Hybrid Diagrams: A Deductive-Algorithmic Approach to Hybrid System Verification*

Luca de Alfaro Arjun Kapur Zohar Manna

Department of Computer Science
Stanford University

Abstract. We present a methodology for the verification of temporal properties of hybrid systems. The methodology is based on the deductive transformation of *hybrid diagrams*, which represent the system and its properties, and which can be algorithmically checked against the specification. This check either gives a positive answer to the verification problem, or provides guidance for the further transformation of the diagrams. The resulting methodology is complete for quantifier-free linear-time temporal logic.

1 Introduction

Specification and verification methodologies for hybrid systems range from algorithmic methods for the verification of linear-time temporal logic properties [2, 1], to deductive approaches for proving linear-time temporal logic properties [11, 7] and interval-based and duration properties [5, 3]. In this paper we present an approach that combines deductive and algorithmic methods into a methodology that is complete (relative to first-order reasoning) for proving linear-time temporal logic properties of hybrid systems, provided no temporal operator appears in the scope of a quantifier. The advantages of the proposed methodology over the rule-based approach of [11, 6] include the visual representation of the proof process, the provision of proof guidance, and the ability to prove specifications expressed by temporal formulas not in canonical form [10].

Hybrid diagrams are related to the *fairness diagrams* of [4] and to the *hybrid automata* of [2, 1]. They consist of a graph whose vertices are labeled by assertions and whose edges are labeled by transition relations; associated with each diagram are *fairness constraints*, that encode acceptance conditions similar to those of ω-automata. The diagrams represent the system behavior and the safety and progress properties that have been proved about it: the vertex and edge labels represent the safety properties, the fairness constraints represent the progress properties. Hybrid diagrams are sufficiently expressive to encode *phase transition systems* (PTSs) [9, 6], which will be the system model adopted in this paper.

The construction of the proof of a temporal specification begins by representing the system as a one-vertex diagram, whose single edge encodes the possible

* The research was supported in part by the National Science Foundation under grant CCR-9527927, by the Defense Advanced Research Projects Agency under contract NAG2-892, by ARO under grant DAAH04-95-1-0317, and by ARO under the MURI grant DAAH04-96-1-0341.

state transitions of the system. This initial diagram can be transformed using a set of rules that preserve the inclusion of system behaviors, producing a chain of diagram transformations. The aim of this process is to obtain a diagram that can be shown to satisfy the specification by purely algorithmic means.

After any number of transformations, an algorithmic procedure can applied to the last diagram, to either establish that the final diagram (and, by behavior inclusion, the original PTS) satisfies the specification, or it returns a set of *candidate counterexample paths* (CCP) in the diagram. The CCPs provide guidance for the extension of the chain of transformations, following the insights of [13]. Additionally, the CCPs can be used to guide the search for counterexamples, by directing the simulation of the original system along the CCPs.

There are four rules to transform diagrams. The *simulation rule* modifies the graph structure of the diagram, enabling the study of safety properties [4]. The *justice* and *compassion rules* prove progress properties of the diagrams, and represent them as additional fairness constraints. The *pruning rule* eliminates portions of the diagram that are never traversed by any computation along which time diverges. These rules generate first-order verification conditions that must be proved to justify the transformation. The justice and compassion rules are one of the main contributions of this paper, and are at the basis of the completeness results of the methodology. By relying on *ranking* and *delay* functions to measure progress towards given goals, the rules enable the proof of justice and compassion properties of the systems; these properties are then represented as fairness constraints which are added to the diagrams.

2 Phase Transition Systems

The hybrid system model we adopt in this paper is that of *phase transition systems* (PTS) [9, 6]. A PTS is a transition system that allows continuous state changes over time periods of positive duration, as well as discrete state changes in zero time. A PTS $\mathcal{S} = (\mathcal{V}, \theta, \mathcal{T}, \Pi, \mathcal{A})$ consists of the following components.

1. A set $\mathcal{V}$ of typed state variables, partitioned into the set $\mathcal{V}_d$ of *discrete variables*, the set $\mathcal{V}_c$ of *clock variables*, and the set $\mathcal{V}_h$ of *hybrid variables*. Clock variables have type $\mathbb{R}^+$ (i.e. the set of non-negative real numbers) and hybrid variables have type $\mathbb{R}$. We distinguish a special clock variable $T \in \mathcal{V}_c$, representing a *master clock* that measures the amount of time elapsed during the system behavior. The state space S consists of all type-consistent interpretations of the variables in $\mathcal{V}$; we denote by $s[\![x]\!]$ the value at state $s \in S$ of variable $x \in \mathcal{V}$.
2. An assertion θ over $\mathcal{V}$, which defines the set $\{s \in S \mid s \models \theta\}$ of initial states.
3. A finite set $\mathcal{T}$ of transition assertions over $\mathcal{V}$, $\mathcal{V}'$ representing the discrete state changes. Each assertion $\pi \in \mathcal{T}$ represents the transition relation $\{(s,t) \mid (s,t) \models \pi\}$, where (s,t) interprets $x \in \mathcal{V}$ as $s[\![x]\!]$ and $x' \in \mathcal{V}'$ as $t[\![x]\!]$. For all $\pi \in \mathcal{T}$, we require that the implication $\pi \rightarrow T = T'$ holds.
4. A *time-progress* assertion Π over $\mathcal{V}$, used to specify a restriction on the progress of time (see [6] for a discussion of its use).
5. A finite set $\mathcal{A}$ of *activities* representing the continuous state changes. Each activity $a \in \mathcal{A}$ consists of an *enabling assertion* C_a over $\mathcal{V}_d$ and of an evolution function $F_a : S \times \mathbb{R} \mapsto S$. At every $s \in S$ there must be exactly one

$a \in \mathcal{A}$ such that $s \models C_a$. If at time t the system is at a state $s \models C_a$, at time $t + \Delta$ the system will be at state $F_a(s, \Delta)$. For every $a \in \mathcal{A}$, the function F_a must satisfy the equations

$$\forall x \in \mathcal{V}_d \,.\, F_a(s,t)[\![x]\!] = s[\![x]\!] \qquad F_a(s,0) = s$$
$$\forall x \in \mathcal{V}_c \,.\, F_a(s,t)[\![x]\!] = s[\![x]\!] + t \qquad F_a(s,t) = F_a(F_a(s,t'), t - t')$$

for every $s \models C_a$, $t \geq 0$ and $0 \leq t' \leq t$. The function F_a is represented by the set of terms $\{F_a^x\}_{x \in \mathcal{V}}$ over $\mathcal{V} \cup \{\Delta\}$, where the term F_a^x gives the temporal evolution of the value of x as a function of the elapsed time Δ.

To define the set of computations of a PTS, we introduce the assertions $\{tick_a[\Delta]\}_{a \in \mathcal{A}}$, where each $tick_a[\Delta]$ is an assertion over $\mathcal{V} \cup \mathcal{V}'$ and over the parameter Δ, whose domain is the set $\mathbb{R}^+$ of non-negative real numbers. Assertion $tick_a[\Delta]$ describes a state change of the system due to activity a when an amount of time $\Delta \geq 0$ elapses, and is given by:

$$C_a \wedge \Big(\bigwedge_{x \in \mathcal{V}} (x' = F_a^x[\Delta]) \Big) \wedge \forall t \,.\, \big(0 \leq t < \Delta \rightarrow \Pi\big[F_a^x[t]/x\big]_{x \in \mathcal{V}}\big) \,.$$

In the above formula, $\Pi[F_a^x[t]/x]_{x \in \mathcal{V}}$ denotes the result of simultaneously replacing for all $x \in \mathcal{V}$ each occurrence of x in Π with $F_a^x[t]$. The form of the assertion $tick_a[\Delta]$ insures that the progress constraint Π holds at every moment of a time-step, except possibly for the final one. As discussed in [6], if Π is used only to encode upper bounds on the transition waiting times, assertion $tick_a[\Delta]$ can be rewritten without quantifiers.

Definition 1 (PTS computations). A *computation* of a PTS $\mathcal{S} = (\mathcal{V}, \theta, \mathcal{T}, \Pi, \mathcal{A})$ is an infinite sequence $\sigma : s_0, s_1, s_2, \ldots$ of states of S that satisfies the following conditions.

1. *Initiality:* $s_0 \models \theta$.
2. *Consecution:* for each $i \geq 0$, either there is a transition $\pi \in \mathcal{T}$ such that $(s_i, s_{i+1}) \models \pi$, or there is an activity $a \in \mathcal{A}$ such that $(s_i, s_{i+1}) \models \exists \Delta \geq 0 \,.\, tick_a[\Delta]$.
3. *Time progress:* for each $t \in \mathbb{R}$ there is $i \in \mathbb{N}$ such that $s_i(T) \geq t$.

We denote by $\mathcal{L}(\mathcal{S})$ the set of computations of a PTS $\mathcal{S}$. □

A Room-Heater Example

As our running example throughout the paper, we consider a variant of the temperature control system presented in [2]. The system, which we call RH, consists of a room with a window and a heater. The window, controlled by some independent agent, may be opened or closed at will. The heater turns on when the temperature is below the threshold temperature of 68°F and turns off when the temperature is above the threshold temperature of 72°F. To prevent mechanical stress, the heater has an embedded clock that prevents it from changing state within 60 seconds of the last change. Initially, the room temperature is below 60°F and the environment temperature (i.e. the temperature outside the room) is 60°F. For simplicity, we assume that the temperature of the environment remains constant at 60°F. Our phase transition system $\mathcal{S} = (\mathcal{V}, \theta, \mathcal{T}, \Pi, \mathcal{A})$, is defined as follows.

1. $\mathcal{V}_d = \{H, W\}$, where H denotes the state of the heater and ranges over domain $\{On, Off\}$, and W denotes the state of the window and ranges over domain $\{Open, Closed\}$. $\mathcal{V}_c = \{T, y\}$, where T is the global clock, and y measures the time elapsed since the last switching *On*/*Off* of the heater. $\mathcal{V}_h = \{x\}$, where x is the temperature of the room.
2. $\theta : H = \mathit{Off} \ \wedge \ W = \mathit{Closed} \ \wedge \ x < 60 \ \wedge \ y = 0 \ \wedge \ T = 0$.
3. $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, where $\tau_i : E_i \wedge \rho_i$ for $i \in \{1, 2, 3\}$, and

$$\begin{array}{ll} E_1 : H = \mathit{Off} \ \wedge \ x \leq 68 \ \wedge \ y \geq 60 & \rho_1 : H' = \mathit{On} \ \wedge \ y' = 0 \\ E_2 : H = \mathit{On} \ \wedge \ x \geq 72 \ \wedge \ y \geq 60 & \rho_2 : H' = \mathit{Off} \ \wedge \ y' = 0 \\ E_3 : \mathit{true} & \rho_3 : W' = \neg W \end{array}$$

   where $\neg \mathit{Open} = \mathit{Closed}$ and $\neg \mathit{Closed} = \mathit{Open}$. Variables not mentioned in ρ_1, ρ_2, and ρ_3, respectively, are left unchanged by the transitions.
4. $\Pi = \neg E_1 \wedge \neg E_2$. This insures that τ_1 and τ_2 are taken as soon as they become enabled.
5. $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$, where $F^T_{a_i} = T + \Delta$, $F^y_{a_i} = y + \Delta$, for every $i \in \{1, 2, 3, 4\}$, and C_{a_i} and $F^x_{a_i}$ are defined as follows:

$$\begin{array}{ll} C_{a_1} : H = \mathit{Off} \ \wedge \ W = \mathit{Closed} & F^x_{a_1} = 60 + e^{-\Delta/105}(x - 60) \\ C_{a_2} : H = \mathit{Off} \ \wedge \ W = \mathit{Open} & F^x_{a_2} = 60 + e^{-\Delta/70}(x - 60) \\ C_{a_3} : H = \mathit{On} \ \wedge \ W = \mathit{Closed} & F^x_{a_3} = 75 + e^{-\Delta/105}(x - 75) \\ C_{a_4} : H = \mathit{On} \ \wedge \ W = \mathit{Open} & F^x_{a_4} = 70 + e^{-\Delta/70}(x - 70) \ . \end{array}$$

The properties we wish to prove about *RH* state that the room temperature eventually reaches the range from 65°F to 75°F, and that once the temperature is in this range, it will remain in this range forever.

3 Hybrid Diagrams

To study the temporal behavior of a PTS, we introduce *hybrid diagrams*, derived from the *fairness diagrams* of [4]. A *hybrid diagram* (diagram, for short) $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ consists of the following components.

1. A set $\mathcal{V}$ of typed state variables that includes the master clock T.
2. A set V of *vertices*.
3. A labeling ρ that assigns to each vertex $v \in V$ an assertion $\rho(v)$ over $\mathcal{V}$. A *location* of a diagram is a pair $(v, s) : v \in V, s \models \rho(v)$ composed of a vertex and of a corresponding state, and represents an instantaneous configuration of the diagram.
4. A labeling θ that assigns to each vertex $v \in V$ an initial assertion $\theta(v)$ over $\mathcal{V}$. This labeling defines the set of initial locations $\{(v, s) \mid v \in V, s \models \theta(v)\}$. For all $v \in V$, we require that $\theta(v) \rightarrow (\rho(v) \wedge T = 0)$.
5. A labeling τ that assigns to each edge $(u, v) \in V \times V$ a transition assertion $\tau(u, v)$ over $\mathcal{V} \cup \mathcal{V}'$ and Δ. For $u, v \in V$, assertion $\tau(u, v)$ represents the possible state changes of the system when going from vertex u to vertex v by a time-step of duration $\Delta \in \mathbb{R}^+$. We require that the assertion $\tau(u, v) \rightarrow T' = T + \Delta$ holds for all $u, v \in V$.
6. A set $\mathcal{J}$ of *justice constraints*, and a set $\mathcal{C}$ of *compassion constraints*. The elements of $\mathcal{J}$ and $\mathcal{C}$ are pairs $(R, G) : R \subseteq V, G \subseteq V^2$.

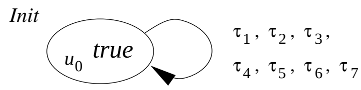

Fig. 1. Hybrid Diagram A_0.

Given an assertion ϕ over $\mathcal{V}$, we denote by ϕ' the formula obtained by replacing each free $x \in \mathcal{V}$ by $x' \in \mathcal{V}'$. Using this notation, we require that a diagram satisfies the requirement $\rho(u) \wedge \tau(u, v) \rightarrow \rho'(v)$, for all $u, v \in V$.

The justice and compassion constraints, collectively called fairness constraints, represent fairness properties that have been proved about the system. For a constraint (R, G), the set $R \subseteq V$ specifies a *request* region; the request is *gratified* when a transition from a vertex u to a vertex v is taken, with $(u, v) \in G$. A just constraint indicates that a request that is performed without interruptions will eventually lead to gratification; a compassionate constraint indicates that a request performed infinitely often will be gratified infinitely often [11, 4].

Definition 2 (diagram computations). A *run* of a diagram is an infinite sequence of locations (v_0, s_0), (v_1, s_1), (v_2, s_2), $\ldots$, satisfying the following conditions.

1. *Initiality:* $s_0 \models \theta(v_0)$.
2. *Consecution:* for all $i \geq 0$, $(s_i, s_{i+1}) \models \exists \Delta \,.\, \tau(v_i, v_{i+1})$.
3. *Time progress:* for each $t \in \mathbb{R}$ there is $i \in \mathbb{N}$ such that $s_i(T) \geq t$.
4. *Justice:* for each constraint $(R, G) \in \mathcal{J}$, if there is $k \in \mathbb{N}$ such that $v_i \in R$ for all $i \geq k$, then there is $j \geq k$ such that $(v_j, v_{j+1}) \in G$.
5. *Compassion:* for each constraint $(R, G) \in \mathcal{C}$, if $v_i \in R$ for infinitely many $i \in \mathbb{N}$, then there are infinitely many $j \in \mathbb{N}$ such that $(v_j, v_{j+1}) \in G$.

If $\sigma : (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots$ is a run of A, the sequence of states $s_0, s_1, s_2, \ldots$ is a *computation* of A. We denote by $\mathit{Runs}(A)$ and $\mathcal{L}(A)$ the sets of runs and computations of A, respectively. □

Every PTS can be represented by a one-vertex diagram, as the following construction shows.

Construction 3. Given a PTS $\mathcal{S} = (\mathcal{V}, \theta, \mathcal{T}, \Pi, \mathcal{A})$, we define the diagram $\mathit{hd}(\mathcal{S}) = (\mathcal{V}, V, \rho, \theta', \tau', \mathcal{J}, \mathcal{C})$ by $V = \{v_0\}$, $\rho(v_0) = \mathit{true}$, $\theta'(v_0) = \theta$, $\mathcal{J} = \emptyset$, $\mathcal{C} = \emptyset$, and

$$\tau'(v_0, v_0) = \Big(\bigvee_{\pi \in \mathcal{T}} (\pi \wedge \Delta = 0)\Big) \vee \Big(\bigvee_{a \in \mathcal{A}} \mathit{tick}_a[\Delta]\Big) . \qquad \square$$

Theorem 4. *For a PTS $\mathcal{S}$, $\mathcal{L}(\mathcal{S}) = \mathcal{L}(\mathit{hd}(\mathcal{S}))$.*

Example 5. In Figure 1, we present the initial diagram $A_0 = \mathit{hd}(RH)$ corresponding to system RH. The transitions τ_1, τ_2, and τ_3 are as in RH (with the added conjunct $\Delta = 0$), and transitions τ_4, τ_5, τ_6, and τ_7 are $\mathit{tick}_{a_1}[\Delta]$, $\mathit{tick}_{a_2}[\Delta]$, $\mathit{tick}_{a_3}[\Delta]$, and $\mathit{tick}_{a_4}[\Delta]$, respectively. The single node u_0 is marked *Init* as a reminder that its initial label $\theta(u_0)$ is equal to the initial condition of the PTS. □

Hybrid diagrams vs. hybrid automata. Hybrid diagrams are related to *hybrid automata*, a formalism widely adopted for the modeling of hybrid systems and for the study of their temporal properties [2, 1]. While sharing a similar labeled-graph structure, the two formalisms differ in some respects.

In a hybrid automaton, the dynamical behavior of the system and the discrete state-transitions are described by different components: the first by differential equations labeling the vertices, the second by transition relations labeling the edges. A hybrid diagram describes both types of evolution using the edge labels: the assertions labeling the vertices represent instead inductive invariants. Moreover, hybrid automata use vertex labels to limit the amount of time for which the system can stay continuously at a vertex. In a hybrid diagram, this role is carried out by the edge labels, which can limit the duration Δ of a time-step.

These differences are motivated by the purposes hybrid automata and hybrid diagrams serve. Hybrid automata were proposed as a formal model of hybrid systems, to which various formal verification methods could be applied. Hybrid diagrams, on the other hand, are meant to provide a deductive representation of a hybrid system and of the safety and progress properties that have been proved about it, and are suited to the application of the diagram transformation rules that will be presented next.

4 Diagram Transformation Rules

The temporal properties of a PTS are studied by means of *transformation rules* [4]. There are four rules: the *simulation rule*, used to study safety properties; the *justice* and *compassion rules*, used to study progress properties; and the *pruning rule*, used to prune portions of a diagram that are never traversed by runs along which time diverges. If a diagram A can be transformed into a diagram B by one of these rules, we write $A \Rightarrow B$, and we indicate by $\stackrel{*}{\Rightarrow}$ the reflexive transitive closure of $\Rightarrow$. The rules preserve language containment: $A \Rightarrow B$ implies $\mathcal{L}(A) \subseteq \mathcal{L}(B)$. Given a PTS $\mathcal{S}$, the rules are used to construct a chain of transformations $hd(\mathcal{S}) \equiv A_0 \Rightarrow A_1 \Rightarrow \cdots \Rightarrow A_n$. At any time, it is possible to check algorithmically whether the last diagram of the chain comply with the specification. This test, discussed in the next section, provides a sufficient condition for the diagram to satisfy the specification, and returns either a positive answer to the verification problem, or guidance for the extension of the chain of transformations.

4.1 Simulation Rule

The *simulation rule*, derived from [4], enables the transformation of a diagram into a new one, such that the second diagram is capable of simulating the first one. A simulation relation between two diagrams A_1 and A_2 is induced by a function $\mu : V_1 \mapsto 2^{V_2}$ from the vertices of A_1 to those of A_2.

Rule 6 (simulation). Let $A_1 = (\mathcal{V}, V_1, \rho_1, \theta_1, \tau_1, \mathcal{J}_1, \mathcal{C}_1)$, $A_2 = (\mathcal{V}, V_2, \rho_2, \theta_2, \tau_2, \mathcal{J}_2, \mathcal{C}_2)$ be two diagrams sharing the same variables. If there is a function $\mu : V_1 \mapsto 2^{V_2}$ that satisfies the conditions below, then $A_1 \Rightarrow A_2$.

1. For all $u \in V_1$, $\theta_1(u) \to \bigvee_{v \in \mu(u)} \theta_2(v)$.

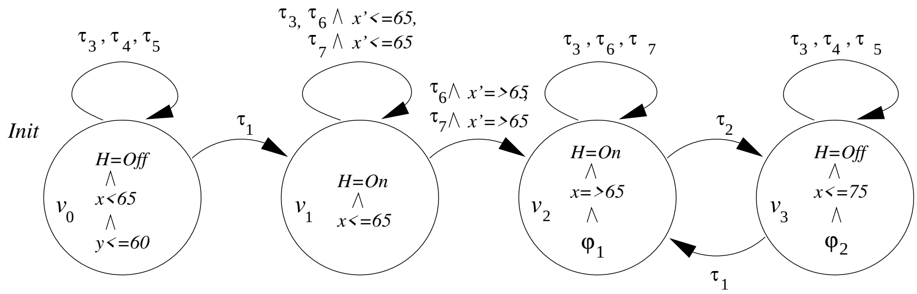

Fig. 2. Hybrid Diagram A_1, where $\varphi_1 : x \leq 75-7\cdot e^{-y/105}$ and $\varphi_2 : x \geq 60+12\cdot e^{-y/70}$. Edges labeled with *false* are not shown.

2. For all $u, u' \in V_1$ and $v \in \mu(u)$,
$$\big(\rho_1(u) \wedge \rho_2(v) \wedge \tau_1(u,u')\big) \rightarrow \bigvee_{v' \in \mu(u')} \tau_2(v,v') \ .$$
3. For each $(R_2, G_2) \in \mathcal{J}_2$ (resp. $\in \mathcal{C}_2$) there is $(R_1, G_1) \in \mathcal{J}_1$ (resp. $\in \mathcal{C}_1$) such that:
   (a) for all $u \in V_1$, if $\mu(u) \cap R_2 \neq \emptyset$ then $u \in R_1$;
   (b) for all $(u,u') \in G_1$ and $v \in \mu(u)$,
   $$\big(\rho_1(u) \wedge \rho_2(v) \wedge \tau_1(u,u')\big) \rightarrow \bigvee_{v' \in H(u',v)} \tau_2(v,v') \ ,$$
   where $H(u',v) = \{v' \mid v' \in \mu(u') \wedge (v,v') \in G_2\}$. □

Theorem 7 (soundness of Rule 6). *If $A_1 \Rightarrow A_2$ by Rule 6, then $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$.*

Example 8. By applying the simulation rule to the diagram A_0 of Figure 1, we obtain the diagram A_1 presented in Figure 2. The application of the rule is based on the function μ defined by $\mu(u_0) = \{v_0, v_1, v_2, v_3\}$. In Figure 2, v_0 is the only vertex satisfying the initial condition specified by θ. □

4.2 Progress Rules

The *justice* and *compassion rules* add new constraints to the justice or compassion sets of a diagram, respectively. Since the rules must preserve language containment, it is possible to add a constraint only if all runs of the diagram already obey it, implying that the constraint represents a progress property of the runs of the diagram. To prove that all runs obey the constraint, the rules rely on *ranking* and *delay* functions to measure progress towards its gratification. The delay functions are similar to the mappings of [8]; our results indicate that to achieve completeness they need to be used in conjunction with ranking functions.

Definition 9 (ranking and delay functions). Recall that *well-founded domain* is a set D together with a relation $>$, such that there is no infinite descending chain $d_0 > d_1 > d_2 > \cdots$ of elements in D.

Given a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$, let $loc(A) = \{(v, s) \in V \times S \mid s \models \rho(v)\}$ denote the set of locations of A. A *ranking function* $\delta : loc(A) \mapsto D$ for a diagram A is a function mapping locations of A into elements of a well-founded domain D. A *delay function* $\gamma : loc(A) \mapsto \mathbb{R}^+$ is a function mapping locations of A into non-negative real numbers. The ranking and delay functions δ, γ are represented by the families $\{\delta(u)\}_{u \in V}$, $\{\gamma(u)\}_{u \in V}$ of terms on $\mathcal{V}$. □

To add a constraint (R, G), the justice rule relies on ranking and delay functions δ, γ. While in R, δ cannot increase unless an edge in G is taken, and γ gives an upper bound to the amount of time before either an edge in G is taken or R is left.

Rule 10 (justice). Consider a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ and a constraint $(R, G) : R \subseteq V, G \subseteq V^2$. Assume that there are ranking and delay functions δ, γ such that, for all $u, v \in R$ with $(u, v) \notin G$, the assertion

$$\rho(u) \wedge \tau(u, v) \quad \rightarrow \quad \delta(u) > \delta'(v) \vee \big(\delta(u) = \delta'(v) \wedge \gamma(u) \geq \gamma'(v) + \Delta\big)$$

holds. Then, $A \Rightarrow A'$, where $A' = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J} \cup \{(R, G)\}, \mathcal{C})$. □

The rule to add compassion constraints is more involved, and requires the use of a family of assertions $\{\phi(v)\}_{v \in V}$, used to represent a set of locations $\{(v, s) \in loc(A) \mid s \models \phi(v)\}$ that plays the same role of R in leading to the goal.

Rule 11 (compassion). Given a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ and a constraint $(R, G) : R \subseteq V, G \subseteq V^2$, assume that there are

1. a family of assertions $\{\phi(v)\}_{v \in V}$ over $\mathcal{V}$, such that $\phi(v) = \mathit{true}$ for all $v \in R$;
2. a ranking function δ and a delay function γ,

such that, for every $u, v \in V$ with $(u, v) \notin G$, the assertions

$$\begin{aligned}
\rho(u) \wedge \tau(u, v) \quad &\rightarrow \quad \delta(u) \geq \delta'(v) \\
\rho(u) \wedge \phi(u) \wedge \tau(u, v) \quad &\rightarrow \quad \delta(u) > \delta'(v) \vee \neg\phi'(v) \vee \gamma(u) \geq \gamma'(v) + \Delta \\
\rho(u) \wedge \neg\phi(u) \wedge \tau(u, v) \quad &\rightarrow \quad \delta(u) > \delta'(v) \vee \neg\phi'(v)
\end{aligned}$$

hold. Then, $A \Rightarrow A'$, where $A' = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C} \cup \{(R, G)\})$. □

Theorem 12 (soundness of Rules 10 and 11). *If a constraint (R, G) is added to a diagram A using Rules 10 or 11, obtaining diagram A', then $Runs(A) = Runs(A')$, and therefore $\mathcal{L}(A) = \mathcal{L}(A')$.*

Example 13. To show that the temperature eventually reaches the desired range, we apply Rule 10 to the diagram A_1 of Figure 2, adding the justice constraint $(\{v_0, v_1\}, \{(v_1, v_2)\})$; we denote by A_2 the resulting diagram. This constraint shows that a run of A_1 cannot stay forever in v_0 or v_1, and must eventually proceed to v_2. The rule uses a ranking function defined by $\delta(v_0) = 1$, $\delta(v_1) = \delta(v_2) = \delta(v_3) = 0$, and a delay function given by $\gamma(v_0) = 60 - y$, $\gamma(v_1) = \mathit{if}\ x \leq 60\ \mathit{then}\ 175 + 105 \ln((75 - x)/49)\ \mathit{else}\ 150 + 70 \ln((70 - x)/10)$, and $\gamma(v_2) = \gamma(v_3) = 0$. □

4.3 Pruning Rule

The *pruning rule* prunes from a diagram a subset of vertices that, because of the presence of a justice constraint, cannot appear in any run of the system.

Rule 14 (pruning). Let $A_1 = (\mathcal{V}, V_1, \rho_1, \theta_1, \tau_1, \mathcal{J}_1, \mathcal{C}_1)$ be a diagram, and let $U_1 \subseteq V_1$ be a subset of its vertices such that the following two conditions hold:

1. there is $(R_1, G_1) \in \mathcal{J}_1$ such that $U_1 \subseteq R_1$, $(U_1 \times V_1) \cap G_1 = \emptyset$;
2. for all $u \in U_1$ and $v \in V_1 - U_1$, $\tau_1(u, v) = \mathit{false}$.

Then, $A_1 \Rightarrow A_2$, where $A_2 = (\mathcal{V}, V_2, \rho_2, \theta_2, \tau_2, \mathcal{J}_2, \mathcal{P}_2)$ is obtained as follows:

1. $V_2 = V_1 - U_1$;
2. ρ_2, θ_2, τ_2 are obtained by restricting the domain of ρ_1, θ_1, and τ_1 to V_2, V_2, and $V_2 \times V_2$, respectively;
3. for each constraint $(R, G) \in \mathcal{J}_1$ (resp. $\in \mathcal{C}_1$), we insert the constraint $(R \cap V_2, G \cap (V_2 \times V_2))$ into $\mathcal{J}_2$ (resp. into $\mathcal{C}_2$). $\square$

This rule can be used in conjunction with Rule 10 to prune from the diagram vertices reached only by invalid runs along which time does not diverge. The soundness of the rule follows from the observation that, if the conditions of the rule are satisfied, no run of the diagram can contain vertices in U. In fact, if a run entered U, it would not be able to leave it, and by staying forever in U it would violate at least one justice constraint of the diagram.

4.4 Completeness Results

Given a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$, we say that A is *deterministic* if $\theta(v) \wedge \theta(w) \leftrightarrow \mathit{false}$ and $\tau(u, v) \wedge \tau(u, w) \leftrightarrow \mathit{false}$ for all $u, v, w \in V$ with $v \neq w$. The following theorem holds.

Theorem 15. *If the set of computations of a PTS $\mathcal{S}$ is a subset of the set of computations of a deterministic diagram A, we can construct a chain of diagram transformations $hd(\mathcal{S}) \stackrel{*}{\Rightarrow} A$ using Rules 6, 10, 11, and 14.*

This completeness result is relative to first-order reasoning, and is proved by giving the construction of the chain of transformations $hd(\mathcal{S}) \stackrel{*}{\Rightarrow} A$ under the assumption $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(A)$. The proof, which uses ideas from [10, 4], has been omitted due to space constraints.

5 Proving Temporal Properties

In this section we present an algorithm to check whether a diagram satisfies a specification written in the linear-time temporal logic TL_s. The formulas of TL_s are obtained by combining first-order logic formulas by means of the future temporal operators $\bigcirc$ (next), $\square$ (always), $\diamond$ (eventually), $\mathcal{U}$ (until), and the corresponding past ones $\ominus$, $\boxminus$, $\diamondminus$ and $\mathcal{S}$ [11]. Given a diagram A and a formula $\phi \in TL_s$, the algorithm provides either a positive answer to $A \models \phi$, or information about the region of the diagram that can contain a counterexample to ϕ. This information can be used as guidance for the extension of the chain of transformations. The first step of the algorithm consists in constructing a Streett automaton $N_{\neg\phi}$ that accepts all the computations that do not satisfy ϕ. The automaton is a first-order version of a classical Streett automaton [12].

Definition 16 (Streett automaton). A (first-order) *Streett automaton* N consists of the components $(\mathcal{V}, (V, E), \rho, Q, \mathcal{B})$, where $\mathcal{V}$, ρ are as in hybrid diagrams; (V, E) is a directed graph with set of vertices V and set of edges $E \subseteq V^2$; $Q \subseteq V$ is the set of *initial vertices*, and $\mathcal{B}$, called the *acceptance list*, is a set of pairs $(P, R) : P, R \subseteq V$. A *run* σ of N is an infinite sequence of locations $(v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots$ such that $v_0 \in Q$, and:

1. for all $i \geq 0$, $s_i \models \rho(v_i)$ and $(v_i, v_{i+1}) \in E$;
2. for each pair $(P, R) \in \mathcal{B}$, either $v_i \in R$ for infinitely many $i \in \mathbb{N}$, or there is $k \in \mathbb{N}$ such that $v_i \in P$ for all $i \geq k$.

If $\sigma : (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots$ is a run of N, the sequence of states $s_0, s_1, s_2, \ldots$ is a *computation* of N. The set of runs (resp. computations) of a Streett automaton N is denoted by $\mathit{Runs}(N)$ (resp. $\mathcal{L}(N)$). □

To show that no behavior of A satisfies $\neg\phi$, the algorithm constructs the *graph product* $A \otimes N_{\neg\phi}$ and checks that no infinite path in it corresponds to a computation of both A and $N_{\neg\phi}$. The construction of the graph product relies on a terminating proof procedure $\vdash$ for the first-order language used in the specification and in the labels of the diagram. The procedure $\vdash$ should be able to prove a subset of the valid sentences that includes all substitution instances of propositional tautologies. Given a first-order formula ψ, we write $\vdash \psi$, $\nvdash \psi$ depending on whether $\vdash$ terminates with or without a proof of ψ, respectively.

Construction 17 (graph product). Given diagram $A = (\mathcal{V}, U, \rho_A, \theta, \tau, \mathcal{J}, \mathcal{C})$ and Streett automaton $N_{\neg\phi} = (\mathcal{V}, (V, E), \rho_N, Q, \mathcal{B})$, the *graph product* $A \otimes N_{\neg\phi} = (W, Z, H)$ consists of a graph (W, H) and of a set of initial vertices $Z \subseteq W$, and is defined by:

1. $W = \{(u, v) \in U \times V \mid \nvdash \neg(\rho_A(u) \wedge \rho_N(v))\}$;
2. $Z = \{(u, v) \in W \mid v \in Q \textit{ and } \nvdash \neg(\theta(u) \wedge \rho_N(v))\}$;
3. $H = \big\{\big((u_1, v_1), (u_2, v_2)\big) \in W^2 \;\big|\; (v_1, v_2) \in E \textit{ and }$
$\nvdash \neg\big(\tau(u_1, u_2) \wedge \rho_N(v_1) \wedge \rho'_N(v_2)\big)\big\}$. □

To show that there is no infinite path in the product that corresponds to a computation of both A and $N_{\neg\phi}$, we check that every infinite path in (W, H) starting from Z violates either a constraint of A or a pair in the acceptance list of $N_{\neg\phi}$. To this end, consider a *strongly connected subgraph* (SCS) $X \subseteq W$ of the graph (W, H). We say that X is *admissible* if the following conditions hold:

1. for all $(R, G) \in \mathcal{J}$ (resp. $\in \mathcal{C}$), if $X \subseteq R \times V$ (resp. if $X \cap (R \times V) \neq \emptyset$), then there are $(u_1, v_1), (u_2, v_2) \in X$ such that $(u_1, u_2) \in G$ and $((u_1, v_1), (u_2, v_2)) \in H$;
2. for all $(P, R) \in \mathcal{B}$, if $X \not\subseteq (U \times P)$ then $X \cap (U \times R) \neq \emptyset$.

The following theorem states that if there are no reachable admissible SCSs in the products, then $A \models \phi$. This check can be done in time polynomial in $|W|$ using efficient graph algorithms.

Theorem 18 (diagram checking). *Given a diagram A and a specification $\phi \in TL_s$, let $A \otimes N_{\neg\phi} = (W, Z, H)$. If all SCSs of (W, H) that are reachable in (W, H) from Z are not admissible, then $A \models \phi$.*

The following theorem states that the verification methodology presented in this paper is complete.

Theorem 19 (completeness for TL_s). *Given a PTS $\mathcal{S}$ and a specification $\phi \in TL_s$, if $\mathcal{S} \models \phi$ then there is a chain of transformations $hd(\mathcal{S}) \overset{*}{\Rightarrow} A$ such that $A \models \phi$ can be proved using Theorem 18.*

Obtaining Guidance

The presence of admissible and reachable SCSs in the product graph can be used to guide the further analysis of the system, following the insights of [13]. Given an admissible and reachable SCS X of $(W, Z, H) = A \otimes N_{\neg\phi}$, let $X_r \subseteq W$ be the set of vertices that can appear along a path from Z to X in (W, H). Consider the projections $Y = \{u \mid (u, v) \in X\}$, $Y_r = \{u \mid (u, v) \in X_r\}$ of X and X_r onto the diagram A: we say that Y_r and Y constitute a *candidate counterexample path* (CCP) in A. The CCPs correspond to regions of the diagram that can contain counterexamples: if a run $\sigma \in \mathit{Runs}(A)$ violates ϕ, there must be a CCP Y_r, Y such that σ first follows Y_r until it reaches Y, and then remains in Y forever while visiting all vertices of Y infinitely often.

The information provided by the CCPs can be used either to guide the search for a counterexample, or to extend the chain of transformations to show that no counterexample is contained in the CCPs.

Search for counterexample. Given a CCP Y_r, Y, it may be possible to prove that there is a behavior shared by the diagram A and the original PTS $\mathcal{S}$ that follows Y_r and then remains in Y forever, visiting all vertices of Y infinitely often. The existence of such a behavior would establish $\mathcal{S} \not\models \phi$.

Alternatively, the CCPs can be used to guide the simulation of the behavior of $\mathcal{S}$ by simulating $\mathcal{S}$ along the CCPs.

Search for proof. The CCPs provide guidance for the extension of the chain of transformations. The aim of the additional transformations is to show that, for every CCP Y_r, Y:

- either there is no path in Y_r from Z to Y;
- or, after following Y_r, a computation cannot remain in Y forever and visit all the vertices of Y infinitely often.

To show that there is no path in Y_r from Z to Y, it is possible to use the simulation rule to strengthen the assertions of the edges and vertices along Y_r, until the path is interrupted by labeling some edge or vertex with *false*. To show that a computation cannot stay in Y forever and visit all vertices of Y infinitely often, the simulation rule can be used to strengthen the labels of vertices and split vertices into new vertices, thus analyzing in more detail the structure of the SCS Y and possibly splitting it into several SCSs. The justice and compassion rules can be used to show that the system cannot stay forever in Y, or infinitely often in some subsets of Y.

Example 20. Using the algorithm presented in this section, it is possible to check that diagram A_1 of Figure 2 satisfies the specification $(65 \leq x \leq 75) \rightarrow \Box(65 \leq x \leq 75)$.

On the other hand, if we check A_1 against the specification $\diamond(65 \leq x \leq 75)$ we obtain two CCPs, corresponding to the SCS $\{v_0\}$, $\{v_1\}$. To prove the specification, we must thus show that either v_0 and v_1 are not reachable (which evidently is not possible), or that a run cannot be forever confined to v_0 or v_1. This is shown by adding the justice constraint $(\{v_0, v_1\}, \{(v_1, v_2)\})$ as in Example 13. The diagram-checking algorithm shows that the resulting diagram A_2 satisfies $\diamond(65 \leq x \leq 75)$. $\square$

Acknowledgments. We thank Todd Neller and Henny Sipma for many useful comments.

References

1. R. Alur, C. Courcoubebetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theor. Comp. Sci.*, 138(1):3–34, 1995.
2. R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho. Hybrid automata: An algorithmic approach to the specification and analysis of hybrid systems. In *Workshop on Hybrid Systems*, volume 736 of *Lect. Notes in Comp. Sci.*, pages 209–229. Springer-Verlag, 1993.
3. Z. Chaochen, A.P. Ravn, and M.R. Hansen. An extended duration calculus for hybrid real-time systems. In *Hybrid Systems*, volume 736 of *Lect. Notes in Comp. Sci.*, pages 36–59. Springer-Verlag, 1993.
4. L. de Alfaro and Z. Manna. Temporal verification by diagram transformations. In *Computer Aided Verification*, volume 1102 of *Lect. Notes in Comp. Sci.*, pages 288–299. Springer-Verlag, 1996.
5. A. Kapur, T.A. Henzinger, Z. Manna, and A. Pnueli. Proving safety properties of hybrid systems. In *FTRTFT'94*, volume 863 of *Lect. Notes in Comp. Sci.*, pages 431–454. Springer-Verlag, 1994.
6. Y. Kesten, Z. Manna, and A. Pnueli. Verifying clocked transition systems. In *Hybrid Systems III*, volume 1066 of *Lect. Notes in Comp. Sci.*, pages 13–40. Springer-Verlag, 1996.
7. L. Lamport. Hybrid systems in TLA+. In *Hybrid Systems*, volume 736 of *Lect. Notes in Comp. Sci.*, pages 77–102. Springer-Verlag, 1993.
8. N.A. Lynch and H. Attiya. Using mappings to prove timing properties. *Distributed Computing*, 6:121–139, 1992.
9. O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *Proc. of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lect. Notes in Comp. Sci.*, pages 447–484. Springer-Verlag, 1992.
10. Z. Manna and A. Pnueli. Completing the temporal picture. *Theor. Comp. Sci.*, 83(1):97–130, 1991.
11. Z. Manna and A. Pnueli. Models for reactivity. *Acta Informatica*, 30:609–678, 1993.
12. S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. Found. of Comp. Sci.*, 1988.
13. H.B. Sipma, T.E. Uribe, and Z. Manna. Deductive model checking. In *Computer Aided Verification*, volume 1102, pages 208–219. Springer-Verlag, 1996.