©Springer-Verlag http://www.springer.de/comp/lncs/index.html

Proceedings of: 6th International Workshop *Hybrid Systems: Computation and Control* (HSCC)

April 2003, Prague, Czech Republic, LNCS 2623, Springer-Verlag

Automatic Verification of a Turbogas Control System with the Murφ Verifier *

Giuseppe Della Penna[1], Benedetto Intrigila[1], Igor Melatti[1], Michele Minichino[3], Ester Ciancamerla[3], Andrea Parisse[1], Enrico Tronci[2,**], and Marisa Venturini Zilli[2]

[1] Dip. di Informatica, Università di L'Aquila, Coppito 67100, L'Aquila, Italy
{dellapenna,intrigila,melatti,parisse}@di.univaq.it

[2] Dip. di Informatica Università di Roma "La Sapienza",
Via Salaria 113, 00198 Roma, Italy
{tronci,zilli}@dsi.uniroma1.it

[3] ENEA , CR Casaccia, Via Anguillarese, 00060 Italy
{minichino,ciancamerlae}@casaccia.enea.it

Abstract. Automatic analysis of *Hybrid Systems* poses formidable challenges both from a modeling as well as from a verification point of view. We present a case study on automatic verification of a *Turbogas Control System* (TCS) using an extended version of the Murφ verifier. TCS is the heart of *ICARO*, a 2MW *Co-generative Electric Power Plant*.

For large hybrid systems, as TCS is, the modeling effort accounts for a significant part of the whole verification activity. In order to ease our modeling effort we extended the Murφ verifier by importing the C language `long double` type (*finite precision real numbers*) into it.

We give experimental results on running our extended Murφ on our TCS model. For example using Murφ we were able to compute an admissible range of values for the variation speed of the *user demand* of electric power to the turbogas.

1 Introduction

Automatic analysis of *Hybrid Systems* poses formidable challenges both from a modeling as well as from a verification point of view. In fact the simultaneous presence of continuous and discrete variables may lead very quickly to *state explosion*, thus preventing completion of the verification process.

Many verification tools (*model checkers*) are available for automatic verification of hybrid systems. Examples are: HyTech [14, 2, 1] and UPPAAL [17, 26]. Also tools originally designed for hardware verification have been used for hybrid systems verification. E.g. in [25] SMV [18, 21] has been used for verification of chemical processing systems.

* This research has been partially supported by MURST projects MEFISTO and SAHARA

** Corresponding Author: Enrico Tronci. Tel: +39 06 4991 8361 Fax: +39 06 8541 842

In this paper we present a case study on automatic verification of a *Turbogas Control System* (TCS) using an extended version of the Murφ verifier [9, 19]. TCS is the main subsystem of *ICARO*, a 2MW *Co-generative Electric Power Plant* in operation at the ENEA Research Center of Casaccia (Italy). TCS is a quite complex control system whose main goal is to regulate the opening of the fuel gas valve of ICARO turbine so that the generated electric power, turbine rotation speed and exhaust smokes temperature are all within given limits notwithstanding changes in the *user demand* for electric power to the turbogas.

We resorted to the Murφ verifier after failing to complete verification using HyTech and SMV. We think Murφ success on this case study can be explained as follows. The transition relations of the many subsystems forming TCS are quite big. This fact is an obstruction for symbolic model checkers as HyTech and SMV. However, when we put such transition relations together the set of reachable states becomes of moderate size. In fact, the controller goal is exactly that of keeping the reachable states within a small neighborhood of the system *setpoint*. An explicit model checker, like Murφ, can easily exploit this fact thus avoiding state explosion.

To ease the hybrid systems modeling activity we decided to *import* within the Murφ verifier the C `long double` type (*finite precision real numbers*). The resulting Murφ verifier still uses a discrete time model. However the C `long double` type is now available for real valued state variables. This turned out to be very useful during the modeling activity, which, in our experience, accounts for a significant part of the whole hybrid system verification activity. Of course our extension to Murφ applies to *Cached* Murφ [23, 7], *Disk* Murφ [20] and *Random* Murφ [24] as well.

Our results show that Murφ, enhanced with finite precision real numbers, is definitively to be considered among the candidate tools available for automatic analysis of hybrid control systems. The main contributions of this paper can be summarized as follows.

- We sketch (Section 3) syntax and semantics of the Murφ verifier enhanced with *finite precision real numbers*.
- We present (Section 4) our case study on verification of a *Turbogas Control System* (TCS).
- We show (Section 5) some of our experimental results on using Murφ enhanced with finite precision real numbers on the Murφ model for the system described in Section 4. Using Murφ we were able to compute an admissible range of values for the variation speed of the *user demand* for electric power to the turbogas.

2 The Murφ verifier

The goal of this section is to give a short overview of the Murφ verifier. For further details we refer the reader to [9, 19].

From a conceptual point of view, Murφ takes as input a *Finite State System* $\mathcal{S}$ and checks that a given invariant property φ for $\mathcal{S}$ is satisfied.

Definition 1. *1. A* Finite State System *(FSS) $\mathcal{S}$ is a 4-tuple $(S,\ I,\ A,\ R)$ where: S is a finite set (of states), $I \subseteq S$ is the set of initial states, A is a finite set (of transition labels) and R is a relation on $S \times A \times S$. R is usually called the* transition relation *of $\mathcal{S}$.*

2. Given states $s, s' \in S$ and $a \in A$ we say that there is a transition from s to s' labeled with a iff $R(s, a, s')$ holds. We say that there is a transition from s to s' (notation $R(s, s')$) iff there exists $a \in A$ s.t. $R(s, a, s')$ holds. The set of successors of state s (notation `next`(s)*) is the set of states s' s.t. $R(s, s')$.*

3. The set of reachable states *of $\mathcal{S}$ (notation* $\mathrm{Reach}(\mathcal{S})$*) is the set of states of $\mathcal{S}$ reachable in zero or more steps from I.*
Formally, $\mathrm{Reach}(\mathcal{S})$ *is the smallest set s.t.*
1. $I \subseteq \mathrm{Reach}(\mathcal{S})$,
2. for all $s \in \mathrm{Reach}(\mathcal{S})$, `next`$(s) \subseteq \mathrm{Reach}(\mathcal{S})$.

In the following we will always refer to a given (once and for all) system $\mathcal{S} = (S,\ I,\ A,\ R)$. Thus, e.g., we will write Reach for $\mathrm{Reach}(\mathcal{S})$. Also we may speak about the set of initial states I as well as about the transition relation R without explicitly mentioning $\mathcal{S}$.

The core of all automatic verification tools is the *reachability analysis*, i.e. the computation of Reach given a definition of $\mathcal{S}$ in some language. In fact checking that *all* reachable states of $\mathcal{S}$ satisfy a given (invariant) property φ entails computing Reach. For this reason we focus on the computation of the set of reachable states of $\mathcal{S}$.

Since the transition relation R of a system defines a graph (*transition graph*) computing Reach means visiting (exploring) the transition graph starting from the initial states in I. This can be done, e.g., using a *Depth First* (DF) visit or a *Breadth First* (BF) visit.

For example the automatic verifier SPIN [22] uses a DF visit. Murφ [19] may use a DF as well as a BF visit. However certain compression options can only be used with a BF visit, for this reason here we focus only on BF visit.

Fig. 1 shows the standard BF state space exploration algorithm. Essentially this is the algorithm used by Murφ to visit the state space of a given system $\mathcal{S}$.

Note that since $\mathcal{S}$ is a finite state system, the algorithm in Fig. 1 always terminates since we never visit the same state more than once.

```
Queue Q;   Hash Table T;
bfs(init_states, next) {
for s in init_states enqueue(Q, s); /* load Q with initial states */
for s in init_states insert(T, s);  /* mark initial states as visited */
while (Q is not empty) { s = dequeue(Q);                /* visit */
 for all s' in next(s)
     if (s' is not in T) {enqueue(Q, s'); insert(T, s');}}}
```

Fig. 1. Explicit Breadth First Visit

The algorithm in Fig. 1 makes use of two main memory data structures: a *Queue*, where states are stored and retrieved (in FIFO order) during the search, and a *Hash Table* used to store all visited states. In Fig. 1 invariants for state `s` may be checked whenever function `enqueue(Q, s)` is called.

Note that Murφ (like SPIN) represents states *explicitly*, i.e. each visited state is stored in RAM (namely in the hash table). There are model checkers (e.g. UPPAAL, SMV, HyTech) that represent states *symbolically*. In symbolic model checking the set V of visited states is represented with its characteristic function f (i.e. $f(s) = \textbf{if } s \in V \textbf{ then } 1 \textbf{ else } 0$) and suitable data structures (e.g. OBDDs, *Ordered Binary Decision Diagrams* [6]) are used to represent f. Examples of symbolic model checkers are SMV [21], UPPAAL [17, 26] (both based on OBDDs) and HyTech [14, 2, 1] which is based on polyedra in multidimensional real space [13, 8, 11, 12].

Explicit model checkers (e.g. as Murφ and SPIN) typically perform better on *software-like* (i.e. asynchronous) systems [15], whereas *symbolic* model checkers (e.g. as SMV) typically perform better on *hardware-like* (i.e. synchronous) systems.

Murφ input consists of a definition of the system $\mathcal{S}$ to be verified and a definition of the property φ to be checked. Both definitions are stored in a file that we call here *Murφ description*.

The Murφ description language for system $\mathcal{S}$ is a high-level programming language for finite-state asynchronous concurrent systems (i.e. software like systems). Murφ description language is high-level in the sense that many features found in common high-level programming languages such as Pascal or C are part of Murφ. For example, Murφ has user-defined data types, procedures, and parameterization of descriptions.

A Murφ description consists of: declarations of constants, types, global variables and procedures; a collection of transition rules; a description of the initial states; and a set of invariants.

The behavioral part of Murφ is a collection of transition rules. Each transition rule is a guarded command which consists of a condition (a Boolean expression on global variables) and an action (a statement that can modify the values of the variables).

The condition and the action are both written in a Pascal-like language. The action can be an arbitrarily complex statement containing loops and conditionals. No matter how complex it is, the action is executed *atomically*, i.e. no other rule can change the variables or otherwise interfere with it while it is being executed.

A *Murφ state* is an assignment of values to all of the global variables of the description.

An execution of the description is generated by executing the endless loop in Fig. 2.

Note that Murφ descriptions are nondeterministic, because of the arbitrary choice in step 2 in Fig. 2. The user has no control over how this choice is made, so a *correct* Murφ program must do the right thing no matter which rules are

```
loop forever {
1. Find all enabled rules, i.e. all rules whose conditions are true
   in the current state;
2. Choose arbitrarily an enabled rule and execute its action,
   thus yielding a new state;  }
```

Fig. 2. Murφ execution loop

chosen. However, once a rule has been chosen, the action is deterministic (there is a unique next state). Note that when using Murφ as a verifier *all* reachable states of a Murφ program are visited.

A small toy example should help to clarify the matter. Let us consider the *Discrete Time System* (DTS) defined by equation 1, where $\mathbf{x}(t)$ is the state value at time t and $\mathbf{d}(t)$ is the disturbance value at time t.

$$\mathbf{x}(t+1) = \begin{cases} \mathbf{x}(t) + \mathbf{d}(t) \text{ if } \mathbf{x}(t) \leq 3 \\ \mathbf{x}(t) - \mathbf{d}(t) \text{ otherwise} \end{cases} \qquad \forall t[\mathbf{d}(t) \in \{0, 1, 2\}], \qquad \mathbf{x}(0) = 0. \quad (1)$$

Fig. 3 shows the FSS corresponding to the DTS defined by Equation 1. The initial state $\mathbf{x}(0) = 0$ is shown with an arrow in Fig. 3, where nodes are labeled with state values and edges are labeled with action (disturbance, in our case) values.

Murφ code for the DTS in Equation 1 is given in Fig. 4 where we have examples of (declarations of) constants, types, global variables, functions, initial states, transition rules and invariants.

Fig. 5 summarizes the output of the Murφ verifier when given the input in Fig. 4. Namely, the Murφ verifier returns an *error trace*, i.e. a (loopless) path in the graph in Fig. 3 from an initial state to a state violating the invariant property. If we replace the $<$ sign in the invariant in Fig. 4 with $\leq$ then the invariant property is always satisfied since all reachable states of the DTS defined by Equation 1 have a value less than or equal to 5 (see Fig. 3).

Remark 1. In the BF algorithm in Fig. 1 only reachable states are visited and thus stored in the hash table `T`. That is the set of reachable states essentially depends only on the *system dynamics*. For example, the set of reachable states for the system defined in Fig. 4 is the integer interval [0, 5]. This set does not depend on `state_type` (the type of variable `x` in Fig. 4) as long as `state_type` contains the integer interval [0, 5]. For example, if in Fig. 4 we change the `state_type` declaration to `state_type : 0..100`, the set of reachable states is still the integer interval [0, 5].

3 Extending Murφ input language with Real Numbers

Murφ built-in types are ranges of integers and enumerative types. To ease the hybrid systems modeling activity we also want to be able to handle *finite preci-*

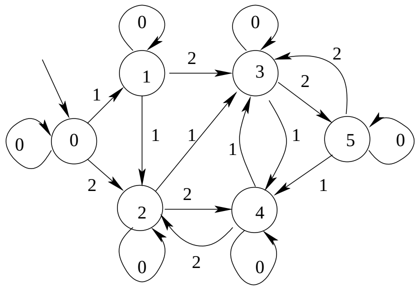

Fig. 3. FSS for the discrete time system in Equation 1.

```
CONST -- constant declarations
MAX_STATE_VALUE : 5;
TYPE    -- type declarations
state_type : 0 .. 10; -- integers from 0 to 10
disturbance_type : 0 .. 2;  -- integers from 0 to 2
VAR    -- (global) variable declarations
x : state_type;  -- x is a variable of type state_type

-- define next state function
function next(x: state_type; d : disturbance_type): state_type;
begin if (x <= 3) then return (x + d); else return (x - d); endif end;

startstate "init" x := 0; end;  -- define initial state

ruleset d : disturbance_type  do  -- define transition rule
   rule "time step" true ==> begin x := next(x, d); end;
end; -- ruleset d

invariant "x is not too big" -- define property to be verified
(x < MAX_STATE_VALUE);
```

Fig. 4. Murφ code for the DTS in Equation 1.

```
Startstate init fired.
x:0
----------
Rule time step, d:1 fired.
x:1
----------
Rule time step, d:2 fired.
x:3
----------
Rule time step, d:2 fired.
The last state of the trace (in full) is:
x:5
----------
```

Fig. 5. Murφ error trace for Murφ model in Fig. 4.

```
const  -- constant declarations
SAMPLING_FREQ : 100.0;  -- Hz. Inverse of the sampling time.
MAX_U: 200.0;     -- Max user demand value (kW)
MAX_D_U: 10.0;    -- Max of time derivative of user demand
MAX_D_P: 0.1;   -- Max of time derivative of compressor pression
MAX_PRES_COMPR: 13.0; -- Max compressor pression (bar)
MIN_PRES_COMPR : 11.0;  -- Min compressor pression (bar)
Power_setpnt : 2000.0;   -- Setpoint of Electric Power (kW)
Texh_setpnt : 552;  -- Setpoint of exhaust smokes temperature(C)
Vrot_setpnt: 75;  -- Setpoint of rotation speed (RPM)
Pow_v_coef : Power_setpnt;      -- valve coefficient in turbogas model
Texh_v_coef : 0.1*Texh_setpnt; -- valve coefficient in turbogas model
Vrot_v_coef : 2*Vrot_setpnt;    -- valve coefficient in turbogas model
FREQ_1 : 100; -- frequency injection disturbances

type  -- type declarations
Disturbance_type : -1..1;
real_type : real(4, 2);  -- used for all real variables
longint_type : -50000 .. +50000;  -- used for counters

var  -- variable declarations
step_counter : longint_type; -- initialized to 0
  -- we do: step_counter := (step_counter + 1)%FREQ_1 at each time step
Power : real_type;   -- Generated Electric Power
```

Fig. 6. A glimpse of the Murφ declarations used in our model.

sion real numbers within the Murφ verifier. Namely, we want to be able to handle within Murφ numbers of the form $s_M\, d_0.d_1 \cdots d_{m-1} \times 10^{s_E\, e_{n-1} \cdots e_0}$ where: d_i and e_i are decimal digits, $d_0 \neq 0$, s_M, $s_E \in \{'+', '-'\}$. As usual we call $s_M\, d_0.d_1 \cdots d_{m-1}$ the *mantissa* and $s_E\, e_{n-1} \cdots e_0$ the *exponent* of the number $s_M\, d_0.d_1 \cdots d_{m-1} \times 10^{s_E\, e_{n-1} \cdots e_0}$.

To this end we add to the Murφ verifier the type `real(m, n)` of real numbers with m digits for the mantissa and n digits for the exponent. Type `real(m, n)` is finite, its cardinality is $2 \times 9 \times 10^{m-1} \times 2 \times 10^n = 36 \times 10^{m+n-1}$. Thus our extension has no impact on Murφ verification algorithms (e.g. as that in Fig. 1) , however makes it easier to model hybrid systems within Murφ.

Note that, as from Remark 1, the huge cardinality of the type `real(m, n)` does not imply, *a priori*, a huge size of the set of reachable states.

The type `real(m, n)` is built on `long double` C type. For this reason the *mantissa* size m and the *exponent* size n in `real(m, n)` must satisfy the following constraints: $1 \leq m \leq$ LDBL_DIG, $2 \leq n \leq \lfloor \log_{10}$LDBL_MAX_10_EXP$\rfloor + 1$, where LDBL_DIG is the maximum number of digits for the mantissa of the `long double` C type and LDBL_MAX_10_EXP is the maximum value of the exponent of the `long double` C type. These constants are defined in the C header `float.h`.

Fig. 6 gives an example of variables and constant declarations within our extended Murφ.

We also extended Murφ by importing all functions available in the C `math` library (header `math.h`). Such functions can now be freely used within the Murφ input language.

Our extension to Murφ is implemented by suitably extending Murφ parser and by adding (low level) functions to store and retrieve finite precision real values into the Murφ internal state representation (namely, state byte vector `bits`, see Murφ documentation [19]).

4 ICARO Turbogas Control System

In this section we will shortly describe the system to be verified and the requirements that we are supposed to verify.

ICARO is a 2MW Electric Co-generative Power Plant, in operation at the ENEA Research Center of Casaccia (Italy), used to provide electricity and heating to the above ENEA Center.

Depending on the operating conditions (e.g. *startup*, *normal*, *shutdown*) ICARO models are widely different. Here we only focus on *normal* operating conditions, i.e. the situation in which ICARO is running at its *nominal* (setpoint) power. In particular our model cannot be used to study system behaviour during transient operating modes (e.g. at startup or shutdown).

ICARO plant consists of many subsystems. Here we only focus on one of the many subsystems of ICARO (e.g. see [5, 3, 4]). Namely we focus on the *Gas Turbine* ICARO subsystem that we call in the following ICARO *Turbogas Control System* (TCS). TCS is the heart of ICARO and is indeed ICARO most critical subsystem. Unfortunately, TCS is also the largest ICARO subsystem, thus making the use of model checking for such hybrid system a challenge.

Unless otherwise stated, all our data (e.g. block diagrams, parameter values, etc) are taken from the ICARO documentation [10].

TCS is a *control system*, that is a (hybrid) system in which we can distinguish two subsystems: the *plant* (i.e. the controlled system) and the *controller* (which sends commands to the plant in order to meet given requirements on the whole system behaviour). Note that many (but not all) hybrid systems are indeed control systems.

Fig. 7 shows the high level block diagram for TCS. The *Turbogas* block in Fig. 7 is the *the plant*. The *Controller* block in Fig. 7 is the *the controller*. In the following we describe the elements of Fig. 7.

4.1 Turbogas Continuous Time Model

The block named *Turbogas* in Fig. 7 models the *Gas Turbine* module. As a matter of fact this module consists of many subsystems (e.g. the compressor, the combustion chamber, the turbine itself and the generator). For our purposes here we can simply look at its input-output model. The turbogas system has the following input variables.

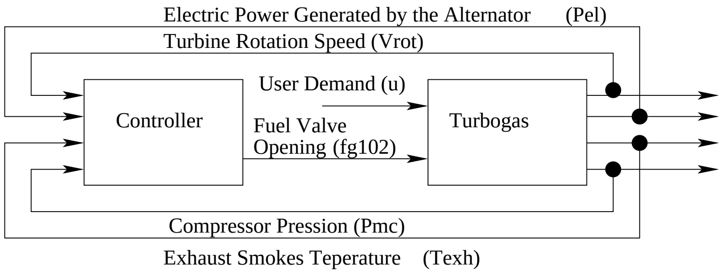

Fig. 7. High level block diagram of ICARO Turbogas Control System

$fg102$ Variable $fg102$ takes value in the real interval [0,1]. This variable gives the opening fraction of the turbogas fuel gas valve (namely valve FG102). For example variable $fg102$ takes value 0 when the valve FG102 is fully closed (no fuel can flow trough the valve) and value 1 when the valve FG102 is fully opened. This is a *control variable*, i.e. a variable whose value can be chosen by the controller so as to achieve predefined goals.

u Variable u models the *User Demand* of electric power. This variable has to be considered as a *disturbance*, i.e. a variable whose value we (i.e. the controller) cannot choose.

The output variables of the turbogas system are the following ones.

P_{el} *Electric power* generated by the alternator.
V_{rot} *Rotation speed* of the gas turbine.
T_{exh} *Temperature* of the exhaust smokes.
P_{mc} *Pression* of the compressor.

The controller goal is to keep the turbogas output variables as close as possible to their *setpoint* notwithstanding variations in the user demand u. The values of the output variables at the setpoint are given in Fig. 8.

- Electric Power setpoint value: $P_{el}^0 = 2000$ (KW).
- Exhaust Smokes Temperature setpoint value: $T_{exh}^0 = 552$ (C).
- Turbine Rotation Speed setpoint value: $V_{rot}^0 = 75$ (RPM)
- Compressor Pression setpoint value: $P_{mc}^0 = 12$ (Bar)

Fig. 8. Turbogas setpoint values.

For the purposes of our analysis we used the ODE (*Ordinary Differential Equation*) model, shown in Fig. 9, to link turbogas input variables with output variables. Of course such a model is only valid in a neighborhood of the setpoint.

Note that, according to the model in Fig. 9, the compressor pression P_{mc} can change value *nondeterministically* as long as it satisfies the constraints given in Fig. 9. We do not need a more detailed model here since the compressor pression is only used as input to the fuel gas valve controller whose requirements do not involve the compressor pression.

We do not know *in advance* the user demand u. However, by making some hypothesis on the user demand u dynamics we can follow for the user demand model in Fig. 9 the same approach we used for the compressor pression. Namely, we simply ask that the user demand $u(t)$ be in the interval $[0, MAX\_U]$ (the user demand is always non-negative since users cannot produce electric power) and the *variation speed* of the user demand $\dot{u}(t)$ be at most MAX_D_U. Note that for the model in Fig. 9 the only *input* variable is $fg102$, all other variables (i.e. P_{el}, V_{rot}, T_{exh}, P_{mc}, u) are *state* as well as *output* variables.

$$\dot{P}_{el}(t) = \alpha_{1,1}P_{el}(t) + \alpha_{1,2}fg102(t) + \alpha_{1,3}u(t)$$

$$\dot{T}_{exh}(t) = \alpha_{2,1}T_{exh}(t) + \alpha_{2,2}fg102(t) + \alpha_{2,3}(P_{el}(t) - P^0_{el}) + \alpha_{2,4}(P_{mc}(t) - P^0_{mc})$$

$$\dot{V}_{rot}(t) = \alpha_{3,1}V_{rot}(t) + \alpha_{3,2}fg102(t) + \alpha_{3,3}(P_{el}(t) - P^0_{el})$$

$$P_{mc}(t) \in [MIN\_P_{mc}, MAX\_P_{mc}] \qquad | \dot{P}_{mc}(t) | \leq MAX\_D\_P_{mc}$$

$$u(t) \in [0, MAX\_U] \qquad | \dot{u}(t) | \leq MAX\_D\_U$$

Fig. 9. Turbogas ODE model used for our analysis.

4.2 Requirements

The goal of the turbogas controller is to set the turbogas control variable $fg102$ so as to keep the value of turbogas output variables P_{el}, V_{rot}, T_{exh} within given limits notwithstanding variations in the user demand u. Such limits are shown in Fig. 10 and are our requirements, i.e. the properties that we will have to check during verification.

$$1300 \leq P_{el}(t) \leq 2500$$
$$200 \leq T_{exh}(t) \leq 580$$
$$40 \leq V_{rot}(t) \leq 120$$

Fig. 10. Turbogas Control System requirements.

4.3 Turbogas Controller

Fig. 11 shows the block diagram for the fuel gas valve controller (i.e. the controller of Fig. 7).

In the following we describe the controller subsystems.

All the controller subsystems (namely: N1 Governor, Power Limiter, Exhaust Temperature Limiter) are built from the *elementary cell* shown in Fig. 12.

Note that in the *elementary cell* in Fig. 12 we have the simultaneous presence of linear blocks (e.g. the integrator block labeled $1/s$ in Fig. 12), *saturation* blocks, *test for* > 0 and logical (AND) blocks. This makes the elementary cell a hybrid system. Since all subsystems in TCS are based on the cell in Fig. 12, it turns out that TCS itself is a (quite big) hybrid system.

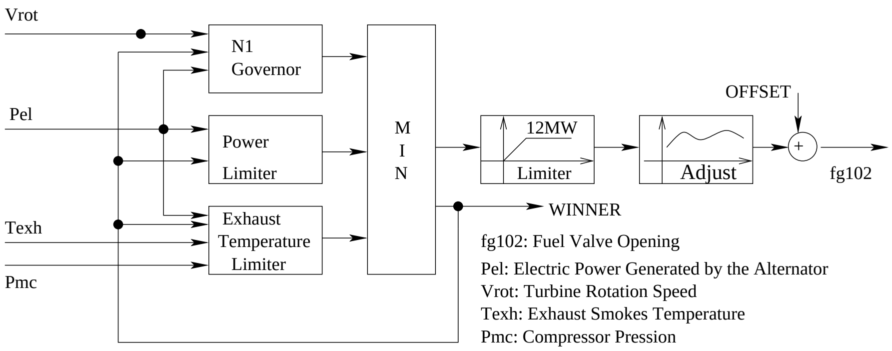

Fig. 11. Turbogas fuel gas valve controller.

From Fig. 11 it is clear that the turbogas controller output is obtained as the minimum (block MIN) of the outputs of the three subsystems N1 Governor, Power Limiter, Exhaust Temperature Limiter.

The *N1 Governor* block in Fig. 13 computes the power demand with the goal of maintaining the turbine rotation speed within given bounds.

The *Power Limiter* block in Fig. 14 computes the power demand with the goal of maintaining the electric power generated within given bounds.

The *Exhaust Temperature Limiter* block in Fig. 15 computes the power demand with the goal of maintaining the temperature of the exhaust smokes within given bounds.

The subsystem MIN in Fig. 11 computes the minimum among its inputs. Moreover, the block MIN returns the name (index) of the winner (i.e. of the input which attained the minimum value) on the output labeled WINNER.

The block *Limiter* in Fig. 11 saturates the power demand to 12MW.

The block *Adjust* together with the OFFSET parameter in Fig. 11 translates the power demand from the Limiter block into a fuel valve opening command, i.e. into a real number in the range [0,1].

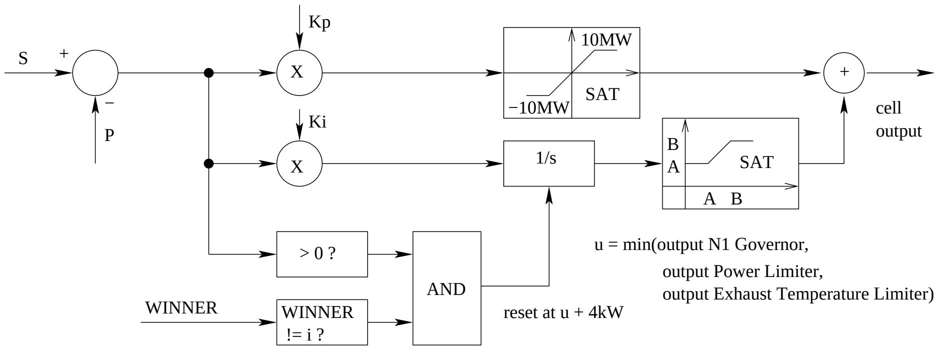

Fig. 12. Elementary cell used for the construction of turbogas controller subsystems: N1 Governor, Power Limiter, Exhaust Temperature Limiter. Cell Inputs: `S`, `P`, `WINNER`. Cell Parameters: `i`, `A`, `B`. Known Constants: `Kp`, `Ki` (from [10])

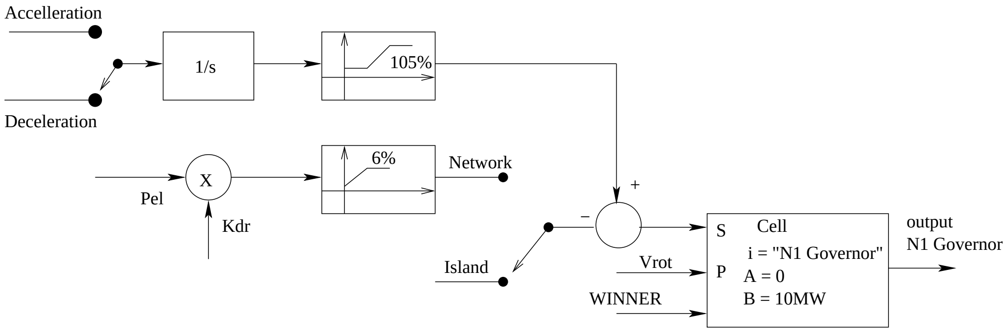

Fig. 13. Turbine Rotation Speed Controller (N1 Governor). The switches `Acceleration/Decelaration`, `Network/Island` change the controller *mode*. Their settings are chosen by a higher level controller or by the human operator. `Kdr` is a known constant (given to us in [10])

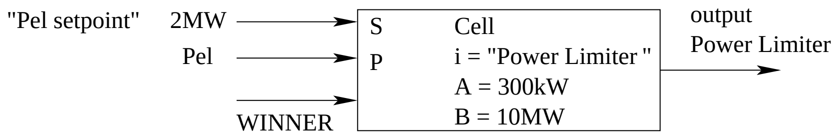

Fig. 14. Electric Power Controller (Power Limiter)

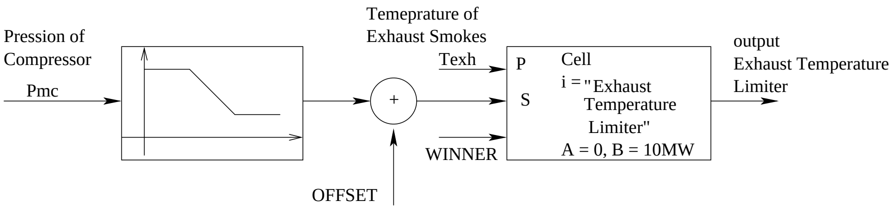

Fig. 15. Exhaust Smoke Temperature Controller (Exhaust Temperature Limiter)

5 Experimental Results

We transformed all the continuous time models into discrete time ones using a sampling time T of 10ms, as suggested in [10].

An example of the Murφ code used in the declaration section of our Murφ model is in Fig. 6. Space constraints do not allow us to show more here.

Using Murφ (with real numbers) we ran several experiments modifying the value of the *speed of variation* of the user demand (`MAX_D_U` in Fig. 6), thus computing an admissible range of values for the variation speed of the user demand.

Intuitively, if `MAX_D_U` is too big, the controller cannot compensate for the sudden user demand variation and the requirements will not be satisfied.

Fig. 16 shows our experimental results.

The size of each state is 470 bits (rounded up to 60 bytes). In all our experiments we used the `--cache` option which replaces Murφ hash table with a cache table [23].

When verification fails (rows 3 and 4 of Fig. 16), Murφ returns an error trace. In such cases the diameter of the reachability graph gives the error trace length. Thus it is quite clear that even for the 4th row of Fig. 16 the error trace is too long (804 states) to be shown here. However, in Fig. 17 we show the first and last state of such an error trace.

Note how, in row 1 and 2 of Fig. 16, when `MAX_D_U` increases, the number of reachable states increases too.

Row Number	`MAX_D_U`	Reachable States	Rules Fired	Reachability Graph Diameter	CPU Time (sec)	Verification Result
1	10.0	2246328	6738984	12904	16988.18	PASS
2	17.5	7492389	22477167	7423	54012.18	PASS
3	25	1739719	5186047	1533	12548.25	FAIL
4	50	36801	109015	804	271.77	FAIL

Fig. 16. Results on a INTEL Pentium 4, 2Ghz Linux PC with 512M RAM. Murφ options used: `-b` (bit compression), `-c` (40 bit hash compaction), `--cache` (use cache rather than hash table, cached Murφ) `-m350` (use 350 MB of RAM for the cache).

From Fig. 16 it is quite clear that only a very small fraction of the 2^{480} states (480 bits are needed to represent a state) is reachable. This is the main reason why we were able to complete verification using Murφ. Note however that if we consider the turbogas model without the controller, the number of reachable states is much higher, out of reach for Murφ. In this respect our findings are similar to those of Kurshan and McMillan in their arbiter verification [16].

The behaviour described above is somehow to be expected. In fact, here we are interested in automatic verification of a control system in a neighborhood of its *setpoint* (i.e. our initial state is the setpoint state). Since the *controller* goal is typically that of keeping the whole system in a (small) neighborhood of the setpoint, we may expect that the set of reachable states from the setpoint is not *too big* (if the controller is well designed).

Taking advantage of this fact, using a symbolic model checker (e.g. as HyTech and SMV are) may be hard. As a result, the representation of the system transition relation can be so large that we may run out of memory even before starting the reachability analysis. Indeed this was our experience when we tried to use HyTech and SMV on our hybrid system verification problem.

For the above reason we decided to try Murφ which is an *explicit state verifier*.

Our experimental results show that for hybrid (control) systems explicit state model checkers (e.g. as Murφ) should be considered among the available verification tools.

```
Startstate initstate fired.
Power:+2.000e+03
Vrot:+7.500e+01
Texh:+5.520e+02
N1_gov:+1.000e+03
Pow_lim:+1.000e+03
Temp_lim:+1.000e+03
valve_fg102:+1.000e-01
v:+7.500e+02
N1_state:+1.000e+03
Powlim_state:+1.000e+03
templim_state:+1.000e+03
minall:+1.000e+03
winner:2
step_counter:0
pression:+1.200e+01
user_demand:+0.000e+00
modality_value:1
................
```

```
................
The last state of the trace (in full) is:
Power:+1.695e+03
Vrot:+3.999e+01
Texh:+5.520e+02
N1_gov:+1.211e+04
Pow_lim:+2.112e+03
Temp_lim:+7.294e+03
valve_fg102:+2.111e-01
v:+1.050e+02
N1_state:+2.115e+03
Powlim_state:+1.808e+03
templim_state:+2.115e+03
minall:+2.112e+03
winner:2
step_counter:5
pression:+1.200e+01
user_demand:+1.500e+02
modality_value:2
----------
End of the error trace.
==============================================
Result: Invariant "rotation speed ok" failed.
```

Fig. 17. First and last state of the 804 states of Murφ error trace for row 4 of Fig. 16.

6 Conclusions

We showed how the C `long double` type (*finite precision real numbers*) can be easily imported within the Murφ verifier. This allows us to easily use Murφ for modeling and verification of hybrid systems.

We presented a case study on automatic verification of a *Turbogas Control System* (TCS) using the Murφ verifier. TCS is is a quite large control system and is the main subsystem of *ICARO*, a 2MW *Co-generative Electric Power Plant.*

We showed experimental results on using Murφ for verification of TCS. Using Murφ we were able to compute an admissible range of values for the variation speed of the *user demand* for electric power to the turbogas.

Our experimental results suggest that Murφ enhanced with finite precision real numbers can be used quite effectively for modeling and for automatic analysis of hybrid control systems.

Testing Murφ on larger hybrid systems is a natural *next step* of our research efforts.

References

[1] url: `http://www.eecs.berkeley.edu/~tah/HyTech`.

[2] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. on Software Engineering*, 22, 1996.

[3] A. Bobbio, E. Ciancamerla, G. Franceschinis, R. Gaeta, M. Minichino, and L. Portinale. Methods of increasing modelling power for safety analysis, applied to a turbine digital control system. In *Proc. of 21st International Conference on "Computer Safety, Reliability and Security" (SAFECOMP)*, LNCS, Catania, Italy, Sept 2002. Springer.

[4] A. Bobbio, E. Ciancamerla, M. Gribaudo, A. Horvath, M. Minichino, and E. Tronci. Model checking based on fluid petri nets for the temperature control system of the icaro co-generative plant. In *Proc. of 21st International Conference on "Computer Safety, Reliability and Security" (SAFECOMP)*, LNCS, Catania, Italy, Sept 2002. Springer.

[5] A. Bobbio, S.Bologna, M. Minichino, E. Ciancamerla, P.Incalcaterra, C.Kropp, and E. Tronci. Advanced tecniques for safety analysis applied to the gas turbine control system of icaro co generative plant. In *Proc. of X Convegno TESEC*, Genova, Italy, June 2001.

[6] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers*, C-35(8), Aug 1986.

[7] url: `http://www.dsi.uniroma1.it/~tronci/cached.murphi.html`.

[8] N. V. Chernikova. Algorithm for discovering the set of all solutions of a linear programming problem. *USSR Computational Mathematics and Mathematical Physics*, 8(6):282–293, 1968.

[9] D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang. Protocol verification as a hardware design aid. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–5, 1992.

[10] ENEA. *Proprietary ICARO Documentation*.

[11] N. Halbwachs. Delay analysis in synchronous programs. In *Proc. of: Computer Aided Verification (CAV)*, number 697 in LNCS, pages 333–346. Springer, 1993.

[12] N. Halbwachs, P. Raymond, and Y.-E. Proy. Verification of linear hybrid systems by means of convex approximation. In *Proc. of: Static Analysis Symposium (SAS)*, number 864 in LNCS, pages 223–237. Springer, 1994.

[13] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: The next generation. In *Proc. of the 16th Annual IEEE Real-time Systems Symposium (RTSS)*, pages 56–65. IEEE, 1995.

[14] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1, 1997.

[15] A. J. Hu, G. York, and D. L. Dill. New techniques for efficient verification with implicitily conjoined bdds. In *31st IEEE Design Automation Conference*, pages 276–282, 1994.

[16] R.P. Kurshan and K.L. McMillan. Analysis of digital circuits through symbolic reduction. *IEEECAD*, 10(11):1356–1371, November 1991.

[17] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL: Status and developments. In Orna Grumberg, editor, *CAV97*, number 1254 in LNCS, pages 456–459. Springer–Verlag, Jun 1997.

[18] K. L. McMillan. *Symbolic model checking*. Kluwer Academic Publishers, Massachusetts, 1993.

[19] url: `http://sprout.stanford.edu/dill/murphi.html`.

[20] G. Della Penna, B. Intrigila, E. Tronci, and M. Venturini Zilli. Exploiting transition locality in the disk based murφ verifier. In *Proc. of 4th International Conference on "Formal Methods in Computer Aided Verification" (FMCAD)*, LNCS, Portland, Oregon, USA, Nov 2002. Springer.

[21] url: `http://www.cs.cmu.edu/~modelcheck/`.

[22] url: `http://netlib.bell-labs.com/netlib/spin/whatispin.html`.

[23] E. Tronci, G. Della Penna, B. Intrigila, and M. Venturini Zilli. Exploiting transition locality in automatic verification. In *IFIP WG 10.5 Advanced Research Working Conference on: Correct Hardware Design and Verification Methods (CHARME)*. LNCS, Springer, Sept 2001.

[24] E. Tronci, G. Della Penna, B. Intrigila, and M. Venturini Zilli. A probabilistic approach to space-time trading in automatic verification of concurrent systems. In *Proc. of 8th IEEE Asia-Pacific Software Engineering Conference (APSEC)*, Macau SAR, China, Dec 2001. IEEE Computer Society Press.

[25] A. L. Turk, S. T. Probst, and G. J. Powers. Verification of real-time chemical processing systems. In *Hybrid and Real-Time Systems*, number 1201 in LNCS, pages 259–272. Springer, 1997.

[26] url: `http://www.docs.uu.se/docs/rtmv/uppaal/`.