Towards Refining Temporal Specifications into Hybrid Systems[1]

Thomas A. Henzinger[2] Zohar Manna[3] Amir Pnueli[4]

Abstract. We propose a formal framework for designing hybrid systems by stepwise refinement. Starting with a specification in hybrid temporal logic, we make successively more transitions explicit until we obtain an executable system.

1 Introduction

We present the foundations of a methodology for the systematic development of hybrid systems. As high-level specification language, we suggest *Abstract Phase Transition Systems* (APTS's). The behavior of an APTS consists of a sequence of phases, and each phase may be described implicitly by temporal constraints. For this purpose we introduce *Hybrid Temporal Logic* (HTL), a hybrid extension of interval temporal logic. The notion of one APTS *refining* (implementing) another is defined, and corresponds to inclusion between the sets of behaviors allowed by each system. We also propose a criterion for judging an APTS to be *executable*, i.e., directly implementable on available architectures. A *development sequence*, then, is envisioned to start at a high-level implicit APTS, which is refined by a sequence of steps into an executable APTS. Ultimately, each refinement step ought to be accompanied by verification.

2 Hybrid Temporal Logic

The behavior of a hybrid system is modeled by a function that assigns to each real-numbered time a system state, i.e., values for all system variables. We require that, at each point, the behavior function has a limit from the left and a limit from the right. Discontinuities are points where the two limits differ. We assume the following *uncertainty principle*: limits of function values (defined over nonsingular intervals) are observable; individual function values (at singular points) are not observable — that is, we cannot know (and do not care) if at a discontinuity the function value coincides with the limit from the left or the limit from the right.

To specify properties of behavior functions, we use an interval temporal logic with a chop operator denoted by semicolon [**?**]. Consistent with our interpretation of behavior functions, only limits and derivatives of the behavior function can be constrained by atomic formulas; individual function values cannot appear in specifications.

[1]The research was supported in part by the National Science Foundation under grants CCR-89-11512 and CCR-92-00794, by the Defense Advanced Research Projects Agency under contract NAG2-703, by the United States Air Force Office of Scientific Research under contracts F49620-93-1-0056 and F49620-93-1-0139, and by the European Community ESPRIT Basic Research Action Project 6021 (REACT).

[2]Department of Computer Science, Cornell University, Ithaca, NY 14853.

[3]Department of Computer Science, Stanford University, Stanford, CA 94305.

[4]Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel.

Syntax

Let V be a set of typed variables, where the allowed types are *boolean*, *integer*, and *real*. We view the booleans and the integers as subsets of the reals, where *false* and *true* correspond to 0 and 1, respectively. For a variable $x \in V$, we write $\overleftarrow{x}$ and $\overrightarrow{x}$ for the limit from the right (the *right limit*) and the limit from the left (the *left limit*) of x, and $\overleftarrow{\dot{x}}$ and $\overrightarrow{\dot{x}}$ for the first derivative from the right (the *right derivative*) and the first derivative from the left (the *left derivative*) of x.

A *local formula* is an atomic formula over the right and left limits and derivatives of variables in V. The formulas ϕ of *Hybrid Temporal Logic* (HTL) are defined inductively as follows:

$$\phi \;:=\; \psi \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1;\phi_2 \mid \forall x.\, \phi$$

where $x \in V$ and ψ is a local formula.

A *state formula* is an atomic formula over the variables in V. If ψ is a state formula, we write $\overleftarrow{\psi}$ (and $\overrightarrow{\psi}$) for the local formula that results from ψ by replacing each variable occurrence x in ψ with its right limit $\overleftarrow{x}$ (left limit $\overrightarrow{x}$, respectively).

Semantics

Let R be the set of real numbers. A *state* $\sigma : V \rightarrow \mathrm{R}$ is a type-consistent interpretation of the variables in V (i.e., boolean variables may only be interpreted as 0 or 1, and a similar restriction holds for integer variables). We write Σ_V for the set of states.

Time is modeled by the nonnegative real line R^+. An *open interval* (a, b), where $a, b \in \mathrm{R}^+$ and $a < b$, is the set of points $t \in \mathrm{R}^+$ such that $a < t < b$; in particular, we consider only open intervals that are nonempty and bounded. Let $I = (a, b)$ be an open interval. A function $f : I \rightarrow \mathrm{R}$ is *piecewise smooth* on I if

- at a, the right limit and all right derivatives of f exist;
- at all points $t \in I$, the right and left limits and all right and left derivatives of f exist, and f is continuous either from the right or from the left;
- at b, the left limit and all left derivatives of f exist.

Two functions $f, g : I \rightarrow \mathrm{R}$ are *indistinguishable* on I if they agree on almost all (i.e., all but finitely many) points $t \in I$. Thus, if two piecewise smooth functions are indistinguishable on the open interval I, then they agree on all limits and derivatives throughout I, on the right limit and right derivatives at a, and on the left limit and left derivatives at b.

A *phase* $P = (b, f)$ over V is a pair consisting of

1. a positive real number $b > 0$, the *length* of P, and
2. a type-consistent family $f = \{f_x \mid x \in V\}$ of functions $f_x : I \rightarrow \mathrm{R}$ that are piecewise smooth on the open interval $I = (0, b)$ and assign to each point $t \in I$ a value for the variable $x \in V$.

It follows that the phase P assigns to every real-valued time $t \in I$ a state $f(t) \in \Sigma_V$. Furthermore, the right limit of f at 0 and the left limit of f at b are defined. We write

$$\overleftarrow{P} \;=\; \lim_{t \rightarrow 0} \{f(t) \mid 0 < t < b\}$$

for the *left limiting state* $\overleftarrow{P} \in \Sigma_V$ of the phase P, and

$$\overrightarrow{P} = \lim_{t \to b} \{f(t) \mid 0 < t < b\}$$

for the *right limiting state* $\overrightarrow{P} \in \Sigma_V$ of P.

Let $P_1 = (b, f)$ and $P_2 = (c, g)$ be two phases. The phases P_1 and P_2 are *indistinguishable* (*equivalent*) if $b = c$ and for all variables $x \in V$, the two functions f_x and g_x are indistinguishable on the open interval $(0, b)$. The phase P_2 is a *subphase* of P_1 at position a, where $0 \leq a < b$, if

- $a + c \leq b$ and
- for all $0 < t < c$, $g(t) = f(a + t)$.

If $a = 0$, then P_2 is a *leftmost* subphase of P_1; if $a = b - c$, a *rightmost* subphase. The two phases P_1 and P_2 *partition* a third phase $P = (d, h)$ if

- $d = b + c$,
- P_1 is a leftmost subphase of P, and
- P_2 is a rightmost subphase of P.

Notice that since P is a phase, for all variables $x \in V$, at b the function h_x is continuous either from the right or from the left. Hence, if P_1 and P_2 partition P, then the value $h_x(b)$ is either the right limit of f_x at b or the left limit of g_x at 0. It follows that there are several phases that are partitioned by the two phases P_1 and P_2. All of these phases, however, are indistinguishable, because they disagree at most at b.

The formulas of hybrid temporal logic are interpreted over phases. A phase $P = (b, f)$ *satisfies* the hybrid temporal formula ϕ, denoted $P \models \phi$, according to the following inductive definition:

$P \models \psi$ iff the local formula ψ evaluates to *true*, where

- $\overleftarrow{x}$ is interpreted as the right limit of f_x at 0,

$$\overleftarrow{x} = \lim_{t \to 0} \{f_x(t) \mid 0 < t < b\};$$

- $\overrightarrow{x}$ is interpreted as the left limit of f_x at b,

$$\overrightarrow{x} = \lim_{t \to b} \{f_x(t) \mid 0 < t < b\};$$

- $\dot{\overleftarrow{x}}$ is interpreted as the right derivative of f_x at 0,

$$\dot{\overleftarrow{x}} = \lim_{t \to 0} \left\{(f_x(t) - \overleftarrow{x})/t \mid 0 < t < b\right\};$$

- $\dot{\overrightarrow{x}}$ is interpreted as the left derivative of f_x at b,

$$\dot{\overrightarrow{x}} = \lim_{t \to b} \left\{(f_x(t) - \overrightarrow{x})/(t - b) \mid 0 < t < b\right\}.$$

$P \models \neg\phi$ iff $P \not\models \phi$.

$P \models \phi_1 \vee \phi_2$ iff $P \models \phi_1$ or $P \models \phi_2$.

$P \models \phi_1; \phi_2$ iff there are two phases P_1 and P_2 that partition P such that $P_1 \models \phi_1$ and $P_2 \models \phi_2$.

$P \models \forall x. \phi$ iff $P' \models \phi$ for all phases $P' = (b, f')$ that differ from P at most in the interpretation f'_x of x.

Notice that right limits and right derivatives are applied at the left end of a phase, while left limits and left derivatives are applied at the right end. Also observe that if two phases P_1 and P_2 are indistinguishable, then $P_1 \models \phi$ iff $P_2 \models \phi$; that is, HTL-formulas cannot distinguish between phases that differ only at finitely many points.

Sample formulas

It is convenient to define abbreviations for common temporal formulas. The following abbreviations express that a leftmost subphase, a rightmost subphase, or any subphase of a phase satisfies the formula ϕ:

$$\begin{array}{lll} \lhd\phi & \text{stands for} & \phi \vee (\phi; \mathit{true}) \\ \rhd\phi & \text{stands for} & \phi \vee (\mathit{true}; \phi) \\ \Diamond\phi & \text{stands for} & (\lhd\phi) \vee (\rhd\phi) \vee (\mathit{true}; \phi; \mathit{true}) \end{array}$$

Thus we can express that all subphases of a phase satisfy ϕ:

$$\Box\phi \quad \text{stands for} \quad \neg\Diamond\neg\phi$$

We now define temporal *until* and *unless* operators over a phase P. The until formula $\phi_1 \mathcal{U} \phi_2$ asserts that the phase P can be partitioned into two subphases P_1 (which may be empty) and P_2 such that ϕ_1 holds throughout P_1 and ϕ_2 holds on a leftmost subphase of P_2; the unless formula $\phi_1 \mathcal{W} \phi_2$ asserts that either ϕ_1 holds throughout the phase P, or P satisfies $\phi_1 \mathcal{U} \phi_2$:

$$\begin{array}{lll} \phi_1 \mathcal{U} \phi_2 & \text{stands for} & (\lhd\phi_2) \vee (\Box\phi_1); (\lhd\phi_2) \\ \phi_1 \mathcal{W} \phi_2 & \text{stands for} & (\Box\phi_1) \vee (\phi_1 \mathcal{U} \phi_2) \end{array}$$

The following formula asserts that the variable $u \in V$ is *rigid* on a phase; that is, the function f_u is constant throughout the phase:

$$u \in \mathit{Rigid} \quad \text{stands for} \quad \Box(\overrightarrow{u} = \overleftarrow{u})$$

Using rigid variables, we can specify that a function is continuous, and that its first derivative is continuous throughout a phase:

$$\begin{array}{lll} x \in C^0 & \text{stands for} & \forall u, v \in \mathit{Rigid}. \left[(\overrightarrow{x} = u); (\overleftarrow{x} = v) \ \rightarrow \ u = v\right] \\ x \in C^1 & \text{stands for} & x \in C^0 \wedge \forall u, v \in \mathit{Rigid}. \left[(\dot{\overrightarrow{x}} = u); (\dot{\overleftarrow{x}} = v) \ \rightarrow \ u = v\right] \end{array}$$

The formula $x \in C^0$ requires that for any partition of a phase P into two subphases, the left and right limits of x at the point of partitioning coincide; the formula $x \in C^1$ adds the analogous requirement for the first derivatives of x.

From now on, we shall write x and $\dot{x}$ synonymous for the right limit $\overleftarrow{x}$ and the right derivative $\dot{\overleftarrow{x}}$, respectively. This convention allows us to read a state formula ψ as a hybrid temporal formula, namely, as the local formula $\overleftarrow{\psi}$.

Hybrid temporal logic subsumes many real-time temporal logics [?] and the duration calculus [?]. For example, the formula

$$\Box \forall x \in C^0.\ \Big[\Big(p \wedge x = 0 \wedge \Box(\dot{x} = 1) \wedge \overrightarrow{x} > 5\Big) \ \rightarrow\ \Diamond(q \wedge x \leq 5)\Big]$$

asserts that every p-state of a phase is followed within 5 time units either by a q-state or by the end of the phase. The variable x is a "clock" that measures the length of all subphases starting with a p-state. The formula

$$\forall x \in C^0.\ \Big[\Big(x = 0 \wedge \Box(p \rightarrow \dot{x} = 1) \wedge \Box(\neg p \rightarrow \dot{x} = 0)\Big) \ \rightarrow\ \overrightarrow{x} \leq 10\Big]$$

asserts that the cumulative time that p is true in a phase is at most 10. Here the variable x is an "integrator" that measures the accumulated duration of p-states.

3 Phase Transition Systems

Following [?] and [?], we model hybrid systems as transition systems. Just as discrete transitions can be represented as binary relations on states, hybrid transitions can be represented as binary relations on phases.

Abstract phase transition systems

An *abstract phase transition system* (APTS) $\mathcal{S} = (V, \mathcal{P}, \mathcal{P}_0, \mathcal{P}_F, \mathcal{T})$ consists of five components:

1. A finite set V of *state variables*.
2. A set $\mathcal{P}$ of *phases* over V.
3. A set $\mathcal{P}_0 \subseteq \mathcal{P}$ of *initial phases*.
4. A set $\mathcal{P}_F \subseteq \mathcal{P}$ of *final phases*.
5. A set $\mathcal{T}$ of *transitions*. Each transition $\tau \in \mathcal{T}$ is a binary relation on the phases in $\mathcal{P}$, i.e., $\tau \subseteq \mathcal{P}^2$.

A *phase sequence* is a finite or infinite sequence of phases. Let $\overline{P} = P_0, P_1, \ldots P_n$ be a finite phase sequence with $P_i = (b_i, f_i)$ for all $0 \leq i \leq n$. The finite phase sequence $\overline{P}$ *partitions* a phase $P = (b, f)$ if

- $b = \sum_{0 \leq i \leq n} b_i$ and
- for all $0 \leq i \leq n$, P_i is a subphase of P at position $\sum_{0 \leq j < i} b_j$.

The finite phase sequence $\overline{P}$ can thus be viewed as a set of indistinguishable phases, namely, those phases that are partitioned by $\overline{P}$. Consequently, we may interpret HTL-formulas over finite phase sequences. The finite phase sequence $\overline{P}$ *satisfies* the hybrid temporal formula ϕ, denoted $\overline{P} \models \phi$, if some phase that is partitioned by $\overline{P}$ satisfies ϕ.

Two finite phase sequences $\overline{P}_1$ and $\overline{P}_2$ are *equivalent* if there are two indistinguishable phases P_1 and P_2 such that $\overline{P}_1$ partitions P_1 and $\overline{P}_2$ partitions P_2. It follows that all equivalence classes of finite state sequences are *closed under stuttering*: if a phase P_i of the finite phase sequence $\overline{P}$ is split into two phases P' and P'' that partition P_i, the resulting finite phase sequence

$$P_0, \ldots P_{i-1}, P', P'', P_{i+1}, \ldots P_n$$

is equivalent to $\overline{P}$.

Let $\overline{P} = P_0, P_1, P_2, \ldots$ be an infinite phase sequence with $P_i = (b_i, f_i)$ for all $i \geq 0$. The infinite phase sequence $\overline{P}$ *diverges* if the infinite sum $\sum_{i \geq 0} b_i$ of phase lengths diverges, i.e., for all nonnegative reals $t \in \mathrm{R}^+$ there is an integer $n \geq 0$ such that $t < \sum_{0 \leq i \leq n} b_i$.

A finite phase sequence $\overline{P}$ is a *run fragment* of the APTS $\mathcal{S}$ if $\overline{P}$ is equivalent to a finite phase sequence $P_0, P_1, \ldots P_n$ that satisfies three conditions:

Initiality $P_0 \in \mathcal{P}_0$.

Continuous activities For all $0 \leq i \leq n$, $P_i \in \mathcal{P}$.

Discrete transitions For all $0 \leq i < n$, there is a transition $\tau \in \mathcal{T}$ such that $(P_i, P_{i+1}) \in \tau$.

The run fragment $\overline{P}$ is *complete* if $P_n \in \mathcal{P}_F$.

An infinite phase sequence $\overline{P}$ is a *run* (*computation*) of the APTS $\mathcal{S}$ if

Safety All finite prefixes of $\overline{P}$ are run fragments of $\mathcal{S}$.

Liveness $\overline{P}$ diverges.

The APTS $\mathcal{S}$ *satisfies* a hybrid temporal formula ϕ, written $\mathcal{S} \models \phi$, if all run fragments of $\mathcal{S}$ satisfy ϕ.

Activity transition graphs

Both timed transition systems [?] and timed safety automata [?] specify APTS's. We use activity transition graphs to specify APTS's.

An *activity transition graph* (ATG) A is a directed labeled multigraph $(V_D, L, E, \mu_1, \mu_2, \mu_3, \kappa)$ consisting of the following components:

- A finite set V_D of *data variables*.
- A finite set L of vertices called *locations*. Each location $\ell \in L$ is labeled by
  - an *initial condition* $\mu_1(\ell)$, a state formula over the variables in V_D,
  - an *activity* $\mu_2(\ell)$, a hybrid temporal formula over V_D, and
  - a *final condition* $\mu_3(\ell)$, a state formula over V_D.
- A finite set E of *edges* between the locations in L. Each edge $e \in E$ is labeled by a guarded command $\kappa(e) = (\gamma \to \alpha)$, where γ is a state formula over the variables in V_D (the *guard* of e) and α is an assignment to some of the variables in V_D.

The ATG A defines the APTS $\mathcal{S}_A = (V, \mathcal{P}, \mathcal{P}_0, \mathcal{P}_F, \mathcal{T})$:

1. The state variables are $V = V_D \cup \{\pi\}$, where the *control variable* π ranges over the locations L.
2. For each location $\ell \in L$, $\mathcal{P}$ contains all phases P such that

$$P \models \Box(\pi = \ell) \wedge \Big(\bigwedge_{x \in V_D} x \in C^0 \Big) \wedge \mu_2(\ell)$$

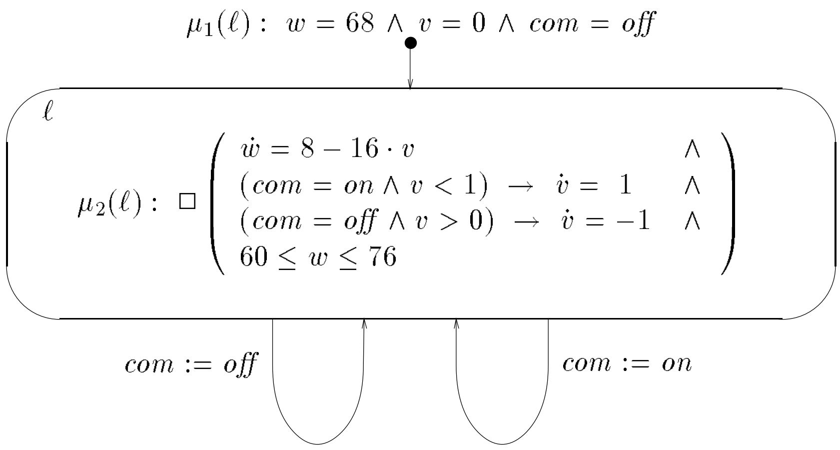

Figure 1: Specification $\widehat{A}$

3. For each location $\ell \in L$, $\mathcal{P}_0$ contains all phases $P \in \mathcal{P}$ such that

$$P \ \models \ (\pi = \ell) \wedge \mu_1(\ell)$$

4. For each location $\ell \in L$, $\mathcal{P}_F$ contains all phases $P \in \mathcal{P}$ such that

$$P \ \models \ (\overrightarrow{\pi} = \ell) \wedge \overrightarrow{\psi}$$

   where $\psi = \mu_3(\ell)$ is the final condition of ℓ.

5. For each edge $e \in E$, $\mathcal{T}$ contains a transition $\tau_e \subseteq \mathcal{P}^2$. Let $\ell_1, \ell_2 \in L$ be the source and target locations of the edge e, and let γ and α be the guard and assignment associated with e. Then $(P_1, P_2) \in \tau_e$ iff

$$\begin{aligned} P_1 \ &\models \ (\overrightarrow{\pi} = \ell_1) \wedge \overrightarrow{\gamma} \\ P_2 \ &\models \ (\pi = \ell_2) \end{aligned}$$

   and $\overleftarrow{P_2}$ results from $\overrightarrow{P_1}$ by executing the assignment α.

Specifications

An ATG with a single location is called a *specification*.

Consider, for example, the specification $\widehat{A}$ presented in Figure **??**. The ATG $\widehat{A}$ specifies a water level controller that opens and closes a valve regulating the outflow of water from a container. The container has an input vent through which water flows at the constant rate 8. When the valve is fully open, there is an outflow of 16, leading to an overall water level decrease of $16 - 8 = 8$ per time unit. The controller can command the valve to open and close using the switch *com*.

The specification $\widehat{A}$ refers to three variables:

- The boolean variable *com*, ranging over the values $\{on, off\}$, represents the latest command issued by the controller. The value *on* causes the opening of the valve; the value *off*, the closing of the valve.

- The real-valued variable v represents the status of the valve. It assumes values between 0 and 1, corresponding to the valve being fully closed (zero water outflow) and fully open, respectively. When $com = on$ and $v < 1$, v increases at the constant rate of 1. When $com = off$ and $v > 0$, v decreases at the same constant rate. By default, when none of these conditions hold, v maintains a constant value, satisfying the equation $\dot{v} = 0$.

- The real-valued variable w represents the water level. When the valve is fully closed, w increases at the rate 8; when the valve is fully open, w decreases at the rate 8. At intermediate values of v, the water level increases at the rate $8 - 16 \cdot v$.

The controller is supposed to keep the water level between 60 and 76 units. The two transitions represent the ability of the controller to set variable *com* to the values *on* and *off*.

We point out that the specification describes both the actions of the controller (giving commands to open and close the valve) and the response of the controlled environment (valve closing and opening and water level rising and falling). When specifying controllers, the set of data variables $V_D = V_C \cup V_E$ of an ATG can be partitioned into a set V_C of *controlled variables*, which may be modified by the controller, and a set V_E of *environment variables*, which vary according to the laws of physics. In our example, the switch *com* is a controlled variable, while the valve v and water level w are environment variables. Note, however, that the equations for the behavior of environment variable v are influenced by the value of the controlled variable *com*.

4 Stepwise Refinement

Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two APTS's over the sets V_1 and V_2 of state variables, respectively. The APTS $\mathcal{S}_1$ *refines* the APTS $\mathcal{S}_2$ if $V_2 \subseteq V_1$ and the projection of every run of $\mathcal{S}_1$ to the variables in V_2 is a run of $\mathcal{S}_2$.

Hierarchical activity transition graphs

An APTS that is given by an ATG can be refined by expanding activities — i.e., hybrid temporal formulas labeling locations — into ATG's. Thus we obtain *hierarchical* (nested) *activity transition graphs* (HATG's), which are defined inductively:

1. Every ATG is an HATG.
2. Let $B = (V_D, L, E, \mu_1, \mu_2, \mu_3, \kappa)$ and C_ℓ be two HATG's, and let A be the tuple $B[\ell := C_\ell]$ that results from B by replacing the activity $\mu_2(\ell)$ of the location $\ell \in L$ with the HATG C_ℓ. Then C is also an HATG.

Every HATG A defines an APTS $\mathcal{S}_A$. Roughly speaking, a location ℓ whose activity is defined by an HATG C_ℓ contributes all phase sequences $\overline{P}$ such that $\overline{P} \models \Box(\pi = \ell)$ and some extension of $\overline{P}$ is a complete run fragment of C_ℓ. The phase sequence $\overline{P}$ needs to be extended by data variables that are local to C_ℓ and a control variable for C_ℓ.

To define the APTS $\mathcal{S}_A$ formally, we inductively translate the nested HATG A into a flat ATG $\mathit{flat}(A)$. If A is an ATG, then $\mathit{flat}(A) = A$; otherwise A is of the form $B[\ell := C_\ell]$, for two ATG's $B = (V_D, L, E, \mu_1, \mu_2, \mu_3, \kappa)$ and $C_\ell = (V_\ell, L_\ell, E_\ell, \nu_1, \nu_2, \nu_3, \lambda)$. Without loss of generality, we assume that the locations in L and L_ℓ are disjoint, and that both B and C are complete graphs (use the guard *false* to label absent edges). Then the ATG $\mathit{flat}(A)$ is defined as follows:

- The data variables of $\mathit{flat}(A)$ are $V_D \cup V_\ell$.

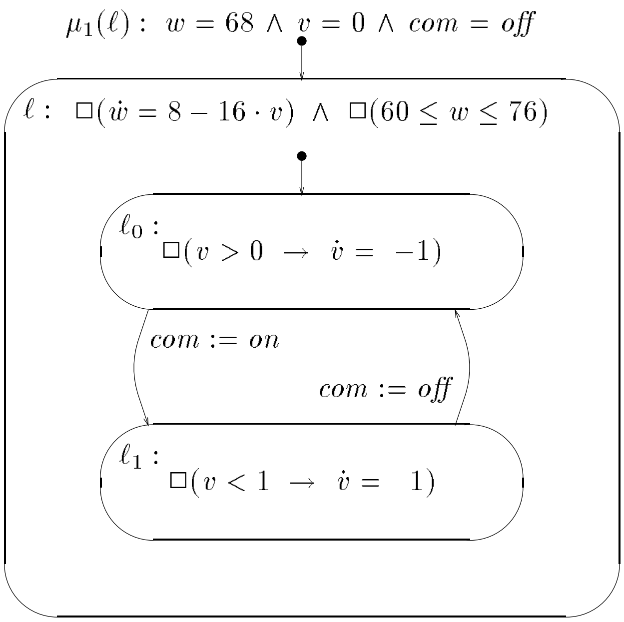

Figure 2: Refinement $\widehat{B}$

- The locations of *flat*(A) are $(L - \{\ell\}) \cup L_\ell$.
  - Each location $\ell' \in (L - \{\ell\})$ is labeled by the initial condition $\mu_1(\ell')$, the activity $\mu_2(\ell')$, and the final condition $\mu_3(\ell')$.
  - Each location $\ell' \in L_\ell$ is labeled by the initial condition $\mu_1(\ell) \wedge \nu_1(\ell')$, the activity $\nu_2(\ell')$, and the final condition $\mu_3(\ell) \wedge \nu_3(\ell')$.
- The graph *flat*(A) is complete.
  - Each edge $(\ell_1, \ell_2) \in (L - \{\ell\})^2$ is labeled by the guarded command $\kappa(\ell_1, \ell_2)$.
  - Each edge $(\ell_1, \ell_2) \in L_\ell^2$ is labeled by the guarded command $\lambda(\ell_1, \ell_2)$.
  - For $\ell_1 \in (L - \{\ell\})$, let $\kappa(\ell_1, \ell) = (\gamma \rightarrow \alpha)$. Each edge (ℓ_1, ℓ_2), with $\ell_2 \in L_\ell$, is labeled by the guarded command
  $$(\mu_1(\ell_2) \wedge \gamma) \ \rightarrow \ \alpha.$$
  - For $\ell_2 \in (L - \{\ell\})$, let $\kappa(\ell, \ell_2) = (\gamma \rightarrow \alpha)$. Each edge (ℓ_1, ℓ_2), with $\ell_1 \in L_\ell$, is labeled by the guarded command
  $$(\mu_3(\ell_1) \wedge \gamma) \ \rightarrow \ \alpha.$$

A *run* (*fragment*) of the HATG A, then, is a run (fragment) of the ATG *flat*(A).

For example, the specification $\widehat{A}$ of the water level controller can be refined into the HATG $\widehat{B}$ presented in Figure **??**, which separates the phases at which $com = on$ from those at which $com = off$. The two conjuncts $\Box(\dot{w} = 8 - 16 \cdot v)$ and $\Box(60 \leq w \leq 76)$, which label the location ℓ, are common to both locations ℓ_0 and ℓ_1 of the inner graph. Also note that the initial conditions of the inner graph are represented by special entry edges: the initial condition of ℓ_0 is *true* and the initial condition of ℓ_1 is *false*. All final conditions are, by default, *true*.

Verification conditions

Let B be an ATG over the set V_D of data variables. A refinement step replaces a hybrid temporal formula ϕ defining the activity of a location ℓ of B with an ATG C_ℓ. The refinement step is *correct* if the projection of every run of the resulting two-level HATG $B[\ell := C_\ell]$ to the variables in $V_D \cup \{\pi\}$ is a run of A.

Suppose that the replaced formula ϕ is of the form $\Box\phi'$. Then the following two conditions suffice to establish the correctness of a refinement step:

1. For every edge of the ATG C_ℓ labeled by the guarded command $\gamma \to \alpha$, the ATG B contains a reflexive edge from ℓ to ℓ labeled by $\gamma' \to \alpha$ such that the state formula γ implies the state formula γ'.

2. There exists a state formula ψ over V_D such that

$$
\begin{array}{lll}
(i) & \mathcal{S}_B & \models \quad (\pi = \ell \;\to\; \psi) \wedge \Box\Big(\pi \neq \ell \;\to\; (\pi \neq \ell)\,\mathcal{W}\,(\pi = \ell \,\wedge\, \psi)\Big) \\
(ii) & \mathcal{S}_{C_\ell} & \models \quad \psi \to \phi
\end{array}
$$

The first condition ensures that all discontinuities of the inner graph C_ℓ are admitted by the outer graph B. The proof obligation (i) asserts that the state formula ψ is true in every left limiting state of a phase in which the control of the outer graph B enters the location ℓ. Now let $\overline{P}$ be a run fragment of the inner graph C_ℓ such that ψ is true in the left limiting state of $\overline{P}$. The proof obligation (ii) guarantees that the hybrid temporal formula ϕ holds over $\overline{P}$, and since ϕ is of the form $\Box\phi'$, it also holds over any continuous segment of $\overline{P}$.

For example, to show that the water level controller $\widehat{B}$ refines the specification $\widehat{A}$, we obtain two proof obligations:

$$
(i) \quad \mathcal{S}_{\widehat{A}} \quad \models \quad \left(\begin{array}{lll}
(\pi = \ell & \to & (w = 68 \,\wedge\, v = 0 \,\wedge\, com = \mathit{off})) \qquad\qquad\qquad \wedge \\
\Box(\pi \neq \ell & \to & (\pi \neq \ell)\,\mathcal{W}\,(w = 68 \,\wedge\, v = 0 \,\wedge\, com = \mathit{off} \,\wedge\, \pi = \ell))
\end{array}\right)
$$

$$
(ii) \quad \mathcal{S}_{\widehat{B}_\ell} \quad \models \quad \underbrace{(w = 68 \,\wedge\, v = 0 \,\wedge\, com = \mathit{off})}_{\psi} \;\to\; \underbrace{\Box\left(\begin{array}{ll}
\dot{w} = 8 - 16 \cdot v & \wedge \\
(com = on \,\wedge\, v < 1) \;\to\; \dot{v} = 1 & \wedge \\
(com = on \,\wedge\, v = 1) \;\to\; \dot{v} = 0 & \wedge \\
(com = \mathit{off} \,\wedge\, v = 0) \;\to\; \dot{v} = 0 & \wedge \\
(com = \mathit{off} \,\wedge\, v > 0) \;\to\; \dot{v} = -1 & \wedge \\
60 \leq w \leq 76 &
\end{array}\right)}_{\phi}
$$

where $\widehat{B}_\ell$ is the ATG labeling the location ℓ in the refinement $\widehat{B}$. In a future paper, we will present a proof system for verifying that an APTS satisfies a hybrid temporal formula.

Executability

The refinement of a specification typically proceeds in several steps until we reach a HATG A such that the ATG $\mathit{flat}(A)$ can be directly implemented. The ATG B is *executable* if two conditions are met:

Effectiveness For each hybrid temporal formula ϕ defining an activity of B, the set of models of ϕ (i.e., the set of phases that satisfy ϕ) is effectively computable.

NonZenoness The APTS $\mathcal{S}_B$ is nonZeno: an APTS $\mathcal{S}$ is *nonZeno* if every run fragment of $\mathcal{S}$ is a finite prefix of a run of $\mathcal{S}$.

The runs of an executable ATG can be generated by adding one phase at a time. The effectiveness condition ensures that, in each state, a stepwise interpreter can compute the set of possible successor phases. The nonZenoness condition ensures that the stepwise interpreter cannot make a nondeterministic choice among the possible successor phases that will result, later on, in a deadlock state from which the system cannot proceed or in a Zeno state from which time cannot diverge [**?**, **?**].

It is not difficult to see that both systems $\widehat{A}$ and $\widehat{B}$ violate the nonZenoness condition and, therefore, are not executable. For the system $\widehat{A}$, consider the phase $P = (1, f)$, where $f(t)$ interprets the data variables as follows, for all $0 < t < 1$:

$$\begin{array}{lcl} f_{com}(t) & = & \mathit{off} \\ f_v(t) & = & 0 \\ f_w(t) & = & 68 + 8 \cdot t \end{array}$$

Thus, as t approaches 1, the value of w approaches 76. This leads to the right limiting state

$$\overrightarrow{P} \;=\; \{com = \mathit{off},\ v = 0,\ w = 76\}.$$

This state is a deadlock state. There is no way to proceed from $\overrightarrow{P}$ without violating the constraint $w \leq 76$. The same phase can be reproduced by the system $\widehat{B}$.

In Figure **??**, we present a further refinement of system $\widehat{B}$. This refinement improves on system $\widehat{B}$ in two respects. First, the phase in which $com = \mathit{off}$ and $v = 0$ has been separated from the phase in which $com = \mathit{off}$ and v is still decreasing. This has been achieved by refining the location ℓ_0 of $\widehat{B}$ with a graph consisting of the two locations ℓ_0 (initial condition *true*, final condition *false*) and ℓ_3 (initial condition *false*, final condition *true*) of $\widehat{C}$. A similar separation has been carried out for the case that $com = on$.

The more important improvement, however, is that the system $\widehat{C}$ forces the setting of com to *on* before w rises above 74, and the setting of com to *off* before w falls below 62. This ensures that the valve starts opening or closing in time to guarantee that w never exceeds the range $[60, 76]$. Indeed, the system $\widehat{C}$ cannot deadlock and is, therefore, executable.

We say that the HATG B *implements* the specification A if $\mathcal{S}_B$ refines $\mathcal{S}_A$ and B is executable. For example, the water level controller $\widehat{C}$ implements the specification $\widehat{A}$.

Environment versus control

In the example of the water level controller, the environment controls the environment variables v and w only through differential equations (activities); the environment does not modify any variables through guarded commands (transitions). If the environment can modify variables through transitions, we have to adopt a stricter notion of refinement. In particular, a refinement step should not constrain the environment [**?**].

Consider, for example, the ATG $\widehat{D}$ presented in Figure **??**. The specification $\widehat{D}$ is a generalization of specification $\widehat{A}$. The container now has two valves, v_i and v_o. The valve v_o controls, as before, the rate of outflow. The valve v_i controls the inflow of water into the container and, similar to v_o, can assume a real value between 0 and 1. The two valves are regulated by the two boolean command variables com_i and com_o that can assume the values 0 (*off*) and 1 (*on*). The conditional differential equation

$$v_i \neq com_i \;\rightarrow\; \dot{v}_i \;=\; 2 \cdot com_i - 1$$

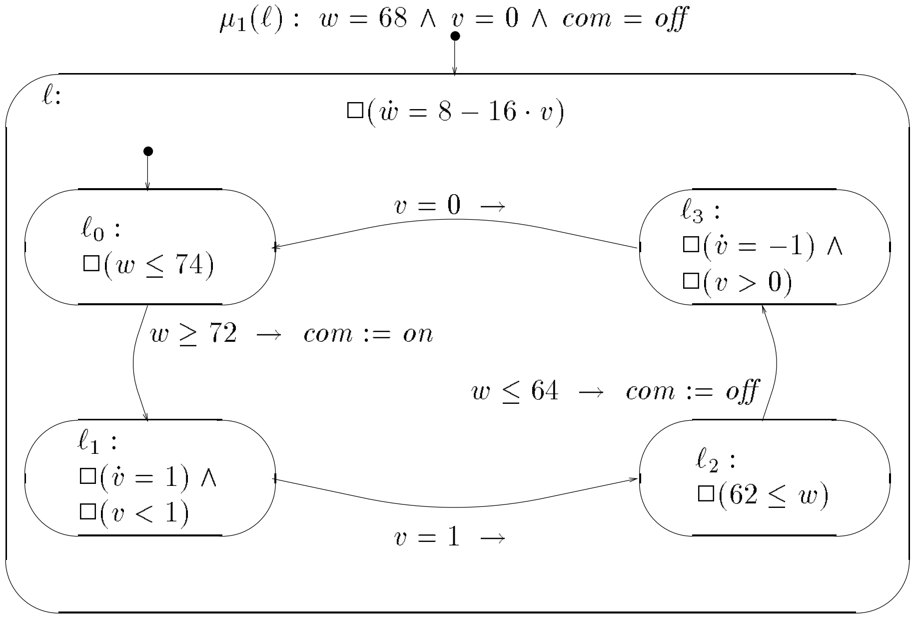

Figure 3: Refinement $\widehat{C}$

should be interpreted as a compact representation of the two cases

$$\begin{array}{lcl} ((com_i = 1 \wedge v_i \neq 1) & \rightarrow & \dot{v}_i = 1) \quad \wedge \\ ((com_i = 0 \wedge v_i \neq 0) & \rightarrow & \dot{v}_i = -1) \end{array}$$

The only variable that can be directly modified by the controller is the outflow switch com_o. All other variables, including the inflow switch com_i, are environment variables. The continuous environment variables w, v_i, and v_o are governed by differential equations. The boolean environment variable com_i, however, is modified by transitions. The notation $com_i :\in \{0, 1\}$ labeling the edge in Figure **??** means that there are two transitions setting com_i nondeterministically to 0 or 1, respectively.

We will assume that all transitions of an APTS can be partitioned into *system transitions*, which modify only controlled variables, and *environment transitions*, which modify only environment variables. For example, the specification $\widehat{D}$ contains two system transitions, which update com_o, and two environment transitions, which update com_i.

Let us consider what type of refinements the specification $\widehat{D}$ can have. If we retain the notion of refinement as defined above, namely that the APTS $\mathcal{S}_1$ refines the APTS $\mathcal{S}_2$ if every computation of $\mathcal{S}_2$ is also a computation of $\mathcal{S}_1$, we can point at the HATG $\widehat{E}$ of Figure **??** as a possible refinement of $\widehat{D}$. This refinement omits all transitions and, as a result, the command variables com_i and com_o and, consequently, the valve variables v_i and vo all remain at 0, leading to a constant value of the water level w at 68, which is well within the prescribed range.

It is quite obvious that $\widehat{E}$ cannot be accepted as a reasonable implementation of $\widehat{D}$. The easy solution that $\widehat{E}$ opted for is to prevent the environment from ever commanding the valve v_i to open. By contrast, the intended meaning of the specification $\widehat{D}$ is that, no matter how the environment commands the input valve to open or close, the controller will find a way to open and close the output valve so that the water level will remain within the prescribed limits.

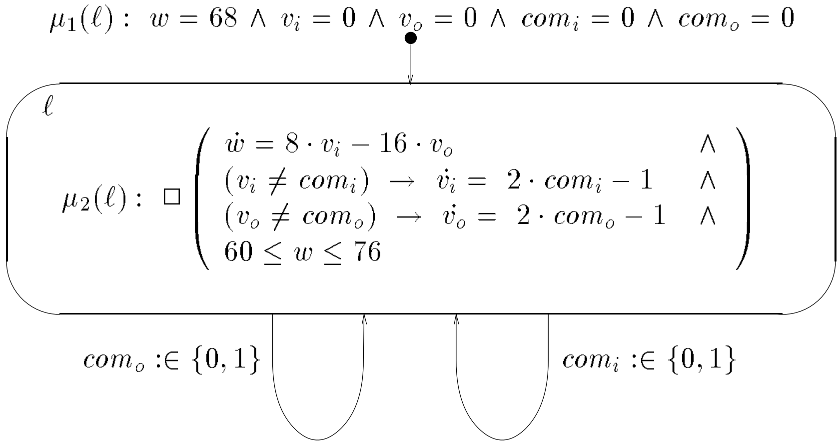

Figure 4: Specification $\widehat{D}$

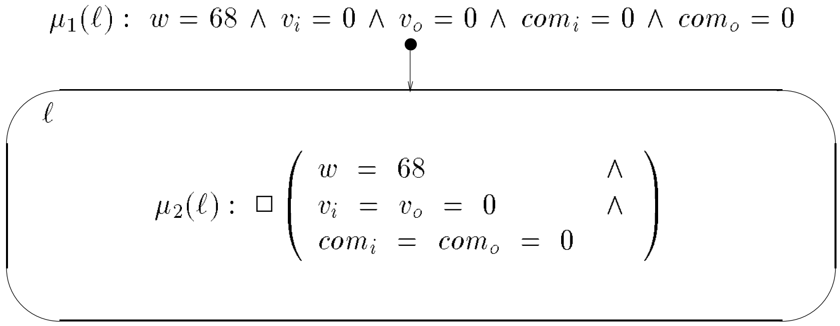

Figure 5: Lazy refinement $\widehat{E}$

Consequently, in the presence of environment transitions, we adopt the following definition for refinement between two APTS's. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two APTS's over the sets V_1 and V_2 of state variables, respectively, such that $V_2 \subseteq V_1$ and $V_E \subseteq V_2$ is a set of environment variables. The APTS $\mathcal{S}_1$ *refines* the APTS $\mathcal{S}_2$ if

1. The projection of every run of $\mathcal{S}_1$ to the variables in V_2 is a run of $\mathcal{S}_2$.
2. Every environment transition of $\mathcal{S}_2$ is duplicated in $\mathcal{S}_1$; that is, if (P_1, P_2) is an environment transition of $\mathcal{S}_2$ and the $\mathcal{S}_1$-phases P_1' and P_2' are extensions of the $\mathcal{S}_2$-phases P_1 and P_2 to the variables in V_1, then (P_1', P_2') is a transition of $\mathcal{S}_1$.

Thus, the refinement of a specification must respect all environment transitions that appear in the specification.

With this more stringent notion of refinement, the HATG $\widehat{E}$ no longer refines the specification $\widehat{D}$, because it fails to duplicate the environment transition of $\widehat{D}$. In Figure ?? we present an HATG $\widehat{F}$ that implements the specification $\widehat{D}$, i.e., $\widehat{F}$ refines $\widehat{D}$ and is executable. The self-loop on the bottom represents the environment transitions that may set com_i to a new value at any point. Drawing a self-loop at an enclosing box is interpreted as if there is a similar self-loop at each of the four internal locations.

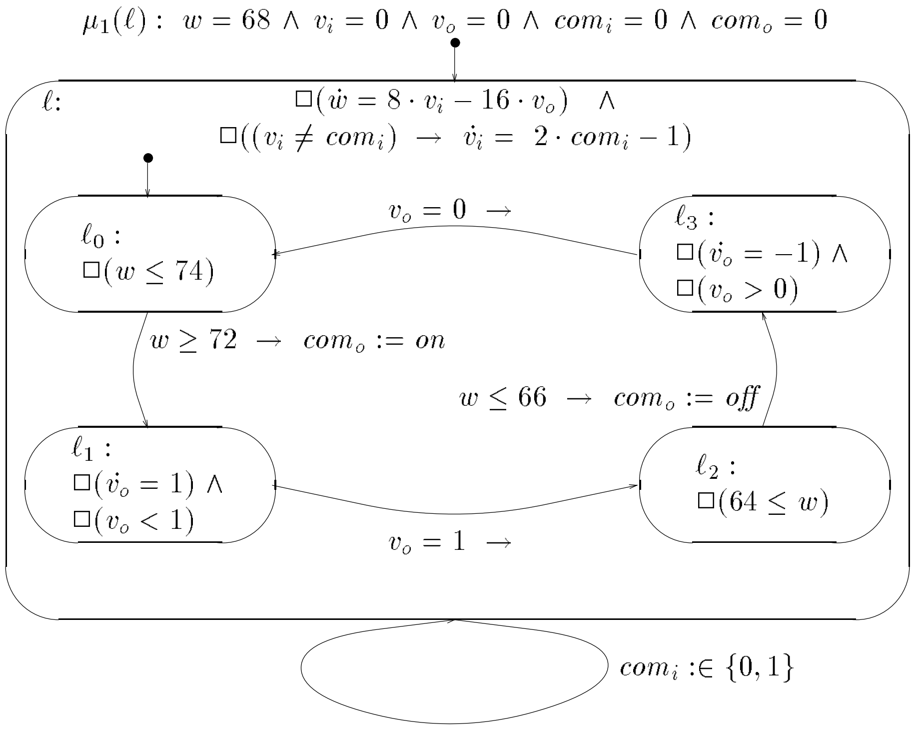

Figure 6: A true implementation of $\widehat{F}$

Acknowledgements. We gratefully acknowledge the help of Luca de Alfaro, Eddie Chang, Arjun Kapur, and Henny Sipma for their careful reading of the manuscript and thank them for many helpful suggestions.

References

[AL90] M. Abadi and L. Lamport. Composing specifications. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, Lecture Notes in Computer Science 430. Springer-Verlag, 1990.

[AH92] R. Alur and T.A. Henzinger. Logics and models of real time: a survey. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 74–106. Springer-Verlag, 1992.

[CHR92] Z. Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40:269–276, 1992.

[Hen92] T.A. Henzinger. Sooner is safer than later. *Information Processing Letters*, 43:135–141, 1992.

[HMP92] T.A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 226–251. Springer-Verlag, 1992.

[HNSY92] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Proceedings of the Seventh Annual Symposium on Logic in Computer Science*, pages 394–406. IEEE Computer Society Press, 1992.

[LA92] L. Lamport and M. Abadi. An old-fashioned recipe for real time. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 1–27. Springer-Verlag, 1992.

[MMP92] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 447–484. Springer-Verlag, 1992.

[Mos85] B. Moszkowski. A temporal logic for multi-level reasoning about hardware. *IEEE Computer*, 18(2):10–19, 1985.

[NSY92] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 549–572. Springer-Verlag, 1992.