Hybrid Diagrams ⋆

Luca de Alfaro

Computer Engineering Department, Jack Baskin School of Engineering, University of California, Santa Cruz, CA 95064, USA.

Arjun Kapur

Intel Corporation, Enterprise Platform Group, Santa Clara, CA 95052, USA

Abstract

Hybrid systems provide a formal model for physical systems controlled by discrete-state controllers. To help with the design of correct controllers, we present a methodology that enables the verification of linear-time temporal logic properties of general, non-linear hybrid systems. The methodology is based on the deductive transformation and algorithmic checking of *hybrid diagrams.*

Hybrid diagrams are graphs whose vertices and edges are labeled with first-order assertions; they represent system abstractions, together with the progress properties that have been proved about them. The verification process begins with the automatic construction of an initial diagram, whose behavior coincides with that of the hybrid system. The proof of a specification is constructed by applying a series of *diagram transformations* to this initial diagram. The transformations preserve behavior containment, and the aim of the transformations is to obtain a diagram that can be algorithmically shown to satisfy the specification. Whenever the algorithmic check of a diagram fails, the check returns guidance for the further transformation of the diagram, or indications about possible counterexamples to the specification.

We present four rules for transforming diagrams: each rule enables the study of a certain class of temporal logic properties. While some rules can be applied unconditionally, others require the proof of first-order verification conditions. We prove that the rules lead to the first verification methodology for general hybrid systems that is complete (relative to first-order reasoning) for proving specifications expressed in first-order linear-time temporal logic, provided no temporal operator appears in the scope of a quantifier.

Preprint submitted to Elsevier Science *1 October 2001*

1 Introduction

Hybrid systems provide a formal model for physical systems interacting with discrete controllers. Their temporal behavior is the combination of two types of behavior: the behavior of the physical system, which is continuous in time, and the behavior of the controller, which consists in a sequence of discrete state changes. To help with the notoriously difficult task of controller design, several methods have been proposed for the formal specification and verification of hybrid systems. These methods range from algorithmic methods for the verification of branching-time temporal logic properties [ACHH93,ACH+95,BL95], to deductive approaches for proving linear-time temporal logic properties [MP93,Lam93] and interval-based and duration properties [CRH93,KHMP94,BLR95].

This paper proposes an approach to the verification of hybrid systems based on the joint use of deductive and algorithmic methods. The approach is suited to the verification of linear-time temporal logic properties of general, non-linear hybrid systems. The approach consists in the deductive construction and refinement of system abstractions, represented as *hybrid diagrams,* followed by the algorithmic check of the abstractions against the specification. The approach provides the first methodology that is complete (relative to first-order reasoning) for proving system specifications written in first-order linear temporal logic, with the proviso that no temporal operator appears in the scope of a quantifier. Aside from this relative completeness result, the advantages of the proposed methodology over the rule-based approach of [MP93,KMP96] 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 [dAMSU97].

Hybrid diagrams are related to the *fairness diagrams* of [dAM96] and to the *hybrid automata* of [ACHH93,ACH+95]. They consist in graphs whose vertices are labeled by first-order assertions and whose edges are labeled by first-order transition relations; associated with each diagram are *fairness constraints* that encode acceptance conditions similar to those of ω-automata. The diagrams represent the hybrid system under consideration, along with the safety and progress properties that have been proved about it. In particular, the vertex

---

⋆ A preliminary version of this paper appeared in: L. de Alfaro, A. Kapur and Z. Manna. Hybrid Diagrams: A Deductive-Algorithmic Approach to Hybrid System Verification. In *Proc. of the 14th Annual Symposium on Theoretical Aspects of Computer Science (STACS 97),* volume 1200 of *Lecture Notes in Computer Science,* Springer-Verlag, pages 151–164, 1997.

Email addresses: `luca@soe.ucsc.edu` (Luca de Alfaro), `arjun@scdt.intel.com` (Arjun Kapur).

URL: `http://www.soe.ucsc.edu/~luca` (Luca de Alfaro).

and edge labels encode the behavior of the system and the safety properties it satisfies, while the fairness constraints encode the progress properties. Hybrid diagrams are sufficiently expressive to encode the *phase transition systems* (PTSs) of [MMP92,KMP96], 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 state transitions (both discrete and continuous) of the hybrid 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 we can apply an algorithmic check to the last diagram of the chain. The check either establishes that the final diagram (and, by behavior inclusion, the original PTS) satisfies the specification, or it returns a set of *candidate counterexample paths* (CCPs) in the diagram. The CCPs provide guidance for the extension of the chain of transformations, following the insights of [SUM96]. Additionally, the CCPs can be used to guide the search for counterexamples by directing the simulation of the original system along the CCPs. Hence, the proposed approach combines the diagnostic information typical of model checking with the generality of deductive methods.

There are four rules for transforming diagrams. The *simulation rule* modifies the graph structure of the diagram, enabling the study of safety properties [dAM96]. 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 progress properties of the systems; these properties are then represented as fairness constraints which are added to the diagrams. Ranking functions are a basic tool for proving progress and termination of reactive systems [MP91]. Ranking functions associate with each state an ordinal number (or an element of a well-founded domain): the number represents the "distance" between the state and a given goal, and decreases when progress towards the goal is made. Our completeness proof shows that in order to study progress and termination of real-time and hybrid systems, it is necessary to augment ranking functions with *delay functions,* which provide an upper bound to the amount of time that can elapse before the ranking function decreases. Delay functions are related to the mappings of [LA92]; our results indicate that the combination of ranking and delay functions is complete for proving progress

properties of real-time and hybrid systems. While the transformation rules have been presented in their full generality, it is possible to construct libraries that list special cases of the rules that occur frequently in practice and that can be applied with little user intervention. We present two such special cases and illustrate them on an example.

2 Phase Transition Systems

The hybrid system model we adopt in this paper is that of *phase transition systems* (PTS) [MMP92,KMP96]. 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.

2.1 Discrete, Continuous and Hybrid Variables

The state of a PTS is described by three types of variables: *discrete variables, clock variables* and *hybrid variables.* The value of discrete variables can change only at discrete points in time. Discrete variables are used to describe the state of the controller, as well as any other quantity that changes value in a discrete fashion.

Clock variables can be set to arbitrary values at discrete points in time; once set, their value increases linearly with time. Among the clock variables is a *master clock* T, that measures the amount of time elapsed since the initial configuration of the system. Properly speaking, the master clock is not a state variable, since its value does not usually correspond to any physical quantity in the underlying system. Nonetheless, the master clock is often useful in writing system specifications, as it provides an absolute reference for measuring the length of intervals of time. We remark that the assumption of the presence of a master clock is not essential for the development of the results presented in the paper: with minor changes, it is possible to adapt the proposed methodology to systems without the notion of a master clock.

Hybrid variables are used to describe the state of the physical system, along with any other derived quantity that can change continuously in time. The value of hybrid variables can change in a continuous way in time, and can also exhibit countably many discontinuity points. The temporal evolution of the values of hybrid variables is often specified by differential equations. In our model we consider the solutions of the differential equations as given, separating the concerns of solving the differential equations from those of studying the temporal properties of the system.

Throughout the paper, we assume that we have a fixed assertion language, used to describe the initial condition and the state transitions of PTSs, and to label the vertices and edges of hybrid diagrams. We assume that the assertion language contains first-order predicate calculus, and that among its domains are the integers and the reals, along with and the usual mathematical functions and relation symbols (e.g. $+, \cdot, \exp, >, \ldots$) interpreted over the reals. By *assertion* over a set of variables $\mathcal{V}$ we intend a formula of this assertion language whose free variables are among those in $\mathcal{V}$. Moreover, in order to obtain our relative completeness results, we assume that the assertion language is powerful enough to represent records of values and lists of values and records [MP91]. Furthermore, the language must also include the *least* and *greatest* fixpoint operators [MP91]. Let $P(x_1, \ldots, x_n)$ be a predicate symbol of arity n, and $\Psi(P, x_1, \ldots, x_n)$ be a formula whose free variables are a subset of $x_1, \ldots, x_n$, and where the predicate P has only *positive* occurrences, i.e., occurrences under an even number of negations. The equation

$$P(x_1, \ldots, x_n) \equiv \Psi(P, x_1, \ldots, x_n) \tag{1}$$

can be interpreted as a fixpoint equation for P, provided all elements except for P and $x_1, \ldots, x_n$ are interpreted. A relation $\widetilde{P}$ is a *solution* of (1) if, when substituted for P, the equation holds for all values of $x_1, \ldots, x_n$. We denote by $\mu P.\Psi$ (resp. $\nu P.\Psi$) the predicate that defines the minimal (resp. maximal) relation satisfying (1). The predicates $\mu P.\Psi$ and $\nu P.\Psi$ are known as the *least* and *greatest* fixpoints of (1). The assertion language must contain the formulas $\mu P.\Psi$ and $\nu P.\Psi$, for any P and Ψ that satisfy the above restrictions.

2.3 Definition and Behavior of Phase Transition Systems

A *Phase Transition System* (PTS) is defined as follows.

Definition 1 (PTS) 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 $\mathbb{R}$. A distinguished clock variable $T \in \mathcal{V}_c$ represents 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 $\mathcal{V}$ that defines the set $\{s \in S \mid s \models \theta\}$ of initial states. We require that the implication $\theta \rightarrow T = 0$ holds.

(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 $\{(s_1, s_2) \in S \times S \mid (s_1, s_2) \models \pi\}$, where (s_1, s_2) interprets $x \in \mathcal{V}$ as $s_1[\![x]\!]$ and $x' \in \mathcal{V}'$ as $s_2[\![x]\!]$. For all $\pi \in \mathcal{T}$, we require that the implication $\pi \rightarrow T = T'$ holds: transitions are instantaneous. We denote by $En(\pi) = \exists \mathcal{V}' \,.\, \pi$ the *enabling condition* of π, denoting the set of states from which π can be taken.

(4) A *time-progress* assertion Π over $\mathcal{V}$ that specifies a condition that must hold whenever time progresses. This assertion can be used to express *urgency requirements* for the scheduling of transitions. For example, the requirement that a transition π be taken as soon as it becomes enabled can be expressed by conjoining the formula $\neg En(\pi)$ to the time-progress assertion. The time-progress assertion can also be used to express other timing requirements for the scheduling of transitions; we refer to [NOSY93] for a more thorough discussion.

(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

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

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 value of x as a function of the elapsed time Δ. $\square$

We study the behavior of a PTS in the *sampling semantics.* This semantics represents the temporal evolution of the system as an enumerable sequence of snapshots, each describing the state of the system at a certain time. A sequence of snapshots is called a *computation,* and with each PTS we associate the set of computations that correspond to possible system evolutions.

To define the set of computations of a PTS, we introduce the family of assertions $\{tick_a[\Delta]\}_{a \in \mathcal{A}}$. 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 possible state change of the system due to activity a when an amount of time $\Delta \geq 0$ elapses, and is given by:

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

In the above formula, $\Pi[F_a^x[t]/x]_{x\in\mathcal{V}}$ denotes the result of simultaneously replacing each occurrence of x in Π with $F_a^x[t]$, for all $x \in \mathcal{V}$. The form of the assertion $tick_a[\Delta]$ ensures that the progress constraint Π holds at every moment of a time-step, except possibly for the final one. As discussed in [KMP96], if Π is used only to encode urgency requirements or upper bounds on the transition waiting times, assertion $tick_a[\Delta]$ can be rewritten without quantifiers.

Definition 2 (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 $i \geq 0$, one of the following holds:
  (a) there is a transition $\pi \in \mathcal{T}$ such that $(s_i, s_{i+1}) \models \pi$;
  (b) 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 $\mathcal{S}$. $\square$

2.4 A Room-Heater Example

As our running example throughout the paper, we consider a variant of the temperature control system presented in [ACHH93]. 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. The general form of the evolution function for x is $F^x = hw + r + (x - hw - r)e^{-\Delta/cw}$, where c is the heat capacity of the room, h is the heat provided by the heater, w is the thermal resistance of the window, and r is the exterior temperature. In our system, we assume that $h_{off} = 0$, $h_{on} = 10$, $w_{open} = 1$, $w_{closed} = 1.5$, $c = 70$, and $r = 60$°F. Initially, the room temperature x is below 60°F. Our phase transition system $\mathcal{S} = (\mathcal{V}, \theta, \mathcal{T}, \Pi, \mathcal{A})$ is defined as follows.

(1) The set of discrete variables is $\mathcal{V}_d = \{H, W\}$, 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 $\{Open, Closed\}$. The set of clock variables is $\mathcal{V}_c = \{T, y\}$, where T is the global clock, and y measures the time elapsed since the last *On*/*Off* switching of the heater. The set of hybrid variables is $\mathcal{V}_h = \{x\}$, where x is the temperature of the room.
(2) The initial condition is
  $\theta : H = Off \;\wedge\; W = Closed \;\wedge\; x < 60 \;\wedge\; y = 0 \;\wedge\; T = 0.$

(3) The set of transitions is $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, where $\tau_i : E_i \wedge R_i$ for $i \in \{1, 2, 3\}$, and

$$\begin{array}{ll} E_1 : H = \mathit{Off} \ \wedge \ x \leq 68 \ \wedge \ y \geq 60 & R_1 : H' = \mathit{On} \ \wedge \ y' = 0 \\ E_2 : H = \mathit{On} \ \wedge \ x \geq 72 \ \wedge \ y \geq 60 & R_2 : H' = \mathit{Off} \ \wedge \ y' = 0 \\ E_3 : \mathit{true} & R_3 : W' = \neg W \end{array}$$

where $\neg \mathit{Open} = \mathit{Closed}$ and $\neg \mathit{Closed} = \mathit{Open}$. Variables not mentioned in R_1, R_2, and R_3, respectively, are left unchanged by the transitions.

(4) The time-progress assertion is $\Pi = \neg E_1 \wedge \neg E_2$. This ensures that τ_1 and τ_2 are taken as soon as they become enabled.

(5) The activities are $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$, where $F^T_{a_i} = T + \Delta$ and $F^y_{a_i} = y + \Delta$ for every $i \in \{1, 2, 3, 4\}$, and C_{a_i} and $F^x_{a_i}$ are defined as follows:

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

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 [dAM96]. Hybrid diagrams are graphs whose vertices are labeled by assertions and whose edges are labeled by transition relations; additional *fairness constraints* represent progress properties of the diagram's computations.

Diagrams are used to represent the PTS under consideration, along with the temporal properties that have been proved about it. Their graph structure provides an abstract representation of the possible state-transitions of the system. Unlike ordinary abstractions, diagrams can preserve all the information that is necessary to recover the behavior of the original system in full detail. This information can be used to refine the abstraction, should the abstraction be inadequate for the proof of a given property. The refinements of the abstractions take the form of *diagram transformations,* and will be discussed in the next section.

3.1 Definition of Hybrid Diagram

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. A *state* is a type-consistent interpretation of all the variables in $\mathcal{V}$; the *state space* S of the diagram is the set of all such variable interpretations.
(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) with $v \in V$ and $s \models \rho(v)$. Hence, a location consists in a vertex and in a corresponding state, and it represents an instantaneous configuration of the diagram. We indicate by $loc(A) = \{(v, s) \in V \times S \mid s \models \rho(v)\}$ the set of locations of A.
(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 $\{(v, s) \in V \times S \mid s \models \theta(v)\}$. For all $v \in V$, we require that assertion $\theta(v) \rightarrow T = 0$ holds.
(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, where Δ is a distinguished variable with domain $\mathbb{R}^+$. 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 $(R, G) : R \subseteq V, G \subseteq V \times V$.

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 of vertices $R \subseteq V$ 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 justice constraint indicates that a request that is performed without interruptions will eventually lead to gratification; a compassion constraint indicates that a request performed infinitely often will be gratified infinitely often [MP93,dAM96].

3.2 Diagram Computations

Given an assertion φ over $\mathcal{V}$, we denote by φ' the formula obtained by replacing each free $x \in \mathcal{V}$ by $x' \in \mathcal{V}'$. With this notation, we define diagram computations as follows.

Definition 3 (diagram computations) A *run* of a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ is an infinite sequence of locations (v_0, s_0), (v_1, s_1),

$(v_2, s_2), \ldots \in (loc(A))^\omega$, satisfying the following conditions.

(1) *Initiality:* $s_0 \models \theta(v_0)$.
(2) *Edge labels:* 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 $Runs(A)$ and $\mathcal{L}(A)$ the sets of runs and computations of A, respectively. □

We note that the above definition of diagram run differs slightly from the one presented in [dAKM97]: since the definition of location ensures that $s_i \models \rho(v_i)$, we can drop the "consecution requirement" used there. When a state transition (s_i, s_{i+1}) occurs along a computation, it must satisfy the formula $\rho(v_i) \wedge \tau(v_i, v_{i+1}) \wedge \rho'(v_{i+1})$. The current choice simplifies the labels of diagram edges, yielding a more concise graphical representation. We define the abbreviation

$$\widehat{\tau}(u, v) \stackrel{\text{def}}{=} \rho(u) \wedge \tau(u, v) \wedge \rho'(v)$$

denoting the possible state changes corresponding to the traversal of edge (u, v).

3.3 The Initial Diagram for a PTS

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

Construction 1 *Given a PTS $\mathcal{S} = (\mathcal{V}, \theta, \mathcal{T}, \Pi, \mathcal{A})$, we define the diagram $hd(\mathcal{S}) = (\mathcal{V}, V, \rho, \varphi, \tau, \mathcal{J}, \mathcal{C})$ by $V = \{v_0\}$, $\rho(v_0) = true$, $\varphi(v_0) = \theta$, $\mathcal{J} = \emptyset$, $\mathcal{C} = \emptyset$, and*

$$\tau(v_0, v_0) = \Big(\bigvee_{\pi \in \mathcal{T}} (\pi \wedge \Delta = 0) \Big) \vee \Big(\bigvee_{a \in \mathcal{A}} tick_a[\Delta] \Big) \,. \qquad \square$$

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

Proof. The result follows by comparing Definitions 2 and 3. □

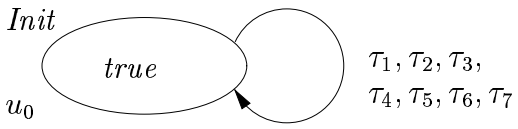

Fig. 1. Hybrid Diagram A_0. The labels τ_1, τ_2, and τ_3 are as the transitions of RH bearing the same name (with the added conjunct $\Delta = 0$); the labels τ_4, τ_5, τ_6, and τ_7 are equal to $tick_{a_1}[\Delta]$, $tick_{a_2}[\Delta]$, $tick_{a_3}[\Delta]$, and $tick_{a_4}[\Delta]$, respectively. In edge labels, we use the comma "," as an alternative symbol for disjunction "$\vee$". The single vertex u_0 is marked *Init* as a reminder that its initial label $\theta(u_0)$ is equal to the initial condition of RH.

In Figure 1, we present the initial diagram $A_0 = hd(RH)$ corresponding to system RH.

3.4 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 [ACHH93,BR95,ACH+95]. 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. In a hybrid diagram, both types of system evolution are described by the traversal of diagram edges. Vertex labels are used to express invariants that have been proved about diagram computations. This difference is 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* [dAM96]. 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 $hd(\mathcal{S}) = A_0 \Rightarrow A_1 \Rightarrow \cdots \Rightarrow A_n$. At any time, it is possible to check algorithmically whether the last diagram of the chain complies 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 [dAM96], 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 $\nu : V_1 \mapsto 2^{V_2}$ mapping each vertex of A_1 into a subset of related vertices in A_2. We say that a function ν induces a simulation relation between A_1 and A_2 if the following three conditions hold:

(1) For each initial location (u, s) of A_1, there is an initial location (v, s) of A_2 with $v \in \nu(u)$.
(2) For all locations (u, s) of A_1 and (v, s) of A_2 such that $v \in \nu(u)$, if it is possible to take a transition from (u, s) to (u', s') in A_1, then it must be possible to take a transition from (v, s) to (v', s') in A_2, with $v' \in \nu(u)$.
(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) If A_2 is in the request region, then A_1 must also be: precisely, if $v \in R_2 \cap \nu(u)$, then $u \in R_1$.
   (b) If A_1 follows a gratify edge, A_2 must be able to do the same by executing one simulation step. Precisely, assume A_1 is at location (u, s) and location A_2 is at (v, s), with $v \in \nu(u)$. If A_1 takes a transition from (u, s) to (u', s') with $(u, u') \in G_1$, then A_2 must be able to take a transition to some (v', s'), with $v' \in \nu(u')$ and $(v, v') \in G_2$.

The *simulation rule* establishes whether there is a simulation relation between two diagrams A_1 and A_2 by checking the validity of a set of logical formulas. In the affirmative case, the rule enables the transformation of A_1 into A_2.

Rule 1 (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* $\nu : V_1 \mapsto 2^{V_2}$ *that satisfies the conditions below, then* $A_1 \Rightarrow A_2$.

(1) For all $u \in V_1$, the following assertion holds:

$$\theta_1(u) \wedge \rho_1(u) \rightarrow \bigvee_{v \in \nu(u)} (\theta_2(v) \wedge \rho_2(v)) \ .$$

(2) For all $u, u' \in V_1$ and $v \in \nu(u)$, the following assertion holds:

$$\left(\widehat{\tau}_1(u, u') \wedge \rho_2(v)\right) \rightarrow \bigvee_{v' \in \nu(u')} \widehat{\tau}_2(v, v') \ .$$

(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 $\nu(u) \cap R_2 \neq \emptyset$ then $u \in R_1$.
(b) For all $(u, u') \in G_1$ and $v \in \nu(u)$, the following assertion holds:

$$\widehat{\tau}_1(u, u') \wedge \rho_2(v) \quad \rightarrow \quad \bigvee_{v' \in H(u', v)} \widehat{\tau}_2(v, v') \ ,$$

where $H(u', v) = \{v' \mid v' \in \nu(u') \wedge (v, v') \in G_2\}$. □

Note that, if A_2 simulates A_1, it is not necessary for every location of A_1 to correspond to a location of A_2. In fact, Condition 1 requires this only for initial locations, while Condition 2 ensures that the *reachable* locations of A_1 correspond to (reachable) locations of A_2. Hence, the non-reachable locations of A_1 may not correspond to any location in A_2.

Theorem 5 (soundness of Rule 1) *If $A_1 \Rightarrow A_2$ by Rule 1, then $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$.*

Example 1 *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 ν defined by $\nu(u_0) = \{v_0^1, v_1^1, v_2^1, v_3^1\}$. In Figure 2, v_0^1 is the only vertex satisfying the initial condition of RH. For the other vertices, the initial labeling is false.*

Intuitively, diagram A_1 says that initially (at vertex v_0^1) the temperature is below $65°F$, the clock is below 60, and the heater is off. After some time at v_0^1, the system may go to vertex v_1^1, where the heater is on, the temperature is still below $65°F$, and the clock can be either below or above 60. After this initial phase, the system oscillates forever between vertices v_2^1 (where the heater is on) and v_3^1 (where it is off). □

Two special cases of the simulation rule are commonly used in practice: *vertex-split*, that splits a vertex of the diagram into several vertices, and *vertex-strengthen,* that strengthens the labels of the diagram vertices.

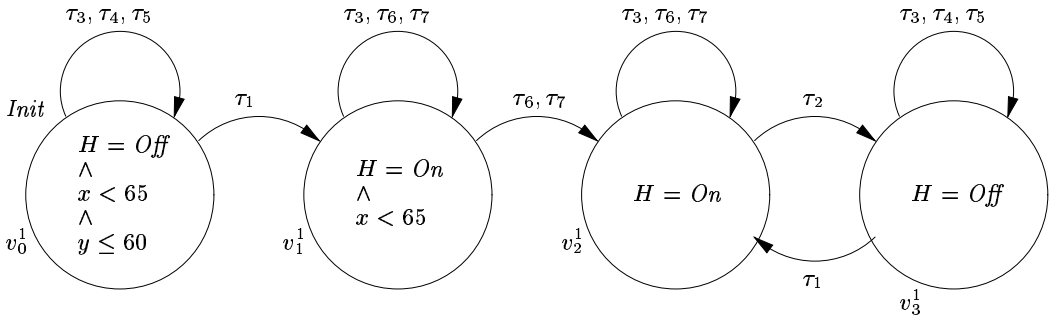

Fig. 2. Hybrid Diagram A_1. Edges labeled with *false* are not shown.

4.1.1 Vertex Split

Rule *vertex-split* enables us to split a vertex v with label φ into a set of vertices $u_1, \ldots, u_n$ with labels $\varphi \wedge \psi_1, \ldots, \varphi \wedge \psi_n$. The rule is used to do a case-analysis on the possible states of the system corresponding to vertex v. In a figurative way, the rule enables us to "zoom-in" into a vertex, analyzing in more detail the transition structure of the system at that vertex.

Rule 2 (vertex-split) *Let $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ be a diagram, $v \in V$ a vertex, $U = \{u_1, \ldots, u_n\}$ a set of new vertices (in which we wish to split v), and $\psi_1, \ldots, \psi_n$ a set of formulas over $\mathcal{V}$ such that $\bigvee_{i \in [1..n]} \psi_i \equiv true$. We can transform A into a diagram A' obtained as follows.*

(1) Replace vertex v with the set of vertices U, where for each $1 \leq i \leq n$, $u_i \in U$ is labeled by $\rho(u_i) = \rho(v) \wedge \psi_i$ and $\theta(u_i) = \theta(v)$.
(2) For all $i, j \in [1..n]$ and $z \in V - \{v\}$, let

$$\tau(z, u_i) = \tau(z, v) \qquad \tau(u_i, z) = \tau(v, z) \qquad \tau(u_i, u_j) = \tau(v, v) \ .$$

For every new edge (w, w') thus labeled, if $\widehat{\tau}(w, w') \equiv false$, then set $\tau(w, w') = false$, thereby eliminating the edge from the diagram.
(3) For each constraint (R, G), if $v \in R$ then replace v with the set U, and for every edge going to or from v in G, replace the edge with the corresponding new edges going to or from U. □

Example 2 *By applying the vertex-split rule twice to the diagram A_1 of Figure 2, we obtain the diagram A_2 presented in Figure 3. The two applications of the rule correspond to splitting v_2^1 into $\{v_2^2, v_4^2\}$ and splitting v_3^1 into $\{v_3^2, v_5^2\}$.*

Intuitively, diagram A_2 analyzes in more detail the behavior of the Room-Heater system corresponding to vertices v_2^1 and v_3^1, distinguishing in each case whether the clock is below or above 60. The distinction is important, since if the clock is below 60 the heater cannot change state. For this reason, the edges from v_2^2 to v_5^2, and from v_5^2 to v_2^2 are labeled with false, *and are not shown in Figure 2.* □

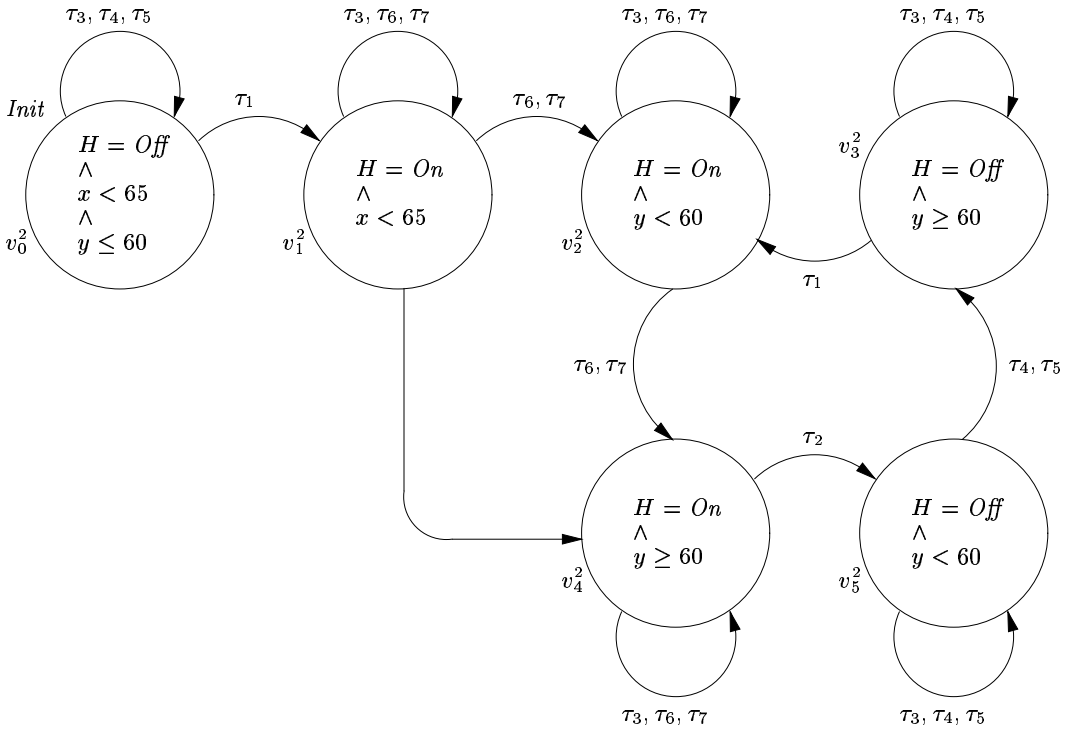

Fig. 3. Hybrid Diagram A_2. Edges labeled with *false* are not shown.

4.1.2 Vertex Strengthen

The other special case of the simulation rule we consider is the *vertex-strengthen* rule. This rule enables us to strengthen the assertions labeling the vertices with inductive invariants.

Rule 3 (vertex-strengthen) *Let $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$, be a diagram, $v_1, \ldots, v_n \subseteq V$ a list of vertices whose vertex labels we wish to strengthen, and $\psi_1, \ldots, \psi_n$ a set of formulas over $\mathcal{V}$. If for all $i, j \in [1..n]$ the following assertions hold*

$$\theta(v_i) \rightarrow \psi_i \qquad (\psi_i \wedge \widehat{\tau}(v_i, v_j)) \rightarrow \psi_j' \ , \tag{2}$$

then we can transform A into diagram $A' = (\mathcal{V}, V, \rho', \theta, \tau, \mathcal{J}, \mathcal{C})$ obtained by defining $\rho'(v_i)$ to be $\rho(v_i) \wedge \psi_i$, for all $i \in [1..n]$ and $\rho'(u) = \rho(u)$ for all $u \in V - \{v_1, \ldots, v_n\}$. $\square$

Proving (2) is equivalent to proving that $\psi_1, \ldots, \psi_n$ are inductive invariants with respect to the vertex and edge labels.

By construction, the set of locations associated with vertex v_i in A' is a subset of the set of locations associated with v_i in A, for all $1 \leq i \leq n$. Thus, while all vertices of A have a corresponding vertex in A', the same does not necessarily

hold for the locations. This indicates how the simulation rule (and in particular its special case vertex-strengthen) can be used to *reduce* the set of locations of a diagram, enabling to construct better approximations of the set of reachable locations.

Example 3 *By applying the vertex-strengthen rule to the diagram A_2 of Figure 3, we obtain the diagram A_3 presented in Figure 4. The application of the rule is based on*

$$\psi_2 = \psi_4 = 65 \leq x \leq 75 \ \wedge \ x \leq 75 - 7 \cdot e^{-y/105}$$

for vertices v_2^3 and v_4^3, and

$$\psi_3 = \psi_5 = 65 \leq x \leq 75 \ \wedge \ x \geq 60 + 12 \cdot e^{-y/70}$$

for vertices v_3^3 and v_5^3.

Intuitively, the purpose of this transformation is to show that indeed the temperature remains within the desired range $65°$–$75°F$ at vertices v_2^3, v_3^3, v_4^3, and v_5^3. The additional conjuncts, denoted by φ_1 and φ_2 in Figure 4, relate the temperature to the value of the clock, ensuring that the clock is above 60 (and the heater is ready to be switched on/off) before the temperature approaches the boundaries of the desired range. □

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, we can only add constraints that do not restrict the language of the diagram: such constraints are said to be *compatible* with the diagram.

Definition 6 (compatible constraints) We say that a constraint (R, G) is *J-compatible* (resp. *C-compatible*) with a diagram A if its addition to the justice (resp. compassion) set of A does not change the set of runs of A. □

Intuitively, if a constraint is J- or C-compatible with a diagram, it expresses a progress property obeyed by the diagram. Hence, in order to add a constraint to a diagram, the justice and compassion rules must prove that the constraint is compatible. In turn, the added constraints represent progress properties that have been proved about the runs of the diagram. Thus, the justice and compassion rules enable us to prove progress properties of diagrams, and to record the proved properties as constraints.

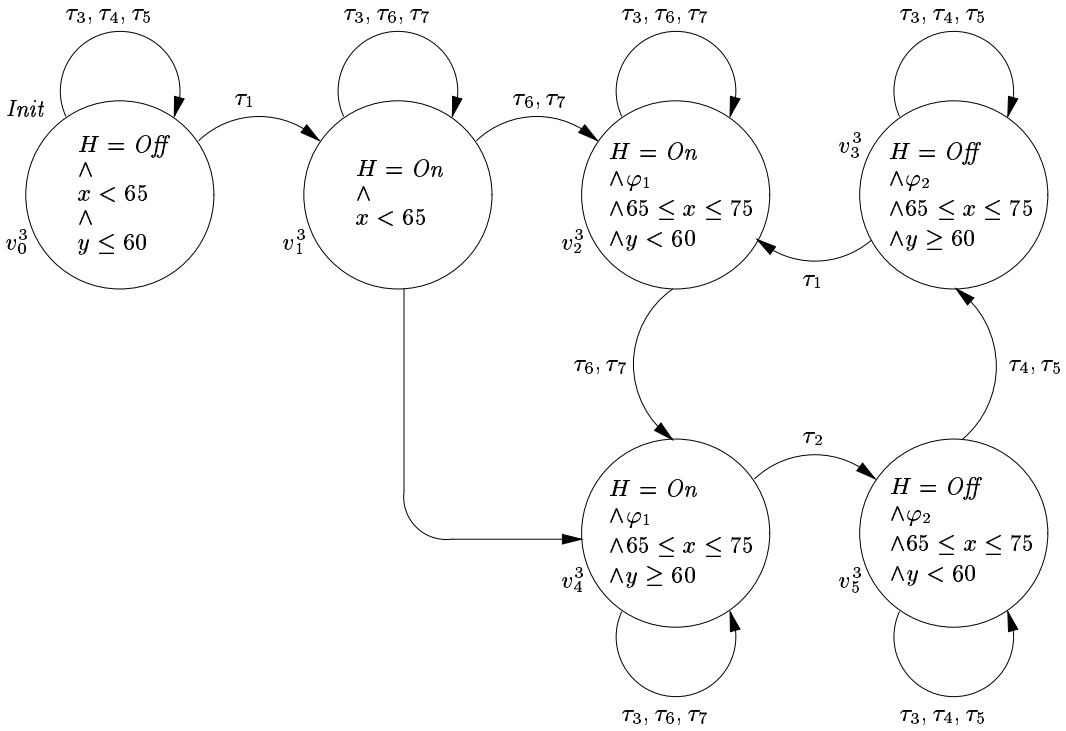

Fig. 4. Hybrid Diagram A_3, where $\varphi_1 : x \leq 75 - 7 \cdot e^{-y/105}$ and $\varphi_2 : x \geq 60 + 12 \cdot e^{-y/70}$. Edges labeled with *false* are not shown.

To prove that all runs obey the constraint, the justice and compassion rules rely on *ranking* and *delay* functions to measure progress towards its gratification. Ranking functions are mappings from diagram locations to ordinal numbers (or to a well-founded set), and they are a standard tool for proving progress and termination properties of systems. In their most basic application, ranking functions are used to prove that a given goal is eventually reached. To this end, it suffices to find a ranking function that does not increase until the goal is reached, and that would decrease infinitely many times along a computation if the goal were not reached. For reactive systems, ranking functions suffice for the proof of arbitrary progress properties expressed in temporal logic [MP91].

Our results indicate that in order to be able to prove arbitrary progress properties of real-time and hybrid systems, it is necessary to use ranking functions together with *delay functions*. Delay functions associate a non-negative real number with each diagram location: roughly, the real number represents an upper bound to the amount of time that can elapse before a decrease in the value of the associated ranking function. Delay functions are reminiscent of the mappings of [LA92]: however, in [LA92] the mappings are used alone, and

completeness for progress properties is not achieved.

Definition 7 (ranking and delay functions) Recall that a *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.

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 function δ is represented by a labeling $\widehat{\delta}$ that associates with each $v \in V$ a term $\widehat{\delta}(v)$ over $\mathcal{V}$ such that $s[\![\widehat{\delta}(v)]\!] = \delta(v, s)$ for all $s \models \rho(v)$. Here, we have extended the interpretation function $[\![\cdot]\!]$ from variable to terms in the obvious way. Similarly, the delay function γ is represented by a labeling $\widehat{\gamma}$ that associates with each $v \in V$ a term $\widehat{\gamma}(v)$ over $\mathcal{V}$ such that $s[\![\widehat{\gamma}(v)]\!] = \gamma(v, s)$ for all $s \models \rho(v)$. $\square$

4.2.1 Justice Rule

The justice rule is used to to add a justice constraint (R, G) to a diagram. Given (R, G) and the ranking and delay functions δ, γ, the rule checks that:

(1) while in R, the value of δ does not increase unless an edge in G is taken;
(2) the value of γ gives an upper bound to the amount of time before an edge in G is taken, R is left, or δ decreases.

If these two conditions hold, then all runs already obey the justice constraint (R, G), which can be added to the diagram.

Rule 4 (justice) *Consider a diagram* $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ *and a constraint* $(R, G) : R \subseteq V, G \subseteq V \times V$. *Assume that there is a ranking labeling* $\widehat{\delta}$ *and a delay labeling* $\widehat{\gamma}$ *such that, for all* $u, v \in R$ *with* $(u, v) \notin G$, *the assertion*

$$\widehat{\tau}(u, v) \quad \rightarrow \quad \widehat{\delta}(u) > \widehat{\delta}'(v) \vee \left(\widehat{\delta}(u) = \widehat{\delta}'(v) \wedge \widehat{\gamma}(u) \geq \widehat{\gamma}'(v) + \Delta\right) \tag{3}$$

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

A special case that occurs frequently is when, for every location (v, s) with $v \in R$, there is an upper bound $t_M(v, s)$ for the time before a G-edge is followed. In this case we can take $s[\![\widehat{\gamma}(v)]\!] = t_M(v, s)$ and $\widehat{\delta}(v) = 0$, and assertion (3) reduces to

$$\widehat{\tau}(u, v) \quad \rightarrow \quad \widehat{\gamma}(u) \geq \widehat{\gamma}'(v) + \Delta \; . \tag{4}$$

The labeling $\widehat{\delta}$ is used to cover the case in which there is no upper bound for the time before a G-edge is followed. An example where the labeling $\widehat{\delta}$ is

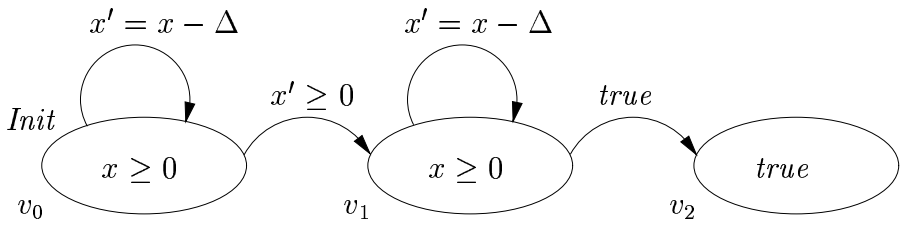

Fig. 5. Hybrid Diagram illustrating ranking labelings. All edges also have the conjunct $T' = T + \Delta$. Edges labeled with *false* are not shown.

required is presented below.

Example 4 *Consider the hybrid diagram depicted in Figure 5. We wish to add the justice requirement* $(R, G) = (\{v_0, v_1\}, \{(v_1, v_2)\})$. *The value of* x *provides a bound for the length of time for which the system can stay at* v_1 *before taking the transition in* G *from* v_1 *to* v_2. *However, the value of* x *does not provide a similar bound from the vertex* v_0, *since* x *gets reset to an arbitrary value during the transition from* v_0 *to* v_1. *Moreover, there is no* global *upper bound on how long we can stay in vertex* v_0 *before taking a gratify transition, even though there is a* local *upper bound on how long we can stay in* v_0 *before going to* v_1. *Thus, we need to use both a ranking and a delay function to prove the justice requirement. To apply the rule we use a ranking labeling defined by* $\widehat{\delta}(v_0) = 1$ *and* $\widehat{\delta}(v_1) = \widehat{\delta}(v_2) = 0$, *and a delay labeling defined by* $\widehat{\gamma}(v_0) = \widehat{\gamma}(v_1) = x$ *and* $\widehat{\gamma}(v_2) = 0$. $\square$

Example 5 *Returning to our Room-Heater example, to show that the temperature eventually reaches the desired range, we apply Rule 4 to the diagram* A_3 *of Figure 4, adding the justice constraint* $(\{v_0^3, v_1^3\}, \{(v_1^3, v_2^3), (v_1^3, v_4^3)\})$; *we denote by* A_4 *the resulting diagram. This constraint shows that a run of* A_3 *cannot stay forever in* v_0^3 *or* v_1^3, *and must eventually proceed to either* v_2^3 *or* v_4^3. *Intuitively, this proves that the temperature eventually reaches the desired range of* $65°$–$75°F$, *and once in that range, it stays forever in the range, since no edge leaves* v_i^3, $i \in \{2, 3, 4, 5\}$.

The rule uses a ranking function defined by $\widehat{\delta}(v_0^3) = 1$ *and* $\widehat{\delta}(v_i^3) = 0$ *for all* $i \in \{1, \ldots, 5\}$. *The delay function is given by*

$$\widehat{\gamma}(v_0^3) = 60 - y;$$
$$\widehat{\gamma}(v_1^3) = \begin{cases} 175 + 105 \ln((75 - x)/49) & \text{if } x \leq 60; \\ 150 + 70 \ln((70 - x)/10) & \text{otherwise;} \end{cases}$$
$$\widehat{\gamma}(v_2^3) = \widehat{\gamma}(v_3^3) = \widehat{\gamma}(v_4^3) = \widehat{\gamma}(v_5^3) = 0 \, .$$

We point out that while this application of the rule uses both a ranking and a delay function, in this example it would have been possible to achieve the same effect using only a delay function. However, this would have required a more complex delay function. $\square$

Theorem 8 (soundness of Rule 4) *If a constraint (R, G) is added by Rule 4 to the justice set of a diagram A, obtaining diagram A', then $Runs(A) = Runs(A')$ and $\mathcal{L}(A) = \mathcal{L}(A')$.*

Proof. Since A' has more constraints than A, it is immediate that $Runs(A') \subseteq Runs(A)$. To show the reverse containment, assume towards the contradiction that the conditions of Rule 4 are satisfied, and that there is a run $\sigma : (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots \in Runs(A)$ that does not satisfy (R, G). By definition, there is $k \in \mathbb{N}$ such that $v_i \in R$, $(v_i, v_{i+1}) \notin G$ for all $i \geq k$. By Condition (3) the value of $\delta(v_i, s_i)$ does not increase for $i \geq k$. Since the domain of δ is well-founded, this value cannot decrease infinitely often, and there must be $k' \geq k$ such that $\delta(v_i, s_i)$ is constant for $i \geq k'$. For $m \geq k'$, let $\xi_m = s_m[\![T]\!] - s_{k'}[\![T]\!]$. From Condition (3) it is easy to prove by induction on $m \geq k'$ that $\xi_m \leq \gamma(v_{k'}, s_{k'}) - \gamma(v_m, s_m)$. Since $\lim_{m\to\infty} \xi_m = \infty$ because of the divergence of time, we have $\lim_{m\to\infty} \gamma(v_m, s_m) = -\infty$, contradicting the non-negativity of γ. □

The completeness result for this rule will be stated and proved in Section 6.

4.2.2 Compassion Rule

The compassion rule is used to add a compassion constraint (R, G) to a diagram. Given (R, G) and the ranking and delay functions δ, γ, the rule checks that:

(1) unless an edge in G is taken, δ does not increase;
(2) whenever R is entered, δ decreases;
(3) while in R, the value of γ gives an upper bound to the amount of time before an edge in G is taken, R is left, or δ decreases.

If these three conditions hold, then all runs already obey the compassion constraint (R, G), which can then be added to the diagram.

Rule 5 (compassion) *Given a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ and a constraint $(R, G) : R \subseteq V, G \subseteq V \times V$, assume that there is a ranking labeling $\widehat{\delta}$ and a delay labeling $\widehat{\gamma}$ such that, for every $u, v \in V$ with $(u, v) \notin G$, the following conditions hold:*

$$\widehat{\tau}(u, v) \quad \rightarrow \quad \widehat{\delta}(u) \geq \widehat{\delta}'(v) \tag{5}$$

$$\textit{If } u \notin R, v \in R: \quad \widehat{\tau}(u, v) \quad \rightarrow \quad \widehat{\delta}(u) > \widehat{\delta}'(v) \tag{6}$$

$$\textit{If } u \in R, v \in R: \quad \widehat{\tau}(u, v) \quad \rightarrow \quad \widehat{\delta}(u) > \widehat{\delta}'(v) \vee \widehat{\gamma}(u) \geq \widehat{\gamma}'(v) + \Delta \tag{7}$$

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

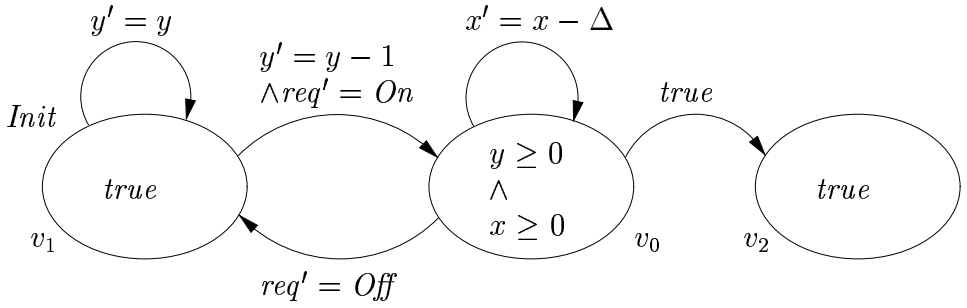

Fig. 6. Hybrid Diagram illustrating the compassion rule. All edges also have the conjunct $T' = T + \Delta$. Edges labeled with *false* are not shown.

Example 6 *Consider the hybrid diagram in Figure 6. The diagram represents a system where a request can be made to do some action (modeled by variable req). This request can be undone at any time (modeled by entry into vertex v_1), and then remade at a later time (modeled by entry into vertex v_0). However, only a finite number of requests can be made (modeled by variable y).*

We would like to prove that if we are infinitely often in vertex v_0 then we will eventually reach vertex v_2, where the edge (v_0, v_2) represents the gratification of the request. That is, we wish to add the compassion requirement $(\{v_0\}, \{(v_0, v_2)\})$. To apply the rule we use a ranking function defined by $\widehat{\delta}(v_0) = \widehat{\delta}(v_1) = y$ and $\widehat{\delta}(v_2) = 0$, and a delay function defined by $\widehat{\gamma}(v_0) = x$ and $\widehat{\gamma}(v_1) = \widehat{\gamma}(v_2) = 0$. □

Theorem 9 (soundness of Rule 5) *If a constraint (R, G) is added by Rule 5 to the compassion set of a diagram A, obtaining diagram A', then $Runs(A) = Runs(A')$, and therefore $\mathcal{L}(A) = \mathcal{L}(A')$.*

Proof. Again, it is immediate that $Runs(A') \subseteq Runs(A)$. To show the reverse containment, assume towards the contradiction that the conditions of Rule 5 are satisfied, and that there is a run $\sigma : (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots \in Runs(A)$ that visits infinitely often R taking only finitely many edges in G. Thus, there is $k \in \mathbb{N}$ such that $(v_i, v_{i+1}) \notin G$ for all $i \geq k$. By (5), the value of δ does not increase along σ after k.

If σ visits $V - R$ infinitely often, it must infinitely often return to R, and by (7) the value of δ beyond k decreases infinitely often, against the hypothesis that the domain of δ is well-founded. Thus, there is $m \geq k$ such that σ stays in R forever beyond position m. Once σ is confined to R, the proof follows that of Theorem 8, due to the similarity between conditions (7) and (3). □

The completeness result for the compassion rule will be stated and proved in Section 6.

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

Rule 6 (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* $v \in V_1 - U_1$, $\widehat{\tau}_1(u, v) \equiv$ *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* $\mathcal{C}_2$*).* $\square$

The soundness of the rule follows from the observation that no run of the diagram can contain vertices in U if the conditions of the rule are satisfied. 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.

This rule can be used in conjunction with Rule 4 to prune from the diagram *livelocking vertices*. A livelocking vertex is a vertex from which there are no runs along which time diverges. This result is a special case of Lemma 19 of Section 6.3.

5 Proving Temporal Properties

The specification language we consider in this paper is the linear-time temporal logic TL_s. The formulas of TL_s are obtained by combining first-order logic formulas by means of propositional connectives, of the future temporal operators $\circ$ (next), $\Box$ (always), $\Diamond$ (eventually), $\mathcal{U}$ (until), and of the corresponding past ones $\ominus$, $\boxminus$, $\diamondminus$ and $\mathcal{S}$ [MP93]. Thus, in a TL_s formula there are no occurrences of temporal operators inside the scope of logical quantifiers.

Given a PTS $\mathcal{S}$ and a specification $\varphi \in TL_s$, we say that $\mathcal{S}$ satisfies φ, written $\mathcal{S} \models \varphi$, if all computations of $\mathcal{S}$ satisfy φ. Denoting by $\mathcal{L}(\varphi)$ the set of behaviors that satisfy φ, this can also be written $\mathcal{L}(S) \subseteq \mathcal{L}(\varphi)$. A proof of $\mathcal{S} \models \varphi$ can

be constructed by producing a chain of diagram transformations

$$hd(\mathcal{S}) = A_0 \Rightarrow A_1 \Rightarrow \cdots \Rightarrow A_n$$

such that $\mathcal{L}(A_n) \subseteq \mathcal{L}(\varphi)$. In fact, $\mathcal{L}(\mathcal{S}) = \mathcal{L}(hd(\mathcal{S}))$ by construction, and $\mathcal{L}(A_i) \subseteq \mathcal{L}(A_{i+1})$ for all $0 \leq i < n$, since diagram transformations preserve language containment. In this section, we present an algorithm to establish whether $\mathcal{L}(A_n) \subseteq \mathcal{L}(\varphi)$, also written $A_n \models \varphi$. Since the verification problem for PTSs is undecidable, the algorithm is not a decision procedure. Rather, the algorithm provides either a positive answer to the verification problem $\mathcal{S} \models \varphi$, or guidance for the search for counterexample or for the further extension of the chain of transformations.

5.1 Algorithmic Checking of Diagrams

Given a diagram A and a formula $\varphi \in TL_s$, the algorithm provides either a positive answer to $A \models \varphi$, 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 $N_{\neg\varphi}$ that accepts all the computations that do not satisfy φ [VW86,Saf88,Saf92]. The automaton is a first-order version of a classical Streett automaton.

Definition 10 (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 $E \subseteq V \times V$; $Q \subseteq V$ is the set of *initial vertices*, and $\mathcal{B}$, called the *acceptance condition*, is a set of pairs $(P, R) : P, R \subseteq V$. Again, a location of N is a pair $(v, s) : v \in V, s \models \rho(v)$ composed of a vertex and of a corresponding state. 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 $i \geq 0$, $(v_i, v_{i+1}) \in E$;
(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 $Runs(N)$ (resp. $\mathcal{L}(N)$). □

To show that no behavior of A satisfies $\neg\varphi$, the algorithm constructs the *graph product* $A \otimes N_{\neg\varphi}$ and checks that no infinite path in it corresponds to a computation of both A and $N_{\neg\varphi}$. 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 2 (graph product) *Given diagram $A = (\mathcal{V}, U, \rho_A, \theta, \tau, \mathcal{J}, \mathcal{C})$ and Streett automaton $N_{\neg\varphi} = (\mathcal{V}, (V, E), \rho_N, Q, \mathcal{B})$, the* graph product $A \otimes N_{\neg\varphi} = (W, Z, H)$ *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) $Z = \{(u, v) \in W \mid v \in Q \text{ and } \nvdash \neg(\theta(u) \wedge \rho_N(v))\}$;
(3) $H = \Big\{\Big((u_1, v_1), (u_2, v_2)\Big) \in W \times W \;\Big|\; (v_1, v_2) \in E \text{ and } \nvdash \neg\Big(\widehat{\tau}(u_1, u_2) \wedge \rho_N(v_1) \wedge \rho'_N(v_2)\Big)\Big\}$. □

To show that there is no infinite path in the product that corresponds to a computation of both A and $N_{\neg\varphi}$, we check that every infinite path in (W, H) starting from Z violates either a fairness constraint of A or a pair in the acceptance condition of $N_{\neg\varphi}$. 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 $(R, G) \in \mathcal{J}$, if $X \subseteq R \times V$ 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 $(R, G) \in \mathcal{C}$, 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 $((u_1, v_1), (u_2, v_2)) \in H$;
(3) 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 $A \models \varphi$. This check can be done in time polynomial in $|W|$ using efficient graph algorithms.

Theorem 11 (diagram checking) *Given a diagram A and a specification $\varphi \in TL_s$, let $A \otimes N_{\neg\varphi} = (W, Z, H)$. If all SCSs of (W, H) that are reachable in (W, H) from Z are not admissible, then $A \models \varphi$.*

Finally, the following theorem states that the verification methodology presented in this paper is complete with respect to first-order linear-time temporal logic.

Theorem 12 (completeness for TL_s) *Given a PTS $\mathcal{S}$ and a specification $\varphi \in TL_s$, if $\mathcal{S} \models \varphi$ then there is a chain of transformations $hd(\mathcal{S}) \stackrel{*}{\Rightarrow} A$ such that $A \models \varphi$ can be proved using Theorem 11.*

This theorem will be proved in Section 6, along with the other completeness results. Note that while the proof procedure $\vdash$ used for the construction of the graph product is incomplete and terminating, the proof of the assertions arising from the transformations may require general first-order reasoning.

5.2 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 [SUM96]. Given an admissible and reachable SCS X of $(W, Z, H) = A \otimes N_{\neg\varphi}$, 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.

5.2.1 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 $\mathcal{S} \not\models \varphi$.

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

5.2.2 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:

(1) either there is no path in Y_r from an initial vertex of Z to Y;
(2) 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 7 *Using the algorithm presented in this section, it is possible to check that diagram A_3 of Figure 4 satisfies the specification* $(65 \leq x \leq 75) \Rightarrow \Box(65 \leq x \leq 75)$.

On the other hand, if we check A_3 against the specification $\Diamond(65 \leq x \leq 75)$ *we obtain two CCPs, corresponding to the SCSs* $\{v_0^3\}$, $\{v_1^3\}$. *To prove the specification, we must thus show that v_0^3 and v_1^3 are not reachable (which is obviously not possible), or that a run cannot be forever confined to v_0^3 or v_1^3. This is shown by adding the justice constraint* $(\{v_0^3, v_1^3\}, \{(v_1^3, v_2^3)\})$ *as in Example 3. The diagram-checking algorithm shows that the resulting diagram A_4 satisfies* $\Diamond(65 \leq x \leq 75)$. $\Box$

6 Completeness Results

In this section we prove Theorem 12, which expresses the completeness of the methodology. Like similar completeness results [MP91], our result is relative to first-order reasoning: in other words, it is relative to the existence of an oracle that is able to prove all valid assertions. We introduce the following definitions.

Definition 13 Consider a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$.

(1) A *run prefix* of A is a finite sequence $(v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots, (v_n, s_n)$ of locations of A satisfying Conditions 1 and 2 of Definition 3.
(2) We say that A is *non-livelocking* if every run prefix of A can be extended to a run of A.
(3) We say that A is *globally reachable* if for every location $(v, s) \in loc(A)$ there is a run prefix $(v_0, s_0), \ldots, (v_n, s_n)$ ending with $(v_n, s_n) = (v, s)$.
(4) We say that A is *deterministic* if $\theta(u) \wedge \theta(v) \leftrightarrow$ *false* and $\widehat{\tau}(u, v) \wedge \widehat{\tau}(u, w) \leftrightarrow$ *false* for all $u, v, w \in V$. $\Box$

The crucial step in the completeness proof for the methodology consists in proving that, if the language of a PTS is contained into that of a determinis-

tic diagram, then there is a chain of transformations beginning with the initial diagram of the PTS and ending with the given deterministic diagram. This fact, together with known results from the theory of omega-automata, will easily lead to the result. The proof of the existence of the chain of transformations consists in several steps. First, we prove relative completeness results for the justice and compassion rules. Next, we show how livelocking locations can be eliminated using the simulation, justice, and pruning rules. Finally, by combining these results we can prove the existence of the chain of transformations.

6.1 Justice

Our first theorem states that if a diagram is totally reachable and has empty set of constraints, then every J-compatible constraint can be added with a single application of Rule 4.

Theorem 14 (completeness, justice) *If a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ is globally reachable, and (R, G) is J-compatible with $A' = (\mathcal{V}, V, \rho, \theta, \tau, \emptyset, \emptyset)$, then (R, G) can be added to the set of justice constraints of A with one application of Rule 4.*

The proof uses the following lemma, that is proved in [LPS81].

Lemma 15 *Let $S, \sqsupset$ be a well-founded ordering. Then there exists a ranking function $\delta_0 : S \mapsto Ord$, where Ord is the set of ordinals, such that:*

(1) for all $s, s' \in S$, we have $s \sqsupset s' \rightarrow \delta_0(s) > \delta_0(s')$;
(2) for all s, if $\delta_0(s) = \alpha$ and $\alpha \sqsupset \beta$, then there exists $s' \in S$ such that $\delta_0(s') = \beta$ and $s \sqsupset s'$.
(3) for all $s, s' \in S$, if $s \sqsupset s''$ implies $s' \sqsupset s''$ for all $s'' \in S$, then $\delta_0(s) \geq \delta_0(s'')$.

Proof of Theorem 14. The proof is based on the completeness proofs for verification rules presented in [MP91]. For $n \geq 0$, we call a finite sequence of locations $(v_0, s_0), (v_1, s_1), \ldots, (v_n, s_n)$ a *j-path* if all its vertices are contained in R, and for $0 \leq i \leq n$, we have $(v_i, v_{i+1}) \notin G$ and $(s_i, s_{i+1}) \models \exists \Delta \,.\, \tau(v_i, v_{i+1})$. Let $\widetilde{R} = \{(v, s) \in loc(A) \mid v \in R\}$, and define the relation $\sqsupset \subseteq \widetilde{R} \times \widetilde{R}$ by

$$(v, s) \sqsupset (v', s') \quad \textit{iff} \quad \left(\begin{array}{l} s'[\![T]\!] - s[\![T]\!] \geq 1, \\ \textit{and there is a j-path from } (v, s) \textit{ to } (v', s') \end{array} \right)$$

The relation $\sqsupset$ is well-founded on $\widetilde{R}$. In fact, assume towards the contradiction that there is an infinite descending chain $\ell_0 \sqsupset \ell_1 \sqsupset \ell_2 \sqsupset \cdots$ of locations of $\widetilde{R}$. Since the diagram is globally reachable, there is a j-path η_0 from an initial location to ℓ_0, and for all $i \in \mathbb{N}$, there is a j-path η_{i+1} from ℓ_i to ℓ_{i+1} of temporal duration at least 1, since $\ell_i \sqsupset \ell_{i+1}$. Thus, the infinite path $\eta_0, \eta_1, \eta_2, \ldots$ is a run of diagram A_1 that stays forever in $\widetilde{R}$ while never following an edge in G, contradicting the J-compatibility of (R, G).

Using Lemma 15, from $\sqsupset$ we define a ranking function $\delta : \widetilde{R} \mapsto \mathit{Ord}$, where Ord is the set of ordinals, that satisfies the following properties:

(1) $\ell \sqsupset \ell' \rightarrow \delta(\ell) > \delta(\ell')$;
(2) if $\ell' \sqsupset \ell'' \rightarrow \ell \sqsupset \ell''$ for all ℓ'', then $\delta(\ell) \geq \delta(\ell')$.

If there is a j-path from ℓ to ℓ' in $\widetilde{R}$, then $\delta(\ell) \geq \delta(\ell')$. In fact, let η be the j-path, and consider any location $\ell'' \in \widetilde{R}$. If $\ell' \sqsupset \ell''$, there must be a j-path η' from ℓ' to ℓ'' of duration at least 1, and because of the existence of j-path $\eta\eta'$ this implies that $\ell \sqsupset \ell''$. Property 2 then leads to the desired conclusion.

Given a j-path $\eta = (v_0, s_0), (v_1, s_1), \ldots, (v_n, s_n)$ we define $\mathrm{len}(\eta) = s_n[\![T]\!] - s_0[\![T]\!]$, and we say that η is *level* if $\delta(v_0, s_0) = \delta(v_1, s_1) = \cdots = \delta(v_n, s_n)$. Given $\ell \in \widetilde{R}$, we denote by $\Upsilon(\ell)$ the set of level j-paths from ℓ, and we define

$$\gamma(\ell) = \sup_{\eta \in \Upsilon(\ell)} \mathrm{len}(\eta) \ . \tag{8}$$

To see that $\gamma : \widetilde{R} \mapsto \mathbb{R}^+$ is well-defined, note that if $\eta : \ell_0, \ell_1, \ldots, \ell_n$ is a level j-path, then $\mathrm{len}(\eta) < 1$, for otherwise $\ell_0 \sqsupset \ell_n$, leading to $\delta(\ell_0) > \delta(\ell_n)$ by Property 1 above. The functions δ, γ can then be extended to $\mathit{loc}(A)$ by assigning an arbitrary value to locations not in $\widetilde{R}$.

Using the methods of [MP91, §7.2.2], from δ we can construct a labeling $\widehat{\delta}$ representing δ that labels each $v \in V$ with an assertion $\widehat{\delta}(v)$ belonging to our assertion language. We can then easily construct the labeling $\widehat{\gamma}$ representing γ from its definition (8), since the language includes fixpoints. We claim that these labelings $\widehat{\delta}$ and $\widehat{\gamma}$ satisfy assertion (3) for all $u, v \in R$ such that $(u, v) \notin G$. To see this, consider two locations $\ell = (u, s) \in \widetilde{R}$, $\ell' = (v, t) \in \widetilde{R}$, and assume that $(s, t) \models \exists \Delta \,.\, \tau(u, v)$. Let $d = t[\![T]\!] - s[\![T]\!]$ be the value of parameter Δ that satisfies the quantifier. We have $\delta(\ell) \geq \delta(\ell')$, since there is a j-path from ℓ to ℓ'. If $\delta(\ell) > \delta(\ell')$, assertion (3) holds. Else, $\delta(\ell) = \delta(\ell')$, and for any $\eta \in \Upsilon(\ell')$, the j-path ℓ, η is also level, and $\mathrm{len}(\ell, \eta) = \mathrm{len}(\eta) + d$: thus, $\gamma(\ell) \geq \gamma(\ell') + d$, and again assertion (3) holds. $\square$

Unlike Rule 4, which is complete for adding J-compatible constraints, Rule 5 is not complete for adding C-compatible constraints, if used in isolation. To show that the methodology, in its whole, is complete for adding compassion constraints, we proceed in two steps. First, we introduce a more complex rule that is complete by itself for adding C-compatible constraints. Then, we show that each application of this rule can be mimicked by the application of Rule 2 to split some vertices, followed by Rule 5 to add the C-compatible constraint, followed by Rule 1 to merge back the vertices.

The new rule requires the use of a family of auxiliary assertions $\{\varphi(v)\}_{v \in V}$ over $\mathcal{V}$, used to represent a set of locations $\{(v, s) \in loc(A) \mid s \models \varphi(v)\}$. This set of locations is a "superset" of R (precisely, is a superset of the locations in R), and it consists of locations that play the same role of R in leading to the goal G.

Rule 7 *(compassion, with auxiliary assertions) Given a diagram* $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ *and a constraint* $(R, G) : R \subseteq V, G \subseteq V \times V$*, assume that there are*

(1) a family of assertions $\{\varphi(v)\}_{v \in V}$ *over* $\mathcal{V}$*, such that* $\varphi(v) = true$ *for all* $v \in R$*;*
(2) a ranking labeling $\widehat{\delta}$ *and a delay labeling* $\widehat{\gamma}$*,*

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

$$\widehat{\tau}(u, v) \quad \rightarrow \quad \widehat{\delta}(u) \geq \widehat{\delta}'(v) \tag{9}$$

$$\varphi(u) \wedge \widehat{\tau}(u, v) \quad \rightarrow \quad \widehat{\delta}(u) > \widehat{\delta}'(v) \vee \neg\varphi'(v) \vee \widehat{\gamma}(u) \geq \widehat{\gamma}'(v) + \Delta \tag{10}$$

$$\neg\varphi(u) \wedge \widehat{\tau}(u, v) \quad \rightarrow \quad \widehat{\delta}(u) > \widehat{\delta}'(v) \vee \neg\varphi'(v) \tag{11}$$

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

The soundness and completeness of this new rule are expressed by the following theorems.

Theorem 16 (soundness of Rule 7) *If a constraint* (R, G) *is added by Rule 7 to the set of compassion constraints of a diagram* A*, obtaining diagram* A'*, then* $Runs(A) = Runs(A')$*, and therefore* $\mathcal{L}(A) = \mathcal{L}(A')$.

Proof. Again, it is immediate that $Runs(A') \subseteq Runs(A)$. To show the reverse containment, assume towards the contradiction that the conditions of Rule 7 are satisfied, and that there is a run $\sigma : (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots \in Runs(A)$ that visits infinitely often R taking only finitely many edges in G. Thus, there

is $k \in \mathbb{N}$ such that $(v_i, v_{i+1}) \notin G$ for all $i \geq k$. By (9), the value of δ does not increase along σ after k.

Let $B_1 = \{(v,s) \in loc(A) \mid s \models \varphi(v)\}$, $B_0 = loc(A) - B_1$, and note that no vertices of R are part of locations of B_0. If σ visits B_0 infinitely often, it must infinitely often return to B_1 to visit R, and by (11) the value of δ beyond k decreases infinitely often, against the hypothesis that the domain of δ is well-founded. Thus, there is $m \geq k$ such that σ stays in B_1 forever beyond position m. Once σ is confined to B_1, the proof follows that of Theorem 8, due to the similarity between conditions (9), (10), and condition (3). $\Box$

Theorem 17 (completeness, compassion) *If a diagram* $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{C}, \mathcal{J})$ *is globally reachable, and* (R, G) *is C-compatible with* $A' = (\mathcal{V}, V, \rho, \theta, \tau, \emptyset, \emptyset)$, *then* (R, G) *can be added to the set of compassion constraints of* A *with one application of Rule 7.*

Proof. The proof of the theorem is related to the proof of Theorem 14, from which we borrow some notation. For $n \geq 0$, we call a finite sequence of locations $(v_0, s_0), (v_1, s_1), \ldots, (v_n, s_n)$ a *g-free-path* if no pair (v_i, v_{i+1}) of consecutive vertices is in G, and $(s_i, s_{i+1}) \models \exists \Delta \,.\, \tau(v_i, v_{i+1})$ for all $0 \leq i < n$. A g-free-path is a *c-path* if it contains at least one vertex in R. Define the relation $\sqsupset \subseteq loc(A) \times loc(A)$ by

$$(v,s) \sqsupset (v',s') \quad \textit{iff} \quad \left(\begin{array}{l} s'[\![T]\!] - s[\![T]\!] \geq 1, \\ \textit{and there is a c-path from } (v,s) \textit{ to } (v',s') \end{array} \right)$$

Since (R, G) is C-compatible with A, relation $\sqsupset$ is well-founded on $loc(A)$. In fact, reasoning as in the previous proof it can be shown that any infinite descending chain provides a counterexample to the C-compatibility of (R, G). Again, on the basis of $\sqsupset$ we can define a ranking function $\delta : loc(A) \mapsto Ord$. For all locations $\ell \in loc(A)$ let

$$\Upsilon(\ell) = \{\eta \mid \eta \textit{ is a level c-path from } \ell\} \,,$$

and define

$$B_1 = \{\ell \in loc(A) \mid \Upsilon(\ell) \neq \emptyset\} \qquad B_0 = loc(A) - B_1 \,.$$

For a location $(v,s) \in loc(A)$, if $v \in R$ then $(v,s) \in B_1$, since $\Upsilon(v,s)$ contains at least the single-location c-path (v,s). Consider a family of assertions $\{\varphi(v)\}_{v \in V}$ such that

$$v \in R \;\rightarrow\; \varphi(v) = \mathit{true} \qquad (v,s) \in B_1 \;\leftrightarrow\; s \models \varphi(v) \wedge \rho(v)$$

for all $v \in V$. These assertions characterize B_1 and, by complementation, also B_0. Define the function $\gamma : loc(A) \mapsto \mathbb{R}^+$ by

$$\ell \in B_1 : \ \gamma(\ell) = \sup_{\eta \in \Upsilon(\ell)} \mathrm{len}(\eta) \qquad\qquad \ell \in B_0 : \ \gamma(\ell) = 0 \ .$$

As in the proof of Theorem 14, it is possible to construct labelings $\widehat{\delta}$, $\widehat{\gamma}$ corresponding to δ and γ and that are expressible in the assertion language [MP91, §7.2.2]. Furthermore, it can be shown that the family of assertions $\{\varphi(v)\}_{v \in V}$ can be constructed on the basis of $\widehat{\delta}$.

We claim that the family of assertions $\{\varphi(v)\}_{v \in V}$, together with the labelings $\widehat{\delta}$ and $\widehat{\gamma}$ corresponding to δ and γ, satisfy the conditions of Rule 7. In fact, let (u, s) and (v, t) be two locations such that $(u, v) \notin G$ and $(s, t) \models \exists \Delta \, . \, \tau(u, v)$, and consider assertions (9), (10) and (11).

(1) Assertion (9) is proved by showing that, for any $\ell \in loc(A)$, we have that $(v, t) \sqsupset \ell$ implies $(u, s) \sqsupset \ell$. The conclusion follows from the properties of δ.
(2) Consider assertion (10), and assume $(u, s) \in B_1$. If $\delta(u, s) > \delta(v, t)$, the conclusion follows. Otherwise, $(u, s), (v, t)$ is a level g-free-path. If $\Upsilon(v, t) = \emptyset$, then $(v, t) \in B_0$, so $t \models \neg\varphi(v)$ and the conclusion follows again. Else, let $d = t[\![T]\!] - s[\![T]\!]$ be the time elapsed from s to t, and consider any $\eta \in \Upsilon(v, t)$. Path $(u, s), \eta$ is a level c-path, so $(u, s), \eta \in \Upsilon(u, s)$; moreover $\mathrm{len}((u, s), \eta) = \mathrm{len}(\eta) + d$. By definition of γ, $\gamma(u, s) \geq \gamma(v, t) + d$, and the conclusion follows once more.
(3) Consider assertion (11), and assume $(u, s) \in B_0$. If $\delta(u, s) > \delta(v, t)$, the conclusion follows. Otherwise, $(u, s), (v, t)$ is a level g-free-path. Since $\Upsilon(u, s) = \emptyset$, we have $\Upsilon(v, t) = \emptyset$. In fact, the existence of $\eta \in \Upsilon(v, t)$ would imply that $(u, s)\eta \in \Upsilon(u, s)$, contradicting $(u, s) \in B_0$. Since $\Upsilon(v, t) = \emptyset$, we have $(v, t) \in B_0$, and thus $t \models \neg\varphi(v)$, from which the conclusion follows. □

The following theorem states that we can mimic an application of Rule 7 with one application of Rule 5 and two applications of Rule 1.

Theorem 18 (simulating Rule 7 by Rules 5 and 1) *Every application of Rule 7 can be mimicked by one application of Rule 5 and two applications of Rule 1.*

Proof. Let $A = (\mathcal{V}, V, \rho, \theta, \tau, \emptyset, \emptyset)$ be the original diagram, let (R, G) be the constraint added by Rule 7, and let δ and $\{\varphi(v)\}_{v \in V}$ be the ranking function

and the auxiliary assertions used by the rule. First, we use Rule 2 to split each vertex $v \in V$ into two new vertices $(v,1)$, $(v,2)$, where

$$\rho(v,1) = \rho(v) \wedge \varphi(v) \qquad \rho(v,2) = \rho(v) \wedge \neg\varphi(v) \ ,$$

obtaining diagram A_1. Second, we apply to A_1 Rule 5, to add the constraint (R', G') defined by

$$R' = V \times \{1\}$$
$$G' = \left\{((u,i),(v,j)) \;\middle|\; (u,v) \in G \wedge i,j \in \{1,2\}\right\}$$

obtaining diagram A_2. The application of the rule relies on the ranking labeling defined by $\widehat{\delta}(v,1) = \widehat{\delta}(v,2) = \widehat{\delta}(v)$. It can be seen that Conditions (9)–(11) of Rule 7 imply that the corresponding Conditions (5)–(7) of Rule 5 hold.

Third, we apply Rule 1 to merge vertices $(v,1)$ and $(v,2)$ back into a single vertex v labeled by $\rho(v)$ again, for all $v \in V$. The function ν corresponding to this merge is specified by $\nu(v,1) = \nu(v,2) = \nu(v)$, for all $v \in V$. The structure of the resulting diagram A_3 will be identical to the structure of A, aside for the presence of the additional constraint (R,G). When applying Rule 1 to merge the vertices, we can justify the constraint (R,G) in A_3 on the basis of the constraint (R',G') of A_2, according to Condition 3. This concludes the proof. □

6.3 Eliminating Livelocking Locations

The following lemma shows how to use the simulation, justice and pruning rules to eliminate the unreachable or livelocking locations of a diagram.

Lemma 19 *Given a diagram $A = (\mathcal{V}, V, \rho, \theta, \tau, \emptyset, \emptyset)$, there is a chain of transformations $A \stackrel{*}{\Rightarrow} B$, where $B = (\mathcal{V}, U, \widetilde{\rho}, \widetilde{\theta}, \widetilde{\tau}, \emptyset, \emptyset)$, and where B is globally reachable and non-livelocking, and $|V| = |U|$.*

Proof. Let $A = (\mathcal{V}, V, \rho, \theta, \tau, \emptyset, \emptyset)$ be the original diagram. Let $F \subseteq loc(A)$ be the set of reachable locations of A. By the methods of [MP91], it is possible to construct a family of assertions $\{\psi(v)\}_{v \in V}$ such that $s \models \psi(v)$ iff $(v,s) \in F$. Using the simulation rule, it is possible to transform A into $A_1 = (\mathcal{V}, V, \bar{\rho}, \theta, \tau, \emptyset, \emptyset)$, where $\bar{\rho}(u) = \rho(u) \wedge \psi(u)$ for all $u, v \in V$.

To obtain a non-livelocking diagram, define the set $E \subseteq loc(A_1)$ as the set of locations of A_1 that appear in some run of A_1. If $\ell \in loc(A_1) - E$, then ℓ is livelocking. Moreover, for all $\ell \in loc(A_1) - E$ and all $\ell' \in E$, there can be no

transition from ℓ to ℓ'. We want to show that there is a family of assertions $\{\varphi(v)\}_{v\in V}$ defining E, i.e. such that $s \models \varphi(v)$ iff $(v,s) \in E$, for all $v \in V$. To this end, we define the relation $\sqsupset \subseteq loc(A) \times loc(A)$ by

$$(v,s) \sqsupset (v',s') \quad \textit{iff} \quad \left(\begin{array}{l} s'[\![T]\!] - s[\![T]\!] \geq 1, \\ \textit{and there is a path from } (v,s) \textit{ to } (v',s') \end{array} \right)$$

We then define the predicates $U(\ell)$ and $H(\ell)$ for $\ell \in loc(A_1)$ by the formulas

$$U(\ell) = \forall \ell' \,.\, \ell \not\sqsupset \ell' \tag{12}$$

$$H(\ell) = \mu Q(\ell) \,.\, \left[U(\ell) \vee \forall \ell' \,.\, (\ell \sqsupset \ell' \rightarrow Q(\ell'))\right], \tag{13}$$

where μ denotes the least fixpoint operator. To show that $H(\ell)$ iff $\ell \notin E$, we reason as follows.

(1) If $H(\ell)$ holds, then there is no infinite sequence of locations $\ell = \ell_0 \sqsupset \ell_1 \sqsupset \ell_2 \sqsupset \cdots$. This implies that ℓ is livelocking, and hence it cannot appear in any run of A_1, so that $\ell \notin E$.
(2) Conversely, assume that $\neg H(\ell)$ holds. From (13), we know that there is at least one ℓ_1 such that $\ell = \ell_0 \sqsupset \ell_1$ and $\neg H(\ell_1)$ holds. Continuing in this way, we can construct an infinite sequence $\ell = \ell_0 \sqsupset \ell_1 \sqsupset \ell_2 \sqsupset \cdots$. Since ℓ is reachable from some initial location (since A_1 is globally reachable), this implies the existence of a run that includes ℓ, from which we conclude $\ell \in E$.

The existence of the family of assertions $\{\varphi(v)\}_{v\in V}$ is then a consequence of the fact that the relation $\sqsupset$ and the other constructs of (12) and (13) can be expressed in our logic [MP91].

By a second application of the simulation rule, it is possible to transform A_1 into $A_2 = (\mathcal{V}, V \times \{1,2\}, \widetilde{\rho}, \theta, \widetilde{\tau}, \emptyset, \emptyset)$, where:

(1) For all $v \in V$, we let $\widetilde{\rho}(v,1) = \bar{\rho}(v) \wedge \varphi(v)$ and $\widetilde{\rho}(v,2) = \bar{\rho}(v) \wedge \neg\varphi(v)$.
(2) For all $v \in V$, we let $\widetilde{\theta}(v,1) = \widetilde{\theta}(v,2) = \theta(v)$.
(3) For all $u, v \in V$ and $i, j \in \{1,2\}$, we let

$$\widetilde{\tau}((u,i),(v,j)) = \begin{cases} \mathit{false} & \text{if } i = 2 \text{ and } j = 1; \\ \tau(u,v) & \text{otherwise.} \end{cases}$$

The function $\nu : V \mapsto V \times \{1,2\}$ is defined by $\nu(u) = \{u\} \times \{1,2\}$ for all $u \in V$.

Note that the constraint $(V \times \{2\}, \emptyset)$ is J-compatible with A_2, since no run of A_2 contains vertices in $V \times \{2\}$. By Theorem 14, using Rule 4 we can transform A_2 into $A_3 = (\mathcal{V}, V \times \{1,2\}, \widetilde{\rho}, \widetilde{\theta}, \widetilde{\tau}, \{(V \times \{2\}, \emptyset)\})$. Finally, using Rule 6 we

can prune from A_3 all the vertices in $V \times \{2\}$, obtaining B. By construction, B is globally reachable, non-livelocking, and has empty justice and compassion sets; moreover, it has the same number of vertices as A. $\Box$

6.4 General Completeness

Combining the previous results we obtain the following theorem, that expresses the completeness of the transformation rules presented in the paper. This theorem will also lead to the completeness of the proposed verification methodology for linear temporal logic specifications, as we will see in the next section.

Theorem 20 (completeness for PTS) *If $\mathcal{S}$ is a PTS and A is a deterministic diagram over $\mathcal{V}$ such that $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(A)$, the simulation, justice, compassion and pruning rules (Rules 1, 4, 5, and 6) enable the construction of a chain of transformations $hd(\mathcal{S}) \stackrel{*}{\Rightarrow} A$.*

Proof. Let $A_0 = hd(\mathcal{S}) = (\mathcal{V}, \{u_0\}, \rho(u_0) = true, \theta_0, \tau_0, \emptyset, \emptyset)$ and $A = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$, where $\mathcal{J} = \{(R_1, G_1), \ldots, (R_m, G_m)\}$, $\mathcal{C} = \{(R_{m+1}, G_{m+1}), \ldots, (R_n, G_n)\}$.

By Lemma 19, A_0 can be transformed into a single-vertex, globally reachable, non-livelocking diagram $A_1 = (\mathcal{V}, \{u_1\}, \rho_1, \theta_1, \tau_1, \emptyset, \emptyset)$ such that $\mathcal{L}(A_0) = \mathcal{L}(A_1)$.

Since diagram A is deterministic, for every sequence of states $\xi : s_0, s_1, s_2, \ldots$ there is at most one sequence of locations $L(\xi) = (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots$ that satisfies Conditions 1 and 2 of Definition 3. Define $Runs_{\mathcal{S}}(A) = \{L(\xi) \mid \xi \in \mathcal{L}(\mathcal{S})\}$, and let $E \subseteq loc(A)$ be the set of locations that appear in some run in $Runs_{\mathcal{S}}(A)$. Again, it is possible to define a family of assertions $\{\varphi(v)\}_{v \in V}$ such that $s \models \varphi(v)$ iff $(v, s) \in E$, for all $(v, s) \in loc(A)$. Consider the diagram $A_2 = (\mathcal{V}, V, \rho_2, \theta_2, \tau_2, \emptyset, \emptyset)$ having the same set of vertices of A, and defined as follows, for all $v, v' \in V$:

(1) $\rho_2(v) = \rho(v) \wedge \varphi(v)$;
(2) $\theta_2(v) = \theta(v) \wedge \theta_1(v)$;
(3) $\tau_2(v, v') = \tau(v, v') \wedge \tau_1(v, v)$.

To show that $\mathcal{L}(A_1) = \mathcal{L}(A_2)$, we show both $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ and $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$.

(1) $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$. Consider a computation $\sigma \in \mathcal{L}(A_1)$. Since $\mathcal{L}(\mathcal{S}) = \mathcal{L}(A_0) = \mathcal{L}(A_1)$, by hypothesis we have $\sigma \in \mathcal{L}(A)$, and there is a run

$\widehat{\sigma} \in \mathit{Runs}_{\mathcal{S}}(A)$ that corresponds to σ. By construction, all locations of $\widehat{\sigma}$ are present in A_2, and all transitions of $\widehat{\sigma}$ are possible in A_2, by definition of τ_2. Thus, $\widehat{\sigma} \in \mathit{Runs}(A_2)$, and $\sigma \in \mathcal{L}(A_2)$, as was to be shown.

(2) $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$. Consider a computation $\widehat{\sigma} \in \mathcal{L}(A_2)$ arising from a run $\sigma \in \mathcal{L}(A_2)$. By construction of θ_2 and τ_2, we see that every state-transition possible along $\widehat{\sigma}$ is also possible in A_1. Since the set of fairness constraints of both diagrams is empty, we conclude that $\widehat{\sigma} \in \mathcal{L}(A_1)$, as was to be shown.

There is a simulation transformation that transforms A_1 into A_2, based on the mapping $\nu : \{u_1\} \mapsto V$. To see this, consider the conditions of Rule 1.

(1) Since $\mathcal{L}(A_1) \subseteq \mathcal{L}(A)$ and A_1 is non-livelocking, it must be $\theta_1(u_1) \rightarrow \bigvee_{u \in V} \theta(u)$. The result then follows from the definition of θ_2.

(2) Let (u_1, s) be a location of A_1 from which there is a transition to (u_1, s'), and let (v, s) be a location of A_2 related to (u_1, s). Since A_1 is globally reachable and non-livelocking, there is a there is a run $\sigma' \in \mathcal{L}(A_1)$ containing (u_1, s). By construction of A_2, this run induces a run $\sigma \in \mathcal{L}(A_2)$ containing (v, s). Let $\eta, (v, s)$ be the run prefix of σ leading to (v, s), and let $\eta', (u_1, s)$ be the corresponding prefix of σ' leading to (u_1, s). Since there is a transition from (u_1, s) to (u_1, s'), and since A_1 is non-livelocking, the run prefix η' can be extended to a run $\eta', (u_1, s), (u_1, s'), \xi' \in \mathit{Runs}(A_1)$. Since A_2 is deterministic, $\eta, (v, s)$ is the only run prefix of A_2 that corresponds to $\eta', (u_1, s)$, and since $\mathcal{L}(A_1) = \mathcal{L}(A_2)$, $\eta, (v, s)$ can also be extended to a run $\eta, (v, s), (v', s'), \xi \in \mathit{Runs}(A_2)$, for some $(v', s') \in \mathit{loc}(A_2)$ and ξ. This shows that indeed A_2 can take a transition from (v, s) to a location (v', s') related to (u_1, s'), as Condition 2 of Rule 1 requires.

(3) Immediate, since A_2 has empty set of fairness constraints.

For $1 \leq i \leq m$, constraint (R_i, G_i) is J-compatible with A_2, and for $m + 1 \leq i \leq n$, constraint (R_i, G_i) is C-compatible with A_2. To see this, assume towards the contradiction that for some $1 \leq i \leq n$ there is a run $\sigma \in \mathit{Runs}(A_2)$ that does not satisfy (R_i, G_i) and let $\widehat{\sigma}$ be the computation of A arising from σ. Since $\mathcal{L}(A_2) = \mathcal{L}(A_1) = \mathcal{L}(\mathcal{S})$, we have $\widehat{\sigma} \in \mathcal{L}(\mathcal{S})$, and since A_2 and A are deterministic, σ is the only run of A_2 that corresponds to $\widehat{\sigma} \in \mathcal{L}(\mathcal{S})$. Thus, if σ does not satisfy (R_i, G_i), $\sigma \notin \mathit{Runs}(A)$, and $\widehat{\sigma} \notin \mathcal{L}(A)$, contradicting $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(A)$. Thus, by Theorems 14 and 17 we can construct a series of transformations $A_2 \Rightarrow A_3 \Rightarrow \cdots \Rightarrow A_{n+2}$, where:

(1) for $1 \leq i \leq m$, A_{i+2} is obtained by adding constraint (R_i, G_i) to the justice set of A_{i+1} using Rule 4;

(2) for $m + 1 \leq i \leq n$, A_{i+2} is obtained by adding constraint (R_i, G_i) to the compassion set of A_{i+1} using Rules 5 and 1, as described by Theorem 18.

The final step consists in proving that there is a simulation transformation based on the identity mapping between A_{m+2} and A. This is a simple consequence of the fact that A_{m+2} and A share the same set of fairness constraints, and the labelings of A_{m+2} are stronger (i.e. they imply) the corresponding labelings of A. $\Box$

6.5 Completeness for Temporal Logic Specifications

The final step in the completeness argument consists in proving Theorem 12, which states the completeness of the proposed verification methodology with respect to specifications written in TL_s. To prove Theorem 12 we need the following construction, that given a Streett automaton M produces a hybrid diagram $hd(M)$ such that $\mathcal{L}(M) = \mathcal{L}(hd(M))$.

Construction 3 (from Streett automaton to diagram) *Given a Streett automaton $M = (\mathcal{V}, (V, E), \rho, Q, \mathcal{A})$ we can construct a hybrid diagram $hd(M) = (\mathcal{V}, V, \rho, \theta, \tau, \mathcal{J}, \mathcal{C})$ as follows.*

(1) For all $u \in V$, $\theta(u) = true$ if $u \in Q$, and $\theta(u) = false$ otherwise.
(2) For all $u, v \in V$, let $\tau(u, v) = true$ if $(u, v) \in E$, and let $\tau(u, v) = false$ otherwise.
(3) $\mathcal{J} = \emptyset$.
(4) $\mathcal{C} = \Big\{(V - P, V \times R) \;\Big|\; (P, R) \in \mathcal{A}\Big\}$. $\Box$

With this construction, we can proceed to the proof of Theorem 12.

6.5.0.1 Proof of Theorem 12. From φ, it is possible to obtain a deterministic Streett automaton M_φ such that $\mathcal{L}(M_\varphi) = \mathcal{L}(\varphi)$ [VW86,Saf88,Saf92]. The diagram $hd(M_\varphi)$ is deterministic, and since $\mathcal{S} \models \varphi$, by Theorem 20 we can construct a chain of transformations $hd(\mathcal{S}) \stackrel{*}{\Rightarrow} hd(M_\varphi)$. It is easy to see that the graph product $hd(M_\varphi) \otimes N_{\neg\varphi}$ does not contain any connected subgraph that is both reachable and admissible. This concludes the argument. $\Box$

7 Conclusions

In this paper, we have presented a diagram-based methodology for the verification of hybrid systems properties expressed in linear-time temporal logic. The proof of a system property consists in a chain of stepwise diagram transformations. The visual representation of the system behavior, coupled with the guidance provided by the algorithm of the previous section, directs the

gradual construction of the proof. The methodology we proposed is the first that is complete (relative to first-order reasoning) with respect to both safety and progress properties.

Diagrams bridge the gap between the system and the specification. In the system $\mathcal{S}$, and in the initial diagram $hd(\mathcal{S})$, all the information about the system's behavior is implicit in the transition relations and in the activities. The temporal specification, on the other hand, is equivalent to an automaton over infinite words: in such an automaton, all the information about the desired behavior is explicit in the vertex labels and in the acceptance conditions. To algorithmically check the system against the specification, we need some finite structure to check, and we need some way of making the implicit information explicit.

The finite structure is that of a diagram, and the purpose of diagram transformations is to take the information that is implicit in the system's transition relations, and make it explicit in the vertex labels and fairness constraints of the diagrams. When all aspects of the system's behavior that are relevant to the specification have been made explicit, the algorithmic check of the diagram against the specification will return an affirmative answer to the verification problem.

While we have chosen phase transition systems as our basic system model, the methodology can be adapted to other models as well, including hybrid automata. In particular, we remark that the definition of a hybrid diagram does not require that the hybrid activities are *deterministic,* as is the case for the definition of PTS. Thus, hybrid diagrams can be used to study systems in which the dynamic evolution of some hybrid variables is specified only by bounds on their derivatives, rather than by exact differential equations.

We conclude by observing that it is possible to formulate *verification rules* in the style of [MP93,KMP96] that correspond to the justice, compassion and pruning rules. These verification rules would yield an alternative methodology for hybrid system verification, which would also be complete for TL_s. We chose to present the rules in the context of diagram transformations, rather than premise-conclusion reasoning, due to the perceived advantages of the diagram-based approach.

Acknowledgements

We would like to thank Zohar Manna for many helpful discussions and suggestions. We also thank Todd Neller and Henny Sipma for useful comments.

References

[ACH+95] 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. *Theoretical Computer Science*, 138(1):3–34, 1995.

[ACHH93] 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.

[BL95] A. Bouajjani and Y. Lakhnech. Logics vs. automata: The hybrid case. In *Hybrid Systems Workshop DIMACS'95*, volume 1066 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1995.

[BLR95] A. Bouajjani, Y. Lakhnech, and R. Robbana. From duration calculus to hybrid automata. In *Proc. 7^{th} Intl. Conference on Computer Aided Verification*, volume 939 of *Lect. Notes in Comp. Sci.*, pages 196–210. Springer-Verlag, 1995.

[BR95] A. Bouajjani and R. Robbana. Verifying ω-regular properties for a subclass of linear hybrid systems. In P. Wolper, editor, *Proc. 7^{th} Intl. Conference on Computer Aided Verification*, volume 939 of *Lect. Notes in Comp. Sci.*, pages 437–450. Springer-Verlag, 1995.

[CRH93] 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.

[dAKM97] L. de Alfaro, A. Kapur, and Z. Manna. Hybrid diagrams: A deductive-algorithmic approach to hybrid system verification. In *Proc. of 14th Annual Symp. on Theor. Asp. of Comp. Sci.*, volume 1200 of *Lect. Notes in Comp. Sci.*, pages 153–164. Springer-Verlag, 1997.

[dAM96] L. de Alfaro and Z. Manna. Temporal verification by diagram transformations. In *Proc. 8th International Conference on Computer Aided Verification*, volume 1102 of *Lect. Notes in Comp. Sci.*, pages 288–299. Springer-Verlag, 1996.

[dAMSU97] L. de Alfaro, Z. Manna, H.B. Sipma, and T.E. Uribe. Visual verification of reactive systems. In *Proc. of TACAS: Tools and Algorithms for the Construction and Analysis of Systems*, volume 1217 of *Lect. Notes in Comp. Sci.*, pages 334–350. Springer-Verlag, 1997.

[KHMP94] A. Kapur, T.A. Henzinger, Z. Manna, and A. Pnueli. Proving safety properties of hybrid systems. In *Proc. of Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *Lect. Notes in Comp. Sci.*, pages 431–454. Springer-Verlag, 1994.

[KMP96] 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.

[LA92] N.A. Lynch and H. Attiya. Using mappings to prove timing properties. *Distributed Computing*, 6:121–139, 1992.

[Lam93] L. Lamport. Hybrid systems in TLA+. In *Hybrid Systems*, volume 736 of *Lect. Notes in Comp. Sci.*, pages 77–102. Springer-Verlag, 1993.

[LPS81] D. Lehmann, A. Pnueli, and J. Stavi. Impartiality, justice and fairness: The etics of concurrent termination. In *Proc. 8th Int. Colloq. Aut. Lang. Prog.*, Lect. Notes in Comp. Sci., pages 264–277. Springer-Verlag, 1981.

[MMP92] 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.

[MP91] Z. Manna and A. Pnueli. Completing the temporal picture. *Theoretical Computer Science*, 83(1):97–130, 1991.

[MP93] Z. Manna and A. Pnueli. Models for reactivity. *Acta Informatica*, 30:609–678, 1993.

[NOSY93] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In *Hybrid Systems*, volume 736 of *Lect. Notes in Comp. Sci.*, pages 149–178. Springer-Verlag, 1993.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. Found. of Comp. Sci.*, 1988.

[Saf92] S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *Proc. ACM Symposium on the Theory of Computing*, pages 275–282, 1992.

[SUM96] H.B. Sipma, T.E. Uribe, and Z. Manna. Deductive model checking. In *Proc. 8th International Conference on Computer Aided Verification*, volume 1102 of *Lect. Notes in Comp. Sci.*, pages 208–219. Springer-Verlag, 1996.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. First IEEE Symposium on Logic in Computer Science*, pages 332–344, 1986.