models \varphi$ (u is called a *bad prefix* for φ). A class of safety LTL formulas is the class of *syntactically safe* LTL formulas, which contain no occurrences of $\mathcal{U}$ in their NNF.

In the rest of the paper we assume that the set $\mathcal{P}$ of atomic propositions is partitioned into disjoint sets of *input* propositions $\mathcal{I}$ and *output* propositions $\mathcal{O}$.

A *transition system* over a set of input propositions $\mathcal{I}$ and a set of output propositions $\mathcal{O}$ is a tuple $\mathcal{T} = (S, s_0, \tau)$, where S is a set of states, s_0 is the initial state, and the transition function $\tau : S \times 2^{\mathcal{I}} \to S \times 2^{\mathcal{O}}$ maps a state s and a valuation $\sigma_I \in 2^{\mathcal{I}}$ of the input propositions to a successor state s' and a valuation $\sigma_O \in 2^{\mathcal{O}}$ to the output propositions. Let $\mathcal{P} = \mathcal{I} \cup \mathcal{O}$ be the set of all propositions. For $\sigma \in \Sigma = 2^{\mathcal{P}}$ we denote $\sigma \cap \mathcal{I}$ by σ_I, and $\sigma \cap \mathcal{O}$ by σ_O.

If the set S is finite, then $\mathcal{T}$ is a finite-state transition system. In this case we define the size $|\mathcal{T}|$ of $\mathcal{T}$ to be the number of its states, i.e., $|\mathcal{T}| \overset{\text{def}}{=} |S|$.

An *execution* of $\mathcal{T}$ is an infinite sequence $s_0, (\sigma_{I0} \cup \sigma_{O0}), s_1, (\sigma_{I1} \cup \sigma_{O1}), s_2 \ldots$ such that s_0 is the initial state, and $(s_{i+1}, \sigma_{Oi}) = \tau(s_i, \sigma_{Ii})$ for every $i \geq 0$. The corresponding sequence $(\sigma_{I0} \cup \sigma_{O0}), (\sigma_{I1} \cup \sigma_{O1}), \ldots \in \Sigma^\omega$ is called a trace. We denote with $\mathsf{Traces}(\mathcal{T})$ the set of all traces of a transition system $\mathcal{T}$.

We say that a transition system $\mathcal{T}$ satisfies an LTL formula φ over atomic propositions $\mathcal{P} = \mathcal{I} \cup \mathcal{O}$, denoted $\mathcal{T} \models \varphi$, if $w \models \varphi$ for every $w \in \mathsf{Traces}(\mathcal{T})$.

The *realizability problem for LTL* is to determine whether for a given LTL formula φ there exists a transition system $\mathcal{T}$ that satisfies φ. The *LTL synthesis problem* asks to construct such a transition system if one exists.

Often, the specification is a combination of multiple requirements, which might not be realizable in conjunction. In such a case, in addition to reporting the unrealizability to the system designer, we would like the synthesis procedure to construct an implementation that satisfies the specification "as much as possible". Such implementation is particularly useful in the case where some of the requirements describe desirable but not necessarily essential properties of the system. To determine what "as much as possible" formally means, a quantitative semantics of the specification language is necessary. In the next subsection we provide such semantics for a fragment of LTL. The quantitative interpretation is based on the standard semantics of LTL formulas of the form $\Box\varphi$.

2.2 Quantitative Semantics of Soft Safety Specifications

Let $\Box\varphi_1, \ldots, \Box\varphi_n$ be LTL specifications, where each φ_i is a safety LTL formula. In order to formalize the maximal satisfaction of $\Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n$, we first give a quantitative semantics of formulas of the form $\Box\varphi$.

Quantitative semantics of safety specifications. For an LTL formula of the form $\Box\varphi$ and a transition system $\mathcal{T}$, we define *the value $val(\mathcal{T}, \Box\varphi)$ of $\Box\varphi$ in $\mathcal{T}$* as

$$val(\mathcal{T}, \Box\varphi) \overset{\text{def}}{=} \begin{cases} (1,1,1) & \text{if } \mathcal{T} \models \Box\varphi, \\ (1,1,0) & \text{if } \mathcal{T} \not\models \Box\varphi \text{ and } \mathcal{T} \models \Diamond\Box\varphi, \\ (1,0,0) & \text{if } \mathcal{T} \not\models \Box\varphi \text{ and } \mathcal{T} \not\models \Diamond\Box\varphi, \text{ and } \mathcal{T} \models \Box\Diamond\varphi, \\ (0,0,0) & \text{if } \mathcal{T} \not\models \Box\varphi, \text{ and } \mathcal{T} \not\models \Diamond\Box\varphi, \text{ and } \mathcal{T} \not\models \Box\Diamond\varphi. \end{cases}$$

Thus, the value of $\Box\varphi$ in a transition system $\mathcal{T}$ is a vector $(v_1, v_2, v_3) \in \{0,1\}^3$, where the value $(1,1,1)$ corresponds to the *true* value in the classical semantics of LTL. When $\mathcal{T} \not\models \Box\varphi$, the values $(1,1,0)$, $(1,0,0)$ and $(0,0,0)$ capture the extent to which φ holds or not along the traces of $\mathcal{T}$. For example, if $val(\mathcal{T}, \Box\varphi) = (1,0,0)$, then φ holds infinitely often on each trace of $\mathcal{T}$, but there exists a trace of $\mathcal{T}$ on which φ is violated infinitely often. When $val(\mathcal{T}, \Box\varphi) = (0,0,0)$, then on some trace of $\mathcal{T}$, φ holds for at most finitely many positions.

Note that by the definition of val, if $val(\mathcal{T}, \Box\varphi) = (v_1, v_2, v_3)$, then *(1)* $v_1 = 1$ iff $\mathcal{T} \models \Box\Diamond\varphi$, *(2)* $v_2 = 1$ iff $\mathcal{T} \models \Diamond\Box\varphi$, and *(3)* $v_3 = 1$ iff $\mathcal{T} \models \Box\varphi$. Thus, the lexicographic ordering on $\{0,1\}^3$ captures the preference of one transition system over another with respect to the quantitative satisfaction of $\square\varphi$.

Example 1. Suppose that we want to synthesize a transition system representing a navigation strategy for a robot working at a restaurant. We require that the robot must serve the VIP area infinitely often, formalized in LTL as $\square\diamondsuit\, \mathit{vip\_area}$. We also desire that the robot never enters the staff's office, formalized as $\square\neg\mathit{office}$. Now, suppose that initially the key to the VIP area is in the office. Thus, in order to satisfy $\square\diamondsuit\, \mathit{vip\_area}$, the robot must violate $\square\neg\mathit{office}$. A strategy in which the office is entered only once, and satisfies $\diamondsuit\square\neg\mathit{office}$, is preferable to one which enters the office over and over again, and only satisfies $\square\diamondsuit\neg\mathit{office}$. Thus, we want to synthesize a strategy $\mathcal{T}$ maximizing $\mathit{val}(\mathcal{T},\square\neg\mathit{office})$.

In order to compare implementations with respect to their satisfaction of a conjunction $\square\varphi_1 \wedge \ldots \wedge \square\varphi_n$ of several safety specifications, we will extend the above definition. We consider the case when the specifier has not expressed any preference for the individual conjuncts. Consider the following example.

Example 2. We consider again the restaurant robot, now with two soft specifications. The soft specification $\square(\mathit{req1} \rightarrow \bigcirc\, \mathit{table1})$ requires that each request by table 1 is served immediately at the next time instance. Similarly, $\square(\mathit{req2} \rightarrow \bigcirc\, \mathit{table2})$, requires the same for table number 2. Since the robot cannot be at both tables simultaneously, formalized as the hard specification $\square(\neg\mathit{table1} \vee \neg\mathit{table2})$, the conjunction of these requirements is unrealizable. Unless the two tables have priorities, it is preferable to satisfy each of $\mathit{req1} \rightarrow \bigcirc\, \mathit{table1}$ and $\mathit{req2} \rightarrow \bigcirc\, \mathit{table2}$ infinitely often, rather than serve one and the same table all the time.

Quantitative semantics of conjunctions. To capture the idea illustrated in Example 2, we define a value function, which, intuitively, gives higher values to transition systems in which a fewer number of soft specifications have low values. Formally, let *the value of* $\square\varphi_1 \wedge \ldots \wedge \square\varphi_n$ *in* $\mathcal{T}$ be

$$\mathit{val}(\mathcal{T},\square\varphi_1 \wedge \ldots \wedge \square\varphi_n) \stackrel{\text{def}}{=} \Big(\sum_{i=1}^{n} v_{i,1}, \sum_{i=1}^{n} v_{i,2}, \sum_{i=1}^{n} v_{i,3}\Big),$$

where $\mathit{val}(\mathcal{T},\square\varphi_i) = (v_{i,1}, v_{i,2}, v_{i,3})$ for $i \in \{1,\ldots,n\}$. To compare transition systems according to these values, we use lexicographic ordering on $\{0,\ldots,n\}^3$.

Example 3. For the specifications in Example 2, the above value function assigns value $(2,0,0)$ to a system satisfying $\square\diamondsuit(\mathit{req1} \rightarrow \bigcirc\, \mathit{table1})$ and $\square\diamondsuit(\mathit{req2} \rightarrow \bigcirc\, \mathit{table2})$, but neither of $\diamondsuit\square(\mathit{req1} \rightarrow \bigcirc\, \mathit{table1})$ and $\diamondsuit\square(\mathit{req2} \rightarrow \bigcirc\, \mathit{table2})$. It assigns the smaller value $(1,1,1)$ to an implementation that gives priority to table 1 and satisfies $\square(\mathit{req1} \rightarrow \bigcirc\, \mathit{table1})$ but not $\square\diamondsuit(\mathit{req2} \rightarrow \bigcirc\, \mathit{table2})$.

According to the above definition, a transition system that satisfies all soft requirements to some extent is considered better in the lexicographic ordering than a transition system that satisfies one of them and violates all the others. We could instead inverse the order of the sums in the triple, thus giving preference to satisfying some soft specification, over having some lower level of satisfaction over all of them. The next example illustrates the differences between the two variations.

Example 4. For the two soft specifications from Example 2, reversing the order of the sums in the definition of $\mathit{val}(\mathcal{T},\square\varphi_1 \wedge \ldots \wedge \square\varphi_n)$ results in giving the higher value $(1,1,1)$ to a transition system that satisfies $\square(\mathit{req1} \rightarrow \bigcirc\,\mathit{table1})$ but not $\square\Diamond(\mathit{req2} \rightarrow \bigcirc\,\mathit{table2})$, and the lower value $(0,0,2)$ to the one that guarantees only $\square\Diamond(\mathit{req1} \rightarrow \bigcirc\,\mathit{table1})$ and $\square\Diamond(\mathit{req2} \rightarrow \bigcirc\,\mathit{table2})$. The most suitable ordering usually depends on the specific application.

In [7] we discuss generalizations of the framework, where the user provides a set of relaxations for each of the soft specifications, and possibly a priority ordering among the soft specifications, or numerical weights.

2.3 Maximum realizability

Using the definition of quantitative satisfaction of soft safety specifications, we now define the maximum realizability problem, which asks to synthesize a transition system that satisfies a given *hard* LTL specification, and is optimal with respect to the satisfaction of a conjunction of *soft* safety specifications.

Maximum realizability problem: Given an LTL formula φ and formulas $\square\varphi_1,\ldots,\square\varphi_n$, where each φ_i is a safety LTL formula, the maximum realizability problem asks to determine if there exists a transition system $\mathcal{T}$ such that $\mathcal{T} \models \varphi$, and if the answer is positive, to synthesize a transition system $\mathcal{T}$ such that $\mathcal{T} \models \varphi$, and such that for every transition system $\mathcal{T}'$ with $\mathcal{T}' \models \varphi$ it holds that $\mathit{val}(\mathcal{T},\square\varphi_1 \wedge \ldots \wedge \square\varphi_n) \geq \mathit{val}(\mathcal{T}',\square\varphi_1 \wedge \ldots \wedge \square\varphi_n)$.

Bounded maximum realizability problem: Given an LTL formula φ and formulas $\square\varphi_1,\ldots,\square\varphi_n$, where each φ_i is a safety LTL formula, and a bound $b \in \mathbb{N}_{>0}$, the bounded maximum realizability problem asks to determine if there exists a transition system $\mathcal{T}$ with $|\mathcal{T}| \leq b$ such that $\mathcal{T} \models \varphi$, and if the answer is positive, to synthesize a transition system $\mathcal{T}$ such that $\mathcal{T} \models \varphi$, $|\mathcal{T}| \leq b$ and such that for every transition system $\mathcal{T}'$ with $\mathcal{T}' \models \varphi$ and $|\mathcal{T}'| \leq b$, it holds that $\mathit{val}(\mathcal{T},\square\varphi_1 \wedge \ldots \wedge \square\varphi_n) \geq \mathit{val}(\mathcal{T}',\square\varphi_1 \wedge \ldots \wedge \square\varphi_n)$.

3 Preliminaries

In this section we recall bounded synthesis, introduced in [17], and in particular the approach based on reduction to SAT. We begin with the necessary preliminaries from automata theory, and the notion of annotated transition systems.

3.1 Bounded Synthesis

A *Büchi automaton* over a finite alphabet Σ is a tuple $\mathcal{A} = (Q, q_0, \delta, F)$, where Q is a finite set of states, q_0 is the initial state, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and $F \subseteq Q$ is a subset of the set of states. A run of $\mathcal{A}$ on an infinite word $w = \sigma_0\sigma_1 \ldots \in \Sigma^\omega$ is an infinite sequence $q_0, q_1, \ldots$ of states, where q_0 is the initial state and for every $i \geq 0$ it holds that $(q_i, \sigma_i, q_{i+1}) \in \delta$.

A run of a Büchi automaton is accepting if it contains infinitely many occurrences of states in F. A *co-Büchi automaton* $\mathcal{A} = (Q, q_0, \delta, F)$ differs from a Büchi automaton in the accepting condition: a run of a co-Büchi automaton is accepting if it contains only *finitely many* occurrences of states in F. For a Büchi automaton the states in F are called *accepting states*, while for a co-Büchi automaton they are called *rejecting states*. A *nondeterministic* automaton $\mathcal{A}$ accepts a word $w \in \Sigma^\omega$ if *some* run of $\mathcal{A}$ on w is accepting. A *universal* automaton $\mathcal{A}$ accepts a word $w \in \Sigma^\omega$ if *every* run of $\mathcal{A}$ on w is accepting.

The *run graph* of a universal automaton $\mathcal{A} = (Q, q_0, \delta, F)$ on a transition system $\mathcal{T} = (S, s_0, \tau)$ is the unique graph $G = (V, E)$ with set of nodes $V = S \times Q$ and set of labelled edges $E \subseteq V \times \Sigma \times V$ such that $((s, q), \sigma, (s', q')) \in E$ iff $(q, \sigma, q') \in \delta$ and $\tau(s, \sigma \cap \mathcal{I}) = (s', \sigma \cap \mathcal{O})$. That is, G is the product of $\mathcal{A}$ and $\mathcal{T}$.

A run graph of a universal Büchi (resp. co-Büchi) automaton is accepting if every infinite path $(s_0, q_0), (s_1, q_1), \ldots$ contains infinitely (resp. finitely) many occurrences of states q_i in F. A transition system $\mathcal{T}$ is accepted by a universal automaton $\mathcal{A}$ if the unique run graph of $\mathcal{A}$ on $\mathcal{T}$ is accepting. We denote with $\mathcal{L}(\mathcal{A})$ the set of transition systems accepted by $\mathcal{A}$.

The bounded synthesis approach is based on the fact that for every LTL formula φ one can construct a universal co-Büchi automaton $\mathcal{A}_\varphi$ with at most $2^{O(|\varphi|)}$ states such that $\mathcal{T} \in \mathcal{L}(\mathcal{A}_\varphi)$ iff $\mathcal{T} \models \varphi$ for every transition system $\mathcal{T}$ [13].

An *annotation* of a transition system $\mathcal{T} = (S, s_0, \tau)$ with respect to a universal co-Büchi automaton $\mathcal{A} = (Q, q_0, \delta, F)$ is a function $\lambda : S \times Q \to \mathbb{N} \cup \{\bot\}$ that maps nodes of the run graph of $\mathcal{A}$ on $\mathcal{T}$ to the set $\mathbb{N} \cup \{\bot\}$. Intuitively, such an annotation is valid if every node (s, q) that is reachable from the node (s_0, q_0) is annotated with a natural number, which is an upper bound on the number of rejecting states visited on any path from (s_0, q_0) to (s, q).

Formally, an annotation $\lambda : S \times Q \to \mathbb{N} \cup \{\bot\}$ is *valid* if

- $\lambda(s_0, q_0) \neq \bot$, i.e., the pair of initial states is labelled with a number, and
- whenever $\lambda(s, q) \neq \bot$, then for every edge $((s, q), \sigma, (s', q'))$ in the run graph of $\mathcal{A}$ on $\mathcal{T}$ we have that (s', q') is annotated with a number (i.e., $\lambda(s', q') \neq \bot$), such that $\lambda(s', q') \geq \lambda(s, q)$, and if $q' \in F$, then $\lambda(s', q') > \lambda(s, q)$.

Valid annotations of finite-state systems correspond to accepting run graphs. An annotation λ is c-bounded if $\lambda(s, q) \in \{0, \ldots, c\} \cup \{\bot\}$ for all $s \in S$ and $q \in Q$.

The synthesis method proposed in [17, 10] employs the following result in order to reduce the bounded synthesis problem to checking the satisfiability of propositional formulas: A transition system $\mathcal{T}$ is accepted by a universal co-Büchi automaton $\mathcal{A} = (Q, q_0, \delta, F)$ iff there exists a $(|\mathcal{T}| \cdot |F|)$-bounded valid annotation for $\mathcal{T}$ and $\mathcal{A}$. One can estimate a bound on the size of the transition system, which allows to reduce the synthesis problem to its bounded version. Namely, if there exists a transition system that satisfies an LTL formula φ, then there exists a transition system satisfying φ with at most $\left(2^{(|\mathsf{subf}(\varphi)| + \log|\varphi|)}\right)!^2$ states.

Let $\mathcal{A} = (Q, q_0, \delta, F)$ be a universal co-Büchi automaton for the LTL formula φ. Given a bound b on the size of the sought transition system $\mathcal{T}$, the bounded synthesis problem can be encoded as a satisfiability problem with the following sets of propositional variables and constraints.

Variables: The variables represent the sought transition system $\mathcal{T}$, and the sought valid annotation λ of the run graph of $\mathcal{A}$ on $\mathcal{T}$. A transition system with b states $S = \{1, \ldots, b\}$ is represented by Boolean variables $\tau_{s,\sigma_I,s'}$ and o_{s,σ_I} for every $s, s' \in S$, $\sigma_I \in 2^{\mathcal{I}}$, and output proposition $o \in \mathcal{O}$. The variable $\tau_{s,\sigma_I,s'}$ encodes the existence of transition from s to s' on input σ_I, and the variable o_{s,σ_I} encodes o being true in the output from state s on input σ_I.

The annotation λ is represented by the following variables. For each $s \in S$ and $q \in Q$, there is a Boolean variable $\lambda^{\mathbb{B}}_{s,q}$ and a vector $\lambda^{\mathbb{N}}_{s,q}$ of $\log(b \cdot |F|)$ Boolean variables: the variable $\lambda^{\mathbb{B}}_{s,q}$ encodes the reachability of (s,q) from the initial node (s_0,q_0) in the corresponding run graph, and the vector of variables $\lambda^{\mathbb{N}}_{s,q}$ represents the bound for the node (s,q). The constraints are as follows.

Constraints for input-enabled $\mathcal{T}$: $C_\mathcal{T} \overset{\text{def}}{=} \bigwedge_{s \in S} \bigwedge_{\sigma_I \in 2^{\mathcal{I}}} \bigvee_{s' \in S} \tau_{s,\sigma_I,s'}$.

Constraints for valid annotation:

$$C_\lambda \overset{\text{def}}{=} \lambda^{\mathbb{B}}_{s_0,q_0} \wedge$$
$$\bigwedge_{q,q' \in Q} \bigwedge_{s,s' \in S} \bigwedge_{\sigma_I \in 2^{\mathcal{I}}} \Big(\big(\lambda^{\mathbb{B}}_{s,q} \wedge \delta_{s,q,\sigma_I,q'} \wedge \tau_{s,\sigma_I,s'}\big) \rightarrow \mathsf{succ}_\lambda(s,q,s',q') \Big),$$

where $\delta_{s,q,\sigma_I,q'}$ is a formula over the variables o_{s,σ_I} that characterizes the transitions in $\mathcal{A}$ between q and q' on labels consistent with σ_I, and $\mathsf{succ}_\lambda(s,q,s',q')$ is a formula over the annotation variables such that $\mathsf{succ}_\lambda(s,q,s',q') \overset{\text{def}}{=} (\lambda^{\mathbb{B}}_{s',q'} \wedge (\lambda^{\mathbb{N}}_{s',q'} > \lambda^{\mathbb{N}}_{s,q}))$ if $q' \in F$, and $\mathsf{succ}_\lambda(s,q,s',q') \overset{\text{def}}{=} (\lambda^{\mathbb{B}}_{s',q'} \wedge (\lambda^{\mathbb{N}}_{s',q'} \geq \lambda^{\mathbb{N}}_{s,q}))$ if $q' \notin F$.

3.2 Maximum Satisfiability (MaxSAT)

While the bounded synthesis problem can be encoded into SAT, for the synthesis of a transition system that satisfies a set of soft specifications "as much as possible", we need to solve an optimization problem. To this end, we reduce the bounded maximum realizability problem to a *partial weighted MaxSAT problem.*

MaxSAT is a Boolean optimization problem. A MaxSAT instance is a conjunction of clauses, each of which is a disjunction of literals, where a literal is a Boolean variable or its negation. The objective in MaxSAT is to compute a variable assignment that maximizes the number of satisfied clauses. In *weighted MaxSAT*, each clause is associated with a positive numerical weight and the objective is now to maximize the sum of the weights of the satisfied clauses. In *partial weighted MaxSAT*, there are two types of clauses, namely *hard* and *soft* clauses, where only the soft clauses have weights. A solution to a partial weighted MaxSAT formula is a variable assignment satisfying all the hard clauses. An optimal solution additionally maximizes the sum of the weights of the soft clauses.

In the encoding in the next section we use hard clauses for the hard specification, and soft clauses to capture the soft specifications in the maximum realizability problem. The weights for the soft clauses will encode the lexicographic ordering on values of conjunctions of soft specifications.

4 From Maximum Realizability to MaxSAT

We now describe the proposed MaxSAT-based approach to maximum realizability. First, we establish an upper bound on the minimal size of an implementation that satisfies a given LTL specification φ and maximizes the satisfaction of a conjunction of the soft specifications $\square\varphi_1, \ldots, \square\varphi_n$ according to the value function defined in Section 2.2. This bound can be used to reduce the maximum realizability problem to its bounded version, which we encode as a MaxSAT problem.

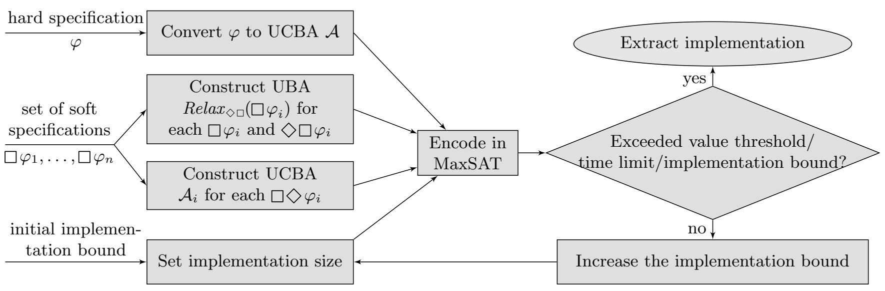

hard specification
φ
Convert φ to UCBA $\mathcal{A}$
set of soft specifications
$\Box\varphi_1, \ldots, \Box\varphi_n$
Construct UBA $Relax_{\Diamond\Box}(\Box\varphi_i)$ for each $\Box\varphi_i$ and $\Diamond\Box\varphi_i$
Construct UCBA $\mathcal{A}_i$ for each $\Box\Diamond\varphi_i$
initial implementation bound
Set implementation size
Encode in MaxSAT
Extract implementation
yes
Exceeded value threshold/ time limit/implementation bound?
no
Increase the implementation bound

Fig. 1: Schematic overview of the maximum realizability procedure.

4.1 Bounded Maximum Realizability

To establish an upper bound on the minimal (in terms of size) optimal implementation, we make use of an important property of the function *val* defined in Section 2.2. Namely, the property that for each of the possible values of $\Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n$ there is a corresponding LTL formula that encodes this value in the classical LTL semantics, as we formally state in the next lemma.

Lemma 1. *For every transition system $\mathcal{T}$ and soft safety specifications $\Box\varphi_1, \ldots, \Box\varphi_n$, if $val(\mathcal{T}, \Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n) = v$, then there exists an LTL formula ψ_v where (1) $\psi_v = \varphi_1' \wedge \ldots \wedge \varphi_n'$, where $\varphi_i' \in \{\Box\varphi_i, \Diamond\Box\varphi_i, \Box\Diamond\varphi_i, true\}$ for $i = 1, \ldots, n$, (2) $\mathcal{T} \models \psi_v$, and for every $\mathcal{T}'$, if $\mathcal{T}' \models \psi_v$, then $val(\mathcal{T}', \Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n) \geq v$.*

The following theorem is a consequence of Lemma 1.

Theorem 1. *Given an LTL specification φ and soft safety specifications $\Box\varphi_1, \ldots, \Box\varphi_n$, if there exists a transition system $\mathcal{T} \models \varphi$, then there exists $\mathcal{T}^*$ such that (1) $val(\mathcal{T}^*, \Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n) \geq val(\mathcal{T}, \Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n)$ for all $\mathcal{T}$ with $\mathcal{T} \models \varphi$, (2) $\mathcal{T}^* \models \varphi$ and $|\mathcal{T}^*| \leq \big((2^{(b+\log b)})!\big)^2$, where $b = \max\{|\mathsf{subf}(\varphi \wedge \varphi_1' \wedge \ldots \wedge \varphi_n')| \mid \forall i : \ \varphi_i' \in \{\Box\varphi_i, \Diamond\Box\varphi_i, \Box\Diamond\varphi_i\}\}$.*

Lemma 1 immediately provides a naive synthesis procedure, which searches for an optimal implementation by enumerating possible ψ_v formulas and solving the corresponding realizability questions. The total number of these formulas is 4^n, where n is the number of soft specifications. The approach that we propose avoids this rapid growth, by reducing the optimization problem to a single MaxSAT instance, making use of the power of state-of-the-art MaxSAT solvers.

Figure 1 gives an overview of our maximum realizability procedure and the automata constructions it involves. As in the bounded synthesis approach, we construct a universal co-Büchi automaton $\mathcal{A}$ for the hard specification φ. For each soft specification $\Box\varphi_i$ we construct a pair of automata corresponding to the relaxations of $\Box\varphi_i$, as shown in Figure 1. The relaxation $\Box\Diamond\varphi_i$ is treated as in bounded synthesis. For $\Box\varphi_i$ and $\Diamond\Box\varphi_i$ we construct a single universal Büchi automaton and define a corresponding annotation function as described next.

4.2 Automata and Annotations for Soft Safety Specifications

We present here the reduction to MaxSAT for the case when each soft specification is of the form $\square\psi$ where ψ is a syntactically safe LTL formula. In this case, we construct a single automaton for both $\square\psi$ and its relaxation $\diamondsuit\square\psi$, and encode the existence of a single annotation function in the MaxSAT problem. The size of this automaton is at most exponential in the length of $\square\psi$.

In the general case, we can treat $\square\psi$ and $\diamondsuit\square\psi$ separately, in the same way that we treat the relaxation $\square\diamondsuit\psi$ of $\square\psi$ in the presented encoding. That would require in total three instead of two annotation functions per soft specification.

We now describe the construction of a universal Büchi automaton $\mathcal{B}_{\square\psi}$ for the safety specification $\square\psi$ and show how we can modify it to obtain an automaton $\mathit{Relax}_{\diamond\square}(\square\psi)$ that incorporates the relaxation of $\square\psi$ to $\diamondsuit\square\psi$.

We first construct a universal Büchi automaton $\mathcal{B}_{\square\psi} = (Q_{\square\psi}, q_0^{\square\psi}, \delta_{\square\psi}, F_{\square\psi})$ for $\square\psi$ such that $\mathcal{L}(\mathcal{B}_{\square\psi}) = \{\mathcal{T} \mid \mathcal{T} \models \square\psi\}$ and $\mathcal{B}_{\square\psi}$ has a unique non-accepting sink state. That is, there exists a unique state $\mathsf{rej}_\psi \in Q_{\square\psi}$ such that $F_{\square\psi} = Q_{\square\psi} \setminus \{\mathsf{rej}_\psi\}$, and $\{q \in Q_{\square\psi} \mid (\mathsf{rej}_\psi, \sigma, q) \in \delta_{\square\psi}\} = \{\mathsf{rej}_\psi\}$ for all $\sigma \in \Sigma$.

From $\mathcal{B}_{\square\psi}$, we obtain a universal Büchi automaton $\mathit{Relax}_{\diamond\square}(\square\psi)$ constructed by redirecting all the transitions leading to rej_ψ to the initial state $q_0^{\square\psi}$. Formally, $\mathit{Relax}_{\diamond\square}(\square\psi) = (Q, q_0, \delta, F)$, where $Q = Q_{\square\psi} \setminus \{\mathsf{rej}_\psi\}$, $q_0 = q_0^{\square\psi}$, $F = F_{\square\psi}$ and $\delta = \big(\delta_{\square\psi} \setminus \{(q,\sigma,q') \in \delta_{\square\psi} \mid q' = \mathsf{rej}_\psi\}\big) \cup \{(q,\sigma,q_0) \mid (q,\sigma,\mathsf{rej}_\psi) \in \delta_{\square\psi}\}$. Let $\mathit{Rej}(\mathit{Relax}_{\diamond\square}(\square\psi)) = \{(q,\sigma,q_0) \in \delta \mid (q,\sigma,\mathsf{rej}_\psi) \in \delta_\psi\}$ be the set of transitions in $\mathit{Relax}_{\diamond\square}(\square\psi)$ that correspond to transitions in $\mathcal{B}_{\square\psi}$ leading to rej_ψ.

The automaton $\mathit{Relax}_{\diamond\square}(\square\psi)$ has the property that its run graph on a transition system $\mathcal{T}$ does not contain a reachable edge corresponding to a transition in $\mathit{Rej}(\mathit{Relax}_{\diamond\square}(\square\psi))$ iff $\mathcal{T}$ is accepted by the automaton $\mathcal{B}_{\square\psi}$, (i.e., $\mathcal{T} \models \square\psi$). Otherwise, if the run graph of $\mathit{Relax}_{\diamond\square}(\square\psi)$ on $\mathcal{T}$ contains a reachable edge that belongs to $\mathit{Rej}(\mathit{Relax}_{\diamond\square}(\square\psi))$, then $\mathcal{T} \not\models \square\psi$. However, if each infinite path in the run graph contains only a finite number of occurrences of such edges, then $\mathcal{T} \models \diamondsuit\square\psi$. Based on these observations, we define an annotation function that annotates each node in the run graph with an upper bound on the number of edges in $\mathit{Rej}(\mathit{Relax}_{\diamond\square}(\square\psi))$ visited on any path reaching the node.

A function $\pi : S \times Q \to \mathbb{N} \cup \{\bot\}$ is a $\diamondsuit\square$*–valid* annotation for a transition system $\mathcal{T} = (S, s_0, \tau)$ and the automaton $\mathit{Relax}_{\diamond\square}(\square\psi) = (Q, q_0, \delta, F)$ if

(1) $\pi(s_0, q_0) \neq \bot$, i.e., the pair of initial states is labelled with a number, and

(2) if $\pi(s,q) \neq \bot$, then for every edge $((s,q),\sigma,(s',q'))$ in the run graph of $\mathit{Relax}_{\diamond\square}(\square\psi)$ on $\mathcal{T}$ we have that $\pi(s',q') \neq \bot$, and
   - if $(q,\sigma,q') \in \mathit{Rej}(\mathit{Relax}_{\diamond\square}(\square\psi))$, then $\pi(s',q') > \pi(s,q)$, and
   - if $(q,\sigma,q') \notin \mathit{Rej}(\mathit{Relax}_{\diamond\square}(\square\psi))$, then $\pi(s',q') \geq \pi(s,q)$.

This guarantees that $\mathcal{T} \models \diamondsuit\square\psi$ iff there exists a $\diamondsuit\square$–valid $|\mathcal{T}|$-bounded annotation π for $\mathcal{T}$ and $\mathit{Relax}_{\diamond\square}(\square\psi)$. Moreover, if π is $|\mathcal{T}|$-bounded and $\pi(s_0,q_0) = |\mathcal{T}|$, then $\mathcal{T} \models \square\psi$, as this means that no edge in $\mathit{Rej}(\mathit{Relax}_{\diamond\square}(\square\psi))$ is reached.

4.3 MaxSAT Encoding of Bounded Maximum Realizability

Let $\mathcal{A} = (Q, q_0, \delta, F)$ be a universal co-Büchi automaton for the LTL formula φ.

For each formula $\square\varphi_j$, $j \in \{1, \ldots, n\}$, we consider two universal automata: the universal Büchi automaton $\mathcal{B}_j = \mathit{Relax}_{\diamond\square}(\square\varphi_j) = (Q_j, q_0^j, \delta_j, F_j)$, constructed as described in Section 4.2, and a universal co-Büchi automaton $\mathcal{A}_j = (\widehat{Q}_j, \widehat{q}_0^j, \widehat{\delta}_j, \widehat{F}_j)$ for the formula $\square\diamond\varphi_j$. Given a bound b on the size of the sought transition system, we encode the bounded maximum realizability problem as a MaxSAT problem with the following sets of variables and constraints.

Variables: The MaxSAT formulation includes the variables from the SAT formulation of the bounded synthesis problem, which represent the sought transition system $\mathcal{T}$ and the sought valid annotation of the run graph of $\mathcal{A}$ on $\mathcal{T}$. Additionally, it includes variables for representing the annotations π_j and λ_j for $\mathcal{B}_j$ and $\mathcal{A}_j$ respectively, similarly to λ in the SAT encoding. More precisely, the annotations for π_j and λ_j are represented respectively by variables $\pi_{s,q}^{\mathbb{B},j}$ and $\pi_{s,q}^{\mathbb{N},j}$ where $s \in S$ and $q \in Q_j$, and variables $\lambda_{s,q}^{\mathbb{B},j}$ and $\lambda_{s,q}^{\mathbb{N},j}$ where $s \in S$ and $q \in \widehat{Q}_j$.

The set of constraints includes C_τ and C_λ from the SAT formulation as hard constraints, as well as the following constraints for the new annotations.

Hard constraints for valid annotations: For each $j = 1, \ldots, n$, let

$$C_\pi^j \stackrel{\text{def}}{=} \bigwedge_{q,q' \in Q_j} \bigwedge_{s,s' \in S} \bigwedge_{\sigma_I \in 2^{\mathcal{I}}} \Big(\big(\pi_{s,q}^{\mathbb{B},j} \wedge \delta_{s,q,\sigma_I,q'}^j \wedge \tau_{s,\sigma_I,s'}\big) \rightarrow \mathsf{succ}_\pi^j(s,q,s',q',\sigma_I) \Big),$$

$$C_\lambda^j \stackrel{\text{def}}{=} \bigwedge_{q,q' \in \widehat{Q}_j} \bigwedge_{s,s' \in S} \bigwedge_{\sigma_I \in 2^{\mathcal{I}}} \Big(\big(\lambda_{s,q}^{\mathbb{B},j} \wedge \widehat{\delta}_{s,q,\sigma_I,q'}^j \wedge \tau_{s,\sigma_I,s'}\big) \rightarrow \mathsf{succ}_\lambda^j(s,q,s',q',\sigma_I) \Big),$$

$$\text{where } \mathsf{succ}_\pi^j(s,q,s',q',\sigma_I) \stackrel{\text{def}}{=} \pi_{s',q'}^{\mathbb{B},j} \wedge \big(\mathsf{rej}^j(s,q,q',\sigma_I) \rightarrow \pi_{s',q'}^{\mathbb{N},j} > \pi_{s,q}^{\mathbb{N},j}\big) \wedge \big(\neg\mathsf{rej}^j(s,q,q',\sigma_I) \rightarrow \pi_{s',q'}^{\mathbb{N},j} \geq \pi_{s,q}^{\mathbb{N},j}\big),$$

and $\mathsf{rej}^j(s,q,q',\sigma_I)$ is a formula over o_{s,σ_I} obtained from $\mathit{Rej}(\mathcal{B}_j)$. The formula $\mathsf{succ}_\lambda^j(s,\widehat{q},s',\widehat{q}',\sigma_I)$ is analogous to $\mathsf{succ}_\lambda(s,q,s',q',\sigma_I)$ defined in Section 3.1.

Soft constraints for valid annotations: Let $b \in \mathbb{N}_{>0}$ be the bound on the size of the transition system. For each $j = 1, \ldots, n$ we define

$$\begin{array}{lll} \mathit{Soft}_\square^j & \stackrel{\text{def}}{=} \pi_{s_0,q_0}^{\mathbb{B},j} \wedge (\pi_{s_0,q_0}^{\mathbb{N},j} = b) & \text{with weight } 1, \\ \mathit{Soft}_{\diamond\square}^j & \stackrel{\text{def}}{=} \pi_{s_0,q_0}^{\mathbb{B},j} & \text{with weight } n, \text{ and} \\ \mathit{Soft}_{\square\diamond}^j & \stackrel{\text{def}}{=} \pi_{s_0,q_0}^{\mathbb{B},j} \vee \lambda_{s_0,\widehat{q}_0}^{\mathbb{B},j} & \text{with weight } n^2. \end{array}$$

The definition of the soft constraints guarantees that $\mathcal{T} \models \square\varphi_j$ if and only if there exist corresponding annotations that satisfy all three of the soft constraints for $\square\varphi_j$. Similarly, if $\mathcal{T} \models \diamond\square\varphi_j$, then $\mathit{Soft}_{\diamond\square}^j$ and $\mathit{Soft}_{\square\diamond}^j$ can be satisfied.

The definition of the weights of the soft clauses reflects the ordering of transition systems with respect to their satisfaction of $\square\varphi_1 \wedge \ldots \wedge \square\varphi_n$. This guarantees that a transition system extracted from an optimal satisfying assignment for the MaxSAT problem is optimal with respect to the value of $\square\varphi_1 \wedge \ldots \wedge \square\varphi_n$, as stated in the following theorem that establishes the correctness of the encoding.

Theorem 2. *Let $\mathcal{A}$ be a given co-Büchi automaton for φ, and for each $j \in \{1, \ldots, n\}$, let $\mathcal{B}_j = Relax_{\Diamond\Box}(\Box\varphi_j)$ be the universal automaton for $\Box\varphi_j$ constructed as in Section 4.2, and let $\mathcal{A}_j$ be a universal co-Büchi automaton for $\Box\Diamond\varphi_j$. The constraint system for bound $b \in \mathbb{N}_{>0}$ is satisfiable if and only if there exists an implementation $\mathcal{T}$ with $|\mathcal{T}| \leq b$ such that $\mathcal{T} \models \varphi$. Furthermore, from the optimal satisfying assignment to the variables $\tau_{s,\sigma_I,s'}$ and o_{s,σ_I}, one can extract a transition system $\mathcal{T}^*$ such that for every transition system $\mathcal{T}$ with $|\mathcal{T}| \leq b$ and $\mathcal{T} \models \varphi$ it holds that $val(\mathcal{T}^*, \Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n) \geq val(\mathcal{T}, \Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n)$.*

Figure 2 shows a transition system extracted from an optimal satisfying assignment for Example 2 with bound 3 on the implementation size. The transitions depicted in the figure are defined by the values of the variables $\tau_{s,\sigma_I,s'}$. The outputs of the implementation (omitted from the figure) are defined by the values of o_{s,σ_I}. The output in state s_1 when *r1* is true is *table1* $\wedge \neg$*table2*, and the output in s_2 when *r2* is true is $\neg$*table1* $\wedge$ *table2*. For all other combinations of state and input the output is $\neg$*table1* $\wedge \neg$*table2*.

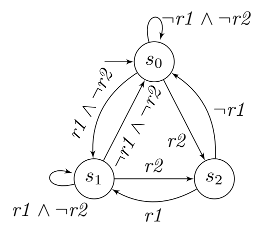


Fig. 2: An optimal implementation for Example 2.

5 Experimental Evaluation

We implemented the proposed approach to maximum realizability[1] in Python 2.7. For the LTL to automata translation we use Spot [8] version 2.2.4. MaxSAT instances are solved by Open-WBO [16] version 2.0. We evaluated our method on instances of two examples. Each experiment was run on machine with a 2.3 GHz Intel Xeon E5-2686 v4 processor and 16 GiB of memory. While the processor is quad-core, only a single core was used. We set a time-out of 1 hour.

Robotic Navigation. We applied our method to the strategy synthesis for a robotic museum guide. The robot has to give a tour of the exhibitions in a specific order, which constitutes the hard specification. Preferably, it also avoids certain locations, such as the staff's office, the library, or the passage when it is occupied. These preferences are encoded in the soft specifications. There is one input variable designating the occupancy of the passage, and eight output variables defining the position of the robot. The formal specifications are given in [7].

Table 1 summarizes the results. With implementation bound of 8, the hard specification is realizable, achieving partial satisfaction of soft specifications. This strategy always selects the passage to transition from Exhibition 1 to Exhibition 2 and hence, always avoids the library. It also temporarily violates the requirement of not entering the staff's office, to acquire access to Exhibition 2. Strategies with higher values exists, but they require larger implementation size. However, for implementation bound 10 the solver reaches a time-out.

[1] The code is available at https://github.com/MahsaGhasemi/max-realizability

Table 1: Results of applying the method to the robotic navigation example, with different bounds on implementation size $|\mathcal{T}|$. We report the number of variables and clauses in the encoding, the satisfiability of hard constraints, the value (and bound) of the MaxSAT objective function, the running times of Spot and Open-WBO, and the time of the solver plus the time for generating the encoding.

	Encoding		Solution		Time (s)		
$\lvert\mathcal{T}\rvert$	# vars	# clauses	sat.	$\Sigma weights$	Spot	Open-WBO	enc.+solve
2	4051	25366	UNSAT	0 (39)	0.93	0.011	0.12
4	19965	125224	UNSAT	0 (39)	0.93	0.079	0.57
6	45897	289798	UNSAT	0 (39)	0.93	1.75	2.9
8	95617	596430	SAT	31 (39)	0.93	956	959
10	152949	954532	SAT	- (39)	0.93	time-out	time-out

Power Distribution Network. We consider the problem of dynamic reconfiguration of power distribution networks. A power network consists of a set P of power supplies (generators) and a set L of loads (consumers). The network is a bipartite graph with edges between supplies and loads, where each supply is connected to multiple loads and each load is connected to multiple supplies. Each power supply has an associated capacity, which determines how many loads it can power at a given time. It is possible that not all loads can be powered all the time. Some loads are critical and must be powered continuously, while others are not and should be powered when possible. Some loads can be initializing, meaning they must be powered only initially for several steps. Power supplies can become faulty during operation, which necessitates dynamic network reconfiguration.

We apply our method to the problem of synthesizing a relay-switching strategy from LTL specifications. The input propositions $\mathcal{I}$ determine which, if any, of the supplies are faulty at each step. We are given an upper bound on the number of supplies that can be simultaneously faulty. The set $\mathcal{O}$ of output propositions contains one proposition $s_{l \to p}$ for each load $l \in L$ and each supply $p \in P$ that are connected. The meaning of $s_{l \to p}$ is that l is powered by p.

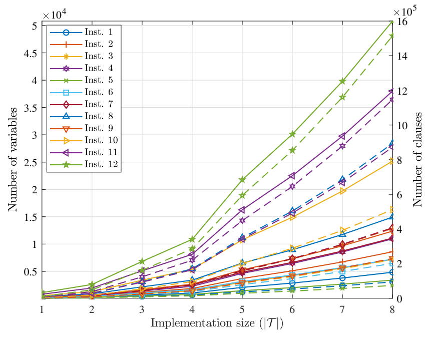
The hard specification asserts that the critical loads must always be powered, the initializing loads should be powered initially, a load is powered by at most one supply, the capacity of supplies is not exceeded, and when a supply is faulty it is not in use. The soft specifications state that non-critical loads are always powered, and that a powered load should remain powered unless its supply fails.

The specifications are given in [7]. Table 2 describes the instances to which we applied our synthesis method. Power supplies have the same capacity E^+ (number of loads they can power) and at most one can be faulty. We consider three categories of instances, depending on the network connectivity (full or sparse), and whether we restrict frequent switching of supplies. In Figure 3, we show the results for the instances defined in Table 2 (detailed results in [7]). In the first set of instances, the specifications have large number of variables (due to full connectivity), and the bottleneck is the translation to automata. In the

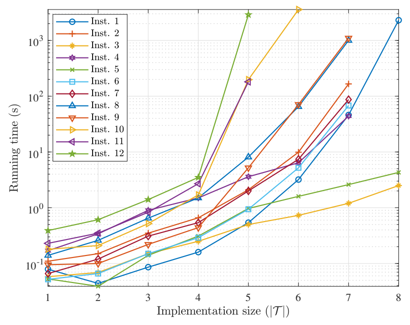
Table 2: Power distribution network instances. An instance is determined by the number supplies $\lvert P\rvert$, the number of loads $\lvert L\rvert$, the capacity of supplies E^+, the number of critical, non-critical and initializing loads. We also show the number of input $\lvert\mathcal{I}\rvert$ and output $\lvert\mathcal{O}\rvert$ propositions and the number of soft specifications.

		Network			Load characterization			Specifications		
	Instance #	$\lvert P\rvert$	$\lvert L\rvert$	E^+	crit.	non-crit.	init.	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{O}\rvert$	# Soft spec.
fully connected, switching allowed	1	3	3	1	1	2	0	2	9	2
	2	3	6	2	2	4	0	2	18	4
	3	3	3	1	0	2	1	2	9	2
	4	3	6	2	1	4	1	2	18	4
sparse, switching allowed	5	4	2	1	1	1	0	3	4	1
	6	4	4	1	1	3	0	3	8	3
	7	4	6	1	1	5	0	3	12	5
	8	4	8	1	1	7	0	3	16	7
sparse, switching restricted	9	4	2	1	1	1	0	3	4	5
	10	4	4	1	1	3	0	3	8	11
	11	4	6	1	1	5	0	3	12	17
	12	4	8	1	1	7	0	3	16	23

third set of instances, the limiting factor is the number of soft specifications, leading to large weights and number of variables in the MaxSAT formulation. We observe that the number of soft specifications is an important factor affecting the scalability of the proposed method. Instance 12, on which the MaxSAT solver reaches time-out for implementation size bound 6, contains 23 soft specifications.



(a) Encoding size



(b) Running time

Fig. 3: Results of applying the method to the instances in Table 2, with different bounds on implementation size $\lvert\mathcal{T}\rvert$. (a) shows the size of the MaxSAT encoding as the number of variables (solid lines) and the number of clauses (dashed lines). (b) shows the running time of the MaxSAT solver plus the time for the encoding.

References

1. Shaull Almagor, Udi Boker, and Orna Kupferman. Formally reasoning about quality. *J. ACM*, 63(3):24:1–24:56, 2016.
2. Rajeev Alur, Aditya Kanade, and Gera Weiss. Ranking automata and games for prioritized requirements. In *Proc. CAV' 08*, volume 5123 of *LNCS*, 2008.
3. Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* 2008.
4. Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In *Proc. CAV'09*, volume 5643 of *LNCS*, pages 140–156, 2009.
5. Alessandro Cimatti, Marco Roveri, Viktor Schuppan, and Andrei Tchaltsev. Diagnostic information for realizability. In *Proc. VMCAI'08*, LNCS, 2008.
6. Alessandro Cimatti, Marco Roveri, Viktor Schuppan, and Stefano Tonetta. Boolean abstraction for temporal logic satisfiability. In *Proc. CAV'07*, volume 4590 of *LNCS*, pages 532–546, 2007.
7. Rayna Dimitrova, Mahsa Ghasemi, and Ufuk Topcu. Maximum realizability for linear temporal logic specifications. *CoRR*, abs/1804.00415, 2018.
8. Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 - A framework for LTL and ω -automata manipulation. In *Proc. ATVA'16*, volume 9938 of *LNCS*, 2016.
9. Rüdiger Ehlers and Vasumathi Raman. Low-effort specification debugging and analysis. In *Proc. SYNT'14*, volume 157 of *EPTCS*, pages 117–133, 2014.
10. Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *STTT*, 15(5-6), 2013.
11. Farah Juma, Eric I. Hsu, and Sheila A. McIlraith. Preference-based planning via maxsat. In *Advances in Artificial Intelligence - 25th Canadian Conference on Artificial Intelligence, Canadian AI 2012, Toronto, ON, Canada, May 28-30, 2012. Proceedings*, volume 7310 of *LNCS*, pages 109–120, 2012.
12. Kangjin Kim, Georgios E. Fainekos, and Sriram Sankaranarayanan. On the minimal revision problem of specification automata. *I. J. Robotics Res.*, 34(12), 2015.
13. Orna Kupferman and Moshe Y. Vardi. Safraless decision procedures. In *Proc. FOCS'05*, pages 531–542, 2005.
14. Morteza Lahijanian, Shaull Almagor, Dror Fried, Lydia E. Kavraki, and Moshe Y. Vardi. This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction. In *Proc. AAAI'15*, 2015.
15. Morteza Lahijanian and Marta Z. Kwiatkowska. Specification revision for markov decision processes with optimal trade-off. In *Proc. CDC'16*, pages 7411–7418, 2016.
16. Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-wbo: A modular maxsat solver,. In *Proc. SAT'14*, volume 8561 of *LNCS*, pages 438–445, 2014.
17. Sven Schewe and Bernd Finkbeiner. Bounded synthesis. In *Proc. ATVA'07*, volume 4762 of *LNCS*, pages 474–488, 2007.
18. Paulo Tabuada and Daniel Neider. Robust linear temporal logic. In *Proc. CSL'16*, volume 62 of *LIPIcs*, pages 10:1–10:21, 2016.
19. Takashi Tomita, Atsushi Ueno, Masaya Shimakawa, Shigeki Hagihara, and Naoki Yonezaki. Safraless LTL synthesis considering maximal realizability. *Acta Informatica*, 54(7), 2017.
20. Jana Tumova, Gavin C. Hall, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. Least-violating control strategy synthesis with safety rules. In *Proc. HSCC'13*, 2013.