Submitted for publication to
Annals of Mathematics and Artificial Intelligence (AMAI)

Focusing qualitative simulation using temporal logic: theoretical foundations

Giorgio Brajnik
Dip. di Matematica e Informatica
Università di Udine, Italy
giorgio@dimi.uniud.it

Daniel J. Clancy
Dept. of Computer Sciences
University of Texas at Austin
Austin, Texas 78712
clancy@cs.utexas.edu

Abstract

We illustrate TeQsim*, a qualitative simulator for continuous dynamical systems that combines the expressive power of qualitative differential equations with temporal logic to constrain and refine the resulting predicted behaviors. Temporal logic is used to specify constraints that restrict the simulation to a region of the state space and to specify trajectories for input variables. A propositional linear–time temporal logic is adopted, which is extended to a three valued logic that allows a formula to be conditionally entailed when quantitative information specified in the formula can be applied to a behavior to refine it. We present a formalization of the logic with correctness and completeness results for the adopted model checking algorithm. We show an example of the simulation of a non–autonomous dynamical system and illustrate possible application tasks, ranging from simulation to monitoring and control of continuous dynamical systems, where* TeQsim *can be applied.*

1 Introduction

Reasoning about change across time is a common problem within artificial intelligence and computer science in general. For systems with discrete state spaces, temporal logic (TL) provides a broad class of formalisms that have been used for such tasks as reasoning about the effect of actions, specifying and verifying correctness of reactive computer programs and synthesizing or analyzing discrete event systems [Thistle and Wonham, 1986; Emerson, 1990; Barbeau *et al.*, 1995; Jonescu and Lin, 1995]. Qualitative reasoning [Kuipers, 1994] has been used to reason about continuous change of dynamical systems

in the presence of incomplete knowledge. The expressiveness of the models used within qualitative simulation, however, is often limited to structural equations constraining the potential values for related variables. The modeler is unable to express behavioral information about the trajectory of a variable or relationships between the trajectories of interconnected variables. Such information allows the modeler to restrict the simulation to a region of the state space and to specify time–varying input variables.

The Temporally Constrained QSIM (TEQSIM, pronounced *tek'sim*) algorithm combines the expressive power of these two paradigms by interleaving temporal logic model checking with qualitative simulation. Temporal logic is used to specify qualitative and quantitative trajectory information that is incorporated into the simulation to constrain and refine the resulting behaviors.

Qualitative simulation constructs a set of possible behaviors consistent with a model of a dynamical system represented by a qualitative differential equation (QDE). The QSIM algorithm [Kuipers, 1994] represents the behavior of a dynamical system by an incrementally generated tree of qualitative states. Each state describes the system at either a time–point or over a time–interval between two points by a tuple of qualitative values for the variables specified within the QDE. Each qualitative value is described by a magnitude and a direction of change: the direction of change represents the sign of the variable's time derivative while the magnitude is defined upon a totally ordered set of distinctive landmark values and is either a landmark value or an interval between two landmark values. The simulator uses the constraints specified within the QDE along with continuity to derive a branching–time behavioral description. Each path within the tree represents a potential behavior of the system; branches result from inherent ambiguity within the qualitative description. Semi–quantitative simulation incorporates quantitative information into the qualitative simulation in the form of numeric ranges for landmark values and bounding envelopes for functions.

TEQSIM interleaves temporal logic model checking with qualitative simulation to obtain two major benefits. *Behavior filtering* tests each partial behavior against the set of temporal logic expressions representing trajectory constraints as the set of behaviors is incrementally generated. A behavior is eliminated from the simulation when it can be shown that all of its possible completions fail to model the set of temporal logic expressions. Thus, the space of the behavioral description is restricted to include only behaviors that can satisfy the temporal logic expressions. *Behavior refinement* integrates numeric information contained within the temporal logic expressions into the qualitative simulation to provide a more precise numerical description. This process restricts an individual behavior to include only those real valued interpretations which model the set of temporal logic expressions and therefore improves the precision of the prediction.

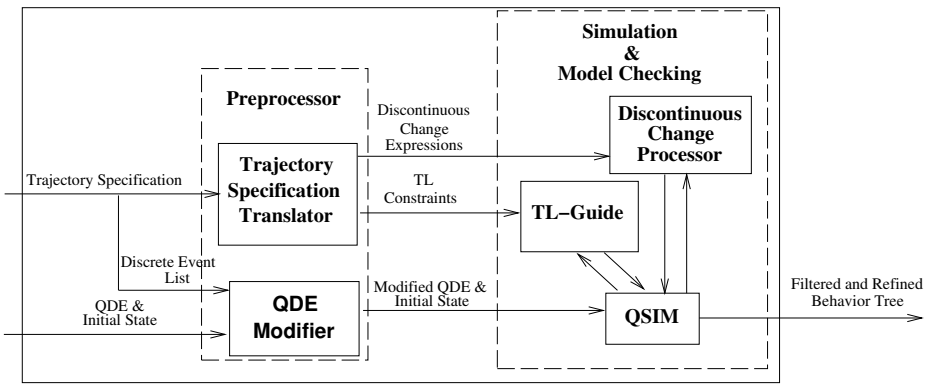


Figure 1: TeQsim architecture

Trajectory constraints extend the expressiveness of the modeling language and can be used to specify the behavior of time–varying input variables of non–autonomous systems; to reduce the complexity of a simulation by focusing the simulation on a region of the state space; to incorporate observations into the simulation; and to reason about boundary condition problems by specifying available information about final or intermediate states.

In this paper, we focus on a formal presentation of the model checking algorithm along with the syntax and semantics of the trajectory constraint language. Both the filtering and refinement of behaviors have been proven to be sound and complete with respect to the trajectory constraint language. The next section provides an overview of the TeQsim algorithm. Section 3 provides an example along with a discussion of some of the applications of this technique while section 4 describes the formal syntax and semantics of our temporal logic. Soundness and completeness theorems are presented along with the model checking algorithm in section 5. Finally, sections 6 and 7 provide a discussion of related work and some of the extensions being investigated.

2 TeQsim Overview

TeQsim accepts as input a model, an initial state, and a *trajectory specification*. The simulation is restricted to the region of the state space described by the constraints in the trajectory specification. A *trajectory* of a set of variables over a time interval $(a, b) \subseteq \Re^* = \Re \cup \{\pm\infty\}$ is defined as a mapping from (a, b) to variable values ($\subseteq \Re^*$). Three different types of expressions are used within the trajectory specification: temporal logic expressions, discontinuous change expressions and external event declarations. Figure 1 provides an overview of the system architecture.

An external event declaration defines a named, quantitatively bounded

event not represented within the QDE. An *event* is a time–point distinguished within the simulation. The declaration of external events introduces new relevant time–points that are inserted during the simulation. For example the declaration `(event open :time (2 4))` defines a time–point called `open` that is quantitatively constrained to occur at time $t \in [2, 4]$. References to external events included in both temporal logic and discontinuous change expressions allow the modeler to temporally constrain the information contained within these expressions and specify correlations between external events and other events generated during simulation. Currently, the specification of external events is limited to a totally ordered sequence.

The term *trajectory constraint* refers to both temporal logic and discontinuous change expressions. Temporal logic expressions (also called temporal logic constraints) are the primary method of specifying trajectory constraints and provide a unifying framework for filtering behaviors. They are discussed later on in the paper. Discontinuous change expressions are used to describe discontinuities in input variables. These expressions both expand (since discontinuous changes are not normally generated) and constrain (since behaviors that do not exhibit the change are filtered) the state space explored during the simulation. The discontinuous change processor injects the changes into the simulation by relaxing the continuity constraints imposed by QSIM. To restrict the simulation to behaviors that exhibit the specified discontinuity, discontinuous change expressions are translated into temporal logic constraints. Discontinuous change expressions and their application are discussed in more detail in [Brajnik and Clancy, 1996a].

The TL–Guide performs model checking and behavior refinement using the temporal logic constraints. Each time QSIM extends a behavior by the addition of a new state, the behavior is passed to TL–Guide to perform behavior filtering and refinement.

2.1 Temporal logic constraint language

The temporal logic constraint language (TLCL) is a variation of propositional linear–time temporal logic (PLTL). PLTL combines state formulæ, that express information about an individual state, with temporal operators such as *until*, *always*, and *eventually* to extend these state formulæ across time and represent change.

The atomic state formulæ used within TLCL describe qualitative and quantitative information about individual states. For example `(qvalue X ((0 X*) inc))` states that variable `X` is between `0` and `X*` and that it is increasing. Propositions such as `value-<=` and `value-in` are used to specify a numeric bound and a numeric range respectively. Boolean combinations of these atomic propositions are also allowed.

Temporal operators are applied to create *path formulæ* to extend the

information within the state formulæ across time. A path formula is defined recursively as either a state formula or a composition of two or more path formulæ using a temporal operator. Path formulæ are evaluated against sequences of states (i.e. behaviors). A path formula comprised of a single state formula is true of a behavior if it is true of the first state in the behavior.

Following Kuipers and Shults [Shults and Kuipers, 1996], we have adopted as basic temporal operators {`until` , `next` , and `strong-next` } and have defined additional ones using boolean operators. The path formula (`until` p q), where both p and q are path formulæ, is true for a behavior if p holds for all suffixes of the behavior preceding the first one where q holds, while (`strong-next` p) is true for a behavior if it contains at least two states and p holds in the behavior starting at the second state. `Next` is similar to `strong-next` except it is also true if the behavior consists of a single state. Other temporal operators can be defined as abbreviations from these two. The following abbreviations are used in the examples presented in the next section. A complete description of the syntax and the semantics for both state formulæ and path formulæ is provided in section 4.

(`always` p)	p is always true.
(`between` p q r)	between the first occurrence of p and the following occurrence of q, r is true.
(`occurs-at` p q)	q is true at the first occurrence of p.

3 Problem solving with TeQsim

The TeQsim algorithm has been tested on a range of examples demonstrating a variety of tasks. The following example demonstrates how TeQsim can be used to derive numeric bounds for a parameterized proportional controller. In addition, this example demonstrates the specification of a time–varying exogenous input to allow simulation of a non–autonomous system.

3.1 Parameter identification example

The model consists of a simple tank with a regulated output flow rate governed by the following equations: $\dot{V} = I - f(L, v)$ and $L = g(V)$. V denotes the volume of liquid in the tank, L the level, I the input flow rate, v the valve opening and $f(\cdot, \cdot)$ the output flow rate. f and g are partially known monotonic functions bounded by certain numeric functions.

TeQsim can be used to design a controller that maintains a constant water level ($L = L_s$) in the tank despite variations in the input flow rate. A proportional controller is added to the model to open or close the valve by an amount proportional to the error: $E = L - L_s$ and $v = v_s + KE$.

TeQsim derives bounds on the constant K that ensure that the closed-loop behavior of the system satisfies performance criteria specified via tra-

jectory constraints. Temporal logic constraints are used to specify a perturbation to the input flow rate, bounds on the overshoot of the controller, and bounds on the time required to return to the nominal state following the termination of the perturbation. The specification of such information via trajectory constraints is straightforward, and the computation of the solution is relatively inexpensive. The bounds provided are used by the quantitative reasoning component of QSIM to infer bounds on K. Figure 2 describes the trajectory constraints provided as input while figure 3 describes the results of the simulation.

Both components of TEQSIM, qualitative simulation and trajectory specification, are crucial when solving a problem of this nature. Qualitative simulation provides a discretization of trajectories essential for supporting a search mechanism. In addition, each qualitative behavior is refined via forward and backward propagation of quantitative information contained within the constraints. Trajectory constraints are used to drive the system by specifying the behavior of the exogenous variable (i.e. input flow rate) and by specifying the controller's performance. This information cannot be represented within the QDE.

Results produced by the method followed in this example should be carefully interpreted. The behavior produced by TEQSIM is not a proof that all real–valued systems satisfying the QDE do behave in the specified manner when K is within the predicted range. Due to QSIM incompleteness, the resulting behavior may be a spurious one. Therefore, TEQSIM provides a *relative* guarantee that *if* there exists a solution, then the solution satisfies the predicted quantitative bounds. Other methods can be applied to ensure that a solution exists, like Monte Carlo simulations.

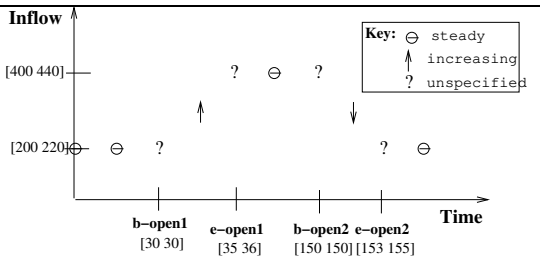
4 Temporal logic constraint language: syntax and semantics

Trajectory constraints are used to incrementally guide and refine the qualitative simulation. Two extensions are required to PLTL:

- A three value logic is used that allows an expression to be *conditionally entailed* when quantitative information contained within the expression can be applied to a behavior to refine the description. A *refinement condition* specifies numerical bounds extracted from the TL expressions. Application of these conditions to the behavior eliminates the region of the state space that extended beyond the quantitative information specified in the TL expression.

- To apply the model checking algorithm incrementally during the simulation, an *undetermined* result may occur when the behavior is insuf-

```
(event b-open1 :time 5)
(event e-open1 :time 6)
(event b-open2 :time 40)
(event e-open2 :time 41)
```



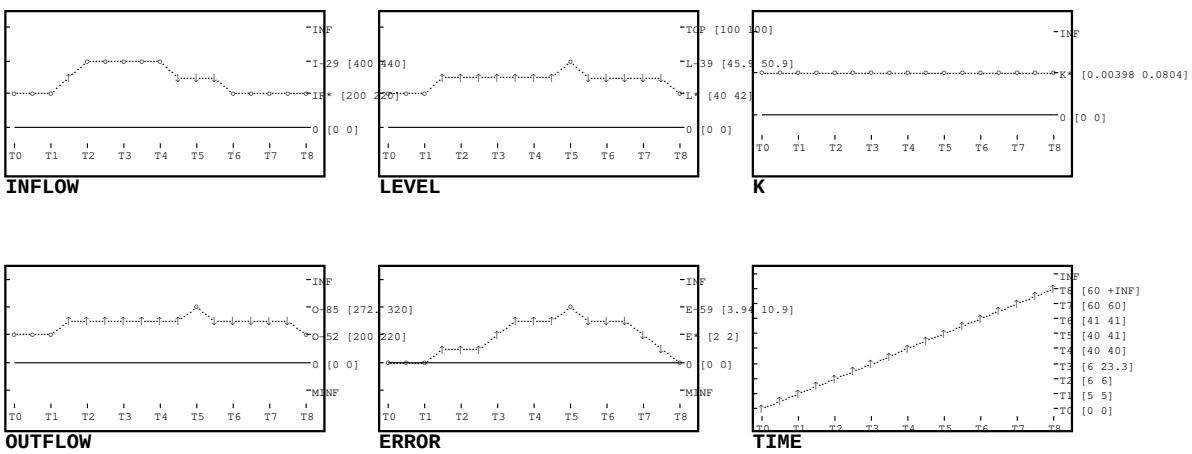
(a) External event declaration (b) Desired input trajectory for `Inflow`

	Temporal logic expression	**Description**
1	`(always (implies` (qvalue error (e* dec)) (value-in time (60 60))))	The system must settle (i.e. the error must go below e*) within 19 seconds of the end of the perturbation. (i.e. at t= 60s).
2	`(always (value-<= outflow 320))`	The outflow must remain below 320 cm/s.
3	`(until` (and (value-in inflow (200 220)) (std valve)) (event b-open1))	The inflow is constant in the range (200 220) until the beginning of the first opening action
4	`(between-events` (qvalue inflow (nil inc)) ((event b-open1) (event e-open1)))	Between events `b-open1` and `e-open1` (*i.e.* the duration of the first opening action) the inflow is increasing.
5	`(occurs-at` (event e-open1) (and (qvalue inflow ((if* inf) std)) (value-in inflow (400 440))))	Inflow is steady and in the range (400 440) after the first opening action.
...		

(c) Trajectory constraints (closing action not described)

- The external event declaration (a) defines four external events corresponding to the beginning and end of an opening and a closing action. A description of the time–varying input (b) is described using using temporal logic constraints that refer to these four external events (c–3,4,5).
- Trajectory constraints are also used to place an upper bound on the settling time along with a bound on the outflow rate (c–1,2).

Figure 2: Trajectory specification for parameter identification example



- TeQsim generates a single behavior using the trajectory constraints specified in figure 2. Notice that the behavior of `Inflow` matches the specification and that the bounds on `outflow` and `time` have been applied.
- Numerical information specified within the trajectory constraints is used to derive an upper and lower bound ([0.00398 0.0804]) for the constant **K**. The bounds can be seen in the time–plot for `K`.

Figure 3: Time plots generated by TeQsim

ficiently determined to evaluate the truth of a TL expression.

This section provides a formal description of the syntax and semantics for TLCL. The syntax and semantics are derived from work done by Emerson [Emerson, 1990] and Kuipers and Shults [Shults and Kuipers, 1996].

4.1 Syntax

The propositional part of the language include the following set $\mathcal{SF}$ of *state formulæ* (where v denotes a Qsim variable, $\mathcal{R}[v,s]$ the range of potential values for v in state s, v_s the unknown value of v in s, and n, n_i denote extended real numbers (*i.e.* $n, n_i \in \Re \cup \{\pm\infty\}$):

`(qvalue` v `(`*qmag qdir*`))` where *qmag* is a landmark or open interval defined by a pair of landmarks in the quantity space associated with v, and *qdir* is one of {`inc`, `std`, `dec`}. `NIL` can be used anywhere to match anything. Such a proposition is true exactly when the qualitative value of v in the state s matches the description `(`*qmag qdir*`)`.

`(value-<=` v n`)` is true iff $\forall x \in \mathcal{R}[v,s] : x \leq n$; it is false iff $\forall x \in \mathcal{R}[v,s] : n < x$; it is unknown otherwise. In such a case the refinement condition is that the least upper bound of the possible real values of v is equal to n (*i.e.* $v_s \leq n$). `(value->=` v n`)` is similar.

`(value-in` v `(`n_1 n_2`))` is true iff $\mathcal{R}[v,s] \subseteq [n_1, n_2]$; it is false iff $\mathcal{R}[v,s] \cap [n_1, n_2] = \emptyset$. It is unknown otherwise, and the refinement condition is that the greatest lower bound is equal to n_1 and the least upper bound is equal to n_2 (*i.e.* $n_1 \leq v_s \wedge v_s \leq n_2$).

Non–atomic propositions are defined using standard boolean operators (`and`, `not`); standard propositional abbreviations are also allowed (`true`, `false`, `or`, `implies`, `iff`). Similar to [Shults and Kuipers, 1996], other proposition schema are defined that enable the reference to attributes of states computed by QSIM (like whether a state is quiescent, is stable or occurs at infinite time).

Path formulæ are constructed from state formulæ by combining them using temporal operators. Apart from the boolean operators, the temporal operators `next`, `strong-next` and `until` are primitive ones, while the other ones can be derived from these using syntactic translation rules. `strong-next` is included besides `next` because we deal also with finite paths and want to provide sufficient expressiveness.

The set of *path formulæ* $\mathcal{PF}$ is defined by the following rule (where $p \in \mathcal{SF}$ and $\varphi \in \mathcal{PF}$):

$$\varphi ::= p | (\varphi \texttt{ and } \varphi) | (\texttt{not } \varphi) | (\texttt{strong-next } \varphi) | (\texttt{next } \varphi) | (\texttt{until } \varphi\ \varphi).$$

The following abbreviations are used:

```
        (or p q)  ≡  (not (and (not p) (not q)))
  (releases p q)  ≡  (not (until (not p) (not q)))
    (before p q)  ≡  (not (until (not p) q))
  (eventually p)  ≡  (until true p)
      (always p)  ≡  (releases false p)
       (never p)  ≡  (always (not p))
    (starts p q)  ≡  (releases p (implies p (always q)))
   (follows p q)  ≡  (releases p
                       (implies p (strong-next (always q))))
 (occurs-at p q)  ≡  (releases p (implies p q))
 (between p q r)  ≡  (releases p
                       (implies p (strong-next (until r q))))
```

These formulæ are translated into a *positive normal form* that is defined as follows: (i) `until`, `releases` and `strong-next` are the only temporal operators in the formula, (ii) for every `not` in the formula, its scope is an atomic proposition, and (iii) such a scope does not include any proposition

constructed using `value-<=`, `value->=` or `value-in`.
The first two requirements do not restrict the expressiveness of the language since the abbreviations shown above can be used to transform a formula to satisfy these conditions. The latter requirement is due to the specific representation of numeric information in QSIM, which does not allow open numeric intervals.

4.2 Semantics

Temporal logic formulæ are given meaning with respect to the interpretation structures defined below. These structures are extended from their typical definition (*e.g.* [Emerson, 1990]) in order to accommodate the refinement process.

Path formulæ are interpreted against an *interpretation structure* $M = <S, \sigma, \Gamma, \mathcal{I}, \mathcal{C}, \mathcal{M}>$ where:

- S is a set of states;
- $\sigma: S \rightarrow S$ is a partial function, mapping states to their successors (we are defining a linear–time logic on finite and infinite paths, hence each state has at most one successor);
- $\mathcal{I}: \mathcal{SF} \times S \rightarrow \{t, f, ?\}$ is an assignment of truth values to propositions and states (? denotes the "ambiguous" truth value);
- Γ is a set of *refinement conditions*. Γ is closed with respect to the standard boolean operators $\{\wedge, \neg\}$ and contains the distinguished item C_{true};
- $\mathcal{C}: \mathcal{SF} \times S \rightarrow \Gamma$ is a function (*condition generator*) that maps state formulæ and states into refinement conditions that disambiguate a formula truth value; we require that $\mathcal{C}(\varphi, s) = C_{\text{true}}$ iff $\mathcal{I}(\varphi, s) = t$ and that $\mathcal{C}(\varphi, s)$ is defined when $\mathcal{I}(\varphi, s) = ?$.
- $\mathcal{M}: \Gamma \times S \rightarrow S$ is a function (*state modifier*) that maps a condition and a state into a refined state. For any state s, $\mathcal{M}(C_{\text{true}}, s) = s$ and $\mathcal{M}(\neg C_{\text{true}}, s) = \perp$.

We require that if φ is an atomic proposition then refinement conditions are *necessary and sufficient* for resolving the ambiguity, *i.e.* if $C = \mathcal{C}(\varphi, s)$ then $\mathcal{I}(\varphi, \mathcal{M}(C, s)) = t$ and $\mathcal{I}(\varphi, \mathcal{M}(\neg C, s)) = f$ (unless $C = C_{\text{true}}$, in which case $\mathcal{M}(\neg C, s) = \perp$).

Given an interpretation M, as customary a *path* is defined as a sequence of states $x = <s_0, s_1, \ldots>$ such that for any pair of consecutive states (s_i, s_{i+1}) we have that $\sigma(s_i) = s_{i+1}$. The length of a path is denoted by $|x|$, which for infinite paths is ∞. For all non negative integers $i < |x|$, x^i denotes the sub–path $<s_i, \ldots>$, $x^{(i,j)}$ denotes $<s_i, \ldots, s_j>$ and $x(i)$ denotes s_i. A *full–path extension* of a finite path x, denoted with $\widehat{x}$, is an infinite path that has

x as a prefix. Finally, $\mathcal{M}$ is naturally extended to paths: if $x = <s_0, \ldots>$ then $x' = \mathcal{M}(C, x) = <s'_0, \ldots>$ where for all i, $s'_i = \mathcal{M}(C, s_i)$. If for some j, $\mathcal{M}(C, s_j) = \bot$ then $\mathcal{M}(C, x) = \bot$.

An interpretation M *is subsumed by* M' (written $M \preceq M'$) iff M' contains all the states and conditions of M, and the four functions of M are restrictions of those of M').

QSIM computes in finite time a set of behaviors, each representing a class of trajectories of the system being simulated. Although a QSIM behavior is a finite structure, it may represent infinite trajectories of the simulated system, as quiescent states are finite descriptions of fixed–point trajectories. A behavior $b = (s_0, \ldots, s_n)$ implicitly identifies a minimal (with respect to $\preceq$) interpretation structure M_b that:

1. $S = \{s_0, \ldots, s_n\}$;
2. (s_i, s_{i+1}) belongs to the QSIM relations `successor` or `transition` and for all $i = 0, \ldots, n-1 : \sigma(s_i) = s_{i+1}$;
3. Γ consists of the set of all possible numerical bounds on QDE variables; these bounds can be represented as (boolean combinations of) inequalities between the unknown value of the variable on a state and a real number (for example, the condition that the QDE variable X in state s has to be less than 5 is "$X_s < 5$").
4. $\mathcal{I}$ is determined by the qualitative values, numerical bindings and statuses of states; $\mathcal{I}$ may give ? only for propositions including `value-<=`, `value->=` and `value-in` as specified in section 4.1;
5. $\mathcal{C}$ is determined by numerical bindings in states;
6. $\mathcal{M}$ is determined by the numerical inference capabilities included in QSIM.

A behavior b is *closed* iff QSIM detected that s_n is a quiescent state or that s_n is a transition state that has no possible successors (signaling that the trajectory of the dynamical system has reached a boundary of the operating region of the model). For a closed behavior b its full–path extensions $\widehat{b} = b$. In the rest of the paper, when discussing a behavior b, we will implicitly assume to deal with interpretations M that subsume M_b.

Formulæ with quantitative information may lead to ambiguity when evaluated (i.e. the behavior only models a portion of the range specified). Ambiguity, however, is not purely a syntactic property, but rather depends on state information. For example, `(value-<= X .3)` will be (unconditionally) true on a state s where $\mathcal{R}[X, s] = [0, 0.25]$, but only conditionally true on s' where $\mathcal{R}[X, s'] = [0, 1.0]$. Because of ambiguity, to define the semantics of formulæ we need to introduce two entailment relations. The first one, called *models* ($\models$), is used to characterize non–ambiguous true formulæ; the

second one, called *conditionally–models* ($\models^?$) characterizes formulæ that are ambiguous. The following definition gives the semantics of the language.

Given an interpretation M, the relations *models* ($\models$) and *conditionally models* ($\models^?$) are defined as follows (we will write $x \models^* \varphi$ to mean ($x \models \varphi$ or $x \models^? \varphi$), and $x \not\models^* \varphi$ to mean that it is not the case that $x \models^* \varphi$):

State formulæ (a ranges over atomic propositions, p and q over $\mathcal{SF}$, s over S):

$$
\begin{array}{lll}
s \models a & \text{iff} & \mathcal{I}(a,s) = t \\
s \models^? a & \text{iff} & \mathcal{I}(a,s) = ? \\
s \models (p \texttt{ and } q) & \text{iff} & s \models p \text{ and } s \models q \\
s \models^? (p \texttt{ and } q) & \text{iff} & s \models^* p \text{ and } s \models^* q \text{ and } \models^? \text{ occurs at least once} \\
s \models (\texttt{not } p) & \text{iff} & s \not\models^* p \\
s \models^? (\texttt{not } p) & \text{iff} & s \models^? p.
\end{array}
$$

Path formulæ ($p \in \mathcal{SF}$ and $\varphi, \psi \in \mathcal{PF}$ and x is a non–empty path):

$$
\begin{array}{lll}
x \models p & \text{iff} & x(0) \models p \\
x \models^? p & \text{iff} & x(0) \models^? p \\
x \models (\texttt{strong-next } \varphi) & \text{iff} & |x| > 1 \text{ and } x^1 \models \varphi \\
x \models^? (\texttt{strong-next } \varphi) & \text{iff} & |x| > 1 \text{ and } x^1 \models^? \varphi \\
x \models (\texttt{next } \varphi) & \text{iff} & |x| > 1 \text{ implies } x^1 \models \varphi \\
x \models^? (\texttt{next } \varphi) & \text{iff} & |x| > 1 \text{ implies } x^1 \models^? \varphi \\
x \models (\texttt{until } \varphi\ \psi) & \text{iff} & \exists i \geq 0 : (x^i \models \psi \text{ and } \forall j < i : x^j \models \varphi) \\
x \models^? (\texttt{until } \varphi\ \psi) & \text{iff} & \exists i \geq 0 : (x^i \models^* \psi \text{ and} \\
 & & \forall j < i : x^j \models^* \varphi) \text{ and} \\
 & & \models^? \text{ occurs at least once} \\
x \models (\texttt{releases } \varphi\ \psi) & \text{iff} & \forall i \geq 0 : x^i \models \psi \text{ or} \\
 & & \exists j \geq 0 : x^i \models \varphi \text{ and } \forall k \leq j : w^k \models \psi \\
x \models^? (\texttt{releases } \varphi\ \psi) & \text{iff} & \forall i \geq 0 : x^i \models^* \psi \text{ or} \\
 & & \exists j \geq 0 : x^i \models^* \varphi \text{ and } \forall k \leq j : w^k \models^* \psi \text{ and} \\
 & & \models^? \text{ occurs at least once}
\end{array}
$$

The semantics of (φ `and` ψ), and (`not` φ) is similar to the propositional case. The definition of `releases` given here agrees with its syntactic equivalence.

To properly handle the refinement process, the usage of ambiguous formulæ must be restricted. The problem is that an arbitrary ambiguous formula may yield several alternative refinement conditions. A disjunction of

refinement conditions cannot be applied to states without requiring a change in the successor function σ and the introduction of a new behavior which is qualitatively identical to the original behavior in the tree. Two different types of disjunction can result from certain ambiguous formula. A *state disjunction* results from a disjunction of ambiguous state formulæ. For example, when interpreted against a particular state `(or (value-<= X 0.5) (value->= Y 15))` may yield the condition $(X_s \leq 0.5 \vee Y_s \geq 15)$. When applying such a condition to a state, $\mathcal{M}(C, s)$ yields two states – s' in which $X_{s'}$ is restricted to be ≤ 0.5 and s'' where $Y_{s''} \geq 15$. A *path disjunction*, on the other hand, occurs when an ambiguous formula is included in a path formula in such a manner that a sub–formula can be conditionally true for more than one sub–path. For example, in the path formula `(until p (value-<= X 0.5))` a disjunction occurs across sub–paths regarding when the condition $(X \leq 0.5)$ should be applied.

The following definitions restrict the syntax to formulæ that are well–behaved. A *potentially ambiguous formula* is: (i) any atomic proposition constructed using one of the following operators `value-<=`, `value->=` or `value-in`, or (ii) a path formula containing a potentially ambiguous sub–formula. *Admissible formulæ* are those formulæ φ that satisfy the following conditions:

1. φ is in positive normal form, and
2. if $\varphi =$ (`until` p q) then q is not potentially ambiguous, and
3. if $\varphi =$ (`releases` p q) then p is not potentially ambiguous, and
4. if $\varphi = (p$ `or` $q)$ then at most one of p and q is potentially ambiguous.

The following lemma guarantees that checking a model against an admissible formula does not lead to disjunction of conditions.

Lemma 1 *For all admissible formulæ φ and for any interpretation M and path x, if $x \stackrel{?}{\models} \varphi$ then any necessary and sufficient condition C for making x a model for φ* (i.e. $\mathcal{M}(C, x) \models \varphi$ *and* $\mathcal{M}(\neg C, x) \not\models \varphi$*) is either a single condition or a conjunction of conditions.*
The proof follows from properties of M on propositions after a case–by–case inductive analysis of formulæ.

Even though the restriction to admissible formulæ reduces expressiveness, such a restriction does not hinder the practical applicability of TeQsim. As long as important distinctions are qualitatively represented (using landmarks or events), most trajectory constraints can be cast into admissible formulæ. For example, the constraint that "until level goes above 50 the input flow rate has to be below 200" could be expressed with the following non–admissible formula `(until (value-<= InFlow 200) (value->= Level 50))` where the

two distinctions (200 and 50) do not correspond to qualitative landmarks. By adding a landmark to the quantity space of `Level` corresponding to the value 50, the formula can be rewritten in an admissible form (*i.e.* `(until (value-<= InFlow 200) (qvalue Level (lm-50 nil)))`).

5 Model checking

The model checking algorithm is designed to evaluate a behavior with respect to a set of admissible formulæ as the behavior is incrementally developed. This allows behaviors to be filtered and refined as early as possible during the simulation.

The algorithm computes (among other things) a truth value from the set $\{\mathtt{T}, \mathtt{F}, \mathtt{U}\}$. A definite answer (*i.e.* `T` or `F`) is provided when the behavior contains sufficient information to determine the truth value of the formula. For example, a non–closed behavior b will *not* be sufficiently determined with respect to the formula `(eventually` p`)` if p is not true for any suffix of b, since p may become true in the future. A behavior is considered to be *sufficiently determined* with respect to a formula whenever there is enough information within the behavior to determine a single truth value for all completions of the behavior. If a behavior is not sufficiently determined for a formula, then `U` is returned and the behavior is not filtered out by TEQSIM. Notice that indeterminacy is a property independent from ambiguity: the former is related to incomplete paths, while the latter deals with ambiguous information present in states of a path[1].

A behavior b is *sufficiently determined* for a positive normal formula φ (written $b \rhd \varphi$) iff $|b| > 0$ and one of the following conditions is met:

1. b is a closed behavior, or φ is a proposition
2. $\varphi = (p_1$ `and` $p_2)$ and either $\forall i : b \rhd p_i$ or $\exists i : b \not\models^{*} p_i$ and $b \rhd p_i$
3. $\varphi = (p_1$ `or` $p_2)$ and either $\forall i : b \rhd p_i$ or $\exists i : b \models^{*} p_i$ and $b \rhd p_i$
4. $\varphi =$ (`strong-next` p) and $b^1 \rhd p$
5. $\varphi =$ (`next` p) and $b^1 \rhd p$
6. $\varphi =$ (`until` $p\ q$) and $\exists i < |b| : (b^i \models^{*} q$ and $b^i \rhd q$ and $\forall j < i : b^j \models^{*} p$ and $b^j \rhd p)$
7. $\varphi =$ (`until` $p\ q$) and $\exists i < |b| : (b^i \not\models p$ and $b^i \rhd p$ and $\forall j \leq i : b^j \not\models q$ and $b^j \rhd q)$

[1]Inferences within the model checking algorithm are limited to the information contained within the behavior. Thus, even though the constraint `(constant X)` is included within a model, the model checking algorithm will not be able to determine that the formula `(eventually (qvalue X (nil inc)))` is false for all completions of a non–closed behavior.

8. $\varphi = (\texttt{releases}\ p\ q)$ and $\exists i < |b| : (b^i \stackrel{*}{\models} p$ and $b^i \rhd p$ and $\forall j \leq i : b^j \stackrel{*}{\models} q$ and $b^j \rhd q)$
9. $\varphi = (\texttt{releases}\ p\ q)$ and $\exists i < |b| : (b^i \stackrel{*}{\not\models} q$ and $b^i \rhd q$ and $\forall j < i : b^j \stackrel{*}{\not\models} p$ and $b^j \rhd p)$.

Graphically, assuming that b is not closed, for the last four cases $b \rhd \varphi$ holds when (symbols `+` and `-` written at position i mean that the formula is (conditionally) true or false, respectively, when checked against the sub–behavior starting at i):

case	formula	example	result

6	$\varphi = (\texttt{until}\ p\ q)$	`b: ooooooioooooooo`
		`p: ++++++`
		`q:  +`
			$b \stackrel{*}{\models} \varphi$

7	$\varphi = (\texttt{until}\ p\ q)$	`b: ooooooooooioooo`
		`p:  -`
		`q: ----------`
			$b \stackrel{*}{\not\models} \varphi$

8	$\varphi = (\texttt{releases}\ p\ q)$	`b: ooooooioooooooo`
		`p:  +`
		`q: ++++++++`
			$b \stackrel{*}{\models} \varphi$

9	$\varphi = (\texttt{releases}\ p\ q)$	`b: ooooooioooooooo`
		`p: -------`
		`q:  -`
			$b \stackrel{*}{\not\models} \varphi$

The relation $\rhd$ is important because when true then the truth of a formula is invariant with respect to the extension of a behavior. More specifically:

Lemma 2 (On extensions) *For any finite path x of an interpretation and any positive normal formula φ such that $x \rhd \varphi$ the following holds:*

$$x \models \varphi \iff \forall \widehat{x} : \widehat{x} \models \varphi, \text{ and}$$
$$x \stackrel{?}{\models} \varphi \iff \forall \widehat{x} : \widehat{x} \stackrel{?}{\models} \varphi$$

The proof is by induction on formula length.

The model checking algorithm is incremental, in that as QSIM extends a behavior with a new state, the model checking algorithm evaluates a certain formula only on the state. This is achieved by progressively decomposing path formulæ into two parts: a present component, that has to be true on the current state for having the original formula be true for the behavior starting from the state, and a future component, that has to be true on the behavior starting from the successor state.

A *progressed* formula is a path formula where these two components have been made explicit. Function Δ achieves this: $\Delta[\cdot]$ is applied to a normalized formula φ and returns a new formula equivalent to φ which is a boolean combination of sub-formulæ that are either propositions or path formulæ scoped by `next` or `strong-next`. Δ is defined as follows:

$$\begin{aligned}
\Delta[p] &= p, \text{ if } p \in \mathcal{SF} \\
\Delta[(\texttt{not } p)] &= (\texttt{not } p) \\
\Delta[(p \texttt{ and } q)] &= \Delta[p] \texttt{ and } \Delta[q] \\
\Delta[(p \texttt{ or } q)] &= \Delta[p] \texttt{ or } \Delta[q] \\
\Delta[(\texttt{next } p)] &= (\texttt{next } p) \\
\Delta[(\texttt{strong-next } p)] &= (\texttt{strong-next } p) \\
\Delta[(\texttt{until } p\ q)] &= \Delta[q] \texttt{ or } (\Delta[p] \texttt{ and } (\texttt{strong-next } (\texttt{until } p\ q))) \\
\Delta[(\texttt{releases } p\ q)] &= \Delta[q] \texttt{ and } (\Delta[p] \texttt{ or } (\texttt{next } (\texttt{releases } p\ q)))
\end{aligned}$$

We also ensure that a progressed formula is in disjunctive normal form (DNF), *i.e.* it is a disjunction of terms, each term being a conjunction of literals. A literal is either an atomic proposition, (`not` p) where p is a proposition, (`next` ϕ) or (`strong-next` ϕ) where ϕ is a path formula.

The following lemma provides an important characterization of progressed formulæ:

Lemma 3 (Equivalence) *For any path x and normalized formula φ:*

$$\begin{aligned}
x \models \varphi &\iff x \models \Delta[\varphi] \\
x \models^{?} \varphi &\iff x \models^{?} \Delta[\varphi] \\
x \rhd \varphi &\iff x \rhd \Delta[\varphi]
\end{aligned}$$

Notice also that φ is admissible iff at most one of the terms of $\Delta[\varphi]$ is potentially ambiguous.

The proof is by induction on formula length.

The following function extracts from a progressed formula its "present component". Sub-formulæ starting with `next` operators are essentially removed. Notice how dependence on s plays a role only for the `strong-next` formula; in all other cases s is ignored. For the `strong-next` formula the state is needed because checking the closedeness of the behavior ending at the current state is the "present component", and this cannot be done by relying solely on syntax. The *present formula extractor* $\mathcal{P}[\cdot,\cdot]$ maps a progressed formula $\varphi = \bigvee \tau_j$ and a state s into a proposition p such that p represents what must be true in s if φ has to be true on s and its successors. It is

defined as follows:

$$\mathcal{P}[\varphi, s] = \bigvee \mathcal{P}[\tau_j, s]$$

$$\mathcal{P}[\tau, s] = \begin{cases} \texttt{FALSE} & \text{if } <s> \text{ is closed and } \tau \text{ contains a} \\ & \quad \texttt{strong-next} \text{ literal} \\ \tau' & \text{where } \tau' \text{ results from } \tau \text{ after removing all } \texttt{next} \text{ and} \\ & \texttt{strong-next} \text{ literals.} \end{cases}$$

In addition, since the output of $\mathcal{P}$ is a proposition, we assume that a simplification step occurs, removing all redundant `TRUE`'s and `FALSE`'s from the resulting formula (for example $(P \texttt{ or FALSE})$ is simplified into P).

$\mathcal{P}$ removes all temporal sub–formulæ from its input.
For example, $\mathcal{P}[(P \texttt{ or } (\texttt{next } Q)), s] = P$ and $\mathcal{P}[(P \texttt{ and } (\texttt{next } Q)), s] = P$. Notice that in a closed behaviors terms containing a `strong-next` literal are "short–circuited" to `FALSE`.

The following function, on the other hand, is responsible for characterizing the future component of a formula. The future component depends on which sub-formulæ of the present component are true or false. For example, if the formula is $(Q \texttt{ or } (P \texttt{ and } (\texttt{next } \varphi)))$, where Q, P are propositions, then when Q is true the future component is the formula `TRUE`, whereas if Q is false and P true the future component is φ.

$\mathcal{F}[\cdot, \cdot]$ maps a progressed formula $\varphi = \bigvee \tau_j$ and a state s into another formula φ' that represents what must be true for the future of s if φ is true from s:

$$\mathcal{F}[\varphi, s] = \bigvee \mathcal{F}[\tau_j, s]$$

$$\mathcal{F}[\tau, s] \quad (\text{assume } \tau = (\bigwedge p_j) \wedge (\bigwedge(\texttt{next } \beta_j)) \wedge (\bigwedge(\texttt{strong-next } \gamma_j)) \text{ where } p_j \text{ are propositions})$$

$$= \begin{cases} \texttt{FALSE} & \text{if exists } p_k : s \not\models^* p_k \\ (\bigwedge \beta_j) \wedge (\bigwedge \gamma_j) & \text{if } s \models^* \mathcal{P}[\tau, s] \text{ and } <s> \text{ is not closed} \\ \texttt{TRUE} & \text{if } s \models^* \mathcal{P}[\tau, s] \text{ and } <s> \text{ is closed} \end{cases}$$

In this case too we assume that a simplification step occurs, removing all redundant `TRUE`'s and `FALSE`'s from the resulting formula.

Notice that if a term contains a `strong-next` literal and the behavior is closed then the term is "dropped" from the result. For example,

$$\mathcal{F}[(P \texttt{ and } (\texttt{next } Q)), s] = \begin{cases} Q & \text{if } s \models^* P \\ \texttt{FALSE} & \text{otherwise} \end{cases}$$

$$\mathcal{F}[(P \texttt{ or } (\texttt{next } Q)), s] = \begin{cases} \texttt{TRUE} & \text{if } s \models^* P \\ Q & \text{otherwise} \end{cases}$$

$$\mathcal{F}[(P \texttt{ or } (\texttt{strong-next } Q)), s] = \begin{cases} \texttt{TRUE} & \text{if } s \models^* P, \text{ non closed} \\ Q & \text{if } s \not\models^* P, \text{ non closed} \\ \texttt{FALSE} & \text{if } s \not\models^* P, \text{ closed} \end{cases}$$

So far we have defined formal tools that will assist in stating properties of actual algorithms for incremental model checking. We describe the algorithm, starting from the following procedure that checks if a state is a model of a state formula and computes refinement conditions.

$\Psi[\cdot,\cdot]$ maps a state formula and a state into a pair (v, c) where $v \in \{\mathtt{T}, \mathtt{F}\}$ and c is a necessary and sufficient refinement condition (possibly C_{true}). Since we don't give the definition of such an algorithm (because tedious and straightforward), we assume that $\Psi[\cdot,\cdot]$ is correct and complete, that is:

$$\Psi[p,s] = (\mathtt{T}, C) \iff s \models^{*} p \text{ and } \mathcal{M}(C,s) \models p \text{ and } \mathcal{M}(\neg C,s) \not\models p, \text{ and}$$
$$\Psi[p,s] = (\mathtt{F}, \neg C_{\text{true}}) \iff s \not\models^{*} p$$

Ψ applies to state formulæ. The next procedure, the core of the model checking algorithm, applies to path formulæ that have been progressed (*i.e.* are the output of Δ). We call it *extended propositional interpretation* because it "short–circuits" the temporal operators in the progressed formula (that is, the sub–formulæ whose top–level operator is `next` or `strong-next`). This function is responsible for determining whether the present component of a formula is true, conditionally true, or false with respect to a state and whether the behavior consisting of that state only is sufficiently determined with respect to the formula. In addition, it also computes the future component of the formula.

The extended propositional interpreter, $\Pi[\cdot,\cdot]$, maps a progressed formula φ and a state s into a triple $(v, c, \varphi') = (\Pi_v[\varphi,s], \Pi_c[\varphi,s], \Pi_f[\varphi,s])$ where $v \in \{\mathtt{T}, \mathtt{F}, \mathtt{U}\}$, c is a refinement condition and φ' is the future component of φ:

φ	$\Pi_v[\varphi,s]$	$\Pi_c[\varphi,s]$
proposition	$\Psi_v[\varphi,s]$	$\Psi_c[\varphi,s]$
(`not` p)	$\begin{cases} \mathtt{T} & \text{if } \Pi_v[p,s] = \mathtt{F} \\ \mathtt{F} & \text{otherwise} \end{cases}$	$\begin{cases} C_{\text{true}} & \text{when } \Pi_v[\varphi,s] = \mathtt{T}; \\ \neg C_{\text{true}} & \text{when } \Pi_v[\varphi,s] = \mathtt{F}; \end{cases}$
(`next` ϕ)	$\begin{cases} \mathtt{T} & \text{if } <s> \text{ is closed} \\ \mathtt{U} & \text{otherwise} \end{cases}$	C_{true}
(`strong-next` ϕ)	$\begin{cases} \mathtt{F} & \text{if } <s> \text{ is closed} \\ \mathtt{U} & \text{otherwise} \end{cases}$	$\begin{cases} \neg C_{\text{true}} & \text{if } \Pi_v[\varphi,s] = \mathtt{F} \\ C_{\text{true}} & \text{otherwise} \end{cases}$
$\bigwedge \phi_j$	$\begin{cases} \mathtt{F} & \text{if } \exists j : \Pi_v[\phi_j,s] = \mathtt{F} \\ \mathtt{T} & \text{if } \forall j : \Pi_v[\phi_j,s] = \mathtt{T} \\ \mathtt{U} & \text{otherwise} \end{cases}$	$\begin{cases} \neg C_{\text{true}} & \text{if } \Pi_v[\varphi,s] = \mathtt{F} \\ \bigwedge \Pi_c[\phi_j,s] & \text{otherwise} \end{cases}$
$\bigvee \tau_j$	$\begin{cases} \mathtt{F} & \text{if } \forall j : \Pi_v[\tau_j,s] = \mathtt{F} \\ \mathtt{T} & \text{if } \exists j : \Pi_v[\phi_j,s] = \mathtt{T} \\ \mathtt{U} & \text{otherwise} \end{cases}$	$\begin{cases} \neg C_{\text{true}} & \text{if } \Pi_v[\varphi,s] = \mathtt{F} \\ \bigvee \Pi_c[\tau_j,s] & \text{otherwise} \end{cases}$

φ	$\Pi_f[\varphi, s]$
proposition	$\begin{cases} \texttt{TRUE} & \text{if } \Pi_v[\varphi, s] = \texttt{T} \\ \texttt{FALSE} & \text{otherwise} \end{cases}$
(not p)	$\begin{cases} \texttt{TRUE} & \text{if } \Pi_v[\varphi, s] = \texttt{F} \\ \texttt{F} & \text{otherwise} \end{cases}$
(next ϕ)	$\begin{cases} \texttt{TRUE} & \text{if } \Pi_v[\varphi, s] = \texttt{T} \\ \phi & \text{otherwise} \end{cases}$
(strong-next ϕ)	$\begin{cases} \texttt{FALSE} & \text{if } \Pi_v[\varphi, s] = \texttt{F} \\ \phi & \text{otherwise} \end{cases}$
$\bigwedge \phi_j$	$\begin{cases} \texttt{TRUE} & \text{if } \Pi_v[\varphi, s] = \texttt{T} \\ \texttt{FALSE} & \text{fi } \Pi_v[\varphi, s] = \texttt{F} \\ \bigwedge \Pi[\phi_j, s] & \text{otherwise} \end{cases}$
$\bigvee \tau_j$	$\begin{cases} \texttt{TRUE} & \text{if } \Pi_v[\varphi, s] = \texttt{T} \\ \texttt{FALSE} & \text{fi } \Pi_v[\varphi, s] = \texttt{F} \\ \bigvee \Pi[\tau_j, s] & \text{otherwise} \end{cases}$

Given a behavior $b = <s_0, \ldots, s_n>$ and a formula φ define the *evaluation sequence* as:

$$\begin{aligned} \varphi_0 &= \Delta[\varphi] \\ v_i &= \Pi_v[\varphi_i, s_i] \\ c_i &= \Pi_c[\varphi_i, s_i] \\ \varphi_{i+1} &= \Delta[\Pi_f[\varphi_i, s_i]]. \end{aligned}$$

The following lemma characterizes useful properties of the evaluation sequence.

Lemma 4 (Properties of the evaluation sequence)

$$b^i \models \varphi_i \implies b^{i+1} \models \varphi_{i+1} \quad (1)$$

$$b^i \models^* \varphi_i \iff b^{i+1} \models^* \varphi_{i+1} \quad (2)$$

$$b^{(i,j)} \rhd \varphi_i \iff b^{(i+1,j)} \rhd \varphi_{i+1} \quad (3)$$

$$b^{(i,i)} \not\rhd \varphi_i \iff v_i = \texttt{U} \quad (4)$$

$$b^{(i,i)} \rhd \varphi_i \wedge s_i \models^* \varphi_i \iff v_i = \texttt{T} \quad (5)$$

$$b^{(i,i)} \rhd \varphi_i \wedge s_i \not\models^* \varphi_i \iff v_i = \texttt{F} \quad (6)$$

$$s \models \mathcal{P}[\varphi, s] \iff \Pi_c[\varphi, s] \equiv C_{\text{true}} \quad (7)$$

$$\mathcal{F}[\varphi, s] \equiv \Pi_f[\varphi, s] \quad (8)$$

The proof follows the definitions of $\mathcal{P}$, $\mathcal{F}$ and Π.

5.1 Temporal Logic Guide algorithm

Given these preliminaries we move now to the definition of the temporal logic guide and refine (TL–Guide) algorithm embedded in TEQSIM.

TL–Guide takes as input a non empty QSIM behavior b and a formula φ representing the user–defined temporal logic constraints. TL–Guide incrementally steps through the behavior calling TLG–1 as each new state is examined. The process terminates once it can be determined whether or not b models φ.

TL–Guide(b, φ) :
  $i := 0$;
  repeat
    v :=TLG–1$(b^{(0,i)}, \varphi)$;
    $i := i + 1$;
  until $i = |b|$ **or** $\neg v$

The core of TL–Guide is the procedure TLG–1[2]. TLG–1 takes as input a non–empty behavior and a formula φ. It evaluates the last state of the behavior with respect to a the progressed version of φ using Π and applies conditions when appropriate.

TLG–1$(<s_0, \ldots, s_i>, \varphi)$:
  done := false;
  if $i = 0$
    then $f := \Delta[\varphi]$
    else $f := s_{i-1}$.future;
  $(v, c, g) := \Pi[f, s_i]$;
  case v
    T: *apply–conditions*(c, s_i); done := true;
    F: *refute*$(<s_0, \ldots, s_i>)$; done := true;
    U: *apply–conditions*(c, s_i);
  endcase;
  if $v =$ U **then** s_i.future $:= \Delta[g]$;
  return(done)

The procedure *apply–conditions* applies refinement conditions to a state (we assume that it is a correct implementation of $\mathcal{M}$) while *refute* marks a behavior as inconsistent and removes it from the simulation agenda.

This straightforward algorithm invokes Π and Δ to compute the evaluation sequence of the behavior $b = <s_0, \ldots, s_i>$ with respect to φ. A progressed version of the formula is stored on the final state to be used in future calls to the function. Only the last two states are used within a single call to the function.

[2] TEQSIM implements TL–Guide by interleaving simulation steps (*i.e.* extending a behavior with a new state) with calls to TLG–1 as each state is created in a way that is equivalent to the definition given here.

A problem arises within this algorithm when a term in a progressed formula leads to refinement conditions whose application needs to be delayed until other terms within the formula are resolved into a definite truth value (*i.e.* `T` or `F`). The following example demonstrates this problem. Suppose $\varphi = (P$ `or` $(\texttt{next}\ \phi))$ where $P =$`(value-<= V .3)`. Assume that on a state s, P is conditionally true. Thus $<s> \rhd \varphi$ and $<s> \not\rhd (\texttt{next}\ \phi)$; notice that while $<s> \stackrel{?}{\models} \varphi$, this may not necessarily hold in the future. In fact, suppose the subsequent state is s':

- if $<s'> \models \phi$ then $<s, s'> \models \varphi$
- if $<s'> \not\models \phi$ then $<s, s'> \stackrel{?}{\models} \varphi$
- if $<s'> \not\rhd \phi$ then the algorithm must continue.

In this case extending the behavior with a new state did not change the status of $\rhd\varphi$, but changed the status of $\stackrel{*}{\models}\varphi$, resolving it into a stricter entailment relation.

The problem is due to the interaction between the local nature of Π and the concept of conditional entailment. Application of refinement conditions when ambiguity exists between $\models$ and $\stackrel{?}{\models}$ forces resolution of $\stackrel{*}{\models}$ with $\stackrel{?}{\models}$, leading to unnecessary elimination of real–valued trajectories and violating completeness. Due to the inability to retract conditions once they are applied to QSIM behaviors, their application must be delayed until the sub-formula in the non–ambiguous branch of the disjunction becomes sufficiently determined.

The following necessary and sufficient conditions characterize the situation described above when evaluating a progressed formula $\varphi = \tau \vee \tau_1 \vee \ldots \tau_k$ against a behavior b:

(i) $b \stackrel{?}{\models} \mathcal{P}[\tau, b(0)]$
(ii) the set $\{\tau_j : b \not\rhd \tau_j\}$ is not empty
(iii) none of the other terms τ_h are such that $b \stackrel{*}{\models} \tau_h$ and $b \rhd \tau_h$.

When these conditions are satisfied, then:

$$b \stackrel{?}{\models} \mathcal{P}[\varphi, b(0)] \not\Rightarrow b \stackrel{?}{\models} \varphi$$

which says that refinement conditions that are sufficient and necessary for the present component may not be necessary for the entire formula. When this situation occurs, φ is called *condition delaying expression*, the refinement condition generated is called a *delayed condition* and the τ_j mentioned in condition (ii) are called the *triggers* for the delayed condition.

After extending b into b' the delayed condition may be resolved. Which means that if a trigger becomes definitely true, then its delayed condition is not necessary any more (that is if $b' \rhd \tau_j \wedge b' \models \tau_j$ then condition (iii) is not true any more and we get $b' \rhd \varphi \wedge b' \models \varphi$). On the other hand, if a

trigger becomes definitely false but (ii) still holds (that is, there are other triggers) nothing has changed for the delayed conditions. If (ii) does not hold, however, then the delayed condition becomes necessary ($b' \rhd \tau_j \wedge b' \not\models \tau_j$ and not (ii) implies $b' \models^{?} \varphi \wedge b' \rhd \varphi$).

Once the problem is detected the following steps must be performed by the model–checking algorithm

1. if the formula φ is sufficiently determined, then model checking — that would normally stop — has to continue evaluating the triggers;
2. delayed conditions must be stored until their triggers are resolved and either condition (ii) or (iii) doesn't hold any more;
3. if a trigger evaluates to true, then its delayed conditions must be dropped;
4. if it evaluates to false, unless there are other triggers, its delayed conditions must be applied;
5. multiple delayed conditions may need to be stored as the algorithm progresses through the behavior.

The complete solution is based on an intertwined implementation of Π and TLG–1 so that the problem is detected and handled efficiently. The report [Brajnik and Clancy, 1996b] contains a detailed description of the algorithm along with soundness and completeness proofs. In this paper we assume that TL–Guide performs a correct detection and application of delayed refinement conditions.

The following theorem, proven in the appendix, characterizes the correctness and completeness properties of TL–Guide. It states that of the behaviors generated by QSIM exactly those that are not models of the temporal logic constraints are removed.

Theorem 1 (TL–Guide is sound and complete) *Given a* QSIM *behavior b and an admissible formula φ then TL–Guide:*

1. *refutes b iff for all full–path extensions $\widehat{b}$: $\widehat{b} \not\models \varphi$ and $b \rhd \varphi$.*
2. *retains b without modifying it iff*
   (a) *$b \models \varphi$ and $b \rhd \varphi$; or*
   (b) *$b \not\rhd \varphi$ and there is no necessary condition C for refining b into a model for φ (*i.e. *$\nexists C \neq C_{\text{true}}$ such that if $b'' = \mathcal{M}(\neg C, b)$ then for all full–path extensions $\widehat{b''}$: $\widehat{b''} \not\models \varphi$).*
3. *replaces b with b' iff*
   (a) *$b \models^{?} \varphi$ and $b \rhd \varphi$ and exists $C \neq C_{\text{true}}$ such that $b' = \mathcal{M}(C, b) \models \varphi$ and C is necessary (*i.e. *if there exists b'' such that $b'' = \mathcal{M}(\neg C, b)$ then for all full–path extensions $\widehat{b''}$: $\widehat{b''} \not\models \varphi$); or*

(b) $b \not\Vdash \varphi$ *and there is a necessary condition* $C \neq C_{\text{true}}$ *(such that if* $b'' = \mathcal{M}(\neg C, b)$ *then for all full–path extensions* $\widehat{b''}$*:* $\widehat{b''} \not\models^{*} \varphi$*) and* $b' = \mathcal{M}(C, b)$*.*

6 Related Work

The incorporation of trajectory information into a qualitative simulation has not been extensively explored in the literature. DeCoste [DeCoste, 1994] introduces *sufficient discriminatory envisionments* to determine whether a goal region is possible, impossible or inevitable from each state of the space. This is performed by generating the simplest state description that is sufficient for inferring these discriminations. Though similar in spirit, our work is: (i) more general, because TeQsim enables the user to address a wide category of problems, not limited to determining reachability of a state; (ii) semantically well–founded, based on temporal logic and (iii) formally proven to provide guaranteed results. Washio and Kitamura [Washio and Kitamura, 1995] present a technique that uses temporal logic to perform a *history oriented envisionment* to filter predictions. TeQsim, within a more rigorously formalized framework, provides a more expressive language, it refines behaviors as opposed to just filtering them, and it incorporates discontinuous changes into behaviors.

The integration of temporal logic model checking and qualitative simulation was initially investigated by Kuipers and Shults [Kuipers and Shults, 1994]. They use a branching–time temporal logic to prove properties about continuous systems by testing the entire behavioral description against a temporal logic expression. The appropriate truth value is returned depending upon whether or not the description satisfies the expression. Our work focuses on *constraining* the simulation as opposed to *querying* it after the simulation is completed.

The trajectory specification language can be compared to other formalisms for specifying temporal constraints. Our language is strictly more expressive than both Allen's *interval algebra* [Allen, 1984] and Dechter, Meiri and Pearl's *temporal constraint networks* [Dechter *et al.*, 1991] due to the ability to arbitrarily compose path formulæ using temporal operators. All of the relations defined by Allen can be expressed using the composition of temporal operators in addition to more complex relations. The usage of the language in TeQsimis also different: instead of asserting temporal constraints in a database of assertions and querying if certain combinations of facts are consistent, TeQsim checks that a database of temporally related facts generated by Qsim satisfy a set of temporal logic constraints.

Bhat et al. [Bhat *et al.*, 1995] present an algorithm for CTL* model checking. They use a set of inference rules that compute formula progression

in a similar way as we do. Our algorithm differs from theirs for two reasons. First, they don't deal with refinement, and this has the consequence that their algorithm does not have to handle (delayed or not) refinement conditions. Second, their algorithm drives the extension of paths, in the sense that the successor of a state is actually generated only when all formulæ have been progressed to `next` operators. In our case, TL–Guide is embedded in the QSIM architecture which drives the model checking activity.

Bacchus and Kabanza utilize temporal logic in a similar manner within the domains of forward chaining planning [Bacchus and Kabanza, 1995] and reactive planning [Bacchus and Kabanza, 1996]. They use temporal logic to place constraints on the path selected to reach a goal state and to express search control information. One of the primary differences between their work and ours is that planning is generally concerned with finding a single solution that satisfies the constraints while qualitative simulation must generate all solution paths. They also use a similar incremental model checking algorithm that uses formula progression.

7 Discussion and future work

TEQSIM supports a general methodology for incorporating arbitrary trajectory information into the qualitative simulation process. The incremental nature of the model checking algorithm aggressively filters and refines behaviors as early as possible to minimize the complexity of the simulation. In terms of model checking complexity, (assuming that the most expensive operation is checking a proposition, *i.e.* computing Ψ) TL–Guide applies Π at most once on each state of the behavior. Π evaluates the present component of the formula, which is a boolean formula in disjunctive normal form. With a caching mechanism this operation can be performed by evaluating each proposition at most once. Therefore TL–Guide performs essentially $O(Nn)$ propositional checks when the temporal constraints contain N different atomic propositions and the behavior contains n states. Furthermore, in all the examples tested, the practical time–complexity of a TEQSIM simulation is dominated by the inferences performed during simulation as opposed to model checking (e.g. quantitative inferences). In fact, often TEQSIM actually decreases the complexity of the simulation by filtering behaviors that are not relevant to the current task.

We are currently investigating the usage of TEQSIM to solve various tasks. In particular, we are applying it to the domain of risk assessment in water supply control, where models of lakes, rivers, and dams are simulated with the purpose of evaluating potential actions (e.g. changing the power requested to a turbine, or the opening of a floodgate). In addition, we are exploring the use of TEQSIM to perform design and validation of robust

controllers for hybrid systems.

The following extensions are being investigated to increase the expressiveness of the trajectory specification language.

Limited first order expressiveness — The temporal logic used is limited to PLTL and is unable to quantify over attributes of states. Certain trajectory constraints require the ability to refer to values across states within the behavior. For example, the specification of a decreasing oscillation requires the ability to compare the magnitude of a variable across states. A limited form of first order logic (like the half–order logic proposed in [Henzinger, 1990]) may provide a sufficiently expressive language while still giving satisfactory performance with respect to complexity.

Metric temporal logic — Due to the introduction of landmarks during the simulation process, QSIM behaviors are potentially infinite structures. Getting a definite answer for formulæ such as `(eventually` p`)` is not always possible when potentially infinite behaviors are encountered since it is always possible for p to occur in the future. Metric temporal logic [Alur and Henzinger, 1993] allows the definition of a horizon for a temporal logic expression. This would allow statements such as "within 50 seconds the tanks level reaches 70 inches." This statements is only expressible within our logic using an external predefined event. Such an extension offers the modeler more flexibility to express relevant constraints.

Functional envelopes — Semi–quantitative reasoning [Berleant and Kuipers, 1988] within TEQSIM uses interval bounds and static functional envelopes for monotonic functions to derive quantitative information about a behavior. NSIM [Kay and Kuipers, 1993] derives dynamic envelopes describing a variable's behavior with respect to time. Currently, only interval information can be specified within TEQSIM trajectory constraints. Extending the language to include information about bounding envelopes with respect to time would increase the precision of solutions computed by TEQSIM.

8 Conclusions

Qualitative simulation and temporal logic provide two alternative formalisms for reasoning about change across time. TEQSIM integrates these two paradigms by incorporating trajectory information specified via temporal logic into the qualitative simulation process. Behaviors that do not model the set of temporal logic expressions are filtered during simulation. Numeric information

specified within the TL expressions can be integrated into the simulation to provide a more precise numerical description for the behaviors which model these expressions.

The correctness of the TL–guide algorithm along with the correctness of QSIM guarantee that all possible trajectories of the modeled system compatible with the model, the initial state and the trajectory constraints are included in the generated behaviors. In addition, the completeness of TL–guide ensures that all behaviors generated by TEQSIM are potential models of the trajectory constraints specified by the modeler.

Acknowledgments

We would like to thank Benjamin Shults for letting us use part of his program to implement TEQSIM, and to the Qualitative Reasoning Group for many fruitful discussions.
QSIM, TEQSIM and other results of the Qualitative Reasoning Group are accessible by World-Wide Web via `http://www.cs.utexas.edu/users/qr`.
This work has taken place in the Qualitative Reasoning Group at the Artificial Intelligence Laboratory, The University of Texas at Austin. Research of the Qualitative Reasoning Group is supported in part by NSF grants IRI-9216584 and IRI-9504138, by NASA grants NCC 2-760 and NAG 2-994, and by the Texas Advanced Research Program under grant no. 003658-242.

References

[Allen, 1984] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[Alur and Henzinger, 1993] R. Alur and T. Henzinger. Real–time logics: complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993.

[Bacchus and Kabanza, 1995] F. Bacchus and F. Kabanza. Using temporal logic to control search in a forward chaining planner. In *Proc of the 3rd European Workshop on Planning (EWSP)*, Assisi, Italy, Aug. 1995. AAAI Press.

[Bacchus and Kabanza, 1996] F. Bacchus and F. Kabanza. Planning for temporally extended goals. In B. Clancey and D. Weld, editors, *Proc. of the Tenth National Conference on Artificial Intelligence*. AAAI Press, Aug. 1996.

[Barbeau *et al.*, 1995] M. Barbeau, F. Kabanza, and R. St-Denis. Synthesizing plant controllers using real-time goals. In *Proc. of IJCAI–95*, pages 791–798. IJCAI, Morgan Kaufmann Publishers, Inc., August 1995.

[Berleant and Kuipers, 1988] D. Berleant and B.J. Kuipers. Using incomplete quantitative knowledge in qualitative reasoning. In *Proc. of the Sixth National Conference on Artificial Intelligence*, pages 324–329, 1988.

[Bhat *et al.*, 1995] G. Bhat, R. Cleaveland, and O. Grumberg. Efficient on–the–fly model checking for CTL*. In *Proc. of Conference on Logic in Computer Science (LICS–95)*, 1995.

[Brajnik and Clancy, 1996a] G. Brajnik and D. J. Clancy. Temporal constraints on trajectories in qualitative simulation. In B. Clancey and D. Weld, editors, *Proc. of the Tenth National Conference on Artificial Intelligence*, pages 979–984. AAAI Press, Aug. 1996.

[Brajnik and Clancy, 1996b] G. Brajnik and D. J. Clancy. Temporal constraints on trajectories in qualitative simulation. Technical Report UDMI–RT–01–96, Dip. di Matematica e Informatica, University of Udine, Udine, Italy, January 1996.

[Dechter *et al.*, 1991] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[DeCoste, 1994] D. DeCoste. Goal–directed qualitative reasoning with partial states. Technical Report 57, The Institute for the Learning Sciences, University of Illinois at Urbana–Champaign, August 1994.

[Emerson, 1990] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 995–1072. Elsevier Science Publishers/MIT Press, 1990. Chap. 16.

[Henzinger, 1990] T. A. Henzinger. Half–order modal logic: how to prove real–time properties. In *Proceedings of the Ninth Annual Symposium on Principles of Distributed Computing*, pages 281–296. ACM Press, 1990.

[Jonescu and Lin, 1995] D. Jonescu and J.Y. Lin. Optimal supervision of discrete event systems in a temporal logic framework. *IEEE Transactions on Systems, Man and Cybernetics*, 25(12):1595–1605, Dec. 1995.

[Kay and Kuipers, 1993] H. Kay and B.J. Kuipers. Numerical behavior envelopes for qualitative models. In *Proc. of the Eleventh National Conference on Artificial Intelligence*. AAAI Press/MIT Press, 1993.

[Kuipers and Shults, 1994] B.J. Kuipers and B. Shults. Reasoning in logic about continuous change. In *Principles of Knowledge Representation and Reasoning (KR–94)*. Morgan Kaufmann Publishers, Inc., 1994.

[Kuipers, 1994] B.J. Kuipers. *Qualitative Reasoning: modeling and simulation with incomplete knowledge*. MIT Press, Cambridge, Massachusetts, 1994.

[Shults and Kuipers, 1996] B. Shults and B. Kuipers. Qualitative simulation and temporal logic: proving properties of continuous systems. TR AI96-244, University of Texas Artificial Intelligence Laboratory, January 1996. To appear in Artificial Intelligence Journal.

[Thistle and Wonham, 1986] J.G. Thistle and W.M. Wonham. Control problems in a temporal logic framework. *International Journal on Control*, 44(4):943–976, 1986.

[Washio and Kitamura, 1995] T. Washio and M. Kitamura. A fast history–oriented envisioning method introducing temporal logic. In *Ninth International Workshop on Qualitative Reasoning (QR–95)*, pages 279–288, Amsterdam, NL, May 1995.

Appendix

Theorem 1 (TL–Guide is sound and complete) *Given a* QSIM *behavior* $b = <s_0, \ldots, s_n>$ *and an admissible formula* φ *then TL–Guide:*

1. *refutes* b *iff for all full–path extensions* $\widehat{b}$*:* $\widehat{b} \not\models^{*} \varphi$ *and* $b \rhd \varphi$.
2. *retains* b *without modifying it iff*
   (a) $b \models \varphi$ *and* $b \rhd \varphi$*; or*
   (b) $b \not\rhd \varphi$ *and there is no necessary condition* C *for refining* b *into a model for* φ (i.e. $\nexists C \neq C_{\text{true}}$ *such that if* $b'' = \mathcal{M}(\neg C, b)$ *then for all full–path extensions* $\widehat{b''}$*:* $\widehat{b''} \not\models^{*} \varphi$).
3. *replaces* b *with* b' *iff*
   (a) $b \not\models^{?} \varphi$ *and* $b \rhd \varphi$ *and exists* $C \neq C_{\text{true}}$ *such that* $b' = \mathcal{M}(C, b) \models \varphi$ *and* C *is necessary* (i.e. *if there exists* b'' *such that* $b'' = \mathcal{M}(\neg C, b)$ *then for all full–path extensions* $\widehat{b''}$*:*$\widehat{b''} \not\models^{*} \varphi$*); or*
   (b) $b \not\rhd \varphi$ *and there is a necessary condition* $C \neq C_{\text{true}}$ *(such that if* $b'' = \mathcal{M}(\neg C, b)$ *then for all full–path extensions* $\widehat{b''}$*:* $\widehat{b''} \not\models^{*} \varphi$*) and* $b' = \mathcal{M}(C, b)$.

Proof. Let (v_i, c_i, φ_i) be the evaluation sequence of b with respect to φ. Numbers in parentheses refer to statements of lemma 4.

(1,$\Longrightarrow$): If b is refuted then $v_n = \texttt{F}$. Therefore (6) leads to $b^n \rhd \varphi_n \wedge b^n \not\models^{*} \varphi_n$. ¿From $b^n \rhd \varphi_n$ it follows that (3) $b^{n-1} \rhd \varphi_{n-1}$ until $b^0 \rhd \varphi_0$. ¿From $b^n \not\models^{*} \varphi$ it follows (2) that $b^n \not\models^{*} \varphi_n$. And by repeating this we get $b^0 \not\models^{*} \varphi_0$ from which the extensions lemma brings $\forall \widehat{b} : \widehat{b} \not\models^{*} \varphi$.

(1,$\Longleftarrow$): Assume $\widehat{b}$ is an arbitrary extension of b. $\widehat{b} \not\models^{*} \varphi$ and $b \rhd \varphi$ imply that (extensions lemma) $b \not\models^{*} \varphi$ and (progression lemma) $b^0 \not\models^{*} \varphi_0$ and $b^0 \rhd \varphi_0$. The latter implies (3) that $b^1 \rhd \varphi_1$ and so forth until $b^n \rhd \varphi_n$. Similarly we get $b1 \not\models^{*} \varphi_1$ and so forth until $b^n \not\models^{*} \varphi_n$. Now $b^n \rhd \varphi_n \wedge b^n \not\models^{*} \varphi_n$ leads (6) to $v_n = \texttt{F}$ and to refutation of b.

(2, $\Longrightarrow$): If b is retained then $v_n = \texttt{T}$ or $\texttt{U}$. If $v_n = \texttt{T}$ then (5) $b^n \models^{*} \varphi_n$ and $b^n \rhd \varphi_n$. Then this implies (1) $b^{n-1} \models^{*} \varphi_{n-1}$ and so forth until $b^0 \models^{*} \varphi_0$. Similarly for $b^0 \rhd \varphi_0$. At this point the extensions lemma leads to $b \models^{*} \varphi$ and $b \rhd \varphi$. If $v_n = \texttt{U}$ then (3,4) imply $b^n \not\rhd \varphi_n$,and repeated application of (3) leads to $b^0 \not\rhd \varphi_0$ and then

to $b \not\rhd \varphi$. To prove that no conditions needs to be applied, observe that if the behavior is not modified then all c_i have to be either C_{true} (which is trivial) or delayed conditions that have not been applied, that is they are not necessary.

(2.a, $\Longleftarrow$): From the hypothesis $b \models \varphi$ the progression lemma leads to $b^0 \models \varphi_0$; by repeatedly applying (1) we get that $b^1 \models \varphi_1, \ldots, b^n \models \varphi_n$ and since $b \rhd \varphi$ it follows that tehere are no delayed conditions that might be applied, hence $b^i \models \mathcal{P}[\varphi_i, s_i]$ and therefore $c_i = C_{\text{true}}$ and TLG–1 never refines any state of b.

(2.b, $\Longleftarrow$): Observe that from $b \not\rhd \varphi$ the progression lemma yields $b^0 \not\rhd \varphi_0$ and that by repeatedly applying (3) we get $b^i \not\rhd \varphi_i$ for all i from 0 to n. Since $b^i \not\rhd \phi$ implies $b^{(i,i)} \not\rhd \phi$, then $v_i = \mathtt{U}$ holds and none of the $b^{(0,i)}$ is ever refuted. To prove that b is not refined if there is no necessary refinement condition, let's assume the contrary, *i.e.* that b is refined. Therefore there is at least one $c_j \not\equiv C_{\text{true}}$ that is applied to b. Hence $b^j \stackrel{?}{\models} \varphi_j$. c_j is also necessary for b, in the sense that any model of φ has to agree with c_j. Or, in other terms, if a behavior agrees with $\neg c_j$ then it cannot be a model for φ. To prove this observe that if $\neg c_j$ is applied to $b(j)$ we obtain a new behavior b'. But $b'^j \not\models \varphi_j$ and so forth until $b' \not\models \varphi$. In addition $b'^j \rhd \varphi_j$ and down to $b' \rhd \varphi$. At this point the extensions lemma yields that none of the possible extensions of b' is a model for φ, contrary to our hypothesis.

(3, $\Longrightarrow$): If b has been refined then at least one of the c_i is $\not\equiv C_{\text{true}}$, and $v_n = \mathtt{T}$ or $\mathtt{U}$. If $v_n = \mathtt{T}$ then $b^n \not\models \varphi_n$ and $b^n \rhd \varphi_n$. The former implies that $b \not\models \varphi$ and the latter $b \rhd \varphi$. Therefore in the case $v_n = \mathtt{T}$ behavior b is a potential model of φ. In case $v_n = \mathtt{U}$ then $b^n \not\rhd \varphi_n$ and then the conclusion $b \not\rhd \varphi$ follows. To prove that condition C is necessary, observe that if b is refined with C then at least one of the c_i has to be $\not\equiv C_{\text{true}}$ and applied. Then $s_i \stackrel{?}{\models} \mathcal{P}[\varphi_i, s_i]$ and therefore C is necessary and sufficient.

(3, $\Longleftarrow$): We have to prove that b is refined using a condition C that is necessary and sufficient. Since C has to be a conjunction of elementary conditions, we have to prove that TL–Guide applies all of these conditions and nothing more. Take any element c' of C. Then c' refers to a state s_j and $s_j \stackrel{?}{\models} \mathcal{P}[\varphi_j, s_j]$ and, since Ψ is complete, $c_j \equiv c'$. c_j is either applied directly or it is delayed (in which case, since c' is necessary then the trigger of c' has to be definitely false and c' is applied). In neither case TL–Guide applies any additional conditions. □