Hybrid Systems in TLA$^+$

Leslie Lamport
Digital Equipment Corporation
Systems Research Center
`lamport@src.dec.com`

6 April 1993

To appear in: Hans Rischel and Anders P. Ravn, editors, *Hybrid Systems*, Lecture Notes in Computer Science, Berlin, 1993. Springer-Verlag. (Proceedings of a Workshop on Theory of Hybrid Systems, held 19–21 October 1992 at Lyngby, Denmark.)

Hybrid Systems in TLA$^+$

Leslie Lamport

Digital Equipment Corporation
Systems Research Center

Abstract. TLA$^+$ is a general purpose, formal specification language based on the Temporal Logic of Actions, with no built-in primitives for specifying real-time properties. Here, we use TLA$^+$ to define operators for specifying the temporal behavior of physical components obeying integral equations of evolution. These operators, together with previously defined operators for describing timing constraints, are used to specify a toy gas burner introduced by Ravn, Rischel, and Hansen. The burner is specified at three levels of abstraction, each of the two lower-level specifications implementing the next higher-level one. Correctness proofs are sketched.

1 Introduction

TLA$^+$ is a formal specification language based on TLA, the Temporal Logic of Actions [5]. We use TLA$^+$ to specify and verify a toy hybrid system—a gas burner described by Ravn, Rischel, and Hansen (RRH) [8]. The TLA$^+$ specification and proof can be compared with the one by RRH that uses the Duration Calculus.

We do not expect TLA$^+$ to permit the best possible specification of this or any other particular example. The specification of a gas burner is likely to be simpler in a formalism devised expressly for the class of hybrid systems that includes the gas burner. The specification of a Modula-3 procedure is likely to be simpler in a formalism for specifying Modula-3 procedures. But, while TLA$^+$ may not be the best method for specifying any particular system, we believe it is quite good for specifying a very wide class of systems, including gas burners and Modula-3 procedures.

There are two reasons for using TLA$^+$ instead of a language tailored to the specific problem. First, specialized languages often have limited realms of applicability. A language that permits a simple specification for one gas burner might require a very complicated one for a different kind of burner. The Duration Calculus seems to work well for real-time properties; but it cannot express simple liveness properties. A formalism like TLA$^+$ that, with no built-in primitives for real-time systems or procedures, can easily specify gas burners and Modula-3 procedures, is not likely to have difficulty with a different kind of gas burner.

The second reason for using TLA$^+$ is that formalisms are easy to invent, but practical methods are not. A practical method must have a precise language and robust tools. Building tools is not easy. It is hard to define a language that is powerful enough to handle practical problems and yet has a precise formal semantics. Such a language is a prerequisite for any sound, practical method. The advantage of not having to implement a new method for every problem domain is obvious.

An important criterion for choosing a formalism is how good it is for formal verification. A method based on a logic has an advantage over one based on an abstract programming language (such as CSP) because one does not have to translate from the specification language to a logic for reasoning. But, not all logics are equal. TLA works well in practice because most of the reasoning is in the domain of actions, which is the realm of "ordinary" mathematical reasoning. The use of temporal logic is minimal and highly ritualized. The temporal structure of the proofs are the same, regardless of whether one is reasoning about a mutual exclusion algorithm or a gas burner.

Modal logics, such as temporal logic and the Duration Calculus, are more difficult to use than ordinary logic. One reason is that the deduction principle, from which one deduces $P \Rightarrow Q$ by assuming P and proving Q, is invalid for most modal logics. In our work on the mechanical verification of TLA [4], we have found formalizing temporal logic reasoning to be much more difficult than formalizing ordinary mathematical reasoning. Temporal logic proofs that look simple when done by hand can be tedious to check mechanically. We believe that mechanical verification of TLA proofs is feasible largely because it involves very little temporal logic. One should be skeptical of claims that reasoning in a modal logic is easy in the absence of experience with mechanically checked proofs.

TLA$^+$ is a complete language, with a precise syntax and formal semantics. At the moment, only a few syntactic issues remain unresolved. Because it is completely formal, some things are a little more awkward to express in TLA$^+$ than in "semi-formal" methods. For example, most semi-formal methods, such as Unity [2] and the temporal logic of Manna and Pnueli [7], allow Boolean-valued variables; TLA$^+$ does not. In TLA$^+$, one cannot declare x to be a Boolean variable and write $\Box x$, one must instead write something like $\Box(x = \text{“T”})$. Although seemingly innocuous, Boolean variables pose the following problem. A specification may require an array $x[1]$, …, $x[17]$ of flexible variables. Formally, such an array is a variable x whose value is a function with domain $\{1, \ldots, 17\}$. If one can declare Boolean variables, then one should also be able to declare that x is a Boolean-valued array variable, with index set $\{1, \ldots, 17\}$. One can then write the formula $\Box(x[14])$. One can also write the formula $\Box(x[i^2 - 23])$. In general, one could write the formula $\Box(x[e])$ for an arbitrary integer-valued expression e. Formalizing Boolean arrays presents the following options:

– Define the meaning of $\Box(x[e])$ for any value of e. Does one consider $x[e]$ to have some special undefined value $\perp$ if e is not in the domain of x? If so, what is the meaning of $\Box\perp$? Is the formula $\Box(\perp \Rightarrow \perp)$ valid?
– Declare $\Box(x[e])$ not to be a wff (well formed formula) if e is not in the index set of x. This leads to two possibilities:
  • The class of expressions e that can appear in the formula $x[e]$ are restricted so it is syntactically impossible for the value of e not to be in the index set of x. This leads to unnatural restrictions on formulas.

- Whether or not a formula is a wff becomes a semantic rather than syntactic property. It is a strange formalism in which it is undecidable whether a formula is a wff.

TLA$^+$ avoids this problem by not having Boolean-valued variables. Neither Manna and Pnueli [7] nor Misra and Chandy [2], in books that are hundreds of pages long, indicate how they formalize Boolean arrays.

There are dozens of similar issues that must be resolved in designing a complete language. A simple informal specification might not look so simple if it had to be written formally. Of particular concern are formalisms based on a type system. It is easy to introduce the informal notation that $Length(s)$ denotes the length of the sequence s. But, will the type system really allow a formal language in which the user can define $Length(s)$ to denote the length of *any* sequence s? It is easy to define such a *Length* operator in TLA$^+$. However, this ability is based on a distinction between a function and an operator—a distinction one won't find in semi-formal methods.

Despite being completely formal, TLA$^+$ is simple enough for practical applications. A general treatment of hybrid systems requires continuous mathematics. Specifying the gas burner requires defining the Riemann integral $\int_a^b f$ of the function f over the closed interval $[a, b]$. Assuming only the set of real numbers with the usual arithmetic operations and ordering relations, the entire definition takes about 15 lines. Our example is a toy one, and we do not claim to have formalized any significant fraction of the mathematics that will be needed in practical applications. We do claim that a language that can define the Riemann integral in 15 lines is powerful enough to express any mathematical concepts likely to arise in real specifications.

TLA$^+$ specifications are written in ASCII. We hope eventually to write a program that converts an ASCII specification to input for a text formater that produces a "pretty-printed" version. The user will specify how individual operators are formated—for example, declaring that the `Integral` operator should be formated so `Integral(a, b, f)` is printed as $\int_a^b f$, and that `[|s|]` should be printed as $|s|$. We have simulated such a program to produce pretty-printed TLA$^+$ specifications.

2 Representing Hybrid Systems with TLA

This section describes the generic operators that can be used to specify hybrid systems. The precise TLA$^+$ definitions of some operators is deferred to Section 3. Figure 1 contains a complete list of all the predefined TLA$^+$ constant operators—the ones that describe data structures. The only additional operators are TLA's action operators (such as *Enabled*) and temporal operators (such as $\Box$). The syntax and formal semantics of these operators, which fit on one page, can be found in [5].

Logic

true false $\wedge$ $\vee$ $\neg$ $\Rightarrow$ $\Leftrightarrow$

$\forall x : p$ $\exists x : p$ $\forall x \in S : p$ $\exists x \in S : p$

choose $x : p$ [Equals some x satisfying p]

Sets

$=$ $\neq$ $\in$ $\notin$ $\cup$ $\cap$ $\subseteq$ $\setminus$ [set difference]

$\{e_1, \ldots, e_n\}$	[Set consisting of elements e_i]
$\{x \in S : p\}$	[Set of elements x in S satisfying p]
$\{e : x \in S\}$	[Set of elements e such that x in S]
subset S	[Set of subsets of S]
union S	[Union of all elements of S]

Functions

domain f $f[e]$ [Function application]

$[x \in S \mapsto e]$	[Function f such that $f[x] = e$ for $x \in S$]
$[S \rightarrow T]$	[Set of functions f with $f[x] \in T$ for $x \in S$]
$[f;\ e_1 \mapsto e_2]$	[Function $\widehat{f}$ equal to f except $\widehat{f}[e_1] = e_2$]
$[f;\ e : S]$	[Set of functions $\widehat{f}$ equal to f except $\widehat{f}[e] \in S$]

Records

$e.x$	[The x-component of record e]
$[[x_1 \mapsto e_1, \ldots, x_n \mapsto e_n]]$	[The record whose x_i component is e_i]
$[[x_1 : S_1, \ldots, x_n : S_n]]$	[Set of all records with x_i component in S_i]
$[[r;\ x \mapsto e]]$	[Record $\widehat{r}$ equal to r except $\widehat{r}.x = e$]
$[[r;\ x : S]]$	[Set of records $\widehat{r}$ equal to r except $\widehat{r}.x \in S$]

Tuples

$e[i]$	[The i^{th} component of tuple e]
$\langle e_1, \ldots, e_n \rangle$	[The n-tuple whose i^{th} component is e_i]
$S_1 \times \ldots \times S_n$	[The set of all n-tuples with i^{th} component in S_i]

Miscellaneous

"$c_1 \ldots c_n$"	[A literal string of n characters]
if p **then** e_1 **else** e_2	[Equals e_1 if p true, else e_2]
case $p_1 \rightarrow e_1, \ldots, p_n \rightarrow e_n$	[Equals e_i if p_i true]
let $x_1 \triangleq e_1 \ \ldots \ x_n \triangleq e_n$ **in** e	[Equals e in the context of the definitions]

Fig. 1. The constant operators of TLA$^+$.

module *RealTime*

import *Reals*

---

parameters

now : **variable**

∞ : **constant**

---

assumption

$$\mathit{InfinityUnReal} \triangleq \infty \notin \mathbf{R}$$

temporal

$$\begin{aligned} RT(v) \triangleq\ & \land\ \mathit{now} \in \mathit{Real} \\ & \land\ \Box[\land\ \mathit{now}' \in \{r \in \mathbf{R} : r > \mathit{now}\} \\ & \qquad \land\ v' = v\ ]_{\mathit{now}} \end{aligned}$$

$$\begin{aligned} & \mathit{VTimer}(x : \textbf{state fcn}, \mathcal{A} : \textbf{action}, \delta, v : \textbf{state fcn}) \triangleq \\ & \quad \land\ x = \textbf{if } \mathit{Enabled}\langle\mathcal{A}\rangle_v \textbf{ then } \mathit{now} + \delta \\ & \qquad\qquad\qquad\qquad\qquad \textbf{else } \infty \\ & \quad \land\ \Box[x' = \textbf{if } (\mathit{Enabled}\langle\mathcal{A}\rangle_v)' \textbf{ then if } \langle\mathcal{A}\rangle_v \textbf{ then } \mathit{now}' + \delta \\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else } x \\ & \qquad\qquad\qquad\qquad\qquad\quad \textbf{else } \infty\ ]_{\langle x,\, v\rangle} \end{aligned}$$

$$\mathit{MaxTimer}(x) \triangleq \Box[(x \neq \infty) \Rightarrow (\mathit{now}' \leq x)]_{\mathit{now}}$$

$$\mathit{MinTimer}(x : \textbf{state fcn}, \mathcal{A} : \textbf{action}, v : \textbf{state fcn}) \triangleq \Box[\langle\mathcal{A}\rangle_v \Rightarrow (\mathit{now} \geq x)]_v$$

Fig. 2. The *RealTime* module.

2.1 Real Time in TLA

A method for writing real-time specifications in TLA is described in [1]. We now review this approach and introduce TLA$^+$ by defining the operators from [1] in the TLA$^+$ module *RealTime* of Figure 2.

The *RealTime* module first imports the module *Reals*, which is assumed to define the set $\mathbf{R}$ of real numbers with its usual arithmetic operators.[1] The module declares two parameters: the (flexible) variable *now* and the constant ∞. The value of *now* represents the current time, which can assume any value in $\mathbf{R}$. The constant ∞ is the usual infinity of mathematicians. (All symbols that appear in a module's formulas are either parameters or symbols that are defined in terms of parameters. The **import** statement includes the parameters of the module *Reals* as parameters of *RealTime*.)

Next comes an assumption, named *InfinityUnReal*, which asserts that ∞ is not a real number. The keyword **temporal** heads a list of definitions of temporal formulas. The first definition in the list defines $RT(v)$ to be a formula asserting

[1] Some people mistakenly think that, because the reals are uncountable, letting time be a real number complicates proofs. The axioms about the real numbers needed to prove such real-time properties as the correctness of Fischer's protocol [1] form a decidable theory.

that *now* is initially in $\mathbf{R}$, and every step either (i) sets the new value of *now* to a real number greater than its current value and (ii) leaves the value of v unchanged, or else it leaves *now* unchanged. (A list of formulas bulleted with $\wedge$ denotes their conjunction. The formula $[\mathcal{A}]_w$ is defined to equal $\mathcal{A} \vee (w' = w)$; a $[\mathcal{A}]_w$ step is thus either an $\mathcal{A}$ step or a step that leaves w unchanged.) In representing a system by a TLA formula, the discrete variables of the system are considered to change instantaneously, meaning that when they change, *now* remains unchanged. This is asserted by the formula $RT(v)$ when v is the tuple whose components are the system's discrete variables.

In [1], timing constraints are expressed through the use of timer variables. The formula $VTimer(x, \mathcal{A}, \delta, v)$ asserts that x is a timer whose value is initially either (i) δ greater than the initial value of *now* or (ii) ∞, depending on whether or not action $\langle\mathcal{A}\rangle_v$ is enabled. The value of x is set to $now + \delta$ when either (i) $\langle\mathcal{A}\rangle_v$ becomes enabled or (ii) an $\langle\mathcal{A}\rangle_v$ step occurs that leaves $\langle\mathcal{A}\rangle_v$ enabled, and it is set to ∞ when $\langle\mathcal{A}\rangle_v$ becomes disabled.[2] (The action $\langle\mathcal{A}\rangle_v$ is defined to equal $\mathcal{A} \wedge (v' \neq v)$; an $\langle\mathcal{A}\rangle_v$ step is thus an $\mathcal{A}$ step that changes v.) The formula $MaxTimer(x)$ asserts that changing *now* cannot make it greater than x. Thus, the formula $VTimer(x, \mathcal{A}, \delta, v) \wedge MaxTimer(x)$ implies that an $\langle\mathcal{A}\rangle_v$ action cannot be continuously enabled for more than δ seconds without having occurred. The formula $MinTimer(x, \mathcal{A}, v)$ asserts that an $\langle\mathcal{A}\rangle_v$ action cannot occur unless $now \geq x$, so $VTimer(x, \mathcal{A}, \delta, v) \wedge MinTimer(x, \mathcal{A}, v)$ implies that an $\langle\mathcal{A}\rangle_v$ action must be enabled for at least δ seconds before it can occur.

The definitions of *VTimer* and *MinTimer* explicitly indicate the *sorts* of the parameters. When no sort is specified, the sort **state fcn**, denoting a mapping from states to values, is assumed—except in the definitions of constants, where the default sort is **constant**, denoting a constant value.

There are many ways of defining timers for expressing real-time constraints, and they are all easily expressed in TLA$^+$. The method used in [1] is probably not optimal for the gas burner example. Although we might be able to simplify the specifications in this example by defining a new kind of timer, in practice one would use a fixed set of operators defined in a standard module like *RealTime*. To make the example more realistic, we have used a pre-existing set of operators.

2.2 Hybrid Systems

To represent hybrid systems in TLA, continuous system variables are represented by variables that change when *now* does. The gas-burner specification of RRH can be expressed using only the timers introduced in [1]. However, RRH's specification is somewhat artificial, apparently chosen to avoid reasoning with continuous mathematics. Instead of the natural requirement that the concentration of unburned gas never exceeds some value, RRH require that unburned gas never be released for more than 4 seconds out of any 30-second period. Because it poses an interesting new challenge for TLA$^+$, we specify the more natural requirement and prove that it is implied by the requirement of RRH.

[2] Another type of timer is also defined in [1], but it is not needed here.

The gas concentration g will satisfy an integral equation of the form

$$g(t) = \int_{t_0}^{t} F(g(t))\,dt$$

where the function F depends on the discrete variables. A more general situation is a continuous variable f that satisfies an equation of the form

$$f(t) = f_0 + \int_{t_0}^{t} G(f(t), t)\,dt \tag{1}$$

A further generalization is to let f be a function whose range is the set of n-tuples of reals. However, this generalization is straightforward and is omitted.

For specifying hybrid systems, we define a TLA formula, pretty-printed as $[x = c + \int G \mid \mathcal{A}, v]$, having the following meaning:

- Initially, x equals c.
- In any step that changes now, the new value x' of x equals $f(now')$, where f is the solution to $f(t) = x + \int_{now}^{t} G(f(s), s)\,ds$.
- Any step that leaves now unchanged leaves x unchanged, unless it is an $\langle \mathcal{A} \rangle_v$ step, in which case x' equals zero.

Thus, $[x = c + \int G \mid \mathsf{false}, v]$ asserts that x represents the solution to (1) with f_0 equal to c and t_0 the initial value of now. The general formula $[x = c + \int G \mid \mathcal{A}, v]$ adds the requirement that x is reset to 0 by an $\langle \mathcal{A} \rangle_v$ action. The precise definition of $[x = c + \int G \mid \mathcal{A}, v]$ is given later, in the *HybridSystems* module.

It is often useful to describe the amount of time a predicate P has been true, which is the integral over time of a step function that equals 1 when P is true and zero when it is false. The formula $[x = 0 + \int \chi(P) \mid \mathsf{false}, v]$ asserts that the value of x always equals this integral, where the function $\chi(P)$ is defined by

$$\chi(P)(r, s) = \begin{cases} 1 \text{ if } P = \mathsf{true} \\ 0 \text{ if } P = \mathsf{false} \end{cases}$$

The formal definition of χ appears in the *HybridSystems* module.

3 The Gas Burner

The example system is the gas burner shown in Figure 3. This is the toy example of RRH. Our goal is to write a specification that is in the spirit of RRH's; the specification of a real gas burner might be much more complex.

The discrete state of the system consists of the states of the gas, the heat-request signal, the flame, and the ignition—each of which can be either on or off. These state components are represented by the four variables declared in the module *BurnerParameters* of Figure 4. A physical state in which the gas is turned on or off will be represented by a state in which the value of the variable *gas* is the string "on" or "off". The *on* and *off* values of the other variables are similarly denoted by the strings "on" and "off". As is customary in TLA

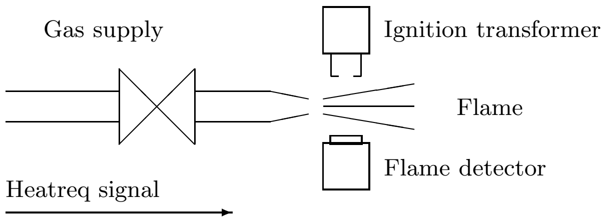

Fig. 3. The gas burner. (Figure taken from [8].)

module *BurnerParameters*

import *HybridSystems*, *RealTime*

---

parameters

$gas,\ heatReq,\ flame,\ ignition$: **variables**

---

state function

$v \ \triangleq\ \langle gas,\ heatReq,\ flame,\ ignition \rangle$

predicates

$Gas \ \triangleq\ gas = \text{“on”}$

$Flame \ \triangleq\ flame = \text{“on”}$

$Heatreq \ \triangleq\ heatReq = \text{“on”}$

$Ignition \ \triangleq\ ignition = \text{“on”}$

Fig. 4. Module *BurnerParameters*

specifications, we define a state function v that equals the tuple of all relevant variables. (In TLA$^+$, angle brackets $\langle\ \rangle$ denote tuples.) To make it easier to compare our specification with theirs, we have also defined state predicates that correspond to the Boolean variables of RRH.

For convenience, the *BurnerParameters* module imports the modules *HybridSystems* and *RealTime* that define the operators described above. Since the *RealTime* module also imports the real numbers, they are transitively imported by the *BurnerParameters* module.

We now specify the requirement that the concentration of gas remains less than some value *MaxCon*. We assume that the gas concentration g is described by the integral equation

$$g(t) = \int_{t_0}^{t} -\delta g(t) + \left\{ \begin{array}{l} \rho \text{ if gas on and flame off} \\ 0 \text{ otherwise} \end{array} \right\} dt$$

where δ is the rate at which gas diffuses away from the burner and ρ is the rate

module *GasConcentration*
import *BurnerParameters*

parameters
$\delta,\ \rho,\ \mathit{MaxCon} : \{r \in \mathbf{R} : r > 0\}$ **const**

state function
$\mathit{accumRate}[r,\ s : \mathbf{R}] \ \triangleq\ (-\delta) * r + (\textbf{if}\ \ \mathit{Gas} \land \neg \mathit{Flame}\ \ \textbf{then}\ \ \rho\ \ \textbf{else}\ \ 0)$
temporal
$\mathit{Req0} \ \triangleq\ \boldsymbol{\forall}\, g : [g = 0 + \int \mathit{accumRate} \mid \mathsf{false}, v] \Rightarrow \Box(g < \mathit{MaxCon})$

Fig. 5. The specification of the gas-concentration requirement

at which gas flows when it is turned on.

Using the TLA formula $[x = c + \int G \mid \mathcal{A}, v]$, it is easy to specify the requirement that the gas concentration is always less than *MaxCon*. First, the function *accumRate* that is substituted for G is defined as follows.

$$\mathit{accumRate}[r,\ s : \mathbf{R}] \ \triangleq\ (-\delta) * r + (\textbf{if}\ \ \mathit{Gas} \land \neg \mathit{Flame}\ \ \textbf{then}\ \ \rho\ \ \textbf{else}\ \ 0)$$

This defines *accumRate* to be a function whose domain is $\mathbf{R} \times \mathbf{R}$ such that $\mathit{accumRate}[r, s]$ equals the expression to the right of the $\triangleq$, for every pair $\langle r,\ s\rangle$ in $\mathbf{R} \times \mathbf{R}$. (In TLA$^+$, square brackets denote function application.)

The formula $[g = 0 + \int \mathit{accumRate} \mid \mathsf{false}, v]$ then asserts that g describes the gas concentration. As usual, the temporal formula $\Box(g < \mathit{MaxCon})$ asserts that g is always less than *MaxCon*, so the formula

$$[g = 0 + \int \mathit{accumRate} \mid \mathsf{false}, v] \Rightarrow \Box(g < \mathit{MaxCon})$$

asserts that if g is the gas concentration, then g is always less than *MaxCon*. We want this formula to be true for an arbitrarily chosen, "fresh" variable g, so the desired property is obtained by universally quantifying g.[3]

The complete definition of this property, which is called *Req0*, is given in module *GasConcentration*, shown in Figure 5. This module imports the *BurnerParameters* module, which defines *Gas* and *Flame*, and declares δ, ρ, and *MaxCon* to be constant parameters whose values lie in the set of positive reals. This "type declaration" is a shorthand for the **assumption**

$$(\delta \in \{r \in \mathbf{R} : r > 0\})\ \land\ (\rho \in \{r \in \mathbf{R} : r > 0\})\ \land\ (\mathit{MaxCon} \in \{r \in \mathbf{R} : r > 0\})$$

Next, we define the three requirements given by RRH. The first requirement is

[3] The symbols $\boldsymbol{\exists}$ and $\boldsymbol{\forall}$ denote quantification over flexible variables; $\exists$ and $\forall$ denote ordinary quantification over rigid variables.

> For Safety, gas must never leak for more than 4 seconds in any period of at most 30 seconds.

This is a requirement for the 30-second interval beginning at time r, for every real r. It is expressed in terms of an array x of timers, where $x[r]$ is the timer used to express the requirement for the interval beginning at r. The amount of gas that has leaked during the interval $[r, r+30]$ is obtained by "integrating" the function $G(r)$ that is defined by

$$G(r)[s, t : \mathbf{R}] \;\triangleq\; \textbf{if}\;\; \mathit{Gas} \wedge \neg \mathit{Flame} \wedge (t \in [r, r+30]) \;\;\textbf{then}\;\; 1 \;\;\textbf{else}\;\; 0$$

This defines G to be an operator such that, for any r, the expression $G(r)$ denotes a function with domain $\mathbf{R} \times \mathbf{R}$.[4]

Using $x[r]$ as a timer, the requirement for this interval is expressed by

$$[x[r] = 0 + \textstyle\int G(r) \mid \mathsf{false}, v] \;\Rightarrow\; \Box(x[r] \leq 4))$$

The requirement *Req1* is obtained by quantifying over all real numbers r, and then universally quantifying x. The complete definition is given in module *BurnerRequirements* in Figure 6. (Our formulas *Req1*, *Req2*, and *Req3* correspond to the formulas $\Box \mathit{Req}_1$, $\Box \mathit{Req}_2$, and $\Box \mathit{Req}_3$ of RRH.)

The second requirement is

> Heat request off shall result in the flame being off after 60 seconds.

This condition is expressed by using a variable x that integrates the amount of time the flame has been on while the heat request was off, and is reset to zero whenever the flame goes off or the heat request comes back on.[5] Such a variable x satisfies the formula

$$[x = 0 + \textstyle\int \chi(\mathit{Flame} \wedge \neg \mathit{HeatReq}) \mid \mathit{Heatreq}' \vee \neg \mathit{Flame}', v]$$

Asserting that x is always less than 60 and quantifying over x yields condition *Req2* of Figure 6.

The final requirement is

[4] A function f has a domain; it is an ordinary value. Thus $G(7)$ and $G(7)[\sqrt{2}, .5]$ both denote values; the first is a function, the second a real number. To define a function, one must specify its domain. However, G is an operator, not a function. It is not a value, and it does not have a domain; the symbol G by itself is not a syntactically correct expression. The definition above defines $G(r)$ for any value of r, not just for real numbers. We had to define $G(r)$ to be a function because it appears as a function, without any arguments, in subsequent formulas. We could have defined G to be a function by writing $G[r : \mathbf{R}][\ldots]$, but there is no need to make G a function, and we didn't feel like writing the "$: \mathbf{R}$". The distinction between operators and functions is another example of the details that arise in defining a precise language with a formal semantics. This distinction is what allows one easily to define in TLA$^+$ an operator *Length* such that $\mathit{Length}(s)$ is the length of *any* sequence s. One could not define *Length* to be a function, since its domain would not be a set. (Its domain would be isomorphic to the "set" of all sets.)

[5] This requirement, which comes from RRH, is satisfied if the heat request is always off and the flame is always on except for flickering out briefly every 59.9 seconds.

module *BurnerRequirements*
import *BurnerParameters*

temporal

$$Req1 \triangleq \textbf{let } G(r)[s, t : \mathbf{R}] \triangleq \textbf{if } Gas \land \neg Flame \land (t \in [r, r+30]) \textbf{ then } 1 \textbf{ else } 0$$
$$\textbf{in } \forall x : \forall r \in \mathbf{R} : [x[r] = 0 + \textstyle\int G(r) \mid \mathsf{false}, v] \Rightarrow \Box(x[r] \leq 4))$$

$$Req2 \triangleq \forall x : [x = 0 + \textstyle\int \chi(Flame \land \neg HeatReq) \mid Heatreq' \lor \neg Flame', v] \Rightarrow \Box(x \leq 60)$$

actions

$$IgniteFailAct(y) \triangleq \land\ Gas \land Ignition \land \neg Flame$$
$$\land\ y' \neq y$$
$$\land\ \textbf{unchanged}\langle v,\ now\rangle$$

$$Req3Reset(y) \triangleq Flame' \lor (y' \neq y) \lor \neg Heatreq'$$

temporal

$$IgniteFail(y) \triangleq \exists z : \land\ \Box[IgniteFailAct(y)]_y$$
$$\land\ VTimer(z, IgniteFailAct(y), 1/2, y)$$
$$\land\ MaxTimer(z)$$
$$\land\ MinTimer(z, IgniteFailAct(y), y)$$

$$Req3 \triangleq \forall x, y : \land\ IgniteFail(y)$$
$$\land\ [x = 0 + \textstyle\int \chi(Heatreq) \mid Req3Reset(y), \langle v,\ y\rangle]$$
$$\Rightarrow \Box(x \leq 60)$$

Fig. 6. The three requirements of Ravn, Rischel, and Hansen.

.

> Heat request shall after 60 seconds result in gas burning unless an ignite or flame failure has occurred. An ignite failure happens when gas does not ignite after 0.5 seconds. The flame fails if it disappears while gas is supplied.

A careful analysis of this condition reveals that replacing “after 60 seconds” by “within 60 seconds” makes the “or flame failure” redundant. We can therefore rewrite this requirement as

> Heat request shall, within 60 seconds, result in gas burning unless an ignite failure has occurred.

The obvious approach is to define a variable x that integrates the time during which the heat request is on and is reset by the presence of a flame, an ignite failure, or the heat request being turned off. We would then define an *IgniteFail* action, define the action *Req3Reset* to be $Flame' \lor IgniteFail \lor \neg Heatreq'$ and write the requirement as

$$\forall x : [x = 0 + \textstyle\int \chi(Heatreq) \mid Req3Reset, v] \Rightarrow \Box(x \leq 60)$$

However, there is no actual *IgniteFail* action. An ignite failure is not a change of the discrete state variables; it is something that is caused by the passage of time. So, we must introduce a variable y that is changed when an ignite failure occurs. Since y should not be a free variable of the formula, it must be "quantified away". We want y to change when the ignition and the gas have been on for precisely .5 seconds while the flame has been off. This is accomplished with a .5-second "vtimer" for the action that changes y. We define $\mathit{IgniteFailAct}(y)$ to be an action that changes y while leaving the other variables unchanged, and is enabled precisely when the gas and ignition are on and the flame is off. An ignition failure happens when this action has been continuously enabled for precisely .5 seconds. So, we let

$$\begin{array}{rcl} \mathit{IgniteFailAct}(y) & \triangleq & \land\ \mathit{Gas} \land \mathit{Ignition} \land \neg \mathit{Flame} \\ & & \land\ (y' \neq y) \land \textbf{unchanged}\ \langle v, \mathit{now} \rangle \end{array}$$

where **unchanged** f denotes $f' = f$. The TLA formula $\Box[\mathit{IgniteFailAct}(y)]_y$ asserts that y changes only when $\mathit{Gas} \land \mathit{Ignition} \land \neg \mathit{Flame}$ is true. To assert that this change occurs only when the $\mathit{IgniteFailAct}(y)$ action has been enabled for precisely .5 seconds, we add a timer z with both a *MaxTimer* and *MinTimer* condition. The temporal formula $\mathit{IgniteFail}(y)$ defined in Figure 6 asserts that an ignition failure has occurred iff y has changed. The definition of the formula *Req3* expressing the third requirement is now straightforward and appears in Figure 6.

We would like to prove the gas concentration condition *Req0* from the three requirements of RRH. This condition actually follows from *Req1*. The precise theorem, expressed in TLA$^+$, is that if

$$\mathit{MaxCon} \geq 4 * \rho * (1 + (1/(1 - \exp[(-30) * \delta])))$$

then $RT(v) \land \mathit{Req1} \Rightarrow \mathit{Req0}$, where exp is the usual exponential function. Before proving this, we finish the specification by defining the operators we have been assuming.

4 The Module *HybridSystems*

The *HybridSystems* module involves the solution to an integral equation. Defining this requires first defining the definite integral. The Riemann integral of a function f on an interval is the signed area under the graph of f. It is defined as the limit of approximations obtained by breaking the interval into subintervals, where the area under the graph of f from p to q is approximated by $f(p)(q-p)$. The definition of $\int_a^b f$ appears in module *Integration* of Figure 7. The informal meanings of the defined operators are explained below.

$\mathbf{R}^+$ The set of positive reals.

$[a, b]$ The closed interval from a to b. This is an "unsigned" interval, where $[a, b]$ equals $[b, a]$. (To avoid writing explicit domains, we have defined $[-, -]$ to

module *Integration*

import *Reals*

constants

$\mathbf{R}^+ \triangleq \{r \in \mathbf{R} : 0 < r\}$

$[a, b] \triangleq \{r \in \mathbf{R} : ((a \leq r) \wedge (r \leq b)) \vee ((a \geq r) \wedge (r \geq b))\}$

$|r| \triangleq$ **if** $r < 0$ **then** $-r$ **else** r

$\{m \dots n\} \triangleq \{i \in Nat : (m \leq i) \wedge (i \leq n)\}$

$$\begin{aligned} &Partition(a, b, n, \delta) \triangleq \\ &\quad \{ p \in [\{0 \dots n{+}1\} \rightarrow [a, b]] : \wedge (p[0] = a) \wedge (p[n{+}1] = b) \\ &\qquad \wedge \forall i \in \{0 \dots n\} : \wedge \vee (a \leq b) \wedge (p[i] \leq p[i{+}1]) \\ &\qquad\qquad \vee (a \geq b) \wedge (p[i] \geq p[i{+}1]) \\ &\qquad\qquad \wedge |p[i{+}1] - p[i]| < \delta \quad \} \end{aligned}$$

$$\begin{aligned} \Sigma(f, p)[n : Nat] \triangleq\ & f[p[n]] * (p[n{+}1] - p[n]) \\ & + \textbf{ if } n = 0 \textbf{ then } 0 \textbf{ else } \Sigma(f, p)[n-1] \end{aligned}$$

$$\begin{aligned} \int_a^b f \triangleq\ & \textbf{choose } r : \wedge r \in \mathbf{R} \\ & \wedge \forall \epsilon \in \mathbf{R}^+ : \exists \delta \in \mathbf{R}^+ : \forall n \in Nat : \\ & \quad \forall p \in Partition(a, b, n, \delta) : |r - \Sigma(f, p)[n]| < \epsilon \end{aligned}$$

$$\begin{aligned} &[c{+}\textstyle\int_a G][r : \mathbf{R}] \triangleq \\ &\quad \textbf{let } f \triangleq \textbf{choose } f : \wedge f \in [[a, r] \rightarrow \mathbf{R}] \\ &\qquad \wedge \forall t \in [a, r] : f[t] = c + \textstyle\int_a^t [s \in \mathbf{R} \mapsto G[f[s], s]] \\ &\quad \textbf{in } f[r] \end{aligned}$$

theorem

$$\begin{aligned} &IntegralOfStepFcn \triangleq \\ &\quad \forall G : \forall a, b, \delta \in \mathbf{R} : \forall n \in Nat : \forall p \in Partition(a, b, n, \delta) : \forall f \in [\mathbf{R} \rightarrow \mathbf{R}] : \\ &\qquad (\forall s, t \in \mathbf{R} : \forall i \in \{0 \dots n\} : (p[i] < t) \wedge (t < p[i+1]) \Rightarrow (G[s, t] = f(p[i]))) \\ &\qquad\quad \Rightarrow ([c{+}\textstyle\int_a G][b] = \Sigma(f, p)[n]) \end{aligned}$$

Fig. 7. Defining integration in TLA$^+$.

be an operator, so $[a, b]$ equals the expression to the right of the $\triangleq$ even if a and b are not real numbers. However, it has the expected meaning only when a and b are real numbers. This remark applies to all the operators in this module.)

$|r|$ The absolute value of r.

$\{m \dots n\}$ The set of natural numbers from m through n. (We assume that the set *Nat* of natural numbers is imported with the *Reals* module.)

$Partition(a, b, n, \delta)$ The set of all possible partitions of the interval $[a, b]$ into $n + 1$ subintervals each of length less than δ. Formally, a partition of an interval $[a, b]$ into $n{+}1$ subintervals is a monotonic function p from $\{0 \dots n{+}1\}$ to $[a, b]$ such that $p[0] = a$ and $p[n + 1] = b$. ($[S \rightarrow T]$ denotes the set of all functions whose domain equals S and whose range is a subset of T.)

module *Exponentials*

import *Integration*

constant

$exp \triangleq [1{+}\int_0 [r, s \in \mathbf{R} \mapsto r]]$

theorems

$$\begin{aligned} ExpFacts \triangleq\ & \land\ exp \in [\mathbf{R} \mapsto \mathbf{R}] \\ & \land\ exp[0] = 1 \\ & \land\ \forall r \in \mathbf{R} : (r > 0) \Rightarrow (1 - r < exp[-r]) \land (exp[-r] < 1) \\ & \land\ \forall r, s \in \mathbf{R} : exp[r] * exp[s] = exp[r + s] \end{aligned}$$

$$\begin{aligned} & DiffusionSolution \triangleq \\ & \quad \forall p, q, a, c \in \mathbf{R} : (p \neq 0) \Rightarrow \\ & \qquad [c{+}\int_a [r, s \in \mathbf{R} \mapsto (p * r) + q]] = \\ & \qquad\quad [t \in \mathbf{R} \mapsto c * exp[p * (t - a)] + (q/p) * (exp[p * (t - a)] - 1)] \end{aligned}$$

Fig. 8. The exponential function.

$\Sigma(f, p)$ This is a function such that, if p is a partition of an interval into $n + 1$ subintervals, then $\Sigma(f, p)[n]$ is the approximation $\sum_{i=0}^{n} f[p[i]](p[i{+}1] - p[i])$ to the area under f on that interval defined by the partition. We have defined $\Sigma(f, p)$ to be a function with domain *Nat* to permit a recursive definition. (In TLA$^+$, only functions, not operators, can be defined recursively. A recursive function definition such as this can be expressed nonrecursively using TLA$^+$'s **choose** operator.)

$\int_a^b f$ The Riemann integral[6] of the function f is defined to be a real number r such that, for every ϵ, there is some δ such that the approximation for every partition with subintervals of length less than δ lies within ϵ of r. (The operator **choose** denotes Hilbert's ε operator [6], so **choose** $r : P(r)$ equals some value r such that $P(r)$ is true, if such an r exists; otherwise, it has an unspecified value.)

$[c{+}\int_a G]$ The function f such that $f[r]$ equals $c{+}\int_a^r G[f[t], t]\, dt$. (The expression $[s \in S \mapsto e(s)]$ denotes the function g with domain S such that $g[s] = e(s)$ for all $s \in S$.)

Finally, the module asserts the result from elementary calculus that if G is a step function on the interval $[a, b]$, then $[c{+}\int_a G][b]$ is equal to its approximation for the appropriate partition. This theorem is needed for the proof of property *Req0*.

Proving *Req0* also requires introducing the exponential function exp, where $\exp[t] = e^t$. It is defined, and some theorems about exp are asserted, in the *Exponentials* module of Figure 8. The theorem named *DiffusionSolution* asserts

[6] Although this integral is commonly written $\int_a^b f(t)\, dt$, rigorous mathematicians usually write $\int_a^b f$. For example, $\int_a^b t^2\, dt$ is just an informal way of denoting $\int_a^b f$ when f is the function defined by $f(t) = t^2$.

module *HybridSystems*

import *Integration*, *RealTime*

temporal

$$[x = c + \textstyle\int G \mid \mathcal{A} : \textbf{action}, v] \;\triangleq\; \begin{array}{l} \wedge\; x = c \\ \wedge\; \Box[x' = \textbf{if } \mathit{now}' = \mathit{now} \\ \qquad\qquad \textbf{then if } \langle \mathcal{A} \rangle_v \textbf{ then } 0 \textbf{ else } x \\ \qquad\qquad \textbf{else } [x + \int_{\mathit{now}} G][\mathit{now}'] \quad]_{\langle \mathit{now},\, x,\, v \rangle} \end{array}$$

state function

$$\chi(P : \textbf{predicate}) \;\triangleq\; \textbf{if } P \textbf{ then } 1 \textbf{ else } 0$$

Fig. 9. The *HybridSystems* module.

that the solution to the equation $f(t) = c + \int_a^t (pf(t) + q)\, dt$ is

$$f(t) = ce^{p(t-a)} + \frac{q}{p}(e^{p(t-a)} - 1)$$

(Such theorems could be proved with the aid of a mathematical package such as *Maple* or *Mathematica*.) The *HybridSystems* module is now straightforward; it appears in Figure 9.

5 The Proof of Property *Req0*

We now sketch the proof of the result mentioned earlier, that *Req1* implies *Req0*. The proof requires only the theorems explicitly asserted in the modules defined above, plus the usual algebraic properties of arithmetic for the real numbers.

The proof is hierarchically structured, using the following notation. The theorem to be proved is statement $\langle 0 \rangle 1$. The proof of statement $\langle i \rangle j$ is either an ordinary paragraph-style proof or the sequence of statements $\langle i+1 \rangle 1$, $\langle i+1 \rangle 2$, ... and their proofs. Within a proof, $\langle k \rangle l$ denotes the most recent statement with that number. A statement has the form

Assume: *Assumption* Prove: *Goal*

which is abbreviated to *Goal* if there is no assumption. The assertion Q.E.D. in statement number $\langle i+1 \rangle k$ of the proof of statement $\langle i \rangle j$ denotes the goal of statement $\langle i \rangle j$. The statement

Case: *Assumption*

is an abbreviation for

Assume: *Assumption* Prove: Q.E.D.

Within the proof of statement $\langle i \rangle j$, assumption $\langle i \rangle$ denotes that statement's assumption.

We begin with the high-level proof, which essentially uses standard predicate logic to reduce the problem to proving a statement with no quantification over flexible variables. The resulting statement, $\langle 1 \rangle 2$, is proved later.

ASSUME: $\mathit{MaxCon} \geq 4 * \rho * (1 + (1/(1 - \exp[(-30) * \delta])))$
PROVE: $RT(v) \land \mathit{Req1} \;\Rightarrow\; \mathit{Req0}$

$\langle 1 \rangle 1.$ $\mathbf{\exists} x : \forall r \in \mathbf{R} : [x[r] = 0 + \int G(r) \mid \mathsf{false}, v]$

PROOF: Follows from a standard theorem about the validity of adding history variables [1], which asserts the validity of $\mathbf{\exists}\, x : (x = f) \land \Box[x' = g]_w$ if x does not occur in f and x' does not occur in g.

$\langle 1 \rangle 2.$ $\land\ \mathit{now} = a$
$\land\ RT(v)$
$\land\ \forall r \in \mathbf{R} : [x[r] = 0 + \int G(r) \mid \mathsf{false}, v]$
$\land\ [g = 0 + \int \mathit{accumRate} \mid \mathsf{false}, v]$
$\land\ \forall r \in \mathbf{R} : \Box(x[r] \leq 4)$
$\Rightarrow\ \Box(g < \mathit{MaxCon})$

PROOF: Sketched below.

$\langle 1 \rangle 3.$ Q.E.D.

$\langle 2 \rangle 1.$ $\land\ \mathit{now} = a$
$\land\ RT(v)$
$\land\ \forall r \in \mathbf{R} : [x[r] = 0 + \int G(r) \mid \mathsf{false}, v]$
$\land\ \forall r \in \mathbf{R} : \Box(x[r] \leq 4)$
$\Rightarrow\ ([g = 0 + \int \mathit{accumRate} \mid \mathsf{false}, v] \Rightarrow \Box(g < \mathit{MaxCon}))$

PROOF: From $\langle 1 \rangle 2$ by propositional logic.

$\langle 2 \rangle 2.$ $\land\ \mathit{now} = a$
$\land\ RT(v)$
$\land\ \forall r \in \mathbf{R} : [x[r] = 0 + \int G(r) \mid \mathsf{false}, v]$
$\land\ \forall r \in \mathbf{R} : \Box(x[r] \leq 4)$
$\Rightarrow\ \mathit{Req0}$

PROOF: $\langle 2 \rangle 1$, the definition of $\mathit{Req0}$, and predicate logic, since g does not occur to the left of the $\Rightarrow$.

$\langle 2 \rangle 3.$ $\land\ \mathit{now} = a$
$\land\ RT(v) \land \mathit{Req1}$
$\Rightarrow\ \mathit{Req0}$

PROOF: By $\langle 1 \rangle 1$, $\langle 2 \rangle 2$, the definition of $\mathit{Req1}$, and simple predicate logic, since x does not occur free in $RT(v)$ or $\mathit{Req0}$.

$\langle 2 \rangle 4.$ Q.E.D.

PROOF: From $\langle 2 \rangle 3$ by simple predicate logic, using the validity of $\exists\, a : \mathit{now} = a$.

Because it involves quantification over flexible variables, this part of the proof cannot be handled by TLP, the mechanical verification system for TLA based on the LP theorem prover [4].[7] The rest of the proof can, in principle, be verified using TLP. However, TLP does not yet contain the full definitional capability of TLA^{+}, so many of the definitions would have to be expanded by hand. In addition to being a possible source of error, this translation might make the

[7] Mechanical checking of this kind of reasoning with quantifiers is not hard. However, such reasoning is always so simple that we have not felt mechanical verification to be worth the effort.

formulas so long that verification would be impractical. We hope that future versions of TLP will permit the necessary definitions.

We now prove $\langle 1\rangle 2$. From the definitions in modules *HybridSystems* and *RealTime*, we can write

$$[x[r]\!=\!0\!+\!\int G(r) \mid \mathsf{false}, v] \;\triangleq\; (x[r] = 0) \land \Box[\mathcal{N}_x(r)]_{\langle now, x[r], v\rangle}$$

$$[g\!=\!0\!+\!\int \mathit{accumRate} \mid \mathsf{false}, v] \;\triangleq\; (g = 0) \land \Box[\mathcal{N}_g]_{\langle now, g, v\rangle}$$

$$RT(v) \;\triangleq\; (now \in \mathbf{R}) \land \Box[\mathcal{N}_{now}]_{\langle now\rangle}$$

for the appropriate actions $\mathcal{N}_x(r)$, $\mathcal{N}_g$, and $\mathcal{N}_{now}$. Let

$$\begin{aligned} \mathit{Init} &\triangleq (now = a) \land (g = 0) \land (now \in \mathbf{R}) \land (\forall\, r \in \mathbf{R} : x[r] = 0) \\ \mathcal{N} &\triangleq [\mathcal{N}_g]_{\langle now, g, v\rangle} \land [\mathcal{N}_{now}]_{\langle now\rangle} \land (\forall\, r \in \mathbf{R} : [\mathcal{N}_x(r)]_{\langle now, x[r], v\rangle}) \\ J &\triangleq \forall\, r \in \mathbf{R} : x[r] \leq 4 \end{aligned}$$

Using the commutativity of conjunction and the fact that conjunction distributes over $\Box$, statement $\langle 1\rangle 2$ can be rewritten as[8]

$$\mathit{Init} \land \Box\mathcal{N} \land \Box J \;\Rightarrow\; \Box(g < \mathit{MaxCon})$$

TLA formulas of this form are proved with the aid of an invariant. First, we define the trivial "type-correctness" part of the invariant:

$$\begin{aligned} T \;\triangleq\; &\land (now \in \mathbf{R}) \land (g \in \mathbf{R}) \\ &\land \forall r \in \mathbf{R} : (x[r] \in \mathbf{R}) \land (0 \leq x[r]) \end{aligned}$$

The nontrivial part I of the invariant is defined as follows, where we first define $\llbracket r \rrbracket$ to be the largest number less than or equal to r that is some multiple of 30 seconds later than a.

$$\begin{aligned} \llbracket r \rrbracket &\triangleq \textbf{choose } s \in \mathbf{R} : \land (s \leq r) \land (r < s + 30) \\ &\phantom{\triangleq \textbf{choose } s \in \mathbf{R} :} \land |s - a|/30 \in \mathit{Nat} \\ k &\triangleq (4 * \rho)/(1 - \mathit{exp}[-30 * \delta]) \\ I &\triangleq g \leq (k * \mathit{exp}[-\delta * (now - \llbracket now \rrbracket)] + \rho * x[\llbracket now \rrbracket]) \end{aligned}$$

The proof of $\langle 1\rangle 2$ is a standard invariance proof, where we prove $\Box I$, using the hypothesis $\Box J$. The high-level structure of the proof is:

$\langle 2\rangle 1$. $\mathit{Init} \land \Box\mathcal{N} \Rightarrow \Box T$
  PROOF: This is a straightforward invariance proof and is omitted.
$\langle 2\rangle 2$. $\mathit{Init} \land \Box\mathcal{N} \land \Box(J \land T) \;\Rightarrow\; \Box(g < \mathit{MaxCon})$
  $\langle 3\rangle 1$. $\mathit{Init} \Rightarrow I$
    PROOF: Trivial, because Init implies $g = 0$.

[8] This notation is rather sloppy, since $\Box\mathcal{N}$ is not a syntactically correct TLA formula. We must write $\Box[\mathcal{N}]_w$ for some state function w. However, it is easy to check that $\mathcal{N}$ equals $[\mathcal{N}]_{\langle now, g, X, v\rangle}$, where $X \;\triangleq\; [r \in \mathbf{R} \mapsto x[r]]$.

$\langle 3\rangle 2.$ $J \land J' \land T \land I \land \mathcal{N} \Rightarrow I'$
PROOF: Sketched below.
$\langle 3\rangle 3.$ $J \land I \Rightarrow (g \leq \mathit{MaxCon})$
PROOF: Follows by simple algebra from the definition of k and assumption $\langle 0\rangle$.
$\langle 3\rangle 4.$ Q.E.D.
$\langle 4\rangle 1.$ $\Box(J \land T) \land I \land \Box\mathcal{N} \Rightarrow \Box I$
PROOF: $\langle 3\rangle 2$ and the standard TLA invariance rule.
$\langle 4\rangle 2.$ $\Box(J \land T) \land \mathit{Init} \land \Box\mathcal{N} \Rightarrow \Box I$
PROOF: $\langle 3\rangle 1$ and $\langle 4\rangle 1$.
$\langle 4\rangle 3.$ $\Box I \Rightarrow \Box(g \leq \mathit{MaxCon})$
PROOF: $\langle 3\rangle 3$ and simple temporal logic reasoning.
$\langle 4\rangle 4.$ Q.E.D.
PROOF: $\langle 4\rangle 2$ and $\langle 4\rangle 3$.
$\langle 2\rangle 3.$ Q.E.D.
PROOF: By $\langle 2\rangle 1$ and $\langle 2\rangle 2$.

We have reduced our goal to proving $\langle 3\rangle 2$, which is an assertion of ordinary mathematics, with no temporal operators. (In the reasoning, primed and unprimed variables are considered to be separate, unrelated values.) The proof of $\langle 3\rangle 2$ is sketched below. The "algebraic calculations", which are omitted, constitute the heart of the proof. They are straightforward, but writing them out at the same level of detail as we have been using would be extremely tedious. Most mechanical verifiers would probably require that they be broken into extremely small steps, though they should not be hard to check with a mathematical package like *Mathematica* or *Maple*.

$\langle 3\rangle 2.$ ASSUME: $J \land J' \land T \land I \land \mathcal{N}$
PROVE: I'
let $\mathit{mid} \triangleq \llbracket \mathit{now} \rrbracket + 30$
$\langle 4\rangle 1.$ CASE: $\mathit{now}' = \mathit{now}$
PROOF: $[\mathcal{N}_g]_{\langle \mathit{now}, g, v\rangle} \land (\mathit{now}' = \mathit{now})$ implies $g' = g$, and $(\mathit{now}' = \mathit{now}) \land \forall r : [\mathcal{N}_x(r)]_{\langle \mathit{now},\, x[r],\, v\rangle}$ implies $x[\llbracket \mathit{now} \rrbracket]' = x[\llbracket \mathit{now} \rrbracket]$, so I' equals I.
$\langle 4\rangle 2.$ CASE: $(\mathit{now}' \neq \mathit{now}) \land (v' = v)$
$\langle 5\rangle 1.$ $\mathit{now}' > \mathit{now}$
PROOF: By $[\mathcal{N}_{\mathit{now}}]_{\langle \mathit{now}\rangle}$ (assumption $\langle 3\rangle$).
$\langle 5\rangle 2.$ CASE: $\mathit{Gas} \land \neg \mathit{Flame}$
let $D(s,t) \triangleq s * \exp[-\delta * t] + (\rho/\delta)(1 - \exp[-\delta * t])$
$\langle 6\rangle 1.$ $g' = D(g, \mathit{now}' - \mathit{now})$
PROOF: By $[\mathcal{N}_g]_{\langle \mathit{now}, g, v\rangle}$ and Theorem *DiffusionSolution* of module *Exponentials*.
$\langle 6\rangle 2.$ CASE: $\llbracket \mathit{now}' \rrbracket = \llbracket \mathit{now} \rrbracket$
$\langle 7\rangle 1.$ $x'[\llbracket \mathit{now}' \rrbracket] = x[\llbracket \mathit{now} \rrbracket] + (\mathit{now}' - \mathit{now})$
PROOF: Case assumptions $\langle 5\rangle$ and $\langle 6\rangle$, $[\mathcal{N}_x(r)]_{\langle \mathit{now}, x[r], v\rangle}$ with $r = \llbracket \mathit{now} \rrbracket$, and Theorem *IntegralOfStepFcn* of module *Integration*, since $\mathit{now} \in [\llbracket \mathit{now} \rrbracket, \llbracket \mathit{now} \rrbracket + 30]$.

$\langle 7\rangle 2$. Q.E.D.
PROOF: Algebraic calculation, using $\langle 6\rangle 1$, $\langle 7\rangle 1$, case assumption $\langle 5\rangle$, and Theorem *ExpFacts* of module *Exponentials*.

$\langle 6\rangle 3$. CASE: $\lfloor\!\lfloor now' \rfloor\!\rfloor \neq \lfloor\!\lfloor now \rfloor\!\rfloor$

$\langle 7\rangle 1$. $g' = D(D(g, mid - now), now' - mid)$
PROOF: By $\langle 6\rangle 1$, since an algebraic calculation using Theorem *ExpFacts* shows that $D(D(s, t_1), t_2) = D(s, t_1 + t_2)$ for any $s, t_1, t_2 \in \mathbf{R}$.

$\langle 7\rangle 2$. $x'[\lfloor\!\lfloor now \rfloor\!\rfloor] = x[\lfloor\!\lfloor now \rfloor\!\rfloor] + mid - now$
PROOF: Case assumption $\langle 4\rangle$, $[\mathcal{N}_x(r)]_{\langle now, x[r], v\rangle}$ with $r = \lfloor\!\lfloor now \rfloor\!\rfloor$, and Theorem *IntegralOfStepFcn*.

$\langle 7\rangle 3$. $D(g, mid - now) \leq k$
PROOF: Algebraic calculation using $\langle 7\rangle 2$, $J' \wedge I$ (Assumption $\langle 3\rangle$), and Theorem *ExpFacts*.

$\langle 7\rangle 4$. $x'[mid] = now' - mid$

$\langle 8\rangle 1$. $x'[mid] = \min(now' - mid, 30)$
PROOF: Case assumption $\langle 6\rangle$, $[\mathcal{N}_x(r)]_{\langle now, x[r], v\rangle}$ with $r = mid$, and Theorem *IntegralOfStepFcn*.

$\langle 8\rangle 2$. Q.E.D.
PROOF: $\langle 8\rangle 1$ and J' (Assumption $\langle 3\rangle$).

$\langle 7\rangle 5$. $\lfloor\!\lfloor now' \rfloor\!\rfloor = mid$

$\langle 8\rangle 1$. $(now' - mid) \leq 4$
PROOF: $\langle 7\rangle 4$ and J' (assumption $\langle 3\rangle$).

$\langle 8\rangle 2$. Q.E.D.
PROOF: $\langle 5\rangle 1$, $\langle 8\rangle 1$, and the definition of *mid*.

$\langle 7\rangle 6$. Q.E.D.
PROOF: Algebraic calculation using $\langle 7\rangle 1$, $\langle 7\rangle 3$, $\langle 7\rangle 4$, $\langle 7\rangle 5$, I (assumption $\langle 3\rangle$), and Theorem *ExpFacts*.

$\langle 6\rangle 4$. Q.E.D.
PROOF: $\langle 6\rangle 2$ and $\langle 6\rangle 3$.

$\langle 5\rangle 3$. CASE: $\neg(Gas \wedge \neg Flame)$

$\langle 6\rangle 1$. $g' = g * \exp[-\delta * (now' - now)]$
PROOF: By $[\mathcal{N}_g]_{\langle now, g, v\rangle}$ and Theorem *DiffusionSolution*.

$\langle 6\rangle 2$. CASE: $\lfloor\!\lfloor now' \rfloor\!\rfloor = \lfloor\!\lfloor now \rfloor\!\rfloor$

$\langle 7\rangle 1$. $x'[\lfloor\!\lfloor now \rfloor\!\rfloor] = x[\lfloor\!\lfloor now \rfloor\!\rfloor]$
PROOF: By $[\mathcal{N}_x(r)]_{\langle now, x[r], v\rangle}$ with $r = \lfloor\!\lfloor now \rfloor\!\rfloor$, case assumption $\langle 6\rangle$, and Theorem *IntegralOfStepFcn*.

$\langle 7\rangle 2$. Q.E.D.
PROOF: Algebraic calculation using $\langle 5\rangle 1$, $\langle 6\rangle 1$, $\langle 7\rangle 1$, I, and Theorem *ExpFacts*.

$\langle 6\rangle 3$. CASE: $\lfloor\!\lfloor now' \rfloor\!\rfloor \neq \lfloor\!\lfloor now \rfloor\!\rfloor$

$\langle 7\rangle 1$. $mid \leq \lfloor\!\lfloor now' \rfloor\!\rfloor$
PROOF: $\langle 5\rangle 1$ and case assumption $\langle 6\rangle$.

$\langle 7\rangle 2$. $g' = g * \exp[-\delta * (mid - now)] * \exp[-\delta * (now' - mid)]$
PROOF: $\langle 6\rangle 1$ and Theorem *ExpFacts*.

⟨7⟩3. $g * \exp[-\delta * (mid - now)] \leq k$

Proof: Algebraic calculation, using I, J (which implies $x[\llbracket now \rrbracket] \leq 4$), and Theorem *ExpFacts* (since $mid > now$).

⟨7⟩4. Q.E.D.

Proof: Algebraic calculation, using ⟨7⟩1, ⟨7⟩2, ⟨7⟩3, T, and Theorem *ExpFacts*.

⟨6⟩4. Q.E.D.

Proof: ⟨6⟩2 and ⟨6⟩3.

⟨5⟩4. Q.E.D.

Proof: ⟨5⟩2 and ⟨5⟩3.

⟨4⟩3. Q.E.D.

Proof: ⟨4⟩1, ⟨4⟩2, and assumption ⟨3⟩, since $[\mathcal{N}_{now}]_{now} \land (now' \neq now)$ implies $v' = v$, by definition of $\mathcal{N}_{now}$.

6 An Implementation

We now specify an implementation of the gas burner inspired by RRH's "control model". RRH also specify an "architecture". However, a comparison of Figures 2 and 3 of [8] reveals that this architecture is just the same implementation expressed in a CSP-like language. A program in a toy language is no closer to a real program than the corresponding TLA$^+$ or Duration Calculus formula is; it just has a syntax that can fool some people into thinking it is closer. So, we see no reason to introduce such a toy programming language.

The control model is described by the state-transition diagram of Figure 10. In the RRH specification, the states of the control model are abstract states that are only loosely coupled with states of the physical variables. For example, the specification of RRH asserts that the ignition and gas are turned on after the Ignite1 state is reached. The easiest way to duplicate the RRH specification would be to translate the state-machine into a simple TLA$^+$ formula, and then translate the "phase requirements" of RRH into TLA$^+$ using the operators

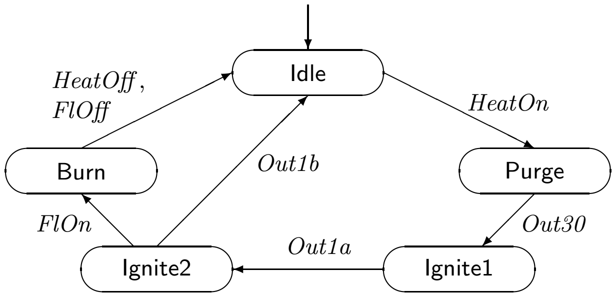

Fig. 10. The control model, adapted from Figures 2 and 3 of [8].

from the *HybridSystems* module. However, we have already presented one level of specification and verification using the *HybridSystems* module's operators, so this would be more of the same. Moreover, this is not the type of specification one would naturally write with TLA. In TLA, it is more natural to write lower level specifications purely in terms of states and transitions. One would draw a state-transition diagram for the *gas* and *ignition* variables—the transitions being enabled and disabled by changes to the control state—and write a TLA$^+$ specification of the resulting system. This representation would be the first step in refining the control model to a realistic representation of an actual implementation. However, developing this specification would take more space than seems appropriate. Instead, we present a simplified version in which the physical variables *gas* and *ignition* that are under the implementer's control change simultaneously with the control state.

The module *BurnerControl* of Figure 11 describes the time-independent behavior of the burner. It is a standard TLA specification; the next-state relation $\mathcal{N}$ is specified as the disjunction of individual actions corresponding to the transitions in Figure 10. Note that this module has a new parameter—the "internal" variable *state*. (Internal variables are ones that are existentially quantified.) The only surprising part of the specification is that the *Out1b* action can occur even if the flame is on. Timing requirements will prevent this action from occurring if the flame ignites quickly enough.

Next, module *BurnerPhaseReqs* of Figure 12 specifies the real-time requirements on the burner. It contains minimum and maximum delays on each of the transitions in Figure 10, which are expressed with the operators from the *RealTime* module. The constraints, and the new parameters ϵ_1 and ϵ_2, are taken directly from the RRH specification. RRH's condition *Ignite2Req* is split into two parts. The first, *Ignite2aReq*, asserts that an *Out1b* action cannot occur until it has been enabled ($state =$ "Ignite2") for at least 1 second, and it must occur if it has been enabled for $1 + \epsilon_1$ seconds. (The formula *Control* asserts that the *Out1b* action remains enabled until an *Out1b* or *FlOn* step occurs.) The second part, *Ignite2bReq*, asserts that a *FlOn* action must be continuously enabled ($state =$ "Ignite2" and flame on) for at least ϵ_2 seconds before it can occur, and it must occur if it has been continuously enabled for ϵ_1 seconds.

Next, module *BurnerEnvironment* in Figure 13 specifies the environment. The burner controls the *gas* and *ignition* variables; the environment controls the *flame* and *heatReq* variables. The specification of the initial condition and next-state relation are standard TLA. Condition ASM_1 of RRH, that "no gas results in no flame within 0.1 seconds," is expressed as usual with a timer. RRH's condition ASM_2, that "gas does not ignite when the ignition transformer is not operating," is a time-independent property that is encoded in the next-state relation $\mathcal{N}$.

Finally, the pieces are put together in the *ControlModel* module of Figure 14. The assumptions about ϵ_1 and ϵ_2 are the same as RRH's, except that our simpler specification is correct with a weaker upper bound on ϵ_1. The statement

include *BurnerEnvironment* **as** *Env*

module *BurnerControl*

import *BurnerParameters*

parameter

$state$: **variable**

predicate

$$w \triangleq \langle gas, ignition, state \rangle$$

$$Init \triangleq w = \langle \text{"off"}, \text{"off"}, \text{"Idle"} \rangle$$

actions

$$\begin{aligned}
HeatOn &\triangleq \land\, HeatReq \land (state = \text{"Idle"}) \\
&\quad \land (state' = \text{"Purge"}) \land \textbf{unchanged } \langle gas, ignition \rangle \\
Out30 &\triangleq \land (state = \text{"Purge"}) \land (state' = \text{"Ignite1"}) \\
&\quad \land \textbf{unchanged } \langle gas, ignition \rangle \\
Out1a &\triangleq \land (state = \text{"Ignite1"}) \land (state' = \text{"Ignite2"}) \\
&\quad \land (ignition' = \text{"on"}) \land (gas' = \text{"on"}) \\
Out1b &\triangleq \land (state = \text{"Ignite2"}) \land (state' = \text{"Idle"}) \\
&\quad \land (ignition' = \text{"off"}) \land (gas' = \text{"off"}) \\
FlOn &\triangleq \land (state = \text{"Ignite2"}) \land Flame \land (state' = \text{"Burn"}) \\
&\quad \land (ignition' = \text{"off"}) \land \textbf{unchanged } gas \\
GoIdle &\triangleq \land (state = \text{"Burn"}) \land (\neg Flame \lor \neg HeatReq) \land (state' = \text{"Idle"}) \\
&\quad \land (gas' = \text{"off"}) \land \textbf{unchanged } ignition
\end{aligned}$$

$$\mathcal{N} \triangleq HeatOn \lor Out30 \lor Out1a \lor Out1b \lor FlOn \lor GoIdle$$

temporal

$$Control \triangleq Init \land \Box[\mathcal{N}]_w$$

Fig. 11. The Control Actions.

is equivalent to inserting all the definitions from the *BurnerEnvironment* module, with the names of the defined symbols preceded by "*Env.*".[9] Similarly, the statement

$$\textbf{include } BurnerPhaseReqs \textbf{ as } Burner(state)$$

is equivalent to including all the definitions of the *BurnerPhaseReqs* environment, with defined symbols renamed by prefacing them with "$Burner(state)$.". This makes *state* a free parameter of all those defined symbols, which must be instantiated when the symbol is used. (Note how it is instantiated with the bound variable s in the definition of *Spec*.) The complete specification is the conjunction of the burner's complete specification and the environment specification,

[9] In contrast to an imported module, whose parameters become parameters of the importing module, parameters of an included module are instantiated. In this case, there are no explicit instantiations, so the included module's parameters are by default instantiated with parameters of the same name.

module *BurnerPhaseReqs*
import *BurnerControl*, *RealTime*

parameters
ϵ_1, ϵ_2 : **R constants**

temporal

$$\begin{aligned}
\mathit{IdleReq} \;&\triangleq\; \exists x, y : \wedge\, \mathit{VTimer}(x, \mathit{HeatOn}, \epsilon_2, w) \;\wedge\; \mathit{MinTimer}(x, \mathit{HeatOn}, w) \\
&\qquad\quad \wedge\, \mathit{VTimer}(y, \mathit{HeatOn}, \epsilon_1, w) \;\wedge\; \mathit{MaxTimer}(y) \\
\mathit{PurgeReq} \;&\triangleq\; \exists x, y : \wedge\, \mathit{VTimer}(x, \mathit{Out30}, 30, w) \;\wedge\; \mathit{MinTimer}(x, \mathit{Out30}, w) \\
&\qquad\quad \wedge\, \mathit{VTimer}(y, \mathit{Out30}, 30 + \epsilon_1, w) \;\wedge\; \mathit{MaxTimer}(y) \\
\mathit{Ignite1Req} \;&\triangleq\; \exists x, y : \wedge\, \mathit{VTimer}(x, \mathit{Out1a}, 1, w) \;\wedge\; \mathit{MinTimer}(x, \mathit{Out1a}, w) \\
&\qquad\quad \wedge\, \mathit{VTimer}(y, \mathit{Out1a}, 1 + \epsilon_1, w) \;\wedge\; \mathit{MaxTimer}(y) \\
\mathit{Ignite2aReq} \;&\triangleq\; \exists x, y : \wedge\, \mathit{VTimer}(x, \mathit{Out1b}, 1, w) \;\wedge\; \mathit{MinTimer}(x, \mathit{Out1b}, w) \\
&\qquad\quad \wedge\, \mathit{VTimer}(y, \mathit{Out1b}, 1 + \epsilon_1, w) \;\wedge\; \mathit{MaxTimer}(y) \\
\mathit{Ignite2bReq} \;&\triangleq\; \exists x, y : \wedge\, \mathit{VTimer}(x, \mathit{FlOn}, \epsilon_2, w) \;\wedge\; \mathit{MinTimer}(x, \mathit{FlOn}, w) \\
&\qquad\quad \wedge\, \mathit{VTimer}(y, \mathit{FlOn}, \epsilon_1, w) \;\wedge\; \mathit{MaxTimer}(y) \\
\mathit{BurnReq} \;&\triangleq\; \exists x, y : \wedge\, \mathit{VTimer}(x, \mathit{GoIdle}, \epsilon_2, w) \;\wedge\; \mathit{MinTimer}(x, \mathit{GoIdle}, w) \\
&\qquad\quad \wedge\, \mathit{VTimer}(y, \mathit{GoIdle}, \epsilon_1, w) \;\wedge\; \mathit{MaxTimer}(y) \\
\mathit{PhaseReqs} \;&\triangleq\; \mathit{IdleReq} \;\wedge\; \mathit{PurgeReq} \;\wedge\; \mathit{Ignite1Req} \;\wedge \\
&\qquad\quad \mathit{Ignite2aReq} \;\wedge\; \mathit{Ignite2bReq} \;\wedge\; \mathit{BurnReq}
\end{aligned}$$

Fig. 12. The control model's timing requirements.

together with the formula $RT(v)$ that describes how *now* changes and asserts that v does not change when *now* does. The burner's specification is obtained by hiding the internal state variable in the conjunction of its time-independent specification, its timing requirements, and the RT-formula asserting that the internal state does not change when *now* does.[10]

The next step is to prove that the Control Model satisfies requirements *Req1*, *Req2*, and *Req3*. This means proving that formula *Spec* of module *ControlModel* implies $\mathit{Req1} \wedge \mathit{Req2} \wedge \mathit{Req3}$. We very briefly sketch the proof that *Spec* implies *Req1*; the other proofs are similar.

We first reduce the problem to proving an assertion without quantification over flexible variables. Let $\mathit{Burner}(s).\mathit{VControl}$ be the formula that is the same as $\mathit{Burner}(s).\mathit{Control}$ except with the quantifiers $\exists\, x, y$ removed and each x and y replaced by a unique variable. For example, let *xIgnite1* and *yIgnite1* be substituted for x and y in *Ignite1Req*. Similarly, let *Env.VSpec* be the formula obtained by removing the quantifier from *Env.Spec* and substituting *xEnv* for x. Simple predicate logic shows that, to prove $\mathit{Spec} \Rightarrow \mathit{Req1}$, it suffices to assume

[10] Note that, since s does not appear in $RT(v)$, the latter formula can be moved inside the quantifier; and $RT(v) \wedge RT(s)$ is equivalent to $RT(\langle v, s\rangle)$.

module *BurnerEnvironment*

import *BurnerParameters*

predicate

$Init \triangleq \langle flame, heatReq \rangle = \langle \text{"off"}, \text{"off"} \rangle$

actions

$ChangeHeatReq \triangleq \land\ heatReq' = \textbf{if}\ HeatReq\ \textbf{then}\ \text{"off"}\ \textbf{else}\ \text{"on"}$
$\qquad\qquad\qquad\quad \land\ \textbf{unchanged}\ flame$

$FlameOn \triangleq \land \neg Flame \land Gas \land Ignition \land (flame' = \text{"on"})$
$\qquad\qquad\quad \land\ \textbf{unchanged}\ heatReq$

$FlameOff \triangleq Flame \land (flame' = \text{"off"}) \land \textbf{unchanged}\ heatReq$

$\mathcal{N} \triangleq ChangeHeatReq \lor FlameOn \lor FlameOff$

temporal

$Spec \triangleq \land\ Init \land \Box[\mathcal{N}]_{\langle flame,\, heatReq \rangle}$
$\qquad\quad \land\ \exists x : \land\ VTimer(x,\ FlameOff \land \neg Gas,\ 1/10,\ \langle flame,\ heatReq \rangle)$
$\qquad\qquad\qquad \land\ MaxTimer(x)$

Fig. 13. Timing assumptions on the environment.

module *ControlModel*

import *BurnerParameters*, *RealTime*

parameters

ϵ_1, ϵ_2 : **R constants**

assumption

$EpsilonAssumption \triangleq (0 < \epsilon_1) \land (\epsilon_2 < \epsilon_1) \land (\epsilon_1 \leq 2/3)$

include *BurnerPhaseReqs* **as** *Burner*(*state*)

include *BurnerEnvironment* **as** *Env*

temporal

$Spec \triangleq \land\ \exists s : Burner(s).Control \land Burner(s).PhaseReqs \land RT(s)$
$\qquad\quad \land\ Env.Spec$
$\qquad\quad \land\ RT(v)$

Fig. 14. The complete control model specification.

$r \in \mathbf{R}$ and prove $\Pi \Rightarrow \Box(z \leq 4)$, where

$$\Pi \triangleq \begin{array}{l} \wedge\ [z = 0 + \int G(r) \mid \mathsf{false}, v] \\ \wedge\ \mathit{Burner}(s).\mathit{Control}\ \wedge\ \mathit{Burner}(s).\mathit{VPhaseReqs} \\ \wedge\ \mathit{Env}.\mathit{VSpec} \\ \wedge\ \mathit{RT}(s)\ \wedge\ \mathit{RT}(v) \end{array}$$

and G is defined by the **let** in the definition of *Req1*.

The proof of $\Pi \Rightarrow \Box(z \leq 4)$ has the same form as the proof of step $\langle 1 \rangle 2$ in the proof that $\mathit{RT}(v) \wedge \mathit{Req1}$ implies $\mathit{Req0}$. We first prove $\Pi \Rightarrow \Box T$ for a simple invariant T, which expresses the expected relations among the values of the variables. For example, one conjunct of T is

$$(s = \text{“}\mathsf{Ignite1}\text{”}) \Rightarrow \begin{array}{l} \wedge\ \mathit{Gas} \wedge \mathit{Ignition} \\ \wedge\ (\mathit{yIgnite1} \in \mathbf{R}) \wedge (\mathit{now} \leq \mathit{yIgnite1}) \end{array}$$

We then prove $\Pi \wedge \Box T \Rightarrow \Box I$ for the following invariant I.

$$\begin{array}{l} \wedge\ z \leq 4 \\ \wedge\ (z > 0) \Rightarrow (z \leq \mathit{now} - r) \\ \wedge\ (\mathit{now} \in [r, r+30]) \Rightarrow \begin{array}[t]{l} \vee\ (s = \text{“}\mathsf{Idle}\text{”}) \\ \vee\ (s = \text{“}\mathsf{Purge}\text{”})\ \wedge\ ((z > 0) \Rightarrow (\mathit{xPurge} \geq r + 30)) \\ \vee\ (s = \text{“}\mathsf{Ignite1}\text{”})\ \wedge\ (z + (\mathit{yIgnite1} - \mathit{now}) \leq 1 + \epsilon_1) \\ \vee\ (s = \text{“}\mathsf{Ignite2}\text{”})\ \wedge\ (z + \mathit{yIgnite2a} - \mathit{now} \leq 2 + 2\epsilon_1) \\ \vee\ \wedge\ (s = \text{“}\mathsf{Burn}\text{”})\ \wedge\ \mathit{Gas} \\ \quad \wedge\ \vee\ \mathit{Flame}\ \wedge\ (z \leq 2 + 2\epsilon_1) \\ \quad\quad \vee\ \neg \mathit{Flame}\ \wedge\ (z + \mathit{yBurn} - \mathit{now} \leq 2 + 3\epsilon_1) \end{array} \end{array}$$

The hard part of proving $\Pi \wedge \Box T \Rightarrow \Box I$ is the analog of step $\langle 3 \rangle 2$ in the proof that $\mathit{RT}(v) \wedge \mathit{Req1}$ implies $\mathit{Req0}$. As usual, this step involves ordinary mathematics, with no temporal operators.

References

1. Martín Abadi and Leslie Lamport. An old-fashioned recipe for real time. Research Report 91, Digital Equipment Corporation Systems Research Center, 1992. An earlier version, without proofs, appeared in [3, pages 1–27].
2. K. Mani Chandy and Jayadev Misra. *Parallel Program Design.* Addison-Wesley, Reading, Massachusetts, 1988.
3. J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors. *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin, 1992. Proceedings of a REX Real-Time Workshop, held in The Netherlands in June, 1991.
4. Urban Engberg, Peter Grønning, and Leslie Lamport. Mechanical verification of concurrent systems with tla. In *Logics of Programs*, Lecture Notes in Computer Science, Berlin, Heidelberg, New York, June 1992. Springer-Verlag.
5. Leslie Lamport. The temporal logic of actions. Research Report 79, Digital Equipment Corporation, Systems Research Center, December 1991.

6. A. C. Leisenring. *Mathematical Logic and Hilbert's ε-Symbol*. Gordon and Breach, New York, 1969.
7. Zohar Manna and Amir Pnueli. *The Temporal Logic of Concurrent Systems*. Springer-Verlag, New York, 1991.
8. Anders P. Ravn, Hans Rischel, and Kirsten M. Hansen. Specifying and verifying requirements of real-time systems. *IEEE Transactions on Software Engineering*, January 1993. to appear.