Linear Constraints 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 embedded software design where system level specifications are much easier to define than the control software behavior itself. Formal analysis of *Embedded Systems* requires modelling both continuous systems (typically, the plant) as well as discrete systems (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 System*, where plant dynamics is modeled as a linear predicate over state, input, and next state variables. MILP solvers requires constraints represented as conjunctive predicates. 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 transformed into an equisatisfiable conjunctive predicate. Moreover, since variable bounds play a key role in this transformation, we present an algorithm that taking as input a linear predicate, computes *implicit variable bounds*.**

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

I. Introduction

Many Embedded Systems are *Software Based Control Systems* (SBCSs). 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*).

Software generation from models and formal specifications forms the core of *Model Based Design* of embedded software [1]. 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. Correct-by-construction software generation as well as formal verification of system level specifications for SBCSs requires modelling both the continuous subsystem (the plant) and discrete systems (the controller). This is typically done using *Hybrid Systems* (e.g., see [2][3]).

Discrete Time Linear Hybrid Systems (DTLHSs) [4][5] 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.

In [6], stemming from a constructive sufficient condition for the existence of a quantized sampling controller for an SBCS modelled as a DTLHS, 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 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, and thus it requires the DTLHS dynamics given as a conjunctive predicate, i.e., a conjunction of linear constraints.

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 represented by an equivalent conjunctive predicate.

Bounds on variables that describe DTLHS behaviour is a reasonable hypothesis. Usually, control software drives the plant towards a goal, while keeping it inside a given bounded admissible region. Bounds on present state variables essentially model the *sensing region*, that is the range of values observable by the sensors, that usually is 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. Non-state variables may model both non-observable plant state variables and uncontrollable inputs (i.e., disturbances). Therefore, bounds on such variables are usually implied by bounds on state variables or by reasonable assumptions about disturbances.

1) Our Main Contributions: In this paper we give an algorithm to transform any linear predicate into an equisatisfiable conjunctive predicate, under the hypothesis that each variable ranges over a bounded interval. This allows a MILP based abstraction technique to be applied on a wider class of DTLHSs (Section III) with respect to [6].

We consider predicates built upon linear constraints (i.e., inequalities of the shape $\sum_{i=1}^{n} a_i x_i \leq b$, Section II), conjunctions and disjunctions. First, we show that, at the price of introducing fresh boolean variables, a predicate can be transformed into an equisatisfiable *guarded predicate* (Section IV), that is a conjunction of guarded constraints, i.e., constraints of the shape $y \to (\sum_{i=1}^{n} a_i x_i \leq b)$. Then, assuming that each variable ranges over a bounded interval, we show that any guarded constraint can be in turn transformed into a *conjunctive predicate*, i.e., a conjunction of linear constraints (Section IV-A). Conjunctive predicates are the input language of MILP solvers. Finally, 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 evaluation of such algorithm is in Sections VI and VII.

A. Related Work

Mixed Integer Linear Programming (MILP) solving based abstraction techniques have been designed for the verification of Discrete Time Hybrid Automata (DHA) [4] and implemented within the symbolic model checker HYSDEL [7]. A MILP based DTLHS abstraction algorithm is the core of automatic control software synthesis from system level specifications in [6], 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 [8].

The automatic procedure that we present here to transform any linear predicate into an equisatisfiable conjunctive predicate is reminiscent of Mixed Integer Programming modeling techniques [9] in Operations Research and boolean formula transformations involved in the conversion of a formula into a conjunctive or disjunctive normal form [5][10].

Finally, an automatic convertion procedure targeting a MILP formulation for automatic synthesis of schedules is presented in [11], where the starting point is a deterministic finite automaton rather than a linear predicate.

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 $J(b)$ be the set of indexes such that $b_j = 1$, i.e., $J(b) = \{j \in [n] \mid b_j = 1\}$.

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 $(L(X) \leq b) \wedge (L(X) \geq a)$.

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 $v(x_1), \ldots, v(x_n)$. Given a predicate $P(Y, X)$, $P(Y, X^*)$ denotes the predicate obtained by substituting each occurrence of x with $v(x)$. We call valuation also the sequence of values X^*. A *satisfying assignment* to a predicate $P(X)$ is a valuation X^* such that $P(X^*)$ holds. 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.

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 conjunctive predicate over X. A *solution* to $(\max, J(X), A(X))$ is a valuation X^* such that $A(X^*)$ and $\forall Z\, (A(Z) \to (J(Z) \leq J(X^*)))$. $J(X^*)$ 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 TRUE if $A(X)$ is satisfiable and FALSE 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 $(\max, -J(X), A(X))$.

III. Discrete Time Linear Hybrid Systems

Discrete Time Linear Hybrid Systems (DTLHSs) provide a suitable model for many embedded control systems since they can effectively model linear algebraic constraints involving both continuous as well as discrete variables. In Ex. 1, we present a DTLHS model of a buck DC-DC converter, i.e., a mixed-mode analog circuit that converts the 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.

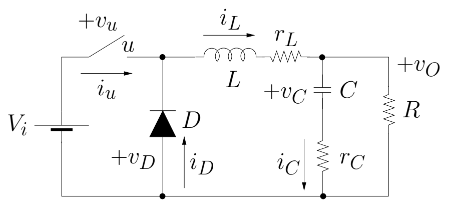

Figure 1. Buck DC-DC converter

$U = U^r \cup U^d$ 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 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 as well as on-chip to support *Dynamic Voltage and Frequency Scaling* (DVFS) in multicore processors. Because of its widespread use, control schemes for buck DC-DC converters have been widely studied (e.g., see [12][13][14]). The typical software based approach is to control the switch u in Figure 1 (typically implemented with a MOSFET) with a microcontroller. The circuit in Figure 1 can be modeled as a DTLHS $\mathcal{H} = (X, U, Y, N)$. 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 $\mathcal{H}$: $X = X^r = i_L, v_O$, $U = U^d = u$, $Y = Y^r \cup Y^d$ 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 [15]) 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 constraints in Eqs. 3–9.

IV. From Linear to Conjunctive Predicates

As shown in [6], MILP solvers can be used to build a suitable discrete abstraction of a DTLHS. As mentioned in Section II-B, MILP solvers require constraints represented as conjunctive predicates. In this section, we show how this limitation can be circumvented. We proceed in two steps. First, in Section IV, we introduce *guarded predicates* and we show that each predicate can be transformed into an equivalent guarded predicate at the price of introducing new auxiliary boolean variables. Then, in Section IV-A, we show that, under the hypothesis that each variable ranges over a bounded interval, each guarded predicate can be in turn transformed into an equivalent conjunctive predicate.

1) Guarded Predicates:

Definition 2: Given a predicate $P(X)$ and a fresh boolean variable $z \notin X$, the predicate $z \to P(X)$ (resp. $\bar{z} \to P(X)$) 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*. If $P(X)$ is a constraint $C(X)$, a predicate of the form $z \to C(X)$ or $\bar{z} \to C(X)$ is called *guarded constraint*. A *generalized guarded constraint* a predicate of the form $z_1 \to (z_2 \to \ldots \to (z_n \to C(X)) \ldots)$ A *guarded predicate* (resp. *generalized* guarded predicate) is a conjunction of either constraints or guarded constraints (resp. generalized guarded constraints).

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

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

Fact 1: A predicate of the form $z \to \bigwedge_{i \in [n]} P_i(X)$ is equivalent to the guarded predicate $\bigwedge_{i \in [n]} (z \to P_i(X))$.

Fact 2: A generalized guarded constraint $z_1 \to (z_2 \to \ldots \to (z_n \to C(X)) \ldots)$ is equisatisfiable to the guarded predicate $(z - \sum_{i \in [n]} z_i \geq 1 - n) \wedge (z \to C(X))$, where z is a fresh boolean variable.

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

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

Lemma 3: Any predicate $P(X)$ is equisatisfiable to a predicate $Q(X, Z) = G(X, Z) \wedge D(Z)$, where G and D are generalized guarded predicates and Z is the set of boolean variables that occur positively as guards in G.

Proof: By induction on the structure of the predicate $P(X)$. If $P(X)$ is a constraint or a conjunction, the statement easily follows from inductive hypothesis.

Let $P(X)$ be the disjunction $P_1(X) \vee P_2(X)$. By inductive hypothesis, there exist two generalized guarded predicates $Q_1(X, Z_1) = G_1(X, Z_1) \wedge D_1(Z_1)$ and $Q_2(X, Z_2) =$

$G_2(X, Z_2) \wedge D_2(Z_2)$ such that $P_1(X) \simeq Q_1(X, Z_1)$ and $P_2(X) \simeq Q_2(X, Z_2)$. 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 and y_2, the predicate $y_1 \to Q_1(X, Z_1) \wedge y_2 \to Q_2(X, Z_2) \wedge y_1 + y_2 \geq 1$ is equisatisfiable to $P(X)$. The predicate $y_1 \to Q_1(X, Z_1)$ has the form $y_1 \to (\bigwedge_{i\in[n]} G_1^i(X, Z_1) \wedge \bigwedge_{j\in[p]} D_1^j(Z_1))$ and therefore it is not a generalized guarded constraint. By Fact 1, it is equivalent to the predicate $\bigwedge_{i\in[n]}(y_1 \to G_1^i(X, Z_1)) \wedge \bigwedge_{j\in[p]}(y_1 \to D_1^j(Z_2))$. By applying Fact 1 also to $y_2 \to Q_2(X, Z_2)$, the statement follows by taking $Z = Z_1 \cup Z_2 \cup \{y_1, y_2\}$, $G(X, Z) = \bigwedge_{i\in[n]} y_1 \to G_1^i(X, Z_1) \wedge \bigwedge_{i\in[m]} y_2 \to G_2^i(X, Z_2)$, and $D(Z) = \bigwedge_{j\in[p]} y_1 \to D_1^j(Z_2) \wedge \bigwedge_{j\in[q]} y_2 \to D_2^j(Z_2) \wedge (y_1 + y_2 \geq 1)$ ■

Proposition 4: Any predicate $P(X)$ is equisatisfiable to a predicate $Q(X, Z) = G(X, Z') \wedge D(Z)$, where G and D are guarded predicates and $Z' \subseteq Z$ is the set of boolean variables that occur positively as guards in G.

Proof: By Lemma 3, any predicate $P(X)$ is equisatisfiable to a generalized guarded predicate $G_1(X, Z_1) \wedge D_1(Z_1)$. By Fact 2, $D_1(Z_1)$ is equisatisfiable to a guarded predicate $D_2(Z_1, Z_2)$. Let $G_1(X, Z_1) = \bigwedge_{i\in[n]} z_1^i \to (z_2^i \to \ldots \to (z_{n_i}^i \to C_i(X)) \ldots) \wedge G_1'(X, Z_3)$, where $G_1'(X, Z_3)$ is a guarded predicate ($Z_3 \subseteq Z_1$). By Fact 2 $G_1(X, Z_1)$ is equisatisfiable to the guarded predicate $\bigwedge_{i\in[n]} w_i \to C_i(X) \wedge \bigwedge_{i\in[n]}(w_i - \sum_{j\in[n_i]} z_j^i \geq 1 - n_i) \wedge G'(X, Z_3)$. The statement follows by taking $Z' = Z_3 \cup \{w_1, \ldots, w_n\}$, $Z = Z' \cup Z_1 \cup Z_2$, $G(X, Z) = \bigwedge_{i\in[n]}(z^i \to C_i(X)) \wedge G''(X, Z')$, and $D(X, Z) = \bigwedge_{i\in[n]}(z^i - \sum_{j\in[n_i]} z_j^i \geq 1 - n) \wedge D''(Z', Z'')$ ■

The function *PtoG* (Alg. 2) summarizes the predicate transformations given in the proof of Prop. 4. It calls function *PtoGG* (Alg. 1) that performs predicate transformations given in the proof of Lemma 3. The function *fresh*( ) returns at each invocation a (globally) fresh variable.

Algorithm 1 From predicates to generalized guarded pred.

```
Input: $P$ predicate over $X$
Output: $\langle G, D, Z\rangle$ where $G$ is a general. guarded predicate,
    $Z$ is the set of its (fresh) guard variables,
    $D(Z)$ is a generalized guarded predicate over Z
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.     $y_1 \leftarrow fresh()$, $y_2 \leftarrow fresh()$, $Z' \leftarrow Z_1 \cup Z_2 \cup \{y_1, y_2\}$
 8.     $D' = \{y_1 \to \gamma \mid \gamma \in D_1\} \cup \{y_2 \to \gamma \mid \gamma \in D_2\} \cup \{y_1 + y_2 \geq 1\}$
 9.     $G' = \{y_1 \to \gamma \mid \gamma \in G_1\} \cup \{y_2 \to \gamma \mid \gamma \in G_2\}$
10.     return $\langle G', D', Z'\rangle$
```

Example 2: Let $\mathcal{H}$ be DTLHS in Ex. 1. Given the predicate N that defines the transition relation of $\mathcal{H}$, function *PtoG* computes the following guarded predicate equisatisfiable to N. Constraints 3–6 remain unchanged, as they are linear constraints in a top-level conjunction. The disjunction 9 is replaced first by the following predicates:

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

and then by constraints 13–16 below, obtained by moving arrows inside the conjunctions, as shown by Fact 1. Similarly, disjunctions 7 and 8 are eliminated by introducing four boolean fresh variables. Summing up, disjunctions 7–9 in Example 1 are replaced by the conjunction of the following (guarded) constraints:

$$\begin{array}{rlrlrl}
z_4 \to (v_u = R_{\text{off}} i_u) & (12) & z_1 \to (i_D \leq 0) & (16) & & \\
z_2 \to (v_D = R_{\text{off}} i_D) & (13) & z_3 \to (u = 1) & (17) & z_1 + z_2 \geq 1 & (20) \\
z_1 \to (i_D \geq 0) & (14) & z_5 \to (u = 0) & (18) & z_3 + z_4 \geq 1 & (21) \\
z_1 \to (v_D = 0) & (15) & z_6 \to (v_u = 0) & (19) & z_5 + z_6 \geq 1 & (22)
\end{array}$$

With respect to the statement of Proposition 4, we have that $Z = \{z_1, z_2, z_3, z_4, z_5, z_6\}$, $G(X, Z')$ is the conjunction of guarded constraints 12–19 and original constraints 3–6, and $D(Z)$ is the conjunction of constraints 20–22.

Algorithm 2 From linear to guarded predicates

```
Input: $P$ predicate over $X$
Output: $\langle G, D, Z', Z\rangle$ where $G$ is a guarded predicate,
    $Z' \subseteq Z$ set of its guard variables,
    $D(Z)$ is a guarded predicate over Z
function $PtoG(P, X)$
 1. $\langle G, D, Z\rangle \leftarrow PtoGG(P, X)$
 2. $G' \leftarrow \varnothing$, $D' \leftarrow \varnothing$, $Z' = \varnothing$
 3. for all $\gamma \in G \cup D$ do
 4.     if $\gamma \equiv z_1 \to (\ldots \to (z_n \to C(W)) \ldots)$ then
 5.         $w \leftarrow fresh()$, $Z \leftarrow Z \cup \{w\}$
 6.         if $W \subseteq X$ then $G' \leftarrow G' \cup \{w \to C(W)\}$
 7.             else $D' \leftarrow D' \cup \{w \to C(W)\}$
 8.         $D' \leftarrow D' \cup \{w - \sum_{i\in[n]} z_i \geq 1 - n\}$
 9.     else if $vars(\gamma) \subseteq X$ then
10.             $G' \leftarrow G' \cup \{\gamma\}$ else $D' \leftarrow D' \cup \{\gamma\}$
11. return $\langle G', D', Z', Z \setminus Z'\rangle$
```

A. From Guarded to Conjunctive Predicates

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 bounded predicate $P(X)$, a real number a, and a variable $x \in X$ 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 5: Each bounded guarded predicate $P(X)$ is equivalent conjunctive predicate $Q(X)$.

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

Together with Prop. 4, Prop. 5 implies that any bounded predicate can be transformed into an equisatisfiable conjunctive predicate, at the cost of adding new auxiliary boolean variables, as stated in the following proposition.

Proposition 6: For each bounded predicate $P(X)$, there exists an equisatisfiable conjunctive predicate $Q(X, Z)$.

Example 3: Let $\mathcal{H}$ be the DTLHS in Examples 1 and 2. We set the parameters of $\mathcal{H}$ as follows:

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

and we assume variables bounds as follows:

$$-2 \cdot 10^4 \leq v_u \leq 15 \quad -4 \leq i_L \leq 4 \quad -1 \leq v_O \leq 7 \quad -4 \leq i'_L \leq 96$$
$$-2 \cdot 10^4 \leq v_D \leq 0 \quad -1.1 \leq v'_O \leq 17 \quad -4 \leq i_u \leq 4 \quad -2 \leq i_D \leq 4$$

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 Prop. 5, guarded constraints 14–19 are replaced by the following linear constraints:

$$2z_1 - i_D \leq 2 \quad (23)$$
$$4 \cdot 10^4 z_4 + v_u - 10^4 i_u \leq 4 \cdot 10^4 \quad (24)$$
$$6 \cdot 10^4 z_4 - v_u + 10^4 i_u \leq 6 \cdot 10^4 \quad (25)$$
$$-2.10^4 z_1 - v_D \leq 2 \cdot 10^4 \quad (26)$$
$$2.10^4 z_2 + v_D - 10^4 i_D \leq 2.10^4 \quad (27)$$
$$6.10^4 z_2 - v_D + 10^4 i_D \leq 6.10^4 \quad (28)$$
$$2 \cdot 10^4 z_6 + v_u \leq 15 \quad (29)$$
$$2 \cdot 10^4 z_4 - v_u \leq 2 \cdot 10^4 \quad (30)$$
$$v_D \leq 0 \quad (31)$$
$$4z_2 + i_D \leq 4 \quad (32)$$
$$z_5 + u \leq 1 \quad (33)$$
$$-u \leq 0 \quad (34)$$
$$15z_6 + v_u \leq 15 \quad (35)$$
$$z_3 - u \leq 1 \quad (36)$$
$$u \leq 1 \quad (37)$$

V. Computing Variable Bounds

In this section, we present an algorithm that checks if a variable x is bounded and that computes an over- and under-approximation of $\sup(x)$ and $\inf(x)$.

Given a guarded predicate $G(X, Z)$, where Z is the set of guard variables, for any valuation Z^*, $G(X, Z^*)$ is equivalent to a conjunctive predicate (Prop. 7). A naïve algorithm to find bounds for a variable x for any valuation Z^* solves the MILP problems *optimalValue*$(x, \max,\ G(X, Z^*))$ and *optimalValue*$(x, \min,\ G(X, Z^*))$. If, *for all* $Z^* \in \mathbb{B}^n$, x is bounded in $G(X, Z^*)$ or $G(X, Z^*)$ is unfeasible, then x is bounded in $G(X, Z)$. Vice versa, if *for some* $Z^* \in \mathbb{B}^n$ $G(X, Z^*)$ is feasible and x is not bounded, then x is not bounded in $G(X, Z)$. Unfortunately, this exhaustive procedure requires to solve $2^{|Z|}$ MILP problems.

The function *computeBounds* in Alg. 3 refines such idea in order to save unnecessary MILP invocations. If all guard literals are positive, 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 $G(X, Z_2^*)$ and 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 (Prop. 7). In the following we establish the correctness of function *computeBounds*.

Proposition 7: 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 guarded constraints, where head variables occurs positively. Then:

1) For any $Z^* \in \mathbb{B}^n$, $G(X, Z^*)$ is equivalent to the conjunctive predicate $\bigwedge_{j \in J(Z^*)} C_j(X)$.
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 $J(a) \subseteq J(b)$ and hence $G(X, b)$ has more constraints than $G(X, a)$. ■

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 $Z = Z_1 \cup Z_2$ and let $|Z_2| = n$. A cut $C \subseteq \mathbb{B}^n$ is (D, Z_2)-minimal, if for all $c \in C$ $D(Z_1, c)$ is satisfiable, and for all $b < C$ $D(Z_1, b)$ is not satisfiable.

To verify that a variable is bounded $G(X, Z') \wedge D(Z)$, where G is a guarded predicate with positive guards in the set $Z' \subseteq Z$ and $D(Z)$ is a conjunctive predicate, it suffices to check if it is bounded in the conjunctive predicate $G(X, c)$, for all c that belong to a (D, Z')-minimal cut.

Algorithm 3 Computing variable bounds in predicate

Input: $\langle G, D, X, Z', Z, x\rangle$ where G is a guarded predicate, $Z' \subseteq Z$ set of its guard variables, $x \in X$ a variable, $D(Z)$ is a conjunctive predicate over Z

Output: $\langle b, \inf, \sup\rangle$, where $b \in \{$B, ¬B, ¬F$\}$.
If b = B, $G(X, Z) \Rightarrow \inf \leq x \leq \sup$

```
function computeBounds($(G, D, X, Z', Z'', x)$)
$C \leftarrow \varnothing$, $r \leftarrow |Z'|$, $\inf \leftarrow +\infty$, $\sup \leftarrow -\infty$, $f \leftarrow$ FALSE
$r' \leftarrow$ optimalValue$(\min, \sum_{i \in [r]} z_i, D(Z))$
$r'' \leftarrow$ optimalValue$(\max, \sum_{i \in [r]} z_i, D(Z))$
for $k = r'$ to $r''$ do
    $end$ = TRUE
    for all $b \in \mathbb{B}_k^r$ do
        if $C \not\leq b$ then $end \leftarrow$ FALSE else continue
        if feasible$(D(Z, c))$ then $C \leftarrow C \cup \{b\}$ else continue
        if feasible$(G(X, b))$ then
            $f \leftarrow$ TRUE
            $M \leftarrow$ optimalValue(max, $x$, $G(X, b)$)
            $m \leftarrow$ optimalValue(min, $x$, $G(X, b)$)
            if $M = \infty$ or $m = \infty$ then return $\langle$¬B, _, _ $\rangle$
            sup $\leftarrow$ max(sup, $M$), inf $\leftarrow$ min(inf, $m$)
    if $end$ then break
if $f$ then return $\langle$B, inf, sup$\rangle$ else return $\langle$¬F, _, _$\rangle$
```

Proposition 8: Let $Q(X, Z) = G(X, Z') \wedge D(Z)$, where G is a guarded predicate such that guard variables in $Z' \subseteq Z$ occur positively and D is a conjunctive predicate. Let C be a (D, 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, Z')-minimal cut, any satisfying assignment (X^*, Z^*) to Q is such that $C \leq Z'^*$. As a consequence, there exists $c \in C$ such that $c \leq Z'^*$. Prop. 7.2 implies that $\max\{x \mid G(X, Z^*)\} \leq \max\{x \mid G(X, c)\}$ 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 8, function *computeBounds* (Alg. 3) 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 4–16 examines possible assignments to guard variables in Z, keeping the invariant $\forall b < C \neg \textit{feasible} G(X, b) \wedge \forall b \geq C \max\{x \mid G(X, Z)\} \leq \max\{x \mid G(X, b)\} \wedge \min\{x \mid G(X, Z^*)\} \geq \min\{x \mid G(X, b)\}$. In the loop in lines 6–14, if the assignment c under consideration is greater than an assignment in C, no further investigation are needed (by Prop. 8 x is bounded in $Q(X, c)$). If $D(Z \setminus Z', b)$ is unfeasible, the assignment c is not relevant, because $c \leq C$, for any (D, Z')-minimal cut C. Otherwise, c is a relevant assignment and it is added to C (line 8). If x is unbounded in $Q(X, c)$ (lines 11 and 13) 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 14). If for all assignments in $c \in \mathbb{B}_k^n$ we have $c \geq C$ ($\mathbb{B}_k^n$ is a cut) we are done, C is a (D, Z')-minimal cut, and $\inf$ and $\sup$ computed so far are over-approximation of x bounds in $Q(X, Z)$ (line 15).

Algorithm 4 From predicates to conjunctive predicates

```
Input: $P$ predicate over $X$
Output: $\langle b, C \rangle$, $b \in \{B, \neg B, \neg F\}$.
    If $b$ = B, then $C \simeq P$
function PtoC($(P, X)$)
1.  $\langle G, D, Z', Z'' \rangle \leftarrow$ PtoG$(P, X)$
2.  $D' \leftarrow$ GtoC$(D, Z' \cup Z'', \langle \mathbf{0}, \mathbf{1} \rangle)$
3.  for all $x \in X$ do
4.      $\langle \mu, m_x, M_x \rangle \leftarrow$ computeBounds$(G, D', X, Z', Z'', x)$
5.      if $\mu \neq$ BOUNDED then return $\langle \mu, \varnothing \rangle$
6.  return $\langle \mu,$ GtoC$(G, X \cup Z' \cup Z'', \langle m, M \rangle)$
```

Example 4: In Ex. 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 variables v_D and i_D, and by computing bounds for variables i'_L, v'_O, i_u, and v_u using Alg. 3.

The function *PtoC* in Alg. 4 presents the overall procedure that transforms a bounded predicate into an equisatisfiable conjunctive predicate. It calls functions in Algs. 1–3 and the function *GtoC* that performs predicate transformations given in the proof of Prop. 5. As a first step, Alg. 4 translates a predicate $P(X)$ into an equisatisfiable guarded predicate $G(X, Z') \wedge D(Z', Z'')$ by calling the function *PtoG*. 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 transformed into a conjunctive predicate D' by calling the function *GtoC* on D. To apply function *GtoC* on $G(X, Z')$, 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. If the function *computeBounds* finds that G' is unfeasible or some x is not bounded in G', the empty constraint is returned together with the failure explanation. Otherwise, the desired conjunctive predicate is returned.

VI. MODELING ISSUES

The disjunction elimination procedure given in Alg. 4 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. On the other hand, guarded predicates 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 5: Disjunctions 7–9 in Ex. 1 can be replaced by the conjunction of the following (guarded) constraints:

$$q \rightarrow v_D = 0 \quad (38) \qquad u \rightarrow v_u = 0 \quad (40) \qquad \bar{q} \rightarrow v_D = R_{\text{off}} i_D \quad (42)$$
$$q \rightarrow i_D \geq 0 \quad (39) \qquad \bar{q} \rightarrow v_D \leq 0 \quad (41) \qquad \bar{u} \rightarrow v_u = R_{\text{off}} i_u \quad (43)$$

The resulting model for the buck DC-DC converter is much more succinct than the guarded model in Ex. 2 and it has two guard variables only, rather than six as in Ex. 2.

Alg. 3 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 $\bar{z} \rightarrow C(X)$ can is equisatisfiable to the guarded predicate $(z' \rightarrow C(X)) \wedge (z' + z = 1)$. This transformation may double the number of guard variables and hence make the application of Alg. 3 less effective than an exhaustive algorithm on the original model with positive and negative guard literals (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. 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 $P(X) = B(x) \wedge P'(X)$, where $B(x) = x \leq b \wedge x \geq a$, for some constants a and b.

Proposition 9: 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 $x'_1 \geq L_1(X, U, Y) \wedge x'_1 \leq L_2(X, U, Y) \wedge N'(X, U, Y, X')$ for a suitable predicate 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 6: 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 9, $\mathcal{H}_1$ 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 $\mathcal{H}_2$ is not bounded.

VII. Experimental Results

In this section, we evaluate the effectiveness of our predicate transformation algorithm *PtoC*. We implemented Alg. 4 in C programming language, using GLPK to solve MILP problems. We present the experimental results obtained by using PtoC on a n-inputs buck DC-DC converter, that we model with three DTLHSs $\mathcal{H}_i = (X_i, U_i, Y_i, N_i)$, with $i \in [3]$, s.t. $X_1 = X_2 = X_3$, $U_1 = U_2 = U_3$, $Y_1 \subset Y_2 \subset Y_3$, N_1 is a *predicate* (Section II-A), N_2 and N_3 are *guarded predicates* (Section IV) and guards in N_3 are positive only.

We then run PtoC on $\mathcal{H}_i$ for increasing values of n (which entails that the number of guards increases), in order to show effectiveness of PtoC. Namely, in Section VII-A1 we show experimental results for the whole algorithm in Alg. 4. Furthermore, in Section VII-A2 we show that exploiting knowledge of the system and modeling it with guarded predicates we obtain better results than those in Section VII-A1. To this aim, we suppose that predicates G, D' and variables sets X, Z', Z'' in Alg. 4 may be directly given as an input to function *PtoC* (thus lines 1 and 2 in Alg. 4 are skipped).

Both in Section VII-A1 and VII-A2 we compare the computation time of function *PtoC* against function *PtoCexh*, which may be obtained from Alg. 4 by replacing the call to function *computeBounds* (our bottleneck here) in line 4 with the naïve algorithm which exhaustively checks all possible assignments to guard variables (see Section V). To this aim, also *PtoCexh* has been implemented inside PtoC. As for *PtoC*, also for *PtoCexh* it is possible to directly specify predicates G, D' and variables sets X, Z', Z''.

A. Multi-Input Buck DC-DC Converter

A Multi-Input Buck DC-DC converter [16] (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 Ex. 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 Ex. 3.

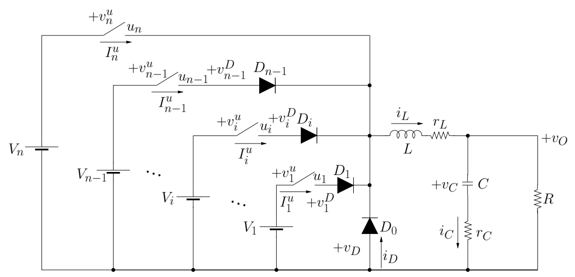

Figure 2. Multi-input Buck DC-DC converter

Table I
PtoC performances (predicates)

n	r	r'	r''	k	$\lvert cut\rvert$	CPU$_r$	CPU$_e$
2	12	6	12	11	64	1.48e+00	1.13e+02
3	18	9	18	17	512	8.33e+01	1.35e+04
4	24	12	24	23	4096	8.73e+03	>1.38e+06

1) 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 $Y_1 = v_D, v_1^D, \ldots, v_{n-1}^D, i_D, I_1^u, \ldots, I_n^u, v_1^u, \ldots, v_n^u$. From a simple circuit analysis (e.g., see [15]), we have that state variables constraints are the same as Eqs. (3) and (4) of the converter in Ex. 1. Analogously, as for the algebraic constraints, we have that Eq. (9) in Ex. 1 also holds for the n-inputs converter. In addition to Eqs. (3), (4) and (9) of Ex. 1, the Eqs. (45)–(48) below must hold.

$$\bigwedge_{i\in[n]}(u_i = 0) \vee (v_i^u = 0) \quad (44) \qquad \bigwedge_{i\in[n]}(u_i = 1) \vee (v_i^u = R_{\text{off}} I_i^u) \quad (45)$$

$$\bigwedge_{i\in[n-1]} ((I_i^u \geq 0) \wedge (v_i^D = 0)) \vee ((I_i^u \leq 0) \wedge (v_i^D = R_{\text{off}} I_i^u)) \quad (46)$$

$$i_L = i_D + \sum_{i=1}^{n} I_i^u \quad (47) \qquad \bigwedge_{i\in[n-1]} v_D = v_i^u + v_i^D - V_i \wedge v_D = v_n^u - V_n \quad (48)$$

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

We call function *PtoC* with parameters $N_1, X_1 \cup U_1 \cup Y_1$ for increasing values of n, and we compare its computation time with that of function *PtoCexh*. Table I shows our experimental results. In Table I, column n shows the number of buck inputs, column r shows the number of guards (see line 1 of Alg. 3), columns r', r'' have the meaning given in lines 2 and 3 of Alg. 3, column k gives the value of k at the end of the for loop of Alg. 3, column $|cut|$ gives the size of cut at the end of the for loop of Alg. 3, and column CPU$_r$ (resp. CPU$_e$) shows the computation time in seconds of function function *PtoC* (resp. *PtoCexh*). Table I shows that heuristics implemented in function *computeBounds* greatly speeds-up variable bounds computation.

2) Multi-Input Buck as a Guarded Predicate: We modify the DTLHS $\mathcal{H}_1$ of Section VII-A1 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 Eqs. (9) and (45)–(48) with Eqs. (38)–(43) (where $q = q_0$, see Section VI), and by adding the following ones ($i \in [n-1]$):

Table II
PtoC performances (guarded predicates)

n	r	r'	r''	k	$\|cut\|$	CPU$_r$	CPU$_e$
4	16	8	8	8	256	1.17e+01	1.24e+01
5	20	10	10	10	1024	1.55e+02	6.93e+01
6	24	12	12	12	4096	2.65e+03	3.78e+02

$$q_i \to v_i^D = 0 \quad (49) \qquad u_i \to v_i^u = 0 \quad (51) \qquad \bar{q}_i \to v_i^D = R_{\text{off}} I_i^u \quad (53)$$
$$q_i \to I_i^u \geq 0 \quad (50) \qquad \bar{q}_i \to v_i^D \leq 0 \quad (52) \qquad \bar{u}_i \to v_i^u = R_{\text{off}} I_i^u \quad (54)$$

Finally, we define $\mathcal{H}_3 = (X_3, U_3, Y_3, N_3)$, where $X_3 = X_2 = X_1$, $U_3 = U_2 = U_1$ and N_3 is obtained from N_2 by eliminating negative guards as described in Section VI. This introduces $2n$ additional auxiliary variables to manage negations of $q_0, \ldots, q_{n-1}, u_1, \ldots, u_n$, thus $Y_3 = Y_2 \cup Y_3' = Y_2 \cup \{q_0', \ldots, q_{n-1}', u_1', \ldots, u_n'\}$.

For $i = 2, 3$, let constraints in N_i be partitioned in G_i and D_i s.t. G_i contains all guarded constraints in N_i. We call function *PtoC* with parameters $G_3, D_3, X_3 \cup U_3 \cup Y_1, Y_2' \cup Y_3', \varnothing$ for increasing values of n, and we compare its computation time with that of function *PtoCexh* with parameters $G_2, D_2, X_2 \cup U_2 \cup Y_1, Y_2', \varnothing$ Note that G_3 only contains positive-guarded constraints, thus it is possible to call function *PtoC* on it. On the other hand, G_2 also contains negative-guarded constraints, thus it cannot be passed to function *PtoC*, whilst it can be managed by function *PtoCexh*.

Table II shows our experimental results. Columns meaning in Table II are the same as of Table I. Predicate translation on the multi-input buck dc-dc model given as guarded predicate is much faster due to a smaller number of auxiliary variables (and constraints). The negative impact of auxiliary boolean variables is clearly showed by the fact that function *PtoCexh*, much slower than function *PtoC* on a model of the same size, performs better than *PtoC* in this case, because it can work on a model with half of the variables. This phenomenon would be greatly amplified in a verification or control software synthesis procedure. These results strongly support guarded predicates as modeling language.

VIII. Conclusions

The results presented in this paper contribute to Model Based Design of embedded software by proposing an expressive modelling language for discrete time linear hybrid systems. Indeed, MILP based abstraction of a DTLHS have been used to synthesize correct-by-construction control software that implements a quantized controller. They require DTLHS dynamics modeled as a conjunctive predicate over state, input, and next state variables.

In this paper, we circumvented such a limitation, by giving an automatic procedure that transforms any predicate into an equisatisfiable conjunctive predicate, provided that each variable ranges over a bounded interval. Moreover, we have presented an 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 algorithms. Most notably, they show that guarded predicates may turn out to be a natural language to describe succinctly DTLHS dynamics.

References

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

[2] 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.

[3] 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.

[4] A. Bemporad and M. Morari, "Verification of hybrid systems via mathematical programming," in *HSCC*, 1999

[5] F. Mari and E. Tronci, "CEGAR based bounded model checking of discrete time hybrid systems," in *HSCC*, 2007

[6] 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.

[7] 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.

[8] 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.

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

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

[11] A. Kobetski and M. Fabian, "Scheduling of discrete event systems using mixed integer linear programming," in *Discrete Event Systems*, 2006

[12] 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.

[13] 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.

[14] 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 Pow. El.*, 23(4), 2008

[15] 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.

[16] 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 Pow. El.*, 25(2), 2010