Linear Constraints and Guarded Predicates as a Modeling Language for Discrete Time Hybrid Systems

Federico Mari, Igor Melatti, Ivano Salvo, and Enrico Tronci
Department of Computer Science – Sapienza University of Rome
Via Salaria 113, 00198 Rome, Italy
Email: {mari,melatti,salvo,tronci}@di.uniroma1.it

***Abstract*—Model based design is particularly appealing in *software based control systems* (e.g., *embedded software*) design, since in such a case system level specifications are much easier to define than the control software behavior itself. In turn, model based design of embedded systems requires modeling both continuous subsystems (typically, the plant) as well as discrete subsystems (the controller). This is typically done using *hybrid systems*. Mixed Integer Linear Programming (MILP) based abstraction techniques have been successfully applied to automatically synthesize correct-by-construction control software for *discrete time linear hybrid systems*, where plant dynamics is modeled as a linear predicate over state, input, and next state variables. Unfortunately, MILP solvers require such linear predicates to be conjunctions of linear constraints, which is not a natural way of modeling hybrid systems. In this paper we show that, under the hypothesis that each variable ranges over a bounded interval, any linear predicate built upon conjunction and disjunction of linear constraints can be automatically translated into an equivalent conjunctive predicate. Since variable bounds play a key role in this translation, our algorithm includes a procedure to compute all implicit variable bounds of the given linear predicate. Furthermore, we show that a particular form of linear predicates, namely *guarded predicates*, are a natural and powerful language to succinctly model discrete time linear hybrid systems dynamics. Finally, we experimentally show the feasibility of our approach on an important and challenging case study taken from the literature, namely the multi-input Buck DC-DC Converter. As an example, the guarded predicate that models (with 57 constraints) a 6-inputs Buck DC-DC Converter is translated in a conjunctive predicate (with 102 linear constraints) in about 40 minutes.**

***Keywords*-Model-based software design; Linear predicates; Hybrid systems**

I. Introduction

Many embedded systems are *Software Based Control Systems* (SBCS). An SBCS consists of two main subsystems: the *controller* and the *plant*. Typically, the plant is a physical system consisting, for example, of mechanical or electrical devices, while the controller consists of control software running on a microcontroller. In an endless loop, each T seconds (*sampling time*), the controller, after an *Analog-to-Digital* (AD) conversion (*quantization*), reads sensor outputs from the plant and, possibly after a *Digital-to-Analog* (DA) conversion, sends commands to plant actuators. The controller selects commands in order to guarantee that the *closed loop system* (that is, the system consisting of both plant and controller) meets given safety and liveness specifications (*system level specifications*).

This paper is an extension of the ICSEA 2012 paper [1], and contributes to *model based design* of embedded software [2]. Software generation from models and formal specifications forms the core of model based design of embedded software. This approach is particularly interesting for SBCSs, since in such a case system level specifications are much easier to define than the control software behavior itself. In this setting, correct-by-construction software generation from (as well as formal verification of) system level specifications for SBCSs requires modeling both the continuous subsystem (the plant) and discrete systems (the controller). This is typically done using *hybrid systems* (e.g., see [3][4]). Here we focus on *Discrete Time Linear Hybrid Systems* (DTLHS) [5][6] which provide an expressive model for closed loop systems. A DTLHS is a discrete time hybrid system whose dynamics is defined as a linear predicate (i.e., a boolean combination of linear constraints) on its continuous as well as discrete (modes) variables. A large class of hybrid systems, including mixed-mode analog circuits, can be modeled using DTLHSs. System level safety as well as liveness specifications are modeled as set of states defined, in turn, as linear predicates. Moreover, discrete time non-linear hybrid systems may be properly overapproximated with DTLHSs [7] in such a way that a controller for the overapproximated system is guaranteed to work also for the original non-linear system.

In our previous work [8][9], stemming from a constructive sufficient condition for the existence of a quantized sampling controller for an SBCS modelled as a DTLHS (which is an undecidable problem [10]), we presented an algorithm that, given a DTLHS model $\mathcal{H}$ for the plant, a quantization schema (i.e., how many bits we use for AD conversion) and system level specifications, returns the C code [11] of a correct-by-construction quantized feedback *control software* (if any) meeting the given system level specifications. The synthesis algorithm rests on the fact that, because of the quantization process, the plant P is seen by the controller as a *Nondeterministic Finite State Automaton* (NFSA) $\hat{P}$, that

is an abstraction of P. The NFSA $\hat{P}$ is computed by solving *Mixed Integer Linear Programming* (MILP) problems which contains the definition of the DTLHS dynamics as a sub-problem. Since available MILP solvers (e.g., GLPK [12] and CPLEX [13]) require conjunctive predicates (i.e., a conjunction of linear constraints) as input, we have that the DTLHS dynamics must be given as a conjunctive predicate.

While this is not a limitation for DTLHSs with a not too complex dynamics, this may turn in an obstruction for more complex systems. As an example, the dynamics of the 6-inputs Buck DC-DC Converter of Section VII-A is described by the conjunction of 102 linear constraints. However, by allowing disjunction, the same dynamics may be written as a linear predicate consisting of 45 linear constraints. Moreover, constants occurring in such a linear constraint are directly linked to the system physical (known) parameters, while the ones in the conjunctive predicate must be suitably computed. This results in a practical limitation for the effective application of the method in [8][9].

This paper is motivated by circumventing such a limitation, by showing that, under the hypothesis that each variable ranges over a bounded interval, any linear predicate can be automatically translated into an equivalent conjunctive predicate.

Note that it is a reasonable hypothesis to assume variables describing a DTLHS behavior to be bounded. In fact, control software drives the plant towards a goal, while keeping it inside a given desired bounded admissible region. Namely, bounds on present state variables essentially model the *sensing region*, that is the range of values observable by the sensors. Such a region is usually a bounded rectangular region (i.e., the Cartesian product of bounded intervals). Bounds on controllable input variables model the *actuation region*, that is the range of values of commands that the actuators may send to the plant and it is also typically a bounded rectangular region. Other variables may model both non-observable plant state variables and uncontrollable inputs (i.e., disturbances). Therefore, bounds on such variables are usually derived from reasonable assumptions or DTLHS knowledge. On the other hand, *next state* variable bounds are typically not explicitly given. However, they may be derived from all other above mentioned variable bounds (as it will be shown in Example 4 of Section V).

Finally, note that the application of the methods outlined here is not limited to the scenario shown above, but may be applied to nearly all possible usages of MILP solvers in any field.

A. Our Main Contributions

In this paper, we give an algorithm to translate any *linear predicate* into an equivalent (as for solving MILP problems, as it will be shown in Proposition 1 of Section II-B) *conjunctive predicate*, i.e., a conjunction of linear constraints. This allows us to circumvent the limitation mentioned above, i.e., that conjunctive predicates must be used to describe DTLHSs dynamics.

We consider predicates built upon conjunctions and disjunctions of linear constraints (i.e., inequalities of the shape $\sum_{i=1}^{n} a_i x_i \leq b$, Section II). In order to translate them into a conjunctive predicate, we employ a two-stage approach. First, we show that, at the price of introducing fresh boolean variables, a predicate can be translated into an equivalent *guarded predicate* (Section IV-A), i.e., a conjunction of *guarded constraints* of the shape $y \rightarrow (\sum_{i=1}^{n} a_i x_i \leq b)$. Guarded predicates themselves are shown to be a powerful means of modeling DTLHSs dynamics in Section VI. Second, once a guarded predicate has been obtained (or a guarded predicate has been directly provided as the input DTLHS model), we show that it can be in turn translated into a conjunctive predicate (Section IV-B). This latter translation needs, as a further input, the (finite) upper and lower bounds for each variable in the predicate. To this end, in Section V we give an algorithm that computes bounds for a variable x in a given guarded predicate $G(X)$, i.e., either it returns two values $m_x, M_x \in \mathbb{R}$ such that if $G(X)$ holds, then $m_x \leq x \leq M_x$, or it concludes that such values do not exist.

An experimental evaluation of the translation algorithm presented in this paper is in Section VII. As an example, we show that the linear predicate that models a 4-inputs Buck DC-DC Converter with 39 linear constraints is translated into a conjunctive predicate of 82 linear constraints in slightly more than 3 hours.

Note that there are two available inputs for our translation algorithm: i) a linear predicate or ii) a guarded predicate. Namely, if a guarded predicate is provided as input, only the second stage mentioned above is performed. Our experimental evaluation also shows that it is more convenient to use guarded predicates instead of linear predicates when modeling DTLHSs dynamics. As an example, the guarded predicate that models a 6-inputs Buck DC-DC Converter with 57 constraints (including 12 different guards), is translated into a conjunctive predicate of 102 linear constraints in about 40 minutes.

B. Paper Outline

The paper is organized as follows. Section II provides the basic definitions to understand our approach. In Section III, we formally define DTLHSs. In Section IV, our two-steps approach (from linear predicates to guarded predicates and then to conjunctive predicates) is outlined, assuming variables bounds to be known. In Section V, we show how we automatically compute bounds for all variables in a guarded predicate, thus completing the description of our approach. Section VI shows that guarded predicates are a powerful and natural modeling language for DTLHSs. Section VII shows experimental results on a meaningful case study,

namely the multi-input Buck DC-DC Converter. Finally, Sections VIII and IX conclude the paper, by comparing the approach presented here with previous work and by providing concluding remarks and future work.

II. Basic Definitions

An initial segment $\{1, \ldots, n\}$ of $\mathbb{N}$ is denoted by $[n]$. We denote with $X = [x_1, \ldots, x_n]$ a finite sequence of distinct variables, that we may regard, when convenient, as a set. Each variable x ranges on a known (bounded or unbounded) interval $\mathcal{D}_x$ either of the reals (continuous variables) or of the integers (discrete variables). The set $\prod_{x \in X} \mathcal{D}_x$ is denoted by $\mathcal{D}_X$. Boolean variables are discrete variables ranging on the set $\mathbb{B} = \{0, 1\}$. If x is a boolean variable we write $\bar{x}$ for $(1 - x)$. The sequence of continuous (discrete, boolean) variables in X is denoted by X^r (X^d, X^b).

The set of sequences of n boolean values is denoted by $\mathbb{B}^n$. The set $\mathbb{B}^n_k \subseteq \mathbb{B}^n$ denotes sequences that contains exactly k elements equal to 1. Given $a, b \in \mathbb{B}^n$, we say that $a \leq b$ if a is point-wise less or equal to b, i.e., if for all $i \in [n]$ we have that $a_i \leq b_i$. Given a set $B \subseteq \mathbb{B}^n$ and $a \in \mathbb{B}^n$ we write $a \leq B$ if there exists $b \in B$ such that $a \leq b$ and $a \geq B$ if there exists $b \in B$ such that $a \geq b$. We denote with $\mathrm{Ones}(b)$ be the set of indexes such that $b_j = 1$, i.e., $\mathrm{Ones}(b) = \{j \in [n] \mid b_j = 1\}$.

A. Predicates

A *linear expression* $L(X) = \sum_{i=1}^{n} a_i x_i$ is a linear combination of variables in X with rational coefficients. A *constraint* is an expression of the form $L(X) \leq b$, where b is a rational constant. We write $L(X) \geq b$ for $-L(X) \leq -b$, $L(X) = b$ for $(L(X) \leq b) \wedge (-L(X) \leq -b)$, and $a \leq L(X) \leq b$ for $(L(X) \leq b) \wedge (L(X) \geq a)$.

(Linear) predicates are inductively defined as follows. A constraint $C(X)$ is a predicate over X. If $A(X)$ and $B(X)$ are predicates, then $(A(X) \wedge B(X))$ and $(A(X) \vee B(X))$ are predicates over X. Parentheses may be omitted, assuming usual associativity and precedence rules of logical operators. A *conjunctive predicate* is a conjunction of constraints.

A *valuation* over X is a function v that maps each variable $x \in X$ to a value $v(x)$ in $\mathcal{D}_x$. We denote with $X^* \in \mathcal{D}_X$ the sequence of values $v(x_1), \ldots, v(x_n)$. We call valuation also the sequence of values X^*. Given a valuation X^*, the value for variable x is $X^*(x)$. Given a predicate $P(Y, X)$, $P(Y, X^*)$ denotes the predicate obtained by replacing each occurrence of x with $X^*(x)$. A *satisfying assignment* to a predicate $P(X)$ is a valuation X^* such that $P(X^*)$ holds. A predicate is said to be *satisfiable* if there exists at least one satisfying assignment. Abusing notation, we denote with P also the set of satisfying assignments to the predicate P. $P(X)$ and $Q(X)$ are *equivalent*, notation $P \equiv Q$, if they have the same set of satisfying assignments. $P(X)$ and $Q(Z)$ are *equisatisfiable*, notation $P \simeq Q$, if P is satisfiable if and only if Q is satisfiable. Finally, two predicates $P(X)$ and $Q(X, Z)$ are *X-equivalent*, notation $P \equiv_X Q$, if the following holds for all valuations X^*, Z^*:

1) if $P(X^*)$ holds, then $Q(X^*, Z)$ is satisfiable;
2) if $Q(X^*, Z^*)$ holds, then $P(X^*)$ holds.

B. Mixed Integer Linear Programming

A *Mixed Integer Linear Programming* (MILP) problem with *decision variables* X is a tuple $(\max,\ J(X),\ A(X))$ where X is a list of variables, $J(X)$ (*objective function*) is a linear expression over X, and $A(X)$ (*constraints*) is a predicate over X. A *solution* to $(\max, J(X), A(X))$ is a valuation X^* such that $A(X^*)$ and $\forall Z\,(A(Z) \rightarrow (J(Z) \leq J(X^*)))$. $J(X^*)$ is the *optimal value* of the MILP problem. A *feasibility* problem is a MILP problem of the form $(\max, 0, A(X))$. We write also $A(X)$ for $(\max, 0, A(X))$. In algorithm outlines, MILP solver invocations are denoted by function *feasible*$(A(X))$ that returns 1 if $A(X)$ is satisfiable and 0 otherwise, and by function *optimalValue*$(\max,\ J(X),\ A(X))$ that returns either the optimal value of the MILP problem $(\max,\ J(X),\ A(X))$ or ∞ if such MILP problem is unbounded. We write $(\min, J(X), A(X))$ for $(\max, -J(X), A(X))$.

Note that available MILP solvers (e.g., GLPK [12] or CPLEX [13]) require $A(X)$ to be a conjunctive predicate. However, as explained in Section I, MILP problems arising in methods like [8][9] are more easily represented as linear predicates. Thus, we need a translation algorithm from a linear predicate A to a conjunctive predicate A' such that a solution to $(\max, J, A')$ (which may be computed by a MILP solver) is also a solution to $(\max, J, A)$ (which may not be computed by a MILP solver). To this end, Proposition 1 clarifies that X-equivalence between predicates must be sought.

Proposition 1: Let $(\max,\ J(X),\ A(X))$ be a MILP problem, let $B(X, Z)$ be a conjunctive predicate which is X-equivalent to $A(X)$ and let $\tilde{J}(X, Z) = J(X) + \sum_{z \in Z} 0z$. Then for all solutions X^*, Z^* of $(\max,\ \tilde{J}(X, Z),\ B(X, Z))$, X^* is a solution of $(\max,\ J(X),\ A(X))$. Moreover, for all solutions X^* of $(\max,\ J(X),\ A(X))$, there exists Z^* such that X^*, Z^* is a solution of $(\max,\ \tilde{J}(X, Z),\ B(X, Z))$. Finally, *optimalValue*$(\max,\ J(X),\ A(X))$ = *optimalValue*$(\max,\ \tilde{J}(X, Z),\ B(X, Z))$.

Proof: Let X^*, Z^* be a solution of $(\max,\ \tilde{J}(X, Z),\ B(X, Z))$. This entails that $B(X^*, Z^*)$ holds, and that $\forall X^+, Z^+$ such that $B(X^+, Z^+)$ holds, $\tilde{J}(X^*, Z^*) \geq \tilde{J}(X^+, Z^+)$. Suppose by absurd that X^* is not a solution for $(\max,\ J(X),\ A(X))$. Then, either i) $A(X^*)$ does not hold or ii) there exist $\tilde{X}$ such that $A(\tilde{X})$ holds and $J(X^*) < J(\tilde{X})$. Case i) is not possible, since $B(X^*, Z^*)$ holds and $B(X, Z)$ is X-equivalent to $A(X)$. Case ii) is not possible since, by X-equivalence of $B(X, Z)$ and $A(X)$ and by definition of $\tilde{J}(X, Z)$, there would exist $\tilde{Z}$ such that $B(\tilde{X}, \tilde{Z})$ holds and $\tilde{J}(\tilde{X}, \tilde{Z}) = J(\tilde{X}) > J(X^*) = \tilde{J}(X^*, Z^*)$.

With a similar reasoning, it is possible to prove the other implication. Finally, equality of optimal values immediately follows from solutions equivalence and definition of $\tilde{J}$.

■

As a consequence of Proposition 1, the translation algorithm we need must take as input a linear predicate $P(X)$ and return as output an X-equivalent conjunctive predicate $Q(X, Z)$. In the following sections, we will show how we achieve this goal.

III. Discrete Time Linear Hybrid Systems

Discrete Time Linear Hybrid Systems (DTLHS) provide a suitable model for many SBCS (including embedded control systems) since they can effectively model linear algebraic constraints involving both continuous as well as discrete variables. This is shown, e.g., in Example 1, that presents a DTLHS model of a buck DC-DC converter, i.e., a mixed-mode analog circuit that converts the *Direct Current* (DC) input voltage to a desired DC output voltage.

Definition 1: A *Discrete Time Linear Hybrid System* is a tuple $\mathcal{H} = (X, U, Y, N)$ where:

- $X = X^r \cup X^d$ is a finite sequence of real and discrete *present state* variables. X' denotes the sequence of *next state* variables obtained by decorating with $'$ variables in X.
- $U = U^r \cup U^d$ is a finite sequence of *input* variables.
- $Y = Y^r \cup Y^d$ is a finite sequence of *auxiliary* variables. Auxiliary variables typically models *modes* (switching elements) or *uncontrollable inputs* (e.g., disturbances).
- $N(X, U, Y, X')$ is a predicate over $X \cup U \cup Y \cup X'$ defining the *transition relation* (*next state*) of the system.

Example 1: The buck DC-DC converter [15] is a mixed-mode analog circuit (Figure 1) converting the DC input voltage (V_i in Figure 1) to a desired DC output voltage (v_O in Figure 1). Buck DC-DC converters are used off-chip to scale down the typical laptop battery voltage (12-24) to the just few volts needed by the laptop processor (e.g., see [15]) as well as on-chip to support *Dynamic Voltage and Frequency Scaling* (DVFS) in multi-core processors. (e.g., see [14]). Because of its widespread use, control schemes for buck DC-DC converters have been widely studied (e.g., see [14][15][16]). The typical software based approach (e.g., see [15]) is to control the switch u in Figure 1 (typically implemented with a MOSFET, i.e., a *Metal-Oxide-Semiconductor Field-Effect Transistor*) with a microcontroller.

The circuit in Figure 1 can be modeled as a DTLHS $\mathcal{H} = (X, U, Y, N)$ as follows. The circuit state variables are i_L and v_C. However we can also use the pair i_L, v_O as state variables in $\mathcal{H}$ model since there is a linear relationship between i_L, v_C and v_O, namely: $v_O = \frac{r_C R}{r_C+R} i_L + \frac{R}{r_C+R} v_C$. Such considerations lead us to the following DTLHS model

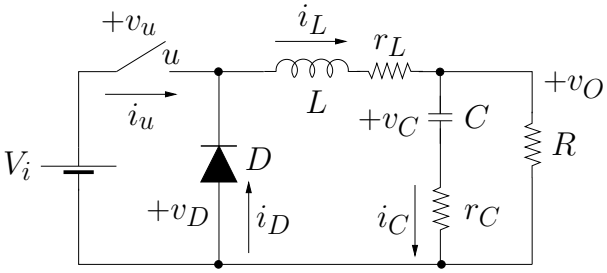

Figure 1. Buck DC-DC converter

$\mathcal{H}$: $X = X^r = [i_L, v_O]$, $U = U^d = [u]$, $Y = Y^r \cup Y^d$ where $Y^r = [i_u, v_u, i_D, v_D]$ and $Y^d = [q]$. Note how $\mathcal{H}$ auxiliary variables Y stem from the constitutive equations of the switching elements (i.e., the switch u and the diode D in Figure 1). From a simple circuit analysis (e.g., see [17]) we have the following equations:

$$\dot{i_L} = a_{1,1} i_L + a_{1,2} v_O + a_{1,3} v_D \quad (1)$$

$$\dot{v_O} = a_{2,1} i_L + a_{2,2} v_O + a_{2,3} v_D \quad (2)$$

where the coefficients $a_{i,j}$ depend on the circuit parameters R, r_L, r_C, L and C as follows: $a_{1,1} = -\frac{r_L}{L}$, $a_{1,2} = -\frac{1}{L}$, $a_{1,3} = -\frac{1}{L}$, $a_{2,1} = \frac{R}{r_c+R}[-\frac{r_c r_L}{L} + \frac{1}{C}]$, $a_{2,2} = \frac{-1}{r_c+R}[\frac{r_c R}{L} + \frac{1}{C}]$, $a_{2,3} = -\frac{1}{L}\frac{r_c R}{r_c+R}$. Using a discrete time model with sampling time T and writing x' for $x(t+1)$, we have:

$$i'_L = (1 + T a_{1,1}) i_L + T a_{1,2} v_O + T a_{1,3} v_D \quad (3)$$

$$v'_O = T a_{2,1} i_L + (1 + T a_{2,2}) v_O + T a_{2,3} v_D. \quad (4)$$

The algebraic constraints stemming from the constitutive equations of the switching elements are the following:

$$v_D = v_u - V_i \quad (5) \qquad (u = 1) \vee (v_u = R_{\text{off}} i_u) \quad (7)$$

$$i_D = i_L - i_u \quad (6) \qquad (u = 0) \vee (v_u = 0) \quad (8)$$

$$((i_D \geq 0) \wedge (v_D = 0)) \vee ((i_D \leq 0) \wedge (v_D = R_{\text{off}} i_D)) \quad (9)$$

The transition relation N of $\mathcal{H}$ is given by the conjunction of the linear predicates (3)–(9).

IV. From Linear to Conjunctive Predicates

As shown in [8][9], MILP solvers can be used to build a suitable discrete abstraction of a DTLHS. As shown in Sections I and II-B (especially in Proposition 1), in order to do this we need a translation algorithm from linear predicates to X-equivalent conjunctive predicates. In this section, we show how we achieve this goal, by designing a two-steps algorithm. First, in Section IV-A, we introduce *guarded predicates* and we show that each predicate $P(X)$ can be translated into an X-equivalent guarded predicate $Q(X, Z)$ at the price of introducing new auxiliary boolean variables Z. Then, in Section IV-B, we show that, under the hypothesis that each variable ranges over a bounded interval, each guarded predicate can be in turn translated into an equivalent conjunctive predicate.

A. Guarded Predicates

As formalized in Definition 2, a *guarded predicate* is an implication between a boolean variable (*guard*) and a predicate.

Definition 2: Given a predicate $P(X)$ and a fresh boolean variable $z \notin X$, the predicate $z \rightarrow P(X)$ (resp. $\bar{z} \rightarrow P(X)$) denotes the predicate $(z = 0) \vee P(X)$ (resp. $(z = 1) \vee P(X)$). We call z the *guard variable* and both z and $\bar{z}$ *guard literals*. Let $C(X)$ be a constraint. A predicate of the form $z \rightarrow C(X)$ or $\bar{z} \rightarrow C(X)$ is called *guarded constraint*. A predicate of the form $z \rightarrow C(X)$ is called *positive guarded constraint*, whilst a predicate of the form $\bar{z} \rightarrow C(X)$ is called *negative guarded constraint*. A *generalized guarded constraint* is a predicate of the form $z_1 \rightarrow (z_2 \rightarrow \ldots \rightarrow (z_n \rightarrow C(X))\ldots)$ A *guarded predicate* (resp. *generalized* guarded predicate, *positive* guarded predicate) is a conjunction of either constraints or guarded constraints (resp. generalized guarded constraints, positive guarded constraints).

To simplify proofs and notations, without loss of generality, we always assume guard literals to be distinct: a conjunction $z \rightarrow C_1(X) \wedge z \rightarrow C_2(X)$ is X-equivalent to the guarded predicate $z_1 \rightarrow C_1(X) \wedge z_2 \rightarrow C_2(X) \wedge z_1 = z \wedge z_2 = z$, being z_1, z_2 fresh boolean variables. Moreover, in algorithm outlines, conjunctive (resp., guarded) predicates will be sometimes regarded as sets of linear (resp., guarded) constraints.

By applying standard propositional equivalences, we have the following facts.

Fact 2: A predicate of the form $z \rightarrow \bigwedge_{i\in[n]} P_i(X)$ is equivalent to the predicate $\bigwedge_{i\in[n]}(z \rightarrow P_i(X))$.

Fact 3: A generalized guarded constraint $z_1 \rightarrow (z_2 \rightarrow \ldots \rightarrow (z_n \rightarrow C(X))\ldots)$ is X-equivalent to the positive guarded predicate $(z - \sum_{i\in[n]} z_i \geq 1 - n) \wedge (z \rightarrow C(X))$, where z is a fresh boolean variable.

Proof: Let z be a fresh boolean variable. We have:

$$
\begin{aligned}
&z_1 \rightarrow (z_2 \rightarrow \ldots \rightarrow (z_n \rightarrow C(X))\ldots)\\
&\equiv z_1 \wedge z_2 \wedge \ldots \wedge z_n \rightarrow C(X)\\
&\equiv_X ((z_1 \wedge z_2 \wedge \ldots \wedge z_n) \rightarrow z) \wedge (z \rightarrow C(X))\\
&\equiv (\bar{z}_1 \vee \bar{z}_2 \vee \ldots \vee \bar{z}_n \vee z) \wedge (z \rightarrow C(X))\\
&\equiv z + \textstyle\sum_{i\in[n]}(1 - z_i) \geq 1 \wedge (z \rightarrow C(X))\\
&\equiv (z - \textstyle\sum_{i\in[n]} z_i \geq 1 - n) \wedge (z \rightarrow C(X))
\end{aligned}
$$

■

Lemma 4 and its constructive proof allow us to translate any predicate $P(X)$ to an X-equivalent generalized guarded predicate $Q(X, Z)$.

Lemma 4: For all predicates $P(X)$, there exists a predicate $Q(X, Z) = G(X, Z) \wedge D(Z)$ such that:

1) $P(X)$ is X-equivalent to $Q(X, Z)$;
2) $G(X, Z)$ and $D(Z)$ (and hence $Q(X, Z)$) are generalized guarded predicates;
3) each generalized guarded constraint in $G(X, Z)$ is of the form $z_1 \rightarrow z_2 \rightarrow \ldots \rightarrow z_m \rightarrow C(X)$, with $z_i \in Z$ and $z_i \notin X$ for all $i \in [m]$.

Proof: The proof is by induction on the structure of the predicate $P(X)$.

- Case $P(X) = C(X)$ for some linear constraint $C(X)$ (base of the induction). Then the thesis holds with $G(X, Z) = P(X)$, $D(Z) = 1$ and $Z = \varnothing$.
- Case $P(X) = P_1(X) \wedge P_2(X)$ for some predicates $P_1(X), P_2(X)$. By inductive hypothesis there exist $Z_1, Z_2, G_1(X, Z_1), D_1(X, Z_1), G_2(X, Z_2), D_2(X, Z_2)$ such that P_i is X-equivalent to $G_i(X, Z_i) \wedge D_i(Z_i)$ for all $i \in \{1, 2\}$. This entails that $P(X)$ is X-equivalent to $G_1(X, Z_1) \wedge G_2(X, Z_1) \wedge D_1(Z_1) \wedge D_2(Z_1)$. By taking $Z = Z_1 \cup Z_2$, $G(X, Z) = G_1(X, Z_1) \wedge G_2(X, Z_2)$ and $D(X, Z) = D_1(Z_1) \wedge D_2(Z_2)$, and recalling that by inductive hypothesis G_i, D_i are generalized guarded predicates and Z_i is the set of boolean variables that occur positively as guards in G_i (for all $i \in \{1, 2\}$), the thesis follows.
- Case $P(X) = P_1(X) \vee P_2(X)$ for some predicates $P_1(X), P_2(X)$. By inductive hypothesis there exist $Z_1, Z_2, G_1(X, Z_1), D_1(X, Z_1), G_2(X, Z_2), D_2(X, Z_2)$ such that P_i is X-equivalent to $Q_i(X, Z_i) = G_i(X, Z_i) \wedge D_i(Z_i)$ for all $i \in \{1, 2\}$. We can always choose auxiliary boolean variables in such a way that $Z_1 \cap Z_2 = \varnothing$.
  Taken two fresh boolean variables $y_1, y_2 \notin Z_1 \cup Z_2$, the predicate $y_1 \rightarrow Q_1(X, Z_1) \wedge y_2 \rightarrow Q_2(X, Z_2) \wedge y_1 + y_2 \geq 1$ is X-equivalent to $P(X)$. For all $i \in \{1, 2\}$, the predicate $\tilde{Q}_i(X, Z_i, y_i) = y_i \rightarrow Q_i(X, Z_i) = y_i \rightarrow (G_i(X, Z_i) \wedge D_i(Z_i))$ is not a generalized guarded predicate. Since $G_i(X, Z_i)$ and $D_i(Z_i)$ are generalized guarded predicates by inductive hypothesis, we have that $G_i(X, Z_i) = \bigwedge_{j\in[n]} \tilde{G}_{i,j}(X, Z_i)$ and $D_i(Z_i) = \bigwedge_{j\in[p]} \tilde{D}_{i,j}(Z_i)$, being $\tilde{G}_{i,j}(X, Z_i), \tilde{D}_{i,j}(Z_i)$ generalized guarded constraints. This allows us to apply Fact 2 to $\tilde{Q}_i(X, Z_i, y_i)$, obtaining an equivalent predicate $R_i(X, Z_i, y_i) = (\bigwedge_{j\in[n]} y_i \rightarrow \tilde{G}_{i,j}(X, Z_i)) \wedge (\bigwedge_{j\in[p]} y_i \rightarrow \tilde{D}_{i,j}(Z_i))$. The thesis follows by taking $Z = Z_1 \cup Z_2 \cup \{y_1, y_2\}$, $G(X, Z) = \bigwedge_{i\in\{1,2\}}(\bigwedge_{j\in[n]} y_i \rightarrow \tilde{G}_{i,j}(X, Z_i))$, and $D(Z) = \bigwedge_{i\in\{1,2\}}(\bigwedge_{j\in[p]} y_i \rightarrow \tilde{D}_{i,j}(Z_i)) \wedge (y_1 + y_2 \geq 1)$. As for point 3, note that this is the only case in which generalized guarded constraints in $G(X, Z)$ are generated, and that the generation takes place by adding boolean fresh guards only. Being the starting predicate only dependent on X, also point 3 is proved.

■

Lemma 4 and its constructive proof are exploited in Algorithm 1, which takes as input a linear predicate $P(X)$ and outputs the generalized guarded predicates $G(X, Z)$ and $D(Z)$. The function *fresh*( ) returns at each invocation a

(globally) fresh variable. Correctness of Algorithm 1 is given as a corollary of Lemma 4.

Corollary 5: For all predicates $P(X)$, Algorithm 1 returns $\langle G, D, Z\rangle$ such that $G(X,Z)\wedge D(Z)$ is X-equivalent to P and fulfills all properties of Lemma 4.

Proposition 6 and its constructive proof allow us to translate any predicate $P(X)$ in an X-equivalent positive guarded predicate $Q(X,Z)$. Moreover, predicate $Q(X,Z)$ has a special form, i.e., it is the conjunction of two positive guarded predicates $G(X,\tilde{Z})$ and $D(Z)$, with $\tilde{Z}\subseteq Z$. This is accomplished by first translating $P(X)$ in an X-equivalent generalized guarded predicate $\tilde{Q}(X,\tilde{Z})$ by using Lemma 4.

Proposition 6: For all predicates $P(X)$, there exists an X-equivalent positive guarded predicate $Q(X,Z) = G(X,\tilde{Z})\wedge D(Z)$, where G and D are positive guarded predicates and $\tilde{Z}\subseteq Z$.

Proof: Let $\tilde{Q}(X,Z_1) = \tilde{G}(X,Z_1)\wedge\tilde{D}(Z_1)$ be the generalized guarded predicate obtained by applying the proof of Lemma 4 (i.e., by applying Algorithm 1 to $P(X)$). Let $\tilde{G}(X,Z_1) = G_1(X,Z_1)\wedge\bigwedge_{i\in[n]}(z_{i,1}\rightarrow z_{i,2}\rightarrow\ldots z_{i,q_i}\rightarrow C_{i,1}(X)))$, being G_1 a positive guarded predicate. By Fact 3, $\tilde{G}(X,Z_1)$ is $(X\cup Z_1)$-equivalent to the positive guarded predicate $G_1(X,Z_1)\wedge\bigwedge_{i\in[n]}\left(\tilde{z}_i\rightarrow C_{i,1}(X)\wedge\tilde{z}_i-\sum_{j\in[q_i]}z_{i,j}\geq 1-q_i\right)$, where $\tilde{z}_i\notin Z_1$ for all $i\in[n]$. Analogously, by Fact 3 $\tilde{D}(Z_1)$ is Z_1-equivalent to the positive guarded predicate $D_1(Z_1)\wedge\bigwedge_{i\in[p]}\left(\hat{z}_i\rightarrow C_{i,2}(Z_1)\wedge\hat{z}_i-\sum_{j\in[r_i]}z_{i,j}\geq 1-r_i\right)$, where $\hat{z}_i\notin Z_1\cup\{\tilde{z}_j\mid j\in[n]\}$ for all $i\in[p]$. Thus the thesis follows by taking:

- $\tilde{Z} = Z_1\cup\{\tilde{z}_j\mid j\in[n]\}$
- $Z = \tilde{Z}\cup\{\hat{z}_j\mid j\in[p]\}$
- $G(X,\tilde{Z}) = G_1(X,Z_1)\wedge\bigwedge_{i\in[n]}\tilde{z}_i\rightarrow C_{i,1}(X)$
- $D(Z) = D_1(Z_1)\wedge\left(\bigwedge_{i\in[n]}\tilde{z}_i-\sum_{j\in[q_i]}z_{i,j}\geq 1-q_i\right)\wedge\left(\bigwedge_{i\in[p]}\hat{z}_i\rightarrow C_{i,2}(Z_1)\wedge\hat{z}_i-\sum_{j\in[r_i]}z_{i,j}\geq 1-r_i\right)$

■

Proposition 6 and its constructive proof are exploited in Algorithm 2, which takes as input a linear predicate $P(X)$ and outputs the positive guarded predicates $G(X,\tilde{Z})$ and $D(Z)$. To this aim, Algorithm 1 is used as an auxiliary procedure. Correctness of Algorithm 2 is given as a corollary of Proposition 6.

Corollary 7: For all predicates $P(X)$, Algorithm 2 returns $\langle G, D, \tilde{Z}, Z\rangle$ such that $G(X,\tilde{Z})\wedge D(Z)$ is X-equivalent to P and $\tilde{Z}\subseteq Z$.

Example 2: Let $\mathcal{H}$ be DTLHS in Example 1. Given the predicate $N(X,U,Y,X')$ that defines the transition relation of $\mathcal{H}$, function *PtoG* computes the guarded predicate $N^{gp}(X,U,Y,X')$ which is $(X\cup U\cup Y\cup X')$-equivalent to N as follows.

Constraints (3)–(6) remain unchanged, as they are linear constraints in a top-level conjunction. The disjunction (9) is first replaced by the conjunction of linear predicates (10)–(12) as follows.

$$z_1\rightarrow(i_D\geq 0\wedge v_D=0)\quad(10)\qquad z_2\rightarrow(i_D\leq 0\wedge v_D=R_{\text{off}}i_D)\quad(11)$$

$$z_1+z_2\geq 1\quad(12)$$

Then, predicates (10)–(11) are replaced by guarded constraints (17)–(20) below, obtained by moving arrows inside the conjunctions, as shown by Fact 2. Similarly, disjunctions (7) and (8) are replaced by guarded linear constraints (21)–(24) and (26)–(27). Summing up, $N^{gp}(X,U,Y,X')$ is given by the conjunction of the following (guarded) constraints:

$$i'_L = (1+Ta_{1,1})i_L + Ta_{1,2}v_O + Ta_{1,3}v_D \quad(13)$$
$$v'_O = Ta_{2,1}i_L + (1+Ta_{2,2})v_O + Ta_{2,3}v_D. \quad(14)$$

$$v_D = v_u - V_i\quad(15)\qquad i_D = i_L - i_u\quad(16)$$
$$z_1\rightarrow(i_D\geq 0)\quad(17)\qquad z_3\rightarrow(u=1)\quad(21)$$
$$z_1\rightarrow(v_D=0)\quad(18)\qquad z_4\rightarrow(v_u=R_{\text{off}}i_u)\quad(22)$$
$$z_2\rightarrow(i_D\leq 0)\quad(19)\qquad z_5\rightarrow(u=0)\quad(23)$$
$$z_2\rightarrow(v_D=R_{\text{off}}i_D)\quad(20)\qquad z_6\rightarrow(v_u=0)\quad(24)$$
$$z_1+z_2\geq 1\quad(25)\qquad z_3+z_4\geq 1\quad(26)\qquad z_5+z_6\geq 1\quad(27)$$

With respect to the statement of Proposition 6, we have that $Z=\tilde{Z}=\{z_1,z_2,z_3,z_4,z_5,z_6\}$, $G(X,Z')$ is the conjunction of guarded constraints (13)–(24) and $D(Z)$ is the conjunction of constraints (25)–(27).

Algorithm 1 From predicates to generalized guarded predicates (auxiliary for Algorithm 5)

Input: P predicate over X
Output: $\langle G, D, Z\rangle$ where $G(X,Z)\wedge D(Z)$ is a generalized guarded predicate X-equivalent to $P(X)$ (see Lemma 4)
function *PtoGG*(P,X)
1. **if** P is a constraint $C(X)$ **then return** $\langle C(X),\varnothing,\varnothing\rangle$
2. **let** $P = P_1\diamond P_2\quad(\diamond\in\{\wedge,\vee\})$
3. $\langle G_1,D_1,Z_1\rangle\leftarrow$*PtoGG*$(P_1)$
4. $\langle G_2,D_2,Z_2\rangle\leftarrow$*PtoGG*$(P_2)$
5. **if** $P=P_1\wedge P_2$ **then return** $\langle G_1\cup G_2, D_1\cup D_2, Z_1\cup Z_2\rangle$
6. **if** $P=P_1\vee P_2$ **then**
7. $y_1\leftarrow$*fresh*$(\,)$, $y_2\leftarrow$*fresh*$(\,)$, $\tilde{Z}\leftarrow Z_1\cup Z_2\cup\{y_1,y_2\}$
8. $\tilde{D}=\{y_1\rightarrow\gamma|\gamma\in D_1\}\cup\{y_2\rightarrow\gamma|\gamma\in D_2\}\cup\{y_1+y_2\geq 1\}$
9. $\tilde{G}=\{y_1\rightarrow\gamma\mid\gamma\in G_1\}\cup\{y_2\rightarrow\gamma\mid\gamma\in G_2\}$
10. **return** $\langle\tilde{G},\tilde{D},\tilde{Z}\rangle$

B. From Guarded Predicates to Conjunctive Predicates

Guarded predicates may be translated into equivalent conjunctive predicates (our final target for Proposition 1) once bounds for all variables occurring in them are known. In this section, we will show how this translation is performed, assuming bounds to be known. In Section V, we will show

Algorithm 2 From predicates to positive guarded predicates (auxiliary for Algorithm 5)

Input: P predicate over X

Output: $\langle G, D, Z, \tilde{Z}\rangle$ where $G(X,\tilde{Z}) \wedge D(Z)$ is a positive guarded predicate X-equivalent to $P(X)$ (see Proposition 6)

function *PtoG*(P, X)

1. $\langle G, D, Z\rangle \leftarrow$*PtoGG*$(P,\ X)$
2. $\tilde{G} \leftarrow \varnothing$, $\tilde{D} \leftarrow \varnothing$, $\tilde{Z} = Z$
3. **for all** $\gamma \in G \cup D$ **do**
4. **if** $\gamma \equiv z_1 \rightarrow (\ldots \rightarrow (z_n \rightarrow C(W))\ldots)$ **then**
5. $w \leftarrow$*fresh*$(\,)$, $Z \leftarrow Z \cup \{w\}$
6. **if** $W \subseteq X$ **then**
7. $\tilde{G} \leftarrow \tilde{G} \cup \{w \rightarrow C(W)\}$, $\tilde{Z} \leftarrow \tilde{Z} \cup \{w\}$
8. **else**
9. $\tilde{D} \leftarrow \tilde{D} \cup \{w \rightarrow C(W)\}$
10. $\tilde{D} \leftarrow \tilde{D} \cup \{w - \sum_{i\in[n]} z_i \geq 1 - n\}$
11. **else if** *vars*$(\gamma) \subseteq X$ **then**
12. $\tilde{G} \leftarrow \tilde{G} \cup \{\gamma\}$
13. **else**
14. $\tilde{D} \leftarrow \tilde{D} \cup \{\gamma\}$
15. **return** $\langle \tilde{G}, \tilde{D}, Z, \tilde{Z}\rangle$

how bounds for variables may be computed if some bounds are already known.

Definition 3: Let $P(X)$ be a predicate. A variable $x \in X$ is said to be *bounded* in P if there exist $a, b \in \mathcal{D}_x$ such that $P(X)$ implies $a \leq x \leq b$. A predicate P is bounded if all its variables are bounded. We write $\sup(P, x)$ and $\inf(P, x)$ for the minimum and maximum value that the variable x may assume in a satisfying assignment for P. When P is clear from the context, we will write simply $\sup(x)$ and $\inf(x)$.

Given a real number a and a variable $x \in X$ over a bounded interval, we write $\sup(ax)$ for $a\sup(x)$ if $a \geq 0$ and for $a\inf(x)$ if $a < 0$. We write $\inf(ax)$ for $a\inf(x)$ if $a \geq 0$ and for $a\sup(x)$ if $a < 0$. Given a linear expression $L(X) = \sum_{i=1}^{n} a_i x_i$ over a set of bounded variables, we write $\sup(L(X))$ for $\sum_{i=1}^{n} \sup(a_i x_i)$ and $\inf(L(X))$ for $\sum_{i=1}^{n} \inf(a_i x_i)$.

Proposition 8: For each bounded guarded predicate $P(X)$ there exists an equivalent conjunctive predicate $Q(X)$.

Proof: The conjunctive predicate $Q(X)$ is obtained from the guarded predicate $P(X)$ by replacing each guarded constraint $C(X)$ of the shape $z \rightarrow (L(X) \leq b)$ in $P(X)$ with the constraint $\tilde{C}(X) = (\sup(L(X)) - b)z + L(X) \leq \sup(L(X))$. If $z = 0$ we have $C(X) \equiv \tilde{C}(X)$ since $C(X)$ holds trivially and $\tilde{C}(X)$ reduces to $L(X) \leq \sup(L(X))$ that holds by construction. If $z = 1$ both $C(X)$ and $\tilde{C}(X)$ reduce to $L(X) \leq b$. Along the same line of reasoning, if $C(X)$ has the form $\bar{z} \rightarrow (L(X) \leq b)$ we set $\tilde{C}(X)$ to $(b - \sup(L(X)))z + L(X) \leq b$. ■

Together with Proposition 6, Proposition 8 implies that any bounded predicate $P(X)$ can be translated into an X-equivalent conjunctive predicate, at the cost of adding new auxiliary boolean variables, as stated in the following proposition.

Proposition 9: For each bounded predicate $P(X)$, there exists an X-equivalent conjunctive predicate $Q(X, Z)$.

Example 3: Let $\mathcal{H}$ be the DTLHS in Example 2. We set the parameters of $\mathcal{H}$ as follows:

$$\begin{array}{llll} r_L = 0.1\Omega & R = 5\Omega & V_i = 15V & L = 2\cdot 10^{-4}H \\ r_C = 0.1\Omega & R_{\text{off}} = 10^4 & T = 10^{-6}\text{secs} & C = 5\cdot 10^{-5}F \end{array}$$

and we assume variables bounds as follows:

$$\begin{array}{llll} -2{\cdot}10^4 \leq v_u \leq 15 & -4 \leq i_L \leq 4 & -1 \leq v_O \leq 7 & -4 \leq i'_L \leq 96 \\ -2{\cdot}10^4 \leq v_D \leq 0 & -1.1 \leq v'_O \leq 17 & -4 \leq i_u \leq 4 & -2 \leq i_D \leq 4 \end{array}$$

By first decomposing equations of the shape $L(X) = b$ in the conjunctive predicate $L(X) \leq b \wedge -L(X) \leq -b$ and then by applying the transformation given in the proof of Proposition 8, guarded constraints (17)–(24) are replaced by the following linear constraints:

$$2z_1 - i_D \leq 2 \quad (28)$$
$$4\cdot 10^4 z_4 + v_u - 10^4 i_u \leq 4\cdot 10^4 \quad (29)$$
$$6\cdot 10^4 z_4 - v_u + 10^4 i_u \leq 6\cdot 10^4 \quad (30)$$
$$-2.10^4 z_1 - v_D \leq 2\cdot 10^4 \quad (31)$$
$$2.10^4 z_2 + v_D - 10^4 i_D \leq 2.10^4 \quad (32)$$
$$6.10^4 z_2 - v_D + 10^4 i_D \leq 6.10^4 \quad (33)$$
$$2\cdot 10^4 z_6 + v_u \leq 15 \quad (34)$$
$$2\cdot 10^4 z_4 - v_u \leq 2\cdot 10^4 \quad (35)$$
$$v_D \leq 0 \quad (36)$$
$$4z_2 + i_D \leq 4 \quad (37)$$
$$z_5 + u \leq 1 \quad (38)$$
$$-u \leq 0 \quad (39)$$
$$15z_6 + v_u \leq 15 \quad (40)$$
$$z_3 - u \leq 1 \quad (41)$$
$$u \leq 1 \quad (42)$$

V. Computing Variable Bounds

In this section, we present two algorithms that check if a variable x is bounded in a guarded predicate $G(X, Z)$, where Z is the set of guard variables. If this is the case, both algorithms return BND (for *bounded*) and compute $a, b \in \mathcal{D}_X$ such that $G(X, Z)$ implies $a \leq x \leq b$. If this is not the case, then INFEAS (for *infeasible*) is returned if $G(X, Z)$ is unfeasible, and UNB (for *unbounded*) is returned otherwise.

The first algorithm, described in function *exhComputeBounds* of Algorithm 3, works for all guarded predicates $G(X, Z)$, where Z is the set of (boolean) variables occurring as guards in $G(X, Z)$. Namely, it is the naïve algorithm which, for all valuations Z^* of $\mathbb{B}^{|Z|}$ (line 5), builds the conjunctive predicate $Q(X) = G(X, Z^*)$. This implies that, for all guarded constraints $\tilde{G}(X, Z) = z \rightarrow C(X, Z)$ inside G, if $Z^*(z)$ is false then Q will not contain $\tilde{G}$, and will contain only $C(X, Z^*)$ otherwise (line 6). Then, if $G(X, Z^*)$ is feasible, the upper and lower bounds for x under $G(X, Z^*)$ are computed (lines 9 and 10). The overall maximum upper bound and minimum lower bound are finally returned in line 15. Unfortunately, this exhaustive procedure requires to solve $2^{|Z|}$ MILP problems.

The second algorithm, described in function *computeBounds* of Algorithm 4, refines Algorithm 3 in order to

Algorithm 3 Computing variable bounds in a guarded predicate (auxiliary for Algorithm 5 and 6)

Input: Guarded predicate $G(X, Z)$ and variable $x \in X$.
Output: $\langle \mu, \inf, \sup \rangle$ with $\mu \in \{\text{BND}, \text{UNBND}, \text{INFEAS}\}$, $\inf, \sup \in \mathcal{D}_x \cup \bot$.
function *exhComputeBounds*(G, X, Z, x)

1. **let** $G(X, Z) = \bigwedge_{i \in [n]} G_i(X, Z)$, being each $G_i(X, Z)$ either a constraint $C_i(X, Z)$ or a guarded constraint $z_i \to C_i(X, Z)$, $\bar{z}_i \to C_i(X, Z)$
2. **let** $g(i)$, for $i \in [n]$, be the guard of G_i, if any , or 1 otherwise
3. **let** $c(i, Z^*)$, for $i \in [n]$, be true iff $(g(i) = z_i \wedge Z^*(z_i) = 1) \vee (g(i) = \bar{z}_i \wedge Z^*(z_i) = 0) \vee g(i) = 1$
4. $\inf \leftarrow +\infty$, $\sup \leftarrow -\infty$, $f \leftarrow 0$
5. **for** $Z^* \in \mathbb{B}^{|Z|}$ **do**
6. $\quad Q(X, Z^*) \leftarrow \bigwedge_{i \in [n] \wedge c(i, Z^*)} C_i(X, Z^*)$
7. $\quad$ **if** *feasible*$(Q(X, Z^*))$ **then**
8. $\quad\quad f \leftarrow 1$
9. $\quad\quad M \leftarrow$ *optimalValue*$(\max, x, Q(X, Z^*))$
10. $\quad\quad m \leftarrow$ *optimalValue*$(\min, x, Q(X, Z^*))$
11. $\quad\quad$ **if** $M = \infty \vee m = \infty$ **then**
12. $\quad\quad\quad$ **return** $\langle \text{UNBND}, \bot, \bot \rangle$
13. $\quad\quad \sup \leftarrow \max(\sup, M)$, $\inf \leftarrow \min(\inf, m)$
14. **if** f **then**
15. $\quad$ **return** $\langle \text{BND}, \inf, \sup \rangle$
16. **else**
17. $\quad$ **return** $\langle \text{INFEAS}, \bot, \bot \rangle$

save unnecessary MILP invocations. Differently from Algorithm 3, Algorithm 4 works only in the case that the input is a positive guarded predicate of form $G(X, \tilde{Z}) \wedge D(Z)$, where $G(X, \tilde{Z})$ is a positive guarded predicate, $D(Z)$ is a conjunctive predicate, and $\tilde{Z} \subseteq Z$ is the set of (boolean) variables occurring as guards in $G(X, \tilde{Z})$. However, such form may be derived from the one output by Algorithm 2 (see Algorithm 5), thus we still have a method to translate any predicate into a conjunctive predicate.

Algorithm 4 is based on the observation that, if an assignment Z_1^* makes true more guards than an assignment Z_2^*, then the conjunctive predicate $G(X, Z_1^*)$ has more constraints than $G(X, Z_2^*)$. Therefore, if x is bounded in $G(X, Z_2^*)$, then it is also bounded in $G(X, Z_1^*)$, and if $G(X, Z_2^*)$ is unfeasible, then also $G(X, Z_1^*)$ is unfeasible (Proposition 10). In the following, we establish the correctness of function *computeBounds*. We begin with a proposition on fixing boolean values in a positive guarded predicate.

Proposition 10: Let $Z = [z_1, \ldots, z_n]$ and let $G(X, Z) = \bigwedge_{i \in [n]} (z_i \to C_i(X))$ be a conjunction of positive guarded constraints. Then:

1) For any $Z^* \in \mathbb{B}^n$, $G(X, Z^*)$ is equivalent to the conjunctive predicate $\bigwedge_{j \in \text{Ones}(Z^*)} C_j(X)$.
2) If $Z_1^* \leq Z_2^*$, then $G(X, Z_2^*) \Rightarrow G(X, Z_1^*)$.

Proof: Statement 1 easily follows by observing that a guarded constraint $z \to C(X)$ is trivially satisfied if z is assigned to 0 and it is equivalent to $C(X)$ if z is assigned to 1. Statement 2 follows from the observation that $a \leq b$ implies $\text{Ones}(a) \subseteq \text{Ones}(b)$ and hence $G(X, b)$ has more constraints than $G(X, a)$. ■

Algorithm 4 is based on the capability of operating *cuts* on the boolean space. Definition 4 formalizes this concept.

Definition 4: We say that a set $C \subseteq \mathbb{B}^n$ is a *cut* if for all $b \in \mathbb{B}^n$ we have $b \leq C$ or $b \geq C$. Let $D(Z)$ be a predicate over a set boolean variables $Z = Z_1 \cup Z_2$. A cut $C \subseteq \mathbb{B}^{|Z_2|}$ is (D, Z_2)-minimal if

- for all $c \in C$, $D(Z_1, c)$, is satisfiable
- for all $b < C$, $D(Z_1, b)$ is not satisfiable.

Proposition 11 shows how cuts are exploited by Algorithm 4. Namely, to verify that a variable x is bounded in the positive guarded predicate $G(X, \tilde{Z}) \wedge D(Z)$, where $D(Z)$ is a conjunctive predicate, it suffices to check if it is bounded in the conjunctive predicate $G(X, c)$, for all c that are $(D, \tilde{Z})$-minimal cuts.

Algorithm 4 Computing variable bounds in a positive guarded predicate (auxiliary for Algorithms 5 and 6)

Input: Positive guarded predicate $G(X, \tilde{Z})$, conjunctive predicate $D(Z)$ with $\tilde{Z} \subseteq Z$ set of guards in $G(X, \tilde{Z})$, and variable $x \in X$.
Output: $\langle \mu, \inf, \sup \rangle$ with $\mu \in \{\text{BND}, \text{UNBND}, \text{INFEAS}\}$, $\inf, \sup \in \mathcal{D}_x \cup \bot$.
function *computeBounds*$(G, D, X, Z, \tilde{Z}, x)$

1. $C \leftarrow \varnothing$, $r \leftarrow |\tilde{Z}|$, $\inf \leftarrow +\infty$, $\sup \leftarrow -\infty$, $f \leftarrow 0$
2. $r' \leftarrow$ *optimalValue*$(\min, \sum_{i \in [r]} z_i, D(Z))$
3. $r'' \leftarrow$ *optimalValue*$(\max, \sum_{i \in [r]} z_i, D(Z))$
4. **for** $k = r'$ **to** r'' **do**
5. $\quad end \leftarrow 1$
6. $\quad$ **for all** $b \in \mathbb{B}_k^r$ **do**
7. $\quad\quad$ **if** $C \not\leq b$ **then**
8. $\quad\quad\quad end \leftarrow 0$
9. $\quad\quad\quad$ **if** *feasible*$(D(Z, c))$ **then**
10. $\quad\quad\quad\quad C \leftarrow C \cup \{b\}$
11. $\quad\quad\quad\quad$ **if** *feasible*$(G(X, b))$ **then**
12. $\quad\quad\quad\quad\quad f \leftarrow 0$
13. $\quad\quad\quad\quad\quad M \leftarrow$ *optimalValue*$(\max, x, G(X, b))$
14. $\quad\quad\quad\quad\quad m \leftarrow$ *optimalValue*$(\min, x, G(X, b))$
15. $\quad\quad\quad\quad\quad$ **if** $M = \infty$ **or** $m = \infty$ **then**
16. $\quad\quad\quad\quad\quad\quad$ **return** $\langle \text{UNBND}, \bot, \bot \rangle$
17. $\quad\quad\quad\quad\quad \sup \leftarrow \max(\sup, M)$, $\inf \leftarrow \min(\inf, m)$
18. $\quad$ **if** end **then break**
19. **if** f **then**
20. $\quad$ **return** $\langle \text{BND}, \inf, \sup \rangle$
21. **else**
22. $\quad$ **return** $\langle \text{INFEAS}, \bot, \bot \rangle$

Proposition 11: Let $Q(X,Z) = G(X,\tilde{Z}) \wedge D(Z)$, where $G(X,\tilde{Z})$ is a positive guarded predicate, $D(Z)$ is a conjunctive predicate, and $\tilde{Z} \subseteq Z$ is the set of (boolean) variables occurring as guards in $G(X,\tilde{Z})$. Let C be a $(D,\tilde{Z})$-minimal cut and $x \in X$. If, for all $c \in C$, x is bounded in $G(X,c)$, then x is bounded in $Q(X,Z)$.

Proof: Since C is a $(D,\tilde{Z})$-minimal cut, any satisfying assignment (X^*,Z^*) to Q is such that $C \leq \tilde{Z}^*$. As a consequence, there exists $c \in C$ such that $c \leq \tilde{Z}^*$. Proposition 10 (point 2) implies that, for all $Z^* \geq C$, $\max\{x \mid G(X,Z^*)\} \leq \max\{x \mid G(X,c)\}$ and $\min\{x \mid G(X,Z^*)\} \geq \min\{x \mid G(X,c)\}$. Therefore, if x is bounded in $Q(X,c)$ for any $c \in C$, then it is bounded in $Q(X,Z)$. ■

Stemming from Proposition 11, function *computeBounds* (Algorithm 4) checks if a variable x is bounded in a guarded predicate by finding a minimal cut. To limit the search space, in line 2 (resp. line 3) it is computed the minimum (resp. maximum) number of 1 that a satisfying assignment to the predicate $D(Z)$ must have. The loop in lines 4–18 examines possible assignments to guard variables in Z, keeping the invariant $\forall b < C[\neg \mathit{feasible}G(X,b)] \wedge \forall b \geq C[\max\{x \mid G(X,Z)\} \leq \max\{x \mid G(X,b)\} \wedge \min\{x \mid G(X,Z^*)\} \geq \min\{x \mid G(X,b)\}]$. In the loop in lines 6–17, if the assignment c under consideration is greater than an assignment in C, no further investigation are needed (by Proposition 11 x is bounded in $Q(X,c)$). If $D(Z \setminus \tilde{Z}, b)$ is unfeasible, the assignment c is not relevant, because $c \leq C$, for any $(D,\tilde{Z})$-minimal cut C. Otherwise, c is a relevant assignment and it is added to C (line 10). If x is unbounded in $Q(X,c)$ (lines 13 and 16) we can immediately conclude that x is unbounded in $Q(X,Z)$. Otherwise, we update the approximations computed for $\inf(x)$ and $\sup(x)$ (line 17). If for all assignments in $c \in \mathbb{B}^n_k$ we have $c \geq C$ ($\mathbb{B}^n_k$ is a cut) we are done, C is a $(D,\tilde{Z})$-minimal cut, and $\inf$ and $\sup$ computed so far are over-approximation of x bounds in $Q(X,Z)$ (line 18).

The above reasoning gives the proof of correctness for function *computeBounds* of Algorithm 4.

Proposition 12: Let $G(X,\tilde{Z})$ be a positive guarded predicate, $D(Z)$ be a conjunctive predicate, where $\tilde{Z}$ is the set of guards in $G(X,\tilde{Z})$ and $\tilde{Z} \subseteq Z$, and let $x \in X$. Then function *computeBounds* of Algorithm 4 returns:

- $\langle \textsc{UnBnd}, \bot, \bot \rangle$ if $G(X,\tilde{Z}) \wedge D(Z)$ is unbounded in x;
- $\langle \textsc{Infeas}, \bot, \bot \rangle$ if $G(X,\tilde{Z}) \wedge D(Z)$ is unfeasible;
- $\langle \textsc{Bnd}, a, b \rangle$ if $G(X,\tilde{Z}) \wedge D(Z)$ is bounded, where a, b are such that $G(X,\tilde{Z}) \wedge D(Z)$ implies $a \leq x \leq b$.

Example 4: In Example 3 we assumed bounds for each variable in the DTLHS $\mathcal{H}$ introduced in Example 1. Such bounds has been obtained by fixing bounds for state variables i_L and v_O and for auxiliary variables i_u, v_u, v_D and i_D, and then by computing bounds for variables i'_L, v'_O using Algorithm 4.

Algorithm 5 From predicates to conjunctive predicates

Input: P predicate over X and modality $\nu \in \{\textsc{Exh}, \textsc{Cut}\}$
Output: result μ and conjunctive predicate $C(X,Z)$ such that $C(X,Z)$ is X-equivalent to $P(X)$ if $P(X)$ is bounded.

function *PtoC*(P, X, ν)
1. $\langle G, D, Z, \tilde{Z} \rangle \leftarrow$*PtoG*$(P, X)$
2. $\tilde{D} \leftarrow$*GtoC*$(D, Z, \mathbf{0}, \mathbf{1})$
3. **for all** $x \in X$ **do**
4. **if** $\nu = \textsc{Cut}$ **then**
5. $\langle \mu, m_x, M_x \rangle \leftarrow$*computeBounds*$(G, \tilde{D}, X, Z, \tilde{Z}, x)$
6. **else**
7. $\langle \mu, m_x, M_x \rangle \leftarrow$*exhComputeBounds*$(G(X,\tilde{Z}) \wedge \tilde{D}(Z), X \cup Z, x)$
8. **if** $\mu \neq \textsc{Bnd}$ **then**
9. **return** $\langle \mu, \bot \rangle$
10. **return** $\langle \textsc{Bnd}, \mathit{GtoC}(G, X \cup Z, m, M) \rangle$

Function *PtoC* of Algorithm 5 presents the overall procedure that translates a bounded predicate $P(X)$ into an X-equivalent conjunctive predicate $C(X,Z)$. Function *PtoC* calls functions in Algorithms 1–4 and function *GtoC*(A, W, m, M), which translates a bounded guarded predicate $A(W)$ with known lower bounds m and upper bounds M for variables in W in a conjunctive predicate, as shown in the proof of Proposition 8. As a first step, Algorithm 5 translates the input predicate $P(X)$ into an X-equivalent guarded predicate $G(X,\tilde{Z}) \wedge D(Z)$ by calling the function *PtoG* (line 1). Since boolean variables are trivially bounded (bounds are vectors $\mathbf{0} = \langle 0, \ldots, 0 \rangle$ and $\mathbf{1} = \langle 1, \ldots, 1 \rangle$), the guarded predicate D can be translated into a conjunctive predicate $\tilde{D}$ by calling the function *GtoC* on D (line 2). To apply function *GtoC* on $G(X,\tilde{Z})$, we need bounds for each variable in X. These bounds are computed by calling $|X|$ times the function *computeBounds* and are stored in the two arrays m, M (lines 3 and 5). If the function *computeBounds* finds that $\tilde{G}$ is unfeasible or some x is not bounded in $\tilde{G}$ (line 8), the empty constraint is returned together with the failure explanation (line 9). Otherwise, the desired conjunctive predicate is returned in line 10.

Correctness of function *PtoC* of Algorithm 5 is stated in Proposition 13.

Proposition 13: Let $P(X)$ be a predicate. Then function *PtoC* of Algorithm 5 returns:

- $\langle \textsc{Unb}, \bot \rangle$ if $P(X)$ is unbounded for some $x \in X$;
- $\langle \textsc{Infeas}, \bot \rangle$ if $P(X)$ is unfeasible;
- $\langle \textsc{Bnd}, C(X,Z) \rangle$ if $P(X)$ is bounded, being $C(X,Z)$ a conjunctive predicate which is X-equivalent to $P(X)$.

Proof: The proof easily follows Propositions 5, 7, 8 and 12. ■

We end this section by proposing a syntactic check, that most of the time may be used to compute variable bounds

avoiding to use the function *computeBounds*.

Definition 5: A variable x is *explicitly bounded* in a predicate $P(X)$, if $P(X) = B(x) \wedge \tilde{P}(X)$, where $B(x) = x \leq b \wedge x \geq a$, for some constants a and b.

Proposition 14: Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS such that each variable $v \in X \cup U \cup Y$ is explicitly bounded in N, and for all $x' \in X'$ there are in N at least two constraints of the form $x' \geq L_1(X, U, Y)$ and $x' \leq L_2(X, U, Y)$. Then N is bounded.

Proof: Since all variables in X, U, and Y are explicitly bounded in N, they are also bounded in N. Therefore $\inf(L_1(X, U, Y))$ and $\sup(L_2(X, U, Y))$ are finite. Since N is guarded, it is a conjunction of guarded constraints and for all $x' \in X'$ it can be written as $x' \geq L_1(X, U, Y) \wedge x' \leq L_2(X, U, Y) \wedge \tilde{N}(X, U, Y, X')$ for a suitable guarded predicate $\tilde{N}$. This implies $\inf(L_1(X, U, Y)) \leq x' \leq \sup(L_2(X, U, Y))$, which in turn implies that x' is bounded in N. ■

Example 5: Let $\mathcal{H}_1$ be the DTLHS $(\{x\}, \{u\}, \varnothing, N_1)$, where $N_1(x, u, x') = (0 \leq x \leq 3) \wedge (0 \leq u \leq 1) \wedge (x' = x + 3u)$. By Proposition 14, $\mathcal{H}_1$ is bounded with $\inf(x') = 0$ and $\sup(x') = 6$. All other variables are explicitly bounded in N. Explicit bounds on present state and input variables do not imply that next state variables are bounded. As an example, let us consider the DTLHS $\mathcal{H}_2 = (\{x\}, \{u\}, \varnothing, N_2)$, where $N_2(x, u, x') = (0 \leq x \leq 3) \wedge (0 \leq u \leq 1) \wedge (x' \geq x + 3u)$. Since, for any value of x and u, x' can assume arbitrary large values, we have that N_2 is not bounded.

VI. Guarded Predicates as Modeling Language

The disjunction elimination procedure given in Algorithm 5 returns a guarded predicate that may contain a large number of fresh auxiliary boolean variables and this may heavily impact on the effectiveness of control software synthesis or verification (as well as the complexity of Algorithm 5 itself, since the auxiliary Algorithm 4 depends on the number of guard variables). On the other hand, guarded predicates, which are used as an intermediate step in Algorithm 5, are themselves a natural language to describe DTLHS behavior: assignments to guard variables play a role similar to modes in hybrid systems and, by using negative literals as guards, we can naturally model different kinds of plant behavior according to different commands sent by actuators.

Example 6: By directly using guarded predicates as modeling language, the DTLHS of Example 1 may be modeled by the conjunction of guarded constraints (43)–(52).

$$i'_L = (1 + Ta_{1,1})i_L + Ta_{1,2}v_O + Ta_{1,3}v_D \quad (43)$$
$$v'_O = Ta_{2,1}i_L + (1 + Ta_{2,2})v_O + Ta_{2,3}v_D. \quad (44)$$

$$v_D = v_u - V_i \quad (45) \qquad q \rightarrow v_D = 0 \quad (49)$$
$$i_D = i_L - i_u \quad (46) \qquad q \rightarrow i_D \geq 0 \quad (50)$$
$$u \rightarrow v_u = 0 \quad (47) \qquad \bar{q} \rightarrow v_D = R_{\text{off}} i_D \quad (51)$$
$$\bar{u} \rightarrow v_u = R_{\text{off}} i_u \quad (48) \qquad \bar{q} \rightarrow v_D \leq 0 \quad (52)$$

Algorithm 6 From guarded predicates to conjunctive predicates

Input: $G(X, Z)$ guarded predicate over X with guards in Z and modality $\nu \in \{\text{EXH}, \text{CUT}\}$.

Output: result μ and conjunctive predicate $C(X, Z)$ such that $C(X, Z)$ is X-equivalent to $G(X, Z)$ if $G(X, Z)$ is bounded.

function *GPtoC*(G, X, Z, ν)

1. **let** G and g be as in lines 1–2 of Algorithm 3
2. **if** $\nu = \text{CUT}$ **then**
3. $\quad \hat{Z} \leftarrow \{z \in Z \mid \exists i : g(i) = \bar{z} \vee g(i) = z\} \cup \{\tilde{z} \in Z \mid \exists i : g(i) = \bar{z}\}$
4. $\quad Z \leftarrow Z \cup \{\tilde{z} \in Z \mid \exists i : g(i) = \bar{z}\}$
5. $\quad G(X, \hat{Z}) \leftarrow \bigwedge_{i \in [n] \wedge g(i) = z_i} g(i) \rightarrow C_i(X, Z) \wedge \bigwedge_{i \in [n] \wedge g(i) = \bar{z}_i} \tilde{z}_i \rightarrow C_i(X, Z)$
6. $\quad D(Z) \leftarrow \bigwedge_{i \in [n] \wedge g(i) = 1} C_i(X, Z) \wedge \bigwedge_{z \in Z \wedge \exists i : g(i) = \bar{z}} \tilde{z} + z = 1$
7. **for all** $x \in X$ **do**
8. $\quad$ **if** $\nu = \text{CUT}$ **then**
9. $\quad\quad \langle \mu, m_x, M_x \rangle \leftarrow$ *computeBnds*$(G, D, X, Z, \hat{Z}, x)$
10. $\quad$ **else**
11. $\quad\quad \langle \mu, m_x, M_x \rangle \leftarrow$ *exhComputeBounds*(G, X, Z, x)
12. $\quad$ **if** $\mu \neq \text{BND}$ **then**
13. $\quad\quad$ **return** $\langle \mu, \bot \rangle$
14. **return** $\langle \text{BND}, \textit{GtoC}(G, X \cup Z, m, M) \rangle$

Note that disjunctions (7)–(9) in Example 1 have been replaced by guarded constraints (47)–(52). The resulting model for the buck DC-DC converter is much more succinct than the guarded model in Example 2 and it has 2 guard variables only, rather than 6 as in Example 2 (and 10 guarded constraints rather than 15).

Algorithm 4 cannot be directly applied to guarded predicates with both positive and negative guard literals. This obstruction can be easily bypassed, by observing that a guarded constraint $\bar{z} \rightarrow C(X)$ is $(X \cup \{z\})$-equivalent to the positive guarded predicate $(\tilde{z} \rightarrow C(X)) \wedge (\tilde{z} + z = 1)$. On the other hand, guarded predicates with both positive and negative guard literals may be directly translated in a conjunctive predicate by using the exhaustive procedure in Algorithm 3 to compute variable bounds. Both such translations are outlined in function *GPtoC* of Algorithm 6. Namely, if $\nu = \text{CUT}$ then the input guarded predicate is translated in a positive guarded predicate and then Algorithm 4 is used. Otherwise, i.e., if $\nu = \text{EXH}$ then the exhaustive Algorithm 3 is used directly on the original guarded predicate. Note that the above described method to obtain a positive guarded predicate from a guarded predicate (lines 3–6 in Algorithm 6) doubles the number of variables originally used as negative guards. Thus, it turns out that it is more convenient to call function *GPtoC* with $\nu = \text{EXH}$ (see experimental results in Section VII).

Summing up, guarded predicates turn out to be a powerful and natural modeling language for describing DTLHS transition relations.

VII. Experimental Results on a Case Study

In this section, we evaluate the effectiveness of our predicate translation functions, i.e., function *PtoC* of Algorithm 5 and function *GPtoC* of Algorithm 6. To this end, we implemented such functions in C programming language, using GLPK to solve MILP problems. We name the resulting tools PtoC (*Predicates to Conjunctive predicates translator*) and GPtoC (*Guarded Predicates to Conjunctive predicates translator*). We will write calls to functions *PtoC* (resp. *GPtoC*) with $\nu = \tilde{\nu}$ as PtoC($\tilde{\nu}$) (resp., GPtoC($\tilde{\nu}$)). PtoC and GPtoC are part of a more general tool named *Quantized feedback Kontrol Synthesizer* (QKS) [8][9].

We present the experimental results obtained by using PtoC and GPtoC on a n-inputs buck DC-DC converter (described in Section VII-A), that we model with two DTLHSs $\mathcal{H}_i = (X_i, U_i, Y_i, N_i)$, with $i \in [2]$, such that $X_1 = X_2$, $U_1 = U_2$, $Y_1 \subset Y_2$, $N_1(X_1, U_1, Y_1, X_1')$ is a predicate, and $N_2(X_2, U_2, Y_2, X_2')$ is a guarded predicate $(X_1 \cup U_1 \cup Y_1 \cup X_1')$-equivalent to N_1. All experiments have been carried out on a 3.00GHz Intel Xeon hyperthreaded Quad Core Linux PC with 8GB of RAM.

We run PtoC on N_1 and GPtoC on N_2 for increasing values of n (which entails that the number of guards increases), in order to show effectiveness of PtoC and GPtoC. To this end, both values for parameter ν will be used, which means that, for each n, 4 experiments are run. In Section VII-B we show experimental results PtoC. Furthermore, in Section VII-C we show that results obtained with GPtoC(Exh) are better than those obtained with both GPtoC(Cut) and PtoC. That is, the best results are obtained by exploiting knowledge of the system and modeling it with guarded predicates, and then using the exhaustive algorithm.

A. Multi-Input Buck DC-DC Converter

A Multi-Input Buck DC-DC Converter [18] (Figure 2), consists of n power supplies with voltage values $V_1 < \ldots < V_n$, n switches with voltage values $v_1^u, \ldots, v_n^u$ and current values $I_1^u, \ldots, I_n^u$, and n input diodes $D_0, \ldots, D_{n-1}$ with voltage values $v_0^D, \ldots, v_{n-1}^D$ and current values $i_0^D, \ldots, i_{n-1}^D$ (in the following, we will also write v_D for v_0^D and i_D for i_0^D). As for the converter in Example 1, the state variables are i_L and v_O, whereas action variables are $u_1, \ldots, u_n$, thus a control software for the n-input buck DC-DC converter has to properly actuate the switches $u_1, \ldots, u_n$. Constant values are the same given in Example 3.

B. Multi-Input Buck as a Predicate

We model the n-input buck DC-DC converter with the DTLHS $\mathcal{H}_1 = (X_1, U_1, Y_1, N_1)$, where $X_1 = [i_L, v_O]$, $U_1 = [u_1, \ldots, u_n]$, and

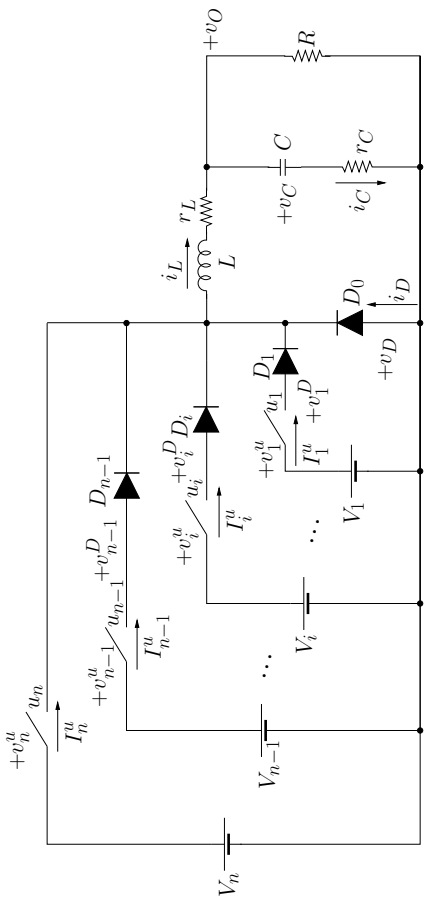

Figure 2. Multi-input Buck DC-DC converter

$Y_1 = [v_D, v_1^D, \ldots, v_{n-1}^D, i_D, I_1^u, \ldots, I_n^u, v_1^u, \ldots, v_n^u]$. From a simple circuit analysis (e.g., see [17]), we have that N_1 is the conjunction of linear predicates (53)–(61).

$$i_L' = (1 + Ta_{1,1})i_L + Ta_{1,2}v_O + Ta_{1,3}v_D \quad (53)$$

$$v_O' = Ta_{2,1}i_L + (1 + Ta_{2,2})v_O + Ta_{2,3}v_D. \quad (54)$$

$$((i_D \geq 0) \wedge (v_D = 0)) \vee ((i_D \leq 0) \wedge (v_D = R_{\text{off}} i_D)) \quad (55)$$

$$\bigwedge_{i \in [n]} (u_i = 0) \vee (v_i^u = 0) \quad (56)$$

$$\bigwedge_{i \in [n]} (u_i = 1) \vee (v_i^u = R_{\text{off}} I_i^u) \quad (57)$$

$$\bigwedge_{i \in [n-1]} ((I_i^u \geq 0) \wedge (v_i^D = 0)) \vee ((I_i^u \leq 0) \wedge (v_i^D = R_{\text{off}} I_i^u)) \quad (58)$$

$$i_L = i_D + \sum_{i=1}^{n} I_i^u \quad (59)$$

$$\bigwedge_{i \in [n-1]} v_D = v_i^u + v_i^D - V_i \quad (60)$$

$$v_D = v_n^u - V_n \quad (61)$$

N_1 also contains the following explicit bounds: $-4 \leq i_L \leq 4 \wedge -1 \leq v_O \leq 7 \wedge -10^3 \leq i_D \leq 10^3 \wedge \bigwedge_{i=1}^{n} -10^3 \leq I_i^u \leq 10^3 \wedge \bigwedge_{i=1}^{n} -10^7 \leq v_i^u \leq 10^7 \wedge \bigwedge_{i=0}^{n-1} -10^7 \leq v_i^D \leq 10^7$.

Table I
PTOC PERFORMANCE (PREDICATES)

n	r	r'	r''	k	$\|cut\|$	CPU_c	Mem_c	CPU_e	Mem_e	$\|In\|$	$\|Out\|$
2	12	6	12	11	64	1.07e+00	5.14e+07	3.07e+01	5.14e+07	21	44
3	18	9	18	17	512	9.63e+01	5.15e+07	2.92e+03	5.14e+07	30	63
4	24	12	24	23	4096	1.15e+04	5.15e+07	>1.38e+06	N/A	39	82

We run PTOC(CUT) with parameters $N_1, X_1 \cup U_1 \cup Y_1 \cup X_1'$ for increasing values of n, and we compare its computation time with that of PTOC(EXH) with the same input parameters. Table I shows our experimental results. In Table I, columns meaning are as follows:

- column n shows the number of buck inputs;
- column r shows the number of guards (see line 1 of Algorithm 4);
- columns r', r'' have the meaning given in lines 2 and 3 of Algorithm 4;
- column k gives the value of k at the end of the outer for loop of Algorithm 4;
- column $|cut|$ gives the size of cut at the end of the for loop of Algorithm 4;
- columns CPU_c and Mem_c (resp. CPU_e and Mem_e) show the computation time in seconds and memory usage in bytes of PTOC(CUT) (resp., of PTOC(EXH))
- column $|In|$ shows the size of the input predicate, as the number of linear constraints (i.e., of the linear predicate atoms) in the input linear predicate N_1;
- column $|Out|$ shows the size of the output conjunctive predicate, as the resulting number of linear constraints in the output conjunctive predicate.

C. Multi-Input Buck as a Guarded Predicate

We modify the DTLHS $\mathcal{H}_1$ of Section VII-B by defining $\mathcal{H}_2 = (X_2, U_2, Y_2, N_2)$, where $X_2 = X_1$, $U_2 = U_1$, $Y_2 = Y_1 \cup Y_2' = Y_1 \cup \{q_0, \ldots, q_{n-1}\}$ and N_2 is obtained from N_1 by replacing disjunctions (55)–(58) with guarded constraints. Thus, N_1 is given by the conjunction of guarded constraints (62)–(76).

$$i_L' = (1 + Ta_{1,1}) i_L + Ta_{1,2} v_O + Ta_{1,3} v_D \quad (62)$$
$$v_O' = Ta_{2,1} i_L + (1 + Ta_{2,2}) v_O + Ta_{2,3} v_D. \quad (63)$$
$$q \rightarrow v_D = 0 \quad (64) \qquad \bar{q} \rightarrow v_D = R_{\mathrm{off}} i_D \quad (66)$$
$$q \rightarrow i_D \geq 0 \quad (65) \qquad \bar{q} \rightarrow v_D \leq 0 \quad (67)$$
$$\bigwedge_{i \in [n-1]} q_i \rightarrow v_i^D = 0 \quad (68) \qquad \bigwedge_{i \in [n-1]} \bar{q}_i \rightarrow v_i^D \leq 0 \quad (71)$$
$$\bigwedge_{i \in [n-1]} q_i \rightarrow I_i^u \geq 0 \quad (69) \qquad \bigwedge_{i \in [n-1]} \bar{q}_i \rightarrow v_i^D = R_{\mathrm{off}} I_i^u \quad (72)$$
$$\bigwedge_{i \in [n]} u_i \rightarrow v_i^u = 0 \quad (70) \qquad \bigwedge_{i \in [n]} \bar{u}_i \rightarrow v_i^u = R_{\mathrm{off}} I_i^u \quad (73)$$
$$i_L = i_D + \sum_{i=1}^{n} I_i^u \quad (74)$$
$$\bigwedge_{i \in [n-1]} v_D = v_i^u + v_i^D - V_i \quad (75)$$
$$v_D = v_n^u - V_n \quad (76)$$

We call both GPTOC(CUT) and GPTOC(EXH) with parameters $N_2, X_2 \cup U_2 \cup Y_2 \cup X_2'$ for increasing values of n, and we compare their computation times.

Table II shows our experimental results. Columns meaning in Table II are the same as of Table I. An additional column $|Y_2|$ shows the number of guard variables in N_2.

D. Evaluation

Results in Table I show that heuristics implemented in function *computeBounds* are indeed effective w.r.t. executing function *exhComputeBounds*. In fact, by comparing columns CPU_c and CPU_e (and recalling that the only difference between PTOC(CUT) shown in CPU_c and PTOC(CUT) shown in CPU_e is that the former calls function *computeBounds* whilst the latter calls function *exhComputeBounds*), we see that such heuristics provide at least a one-order-of-magnitude speed-up in variable bounds computation. Such speed-up rapidly grows with the size of the input. In fact, for the 4-bits buck DC-DC converter, PTOC(EXH) requires more than 2 weeks, whilst PTOC(CUT) terminates in about 3 hours. Moreover, we also note that the resulting number of linear constraints output by PTOC is at most twice the starting number of linear constraints.

PTOC(CUT) is however not effective on the n-input buck DC-DC converter for $n \geq 5$. In fact, for $n = 5$, there are 30 boolean guards (i.e., $r = 30$), and the heuristics do not provide enough speed-up to obtain termination in a reasonable time. However, if we directly use guarded predicates as input language as in Section VII-C, we are able to generate the conjunctive predicate for both $n = 5$ and $n = 6$. This is due to the smaller number of guard variables used in Section VII-C than that used in Section VII-B. The negative impact of auxiliary boolean variables is clearly showed by the fact that GPTOC(EXH), much slower than GPTOC(CUT) on a model of the same size, performs better than GPTOC(CUT) in this case, because it can work on a model with half of the variables (see columns $|Y_2|$ and r). The same holds if we compare results of GPTOC(EXH) with those of PTOC(EXH) and PTOC(CUT). This phenomenon would be greatly amplified in a verification or control software synthesis procedure. These results strongly support guarded predicates as modeling language.

Table II
GPTOC PERFORMANCE (GUARDED PREDICATES)

n	$\lvert Y_2\rvert$	r	r'	r''	k	$\lvert cut\rvert$	CPU_c	Mem_c	CPU_e	Mem_e	$\lvert In\rvert$	$\lvert In_{\mathrm{pos}}\rvert$	$\lvert Out\rvert$
2	4	8	4	4	4	16	2.80e-01	5.14e+07	2.50e-01	5.14e+07	17	21	38
3	6	12	6	6	6	64	9.70e-01	5.15e+07	9.70e-01	5.15e+07	24	30	54
4	8	16	8	8	8	256	1.04e+01	5.16e+07	3.41e+00	5.15e+07	31	39	70
5	10	20	10	10	10	1024	1.75e+02	5.17e+07	1.69e+01	5.16e+07	38	48	86
6	12	24	12	12	12	4096	2.55e+03	5.17e+07	8.57e+01	5.17e+07	45	57	102

VIII. RELATED WORK

This paper is an extended version of [1]. With respect to [1], this paper provides more details in the introduction and in the related work description, extends basic definitions and algorithms descriptions, gives more detailed proofs for theorems, and provides a revised and enriched version of the experiments.

MILP problems solving based abstraction techniques have been designed for the verification of *Discrete Time Hybrid Automata* (DHA) [5] and implemented within the symbolic model checker HYSDEL [19]. A MILP based DTLHS abstraction algorithm is the core of automatic control software synthesis from system level specifications in [8][9], and it requires DTLHS dynamics modeled as a conjunctive predicate. The same limitation occurs in abstraction techniques based on the Fourier-Motzkin procedure for existential quantifier elimination [20]. All such approaches may exploit the translation algorithm presented here in order to improve their applicability.

Automatic or automatable translation procedures targeting MILP formulations have been presented in [21] and [22]. Namely, in [22] the authors propose an approach to translate (*reformulate* in their parlance) mixed integer bilinear problems (i.e., problems in which constraints may contain products of a nonnegative integer variable and a nonnegative continuous variable) into MILP problems. This reformulation is obtained by first replacing a general integer variable with its binary expansion and then using McCormick envelopes to linearize the resulting product of continuous and binary variables. In [21], the authors present an automatic conversion from deterministic finite automata to MILP formulations. This allows to efficiently combine supervisory control theory and MILP to automatically generate time-optimal, collision-free and non-blocking working schedules for a flexible manufacturing system. Both these works differ from ours in the starting point of the translation procedure (and of course in the actual algorithms designed): in [21] they are interested in translating deterministic finite automata, whilst in [22] the goal is to translate mixed integer bilinear problems. On the other hand, in this paper we are interested in translating conjunctions and disjunctions of linear constraints (see Section II-A), thus the approaches in [21][22] cannot be used in our context.

Many works in the literature deal with automatic specification of MILP problems in order to solve customized synthesis problem. As an example, in [23] the target is a formal synthesis approach to design of optimal application-specific heterogeneous multiprocessor systems. As a further example, in [24], a topology synthesis method for high performance System-on-Chip design is presented. Finally, in [25] the development of a technique to target fresh water consumption and wastewater generation for systems involving multiple contaminants is presented. In this paper, rather than giving a MILP scheme to be properly customized to solve a problem of a given type, we provide a translation from a general-purpose predicate to an equivalent MILP problem.

Finally, we note that the automatic procedure presented in this paper is reminiscent of Mixed Integer Programming modeling techniques [26] in Operations Research and boolean formula transformations involved in the conversion of a formula into a conjunctive or disjunctive normal form [6][27].

IX. CONCLUSIONS AND FUTURE WORK

The results presented in this paper contribute to *model based design* of SBCS (most notable, of embedded software) by proposing an expressive modeling language for DTLHS. In fact, in our previous work MILP based approaches have been used to synthesize correct-by-construction control software for DTLHSs. However, such approaches require DTLHS dynamics to be modeled as a conjunctive linear predicate over state, input, and next state variables. This may turn out to be not practically feasible for DTLHSs with complex dynamics.

In this paper, we circumvented such a limitation, by giving an automatic procedure that translates any disjunction-conjunction of linear constraints into an equisatisfiable conjunctive predicate, provided that each variable ranges over a bounded interval. This last proviso is automatically enforced by our procedure, since it includes a routine algorithm that, taking a linear predicate P and a variable x, verifies if x is bounded in P, by computing (an over-approximation of) bounds for x.

Finally, our experimental results show the effectiveness of our approach on an important and challenging case study

taken from the literature, namely the multi-input Buck DC-DC Converter. As an example, the linear predicate that models a 4-inputs buck DC-DC converter with 39 linear constraints is translated into a conjunctive predicate of 82 linear constraints in slightly more than 3 hours. Most notably, our experimental results show that guarded predicates, which are used by our translation procedure as an intermediate language, turn out to be a natural language to succinctly describe DTLHS dynamics. In fact, the guarded predicate that models a 6-inputs Buck DC-DC Converter with 57 constraints (including 12 different guards), is translated into a conjunctive predicate of 102 linear constraints in about 40 minutes.

The presented approach has the main drawback to be exponential on the number of boolean guards used in the (initial or intermediate) guarded predicate. As a future work, we aim to counteract such a limitation by recognizing if the input predicate is of some known structure. As an example, if the guarded predicate is composed by k blocks of the same structure, we may translate just one of such blocks and then suitably copy the resulting conjunctive predicate k times.

ACKNOWLEDGMENTS

Our work has been partially supported by: MIUR project DM24283 (TRAMP) and by the EC FP7 projects GA600773 (PAEON) and GA317761 (SmartHG).

ACRONYMS

AD Analog-to-Digital. 1
DA Digital-to-Analog. 1
DC Direct Current. 4
DHA Discrete Time Hybrid Automata. 12
DTLHS Discrete Time Linear Hybrid System. 1, 2, 4, 6, 7, 9–13
DVFS Dynamic Voltage and Frequency Scaling. 4
MILP Mixed Integer Linear Programming. 1–4, 7, 10, 12, 13
NFSA Nondeterministic Finite State Automaton. 1
QKS Quantized feedback Kontrol Synthesizer. 10
SBCS Software Based Control System. 1

REFERENCES

[1] F. Mari, I. Melatti, I. Salvo, and E. Tronci, "Linear constraints as a modeling language for discrete time hybrid systems," in *ICSEA*, 2012, pp. 664–671.

[2] T. A. Henzinger and J. Sifakis, "The embedded systems design challenge," in *FM*, ser. LNCS 4085, 2006, pp. 1–15.

[3] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theoretical Computer Science*, vol. 138, no. 1, pp. 3 – 34, 1995.

[4] R. Alur, T. A. Henzinger, and P.-H. Ho, "Automatic symbolic verification of embedded systems," *IEEE Trans. Softw. Eng.*, vol. 22, no. 3, pp. 181–201, 1996.

[5] A. Bemporad and M. Morari, "Verification of hybrid systems via mathematical programming," in *HSCC*, ser. LNCS 1569, 1999, pp. 31–45.

[6] F. Mari and E. Tronci, "CEGAR based bounded model checking of discrete time hybrid systems," in *HSCC*, ser. LNCS 4416, 2007, pp. 399–412.

[7] V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, and E. Tronci, "Automatic control software synthesis for quantized discrete time hybrid systems," in *CDC*. IEEE, 2012, pp. 6120–6125.

[8] F. Mari, I. Melatti, I. Salvo, and E. Tronci, "Synthesis of quantized feedback control software for discrete time linear hybrid systems," in *CAV*, ser. LNCS 6174, 2010, pp. 180–195.

[9] ——, "Model based synthesis of control software from system level formal specifications," *ACM Trans. on Soft. Eng. and Meth.*, vol. To appear. [Online]. Available: http://mclab.di.uniroma1.it/publications/papers/federicomari/2013/110_FedericoMari2013.pdf

[10] ——, "Undecidability of quantized state feedback control for discrete time linear hybrid systems," in *Proceedings of the International Colloquium on Theoretical Aspects of Computing, ICTAC*, ser. LNCS, A. Roychoudhury and M. D'Souza, Eds., vol. 7521. Springer-Verlag Berlin Heidelberg, 2012, pp. 243–258.

[11] ——, "Synthesizing control software from boolean relations," *Int. J. on Advances in SW*, vol. 5, no. 3&4, pp. 212–223, 2012.

[12] "Gnu GLPK Web Page: http://www.gnu.org/software/glpk/," last accessed 6 mar 2013.

[13] "CPLEX Web Page: http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/," last accessed 6 mar 2013.

[14] W. Kim, M. S. Gupta, G.-Y. Wei, and D. M. Brooks, "Enabling on-chip switching regulators for multi-core processors using current staggering," in *ASGI*, 2007.

[15] W.-C. So, C. Tse, and Y.-S. Lee, "Development of a fuzzy logic controller for dc/dc converters: design, computer simulation, and experimental evaluation," *IEEE Trans. on Power Electronics*, vol. 11, no. 1, pp. 24–32, 1996.

[16] V. Yousefzadeh, A. Babazadeh, B. Ramachandran, E. Alarcon, L. Pao, and D. Maksimovic, "Proximate time-optimal digital control for synchronous buck dc–dc converters," *IEEE Trans. on Power Electronics*, vol. 23, no. 4, pp. 2018–2026, 2008.

[17] P.-Z. Lin, C.-F. Hsu, and T.-T. Lee, "Type-2 fuzzy logic controller design for buck dc-dc converters," in *FUZZ*, 2005, pp. 365–370.

[18] M. Rodriguez, P. Fernandez-Miaja, A. Rodriguez, and J. Sebastian, "A multiple-input digitally controlled buck converter for envelope tracking applications in radiofrequency power amplifiers," *IEEE Trans. on Power Electronics*, vol. 25, no. 2, pp. 369–381, 2010.

[19] F. Torrisi and A. Bemporad, "HYSDEL — A tool for generating computational hybrid models for analysis and synthesis problems," *IEEE Transactions on Control System Technology*, vol. 12, no. 2, pp. 235–249, 2004.

[20] S. K. Jha, B. H. Krogh, J. E. Weimer, and E. M. Clarke, "Reachability for linear hybrid automata using iterative relaxation abstraction," in *HSCC*, ser. LNCS 4416, 2007, pp. 287–300.

[21] A. Kobetski and M. Fabian, "Scheduling of discrete event systems using mixed integer linear programming," in *Discrete Event Systems, 2006 8th International Workshop on*, july 2006, pp. 76 –81.

[22] A. Gupte, S. Ahmed, M. S. Cheon, and S. S. Dey, "Solving mixed integer bilinear problems using milp formulations," *SIAM J. on Optimization*, vol. To appear. [Online]. Available: http://www.optimization-online.org/DB_FILE/2011/07/3087.pdf

[23] S. Prakash and A. C. Parker, "Readings in hardware/software co-design," G. De Micheli, R. Ernst, and W. Wolf, Eds. Norwell, MA, USA: Kluwer Academic Publishers, 2002, ch. SOS: synthesis of application-specific heterogeneous multiprocessor systems, pp. 324–337. [Online]. Available: http://dl.acm.org/citation.cfm?id=567003.567031

[24] M. Jun, S. Yoo, and E.-Y. Chung, "Mixed integer linear programming-based optimal topology synthesis of cascaded crossbar switches," in *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '08. Los Alamitos, CA, USA: IEEE Computer Society Press, 2008, pp. 583–588. [Online]. Available: http://dl.acm.org/citation.cfm?id=1356802.1356945

[25] Z. Handani, H. Hashim, S. Alwi, and Z. Manan, "A mixed integer linear programming (milp) model for optimal design of water network," in *Modeling, Simulation and Applied Optimization (ICMSAO), 2011 4th International Conference on*, april 2011, pp. 1 –6.

[26] F. S. Hillier and G. J. Lieberman, *Introduction to operations research*. McGraw-Hill Inc., 2001.

[27] D. Sheridan, "The optimality of a fast cnf conversion and its use with sat," in *SAT*, 2004.