On the Computational Power of Timed Differentiable Petri Nets

Serge Haddad[1], Laura Recalde[2], Manuel Silva[2]

[1] LAMSADE-CNRS UMR 7024, University Paris-Dauphine, France
E-mail: haddad@lamsade.dauphine.fr

[2] GISED, University Zaragossa, Spain
E-mail: {lrecalde | silva}@unizar.es

Abstract. Well-known hierarchies discriminate between the computational power of discrete time and space dynamical systems. *A contrario* the situation is more confused for dynamical systems when time and space are continuous. A possible way to discriminate between these models is to state whether they can simulate Turing machine. For instance, it is known that continuous systems described by an ordinary differential equation (ODE) have this power. However, since the involved ODE is defined by overlapping local ODEs inside an infinite number of regions, this result has no significant application for differentiable models whose ODE is defined by an explicit representation. In this work, we considerably strengthen this result by showing that Time Differentiable Petri Nets (TDPN) can simulate Turing machines. Indeed the ODE ruling this model is expressed by an explicit linear expression enlarged with the "minimum" operator. More precisely, we present two simulations of a two counter machine by a TDPN in order to fulfill opposite requirements: robustness and boundedness. These simulations are performed by nets whose dimension of associated ODEs is constant. At last, we prove that marking coverability, submarking reachability and the existence of a steady-state are undecidable for TDPNs.

1 Introduction

Hybrid systems. Dynamic systems can be classified depending on the way time is represented. Generally, trajectories of discrete-time systems are obtained by iterating a transition function whereas the ones of continuous-time systems are often solutions of a differential equation. When a system includes both continuous and discrete transitions it is called an *hybrid system*. On the one hand, the expressive power of hybrid systems can be strictly greater than the one of Turing machines (see for instance [12]). On the other hand, in restricted models like timed automata [1], several problems including reachability can be checked in a relatively efficient way (i.e. they are $PSPACE$-complete). The frontier between decidability and undecidability in hybrid systems is still an active research topic [8,10,4,11].

Continuous systems. A special kind of hybrid systems where the trajectories are continuous (w.r.t. standard topology) and right-differentiable functions of time have been intensively studied. They are defined by a finite number regions and associated ordinary differential equations ODEs such that inside a region r, a trajectory fulfills the equation $\dot{x}_d = f_r(x)$ where x is the trajectory and $\dot{x}_d$ its right derivative. These additional requirements are not enough to limit their expressiveness. For instance, the model of [2] has piecewise constant derivatives inside regions which are polyhedra and it is Turing equivalent if its space dimension is at least 3 (see also [3,5] for additional expressiveness results).

Differentiable systems. A more stringent requirement consists in describing the dynamics of the system by a single ODE $\dot{x} = f(x)$ where f is continuous, thus yielding continuously differentiable trajectories. We call such models, *differentiable systems*. In [6], the author shows that differentiable systems in $\mathbb{R}^3$ can simulate Turing machine. The corresponding ODE is obtained by extrapolation of the transition function of the Turing machine over every possible configuration. Indeed such a configuration is represented as a point in the first dimension of the ODE (and also in the second one for technical reasons) and the third dimension corresponds to the time evolution. The explicit local ODE around every representation of a configuration is computed from this configuration and its successor by the Turing machine. Thus the explicit equations of the ODE are piecewise defined inside *an infinite number regions* which is far beyond the expressiveness of standard ODE formalisms used for the design and analysis of dynamical systems. So the question to determine which (minimal) set of operators in an explicit expression of f is required to obtain Turing machine equivalence, is still open.

Our contribution. In this work, we (partially) answer this question by showing that Time Differentiable Petri Nets (a model close to Time Continuous Petri Nets [7,13]) can simulate Turing machines. Indeed the ODE ruling this model is particularly simple. First its expression is a linear expression enlarged with the "minimum" operator. Second, it can be decomposed into a finite number of linear ODEs $\dot{x} = \mathbf{M} \cdot x$ (with $\mathbf{M}$ a matrix) inside polyhedra.

More precisely, we present two simulations of two counter machines in order to fulfill opposite requirements: robustness (allowing some perturbation of the simulation) and boundedness of the simulating net system. Our simulation is performed by a net with a constant number of places, i.e. whose dimension of its associated ODE is constant (in $(\mathbb{R}_{\geq 0})^6$ for robust simulation and in $[0, K]^{14}$ for bounded simulation). Afterwards, by modifying the simulation, we prove that marking coverability, submarking reachability and the existence of a steady-state are undecidable for (bounded) TDPNs.

Outline of the paper. In section 2, we recall notions of dynamical systems and simulations. In section 3, we introduce TDPNs. Then we design a robust simulation of counter machines in section 4 and a bounded one in section 5. Afterwards, we establish undecidability results in section 6. At last, we conclude and give some perspectives to this work.

2 Dynamical systems and simulation

Notations. $\mathbb{N}$ (resp. $\mathbb{R}_{\geq 0}, \mathbb{R}_{>0}$) is the set of non negative integers (resp. non negative, positive reals).

Definition 1. *A deterministic dynamical system $(X, \mathcal{T}, f)$ is defined by:*
- *a state space X, a time space $\mathcal{T}$ ($\mathcal{T}$ is either $\mathbb{N}$ or $\mathbb{R}_{\geq 0}$),*
- *a transition function f from $X \times \mathcal{T}$ to X fulfilling:*

$$\forall x \in X, \forall \tau_1, \tau_2 \in \mathcal{T}, f(x, 0) = x \wedge f(x, \tau_1 + \tau_2) = f(f(x, \tau_1), \tau_2)$$

In the sequel, we will only deal with deterministic systems. In a *discrete* (resp. *continous*) system $X \subseteq \mathbb{N}^d$ for some d (resp. $X \subseteq (\mathbb{R}_{\geq 0})^d$) and $\mathcal{T} = \mathbb{N}$ (resp. $\mathcal{T} = \mathbb{R}_{\geq 0}$). The simulation of a discrete system by a continuous one involves a mapping from the set of states of the discrete system to the powerset of states of the continuous systems and an observation epoch. A simulation ensures that, starting from some state in the image of an initial state of the discrete system and observing the state reached after some multiple n of the epoch, one can recover the state of the discrete system after n steps. If the continuous system evolves in some bounded subset of $(\mathbb{R}_{\geq 0})^d$, the simulation is said *bounded*.

Definition 2. *A continuous dynamical system $(Y, \mathbb{R}_{\geq 0}, g)$ simulates a discrete dynamical system $(X, \mathbb{N}, f)$ if there is a mapping ϕ from X to 2^Y and $\tau_0 \in \mathbb{R}_{>0}$ such that:*
- $\forall x \neq x' \in X, \phi(x) \cap \phi(x') = \emptyset$
- $\forall x \in X, \forall y \in \phi(x), g(y, \tau_0) \in \phi(f(x, 1))$

The simulation is said bounded *if $Y \subset [0, K]^d$ for some $K \in \mathbb{R}_{\geq 0}$.*

Roughly speaking, a *robust* simulation is insensitive to small perturbations of the simulation mapping and the observation instants. In order to define robust simulation, we refine the notion of simulation. First, a *two-level* dynamical system $(Y = Y_1 \times Y_2, \mathbb{R}_{\geq 0}, g)$ is such that g is defined by g_1 from $Y_1 \times \mathbb{R}_{\geq 0}$ to Y_1 and by g_2 from $Y \times \mathbb{R}_{\geq 0}$ to Y_2 as: $g((y_1, y_2), \tau) = (g_1(y_1, \tau), g_2((y_1, y_2), \tau))$. In words, the behaviour of the first component depends only on its local state.

Definition 3. *A two-level continuous dynamical system $(Y, \mathbb{R}_{\geq 0}, g)$* consistently *simulates a discrete dynamical system $(X, \mathbb{N}, f)$ if there is $y_0 \in Y_1$, a mapping ϕ from X to 2^{Y_2} and $\tau_0 \in \mathbb{R}_{>0}$ such that:*
- $\forall x \neq x' \in X, \phi(x) \cap \phi(x') = \emptyset$,
- $g_1(y_0, \tau_0) = y_0$,
- $\forall x \in X, \forall y \in \phi(x), g_2((y_0, y), \tau_0) \in \phi(f(x, 1))$.

Note that the first part of component is a "fixed" part of the system since its whole trajectory does not depend on the input of the simulated system.

Definition 4. *A simulation (by a two-level system) is* robust *iff there exists $\delta, \epsilon \in \mathbb{R}_{>0}$ such that:*
- $\forall x \neq x' \in X, dist(\phi(x), \phi(x')) > 2\epsilon$
- $\forall x \in X, \forall y_2 \in Y_2, \forall n \in \mathbb{N}, \forall \tau \in \mathbb{R}_{\geq 0}$,

  $max(dist(y_2, \phi(x)), dist(\tau, n\tau_0)) \leq \delta \Rightarrow dist(g_2((y_0, y_2), \tau), \phi(f(y, n))) \leq \epsilon$

where $dist(Y, Y') = \inf(|y - y'|_\infty \mid y \in Y, y' \in Y')$

Thus, if the simulation is robust, starting with an initial state no more pertubated than δ and delaying or anticipating the observation of the system by no more than δ, the state of the simulated system can be recovered. For obvious reasons, the simulation of an infinite-state system cannot be *simultaneously robust and bounded*.

3 Timed Differentiable Petri Nets

Notations. Let f be a *partial* mapping then $f(x) = \perp$ means that $f(x)$ is undefined. Let $\mathbf{M}$ be a matrix whose domain is $A \times B$, with $A \cap B = \emptyset$ and $a \in A$ (resp. $b \in B$) then $\mathbf{M}(a)$ (resp. $\mathbf{M}(b)$) denotes the vector corresponding to the row a (resp. the column b) of $\mathbf{M}$.

Definition 5 (Timed Differentiable Petri Nets). *A Timed Differentiable Petri Net* $\mathcal{D} = \langle P, T, \mathbf{C}, \mathbf{W} \rangle$ *is defined by:*

- P*, a finite set of places,*
- T*, a finite set of transitions with* $P \cap T = \emptyset$*,*
- $\mathbf{C}$*, the incidence matrix from* $P \times T$ *to* $\mathbb{Z}$*, we denote by* ${}^\bullet t$ *(resp.* $t^\bullet$*) the set of input places (resp. output places) of* t*,* $\{p \mid \mathbf{C}(p,t) < 0\}$ *(resp.* $\{p \mid \mathbf{C}(p,t) > 0\}$*).* $\mathbf{C}(t)$ *is called the incidence of* t*.*
- $\mathbf{W}$*, the speed control matrix a partial mapping from* $P \times T$ *to* $\mathbb{R}_{>0}$ *such that:*
  - $\forall t \in T, \exists p \in P, \mathbf{W}(p,t) \neq \perp$
  - $\forall t \in T, \forall p \in P, \mathbf{C}(p,t) < 0 \Rightarrow \mathbf{W}(p,t) \neq \perp$

A time differentiable Petri net is a Petri net enlarged with a *speed control matrix*. In a Petri net, a state $\mathbf{m}$, called a *marking*, is a positive integer vector over the set of places (i.e. an item of $\mathbb{N}^P$) where an unit is called a token. The state change is triggered by transition *firings*. In $\mathbf{m}$, the firing of a transition t with multiplicity $k \in \mathbb{N}$ yielding marking $\mathbf{m}' = \mathbf{m} + k\mathbf{C}(t)$ is only possible if $\mathbf{m}'$ is positive. Note that in Petri nets, both the choice of the transition firing and the number of simultaneous firings are non deterministic.

In a TDPN, a marking $\mathbf{m}$, is a positive real vector over the set of places (i.e. an item of $(\mathbb{R}_{\geq 0})^P$). The non determinism of Petri nets is solved by computing at any instant the instantaneous firing rate of every transition and then applying the incidence matrix in order to deduce the infinitesimal variation of the marking. The instantaneous firing rate of transitions $\mathbf{f}(\mathbf{m})(t)$ depends on the current marking *via* the speed control matrix $\mathbf{W}$: $\mathbf{f}(\mathbf{m})(t) = min(\mathbf{W}(p,t) \cdot \mathbf{m}(p) \mid \mathbf{W}(p,t) \neq \perp)$.

The first requirement about $\mathbf{W}$ ensures that the firing rate of any transition may be determined whereas the second one ensures that the marking remains non negative since any input place p of a transition t controls its firing rate.

Definition 6 (Trajectory). *Let* $\mathcal{D}$ *be a TDPN, then a trajectory is a* ***continuously differentiable*** *mapping* $\mathbf{m}$ *from time (i.e.* $\mathbb{R}_{\geq 0}$*) to the set of markings (i.e.* $(\mathbb{R}_{\geq 0})^P$*) which satisfies the following differential equation system:*

$$\dot{\mathbf{m}} = \mathbf{C} \cdot \mathbf{f}(\mathbf{m}) \tag{1}$$

If $\mathbf{m}(0)$ is non negative, the requirement of non negativity is a consequence of the definition of TDPNs and one can also prove (by a reduction to the linear equation case) that given an initial marking there is always a single trajectory. Equation 1 is particularly simple since it is expressed as a linear equation enlarged with the *min* operator. We introduce the concept of *configurations*: a configuration assigns to a transition, the place that controls its firing rate.

Definition 7 (Configuration). *Let $\mathcal{D}$ be a TDPN, then a configuration cf of $\mathcal{D}$ is a mapping from T to P such that $\forall t \in T, \mathbf{W}(cf(t), t) \neq \perp$. Let cf be a configuration, then $[cf]$ denotes the following polyhedron:*

$$[cf] = \{\mathbf{m} \in (\mathbb{R}_{\geq 0})^P \mid \forall t \in T, \forall p \in P, \mathbf{W}(p,t) \neq \perp \Rightarrow \mathbf{W}(p,t) \cdot \mathbf{m}(p) \geq \mathbf{W}(cf(t),t) \cdot \mathbf{m}(cf(t))\}$$

By definition, there are $\prod_{t \in T} |\{p \mid \mathbf{W}(p,t) \neq \perp\}| \leq |P|^{|T|}$ configurations. In the sequel, we use indifferently the word configuration to denote both the mapping cf and the polyhedron $[cf]$. Inside the polyhedron $[cf]$, the differential equation ruling $\mathcal{D}$ becomes linear:

$$\forall p \in P, \dot{\mathbf{m}}(p) = \sum_{t \in T} \mathbf{C}(p,t) \cdot \mathbf{W}(cf(t),t) \cdot \mathbf{m}(cf(t))$$

Graphical notations. We extend the graphical notations of Petri nets in order to take into account matrix $\mathbf{W}$. A Petri net is a bipartite graph where places are represented by circles (sometimes with their initial marking inside) and transitions by rectangles. An arc denotes a relation between a place and a transition. Note that arcs corresponding to matrix $\mathbf{C}$ are oriented whereas arcs corresponding to matrix $\mathbf{W}$ are not oriented. There are four possible patterns illustrated in figure 1. When $\mathbf{W}(p,t) = \perp \wedge \mathbf{C}(p,t) > 0$, place p receives tokens from t and does not control its firing rate. There is an oriented arc from t to p labelled by $\mathbf{C}(p,t)$. When $\mathbf{W}(p,t) \neq \perp \wedge \mathbf{C}(p,t) < 0$, place p provides tokens to t. So it *must control* its firing rate. The non oriented arc between p and t is redundant, so we will not draw it and represent only an oriented arc from p to t both labelled by $-\mathbf{C}(p,t)$ and $\mathbf{W}(p,t)$. In order to distinguish between these two labels, $\mathbf{W}(p,t)$ will always be drawn inside a box. When $\mathbf{W}(p,t) \neq \perp \wedge \mathbf{C}(p,t) > 0$, place p receives tokens from t and controls its firing rate. There is both an oriented arc from t to p and a non oriented arc between p and t with their corresponding labels. When $\mathbf{W}(p,t) \neq \perp \wedge \mathbf{C}(p,t) = 0$, place p controls the firing rate of t and t does not modify the marking of p, so there is a non oriented arc between p and t. We omit labels $\mathbf{C}(p,t)$, $-\mathbf{C}(p,t)$ and $\mathbf{W}(p,t)$ when they are equal to 1.

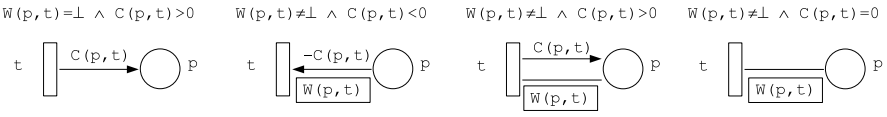

Fig. 1. Graphical notations

The net of Figure 2 illustrates TDPNs. In order to simplify the notations, when writing the differential equations, we use p as a notation for $\mathbf{m}(p)$ (the

trajectory projected on p). The ODE corresponding to this net is (note that place pk holds a constant number of tokens):

$$\dot{x}_1 = \mathbf{f}(t_2) - \mathbf{f}(t_4) = \min\{\omega \cdot x_2, 2a\omega \cdot y_1\} - \min\{a\omega \cdot x_1, a\omega\}$$
$$\dot{x}_2 = \mathbf{f}(t_1) - \mathbf{f}(t_3) = \min\{a\omega \cdot y_2, a\omega\} - \min\{2a\omega \cdot x_2, \omega \cdot x_1\}$$
$$\dot{y}_1 = \mathbf{f}(t_4) - \mathbf{f}(t_2) = \min\{a\omega \cdot x_1, a\omega\} - \min\{\omega \cdot x_2, 2a\omega \cdot y_1\}$$
$$\dot{y}_2 = \mathbf{f}(t_3) - \mathbf{f}(t_1) = \min\{2a\omega \cdot x_2, \omega \cdot x_1\} - \min\{a\omega \cdot y_2, a\omega\}$$

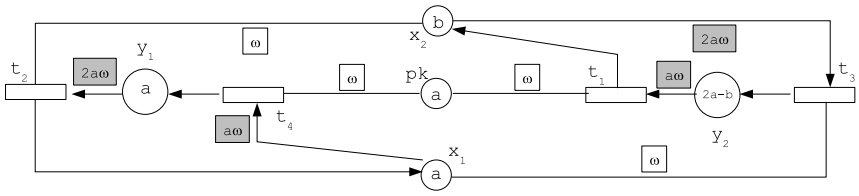

Fig. 2. A periodic TDPN

However, it can be observed that $y_1 + x_1$ and $y_2 + x_2$ are constant. Hence the system (with the initial condition described by the marking in the figure) is equivalent to:

$$\dot{x}_1 = \min\{\omega \cdot x_2, 2a\omega \cdot y_1\} - \min\{a\omega \cdot x_1, \omega \cdot a\},\ y_1 = 2a - x_1$$
$$\dot{x}_2 = \min\{a\omega \cdot y_2, \omega \cdot a\} - \min\{2a\omega \cdot x_2, \omega \cdot x_1\},\ y_2 = 2a - x_2$$

This corresponds to a set of sixteen configurations. Let us solve the differential system with $1 \leq a \leq b \leq 2a - 1$. The linear system that applies initially is:

$$\dot{x}_1 = \omega \cdot x_2 - \omega \cdot a,\ y_1 = 2a - x_1,\ \dot{x}_2 = \omega \cdot a - \omega \cdot x_1,\ y_2 = 2a - x_2$$

In figure 2, we have represented the "inactive" items of matrix $\mathbf{W}$ in a shadowed box. In the sequel, we use this convention when it will be relevant. The solution of this system is:

$$x_1(\tau) = a + (b-a)\sin(\omega \cdot \tau),\ x_2(\tau) = a + (b-a)\cos(\omega \cdot \tau)$$
$$y_1(\tau) = a - (b-a)\sin(\omega \cdot \tau),\ y_2(\tau) = a - (b-a)\cos(\omega \cdot \tau)$$

This trajectory stays infinitely in the initial configuration and consequently it is the behaviour of the net. Note that the dimension of the ODE may be strictly smaller than the number of places. Indeed, the existence of a linear invariant such like $\sum_{p \in P} \mathbf{m}(p) = cst$ decreases by one unit the number of dimensions. Otherwise stated, the dimension of the ODE is not $|P|$ but $\mathbf{rank}(\mathbf{C})$. Here, $|P| = 5$ and $\mathbf{rank}(\mathbf{C}) = 2$.

4 A robust simulation

4.1 Two counter machines

We will simulate two (non negative integer) counter machines (equivalent to Turing machines [9]). Their behaviour is described by a set of instructions. An instruction `I` may be one of the following kind with an obvious meaning ($\mathtt{cpt_u}$

is a counter with $u \in \{1, 2\}$):

- I : goto I′;
- I : increment($\mathtt{cpt_u}$); goto I′;
- I : decrement($\mathtt{cpt_u}$); goto I′;
- I : if $\mathtt{cpt_u} = 0$ then goto I′ else goto I";
- I : STOP;

W.l.o.g. a decrementation must be preceded by a test on the counter and the (possible) successor(s) of an instruction is (are) always different from it.

4.2 Basic principles of the simulation

Transition pairs. In a TDPN, when a transition begins to fire, it will never stop. Thus we use *transition pairs* in order to *temporarily* either move tokens from one place to another one, or produce/consume tokens in a place.

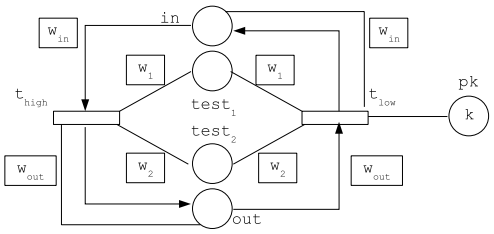

Fig. 3. A transition pair

Let us examine transitions t_{high} and t_{low} of figure 3. Their incidence is opposite. So if their firing rate is equal no marking change will occur. Let us examine $\mathbf{W}$, all the items of $\mathbf{W}(t_{high})$ and $\mathbf{W}(t_{low})$ are equal except $\mathbf{W}(pk, t_{low}) = k$ and $\mathbf{W}(pk, t_{high}) = \perp$. Thus, if any other place controls the firing rate of t_{low} it will be equal to the one of t_{high}. Place pk is a *constant* place meaning that its marking will always be k. Summarizing:

- if $w_{in}\mathbf{m}(in) > k \wedge w_{out}\mathbf{m}(out) > k \wedge w_1\mathbf{m}(test_1) > k \wedge w_2\mathbf{m}(test_2) > k$ then this pair transfers some amount of tokens from *in* to *out*,
- otherwise, there will be no marking change.

The clock subnet. The net that we build consists in two subnets: an instance of the subnet of figure 2, called in the sequel the *clock* subnet, and another subnet depending on the counter machine called the *operating* subnet. The clock subnet has k as average value, 1 as amplitude and π as period (i.e. $\omega = 2$). We recall the behavioural equations of the place markings that will be used by the operating subnet: $\mathbf{m}(x_1)(\tau) = k + sin(2\tau), \mathbf{m}(y_1)(\tau) = k - sin(2\tau)$.

Figure 4 represents the evolution of markings for x_1, y_1 and x_2 (the marking of place y_2 is symmetrical to x_2 w.r.t. the axis $\mathbf{m} = k$). Note that the mottled area is equal to 1.

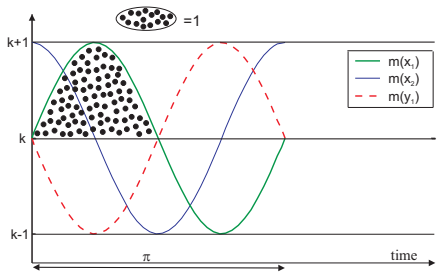

Fig. 4. The behaviour of the clock subnet

The marking changes of the operating subnet will be ruled by the places x_1 and y_1. An *execution cycle* of the net will last π. The first part of the cycle (i.e. $[h\pi, h\pi + \pi/2]$ for some $h \in \mathbb{N}$) corresponds to $\mathbf{m}(x_1) \geq k$ and the second part of the cycle (i.e. $[h\pi + \pi/2, (h+1)\pi]$) corresponds to $\mathbf{m}(y_1) \geq k$. So, the period of observation τ_0 is equal to π.

Specialisation of the transition pairs pattern.

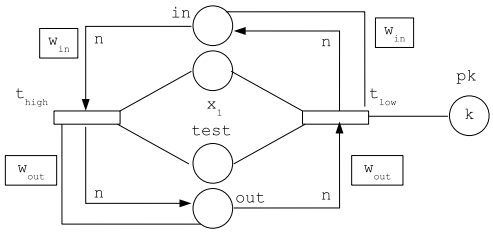

Fig. 5. A specialised transition pair

Using place x_1 (or y_1), we specialise transition pairs as illustrated in figure 5 (pk is the constant place of the clock subnet). In this subnet, one of the test place is x_1 and the control weights of the two test places (x_1 and $test$) are 1. First due to the periodical behaviour of $\mathbf{m}(x_1)$, no tokens transfer will occur during the second part of the cycle. Let us examine the different cases during a time interval $[h\pi, h\pi + \pi/2]$. We assume that within this interval $\mathbf{m}(test), \mathbf{m}(in)$ and $\mathbf{m}(out)$ are not modified by the other transitions.

- If $\mathbf{m}(test)(h\pi) \leq k$ then there will be no transfer of tokens.
- If $\mathbf{m}(test)(h\pi) \geq k+1 \wedge w_{in}(\mathbf{m}(in)(h\pi) - n) \geq k+1 \wedge w_{out}\mathbf{m}(out)(h\pi) \geq k+1$ then t_{high} will be controlled by x_1 and t_{low} will be controlled by pk. Hence (see the integral of figure 4) exactly n tokens will be transfered from in to out.
- Otherwise, some amount of tokens in $[0, n]$ will be transfered from in to out.

From a simulation point of view, one wants to avoid the last case. For the same reason, when possible, we choose w_{in} and w_{out} enough large so that it ensures that *in* and *out* will never control t_{high} and t_{low}.

4.3 The operating subnet

Places of the operating subnet and the simulation mapping. Let us suppose that the counter machine has l instructions $\{\mathtt{I}_1, \ldots, \mathtt{I}_l\}$ and two counters $\{\mathtt{cpt}_1, \mathtt{cpt}_2\}$. The operating subnet has the following places: pc, qc, pn, qn, c_1, c_2. The forth first places simulate the program counter whereas the last ones simulate the counters. Furthermore by construction, the following invariants will hold for every reachable marking $\mathbf{m}$: $\mathbf{m}(pc)+\mathbf{m}(qc) = l+1$ and $\mathbf{m}(pn)+\mathbf{m}(qn) = l+1$. We now define the simulation mapping ϕ. Assume that, in a state s of the counter machine, $\mathtt{I}_\mathtt{i}$ is the next instruction and the value of the counter $\mathtt{cpt}_\mathtt{u}$ is v_u. Then a marking $\mathbf{m} \in \phi(s)$ iff:

- The submarking corresponding to the clock subnet is its initial marking.
- $\mathbf{m}(pn) = i$, $\mathbf{m}(qn) = l + 1 - i$,
  if $1 < i < l$ then
    $\mathbf{m}(pc) \in [i - l/k, i + l/k]$ and $\mathbf{m}(qc) \in [l + 1 - i - l/k, l + 1 - i + l/k]$
  else if $i = 1$
    $\mathbf{m}(pc) \in [1, 1 + l/k]$ and $\mathbf{m}(qc) \in [l - l/k, l]$
  else if $i = l$
    $\mathbf{m}(pc) \in [l - l/k, l]$ and $\mathbf{m}(qc) \in [1, 1 + l/k]$
- $\mathbf{m}(c_1) = k - 1 + 3v_1, \mathbf{m}(c_2) = k - 1 + 3v_2$.
  Moreover, we choose $k \geq 6l^2$ for technical reasons.

Principle of the instruction simulation. The simulation of an instruction $\mathtt{I}_\mathtt{i}$ takes exactly the time of the cycle of the clock subnet and is decomposed in two parts ($\mathbf{m}(x_1) \geq k$ followed by $\mathbf{m}(y_1) \geq k$).

The first stage is triggered by $((k + 3l)/i)\mathbf{m}(pc) \geq k + 1 \wedge ((k + 3l)/(l + 1 - i)\mathbf{m}(qc) \geq k + 1$ and performs the following tasks:

- updating $\mathbf{m}(pn)$ by producing (resp. consuming) $j - i$ tokens if $j > i$ (resp. $j < i$) where $\mathtt{I}_\mathtt{j}$ is the next instruction; simultaneously updating $\mathbf{m}(qn)$ accordingly. If $\mathtt{I}_\mathtt{i}$ is a conditional jump, this involves to find the appropriate j. The marking of pn will vary from i to j and the one of qn from $l + 1 - i$ to $l + 1 - j$,
- updating the counters depending on the instruction.

The second stage is triggered by $((k + 3l)/j)\mathbf{m}(pn) \geq k + 1 \wedge ((k + 3l)/(l + 1-j)\mathbf{m}(qn) \geq k+1$ and performs the following task: updating $\mathbf{m}(pc)$ and $\mathbf{m}(qc)$ by a variable value in such a way that their marking still belong to the intervals associated with the simulation mapping.

First stage: simulation of an unconditional jump. This part of the simulation applies to both an unconditional jump, an incrementation and a decrementation. The simulation of the counter updates is straightforward once this pattern is presented. For this kind of instructions, the next instruction say $\mathtt{I}_\mathtt{j}$ is *a priori* known.

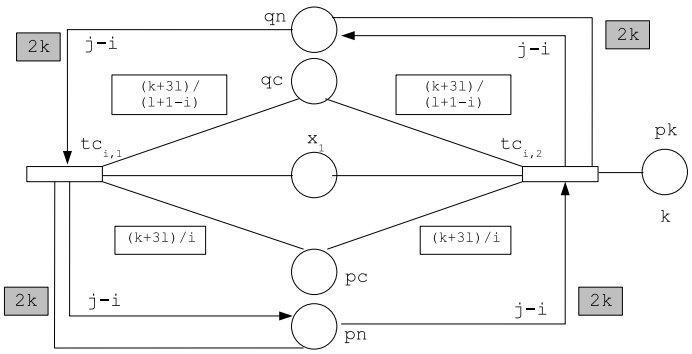

Fig. 6. First stage: simulation of an unconditional jump

The subnet we build depends on the relative values of i (the index of the current instruction) and j (the index of the next instruction). Here, we assume that $i < j$, the other case is similar. The transition pair of figure 6 is both triggered by pc and qc.

- Assume that the current instruction is $\mathtt{I_{i'}}$ with $i' \neq i$. If $i' < i$ then pc disables the transition pair whereas if $i' > i$ then qc disables the transition pair. We explain the first case. $\mathbf{m}(pc) \leq i' + l/k \leq i - 1 + l/k$; thus $((k+3l)/i)\mathbf{m}(pc) \leq ((k+3l)/i)(i-1+l/k) \leq k-1$ (due to our hypothesis on k).
- Assume that the current instruction is $\mathtt{I_i}$. Then both $((k+3l)/i)\mathbf{m}(pc) \geq ((k+3l)/i)(i-l/k) \geq k+2$ and $((k+3l)/(l+1-i))\mathbf{m}(qc) \geq ((k+3l)/(l+1-i))((l+1-i)-l/k) \geq k+2$.

Thus in the second case, the pair is activated and transfers $j-i$ tokens from qn to pn during the first part of the cycle as required.

Note that $\mathbf{W}(pn, tc_{i,1}) = \mathbf{W}(pn, tc_{i,2}) = \mathbf{W}(qn, tc_{i,1}) = \mathbf{W}(qn, tc_{i,2}) = 2k$ ensures that places pn and qn do not control these transitions (since $2k \geq k+2$ for k enough large).

First stage: simulation of a conditional jump. The first stage for simulating the instruction $\mathtt{I_i}$: `if` $\mathtt{cpt_u} = 0$ `then goto` $\mathtt{I_j}$ `else goto` $\mathtt{I_{j'}}$; is illustrated in figure 7 in case $i < j < j'$ (the other cases are similar).

It consists in two transition pairs. Pair $tc_{i,1}, tc_{i,2}$ mimics the first stage of an unconditional jump from $\mathtt{I_i}$ to $\mathtt{I_j}$. It will transfer during the first part of the cycle $j-i$ tokens from qn to pn. Pair $tc_{i,3}, tc_{i,4}$ is triggered if $c_u \geq k+1$ (i.e. the counter $\mathtt{cpt_u}$ is non null). If it is the case it will transfer $j'-j$ tokens from qn to pn. Thus:

- If $\mathbf{m}(c_u) = k-1$ then only the first pair is triggered and $j-i$ tokens will be transfered from qn to pn.
- otherwise $\mathbf{m}(c_u) \geq k+2$, the two pairs are simultaneously triggered and $j-i$ tokens will be transfered from qn to pn and $j'-j$ from qn to pn. Summing, $j'-i$ tokens will be transfered from qn to pn as required.

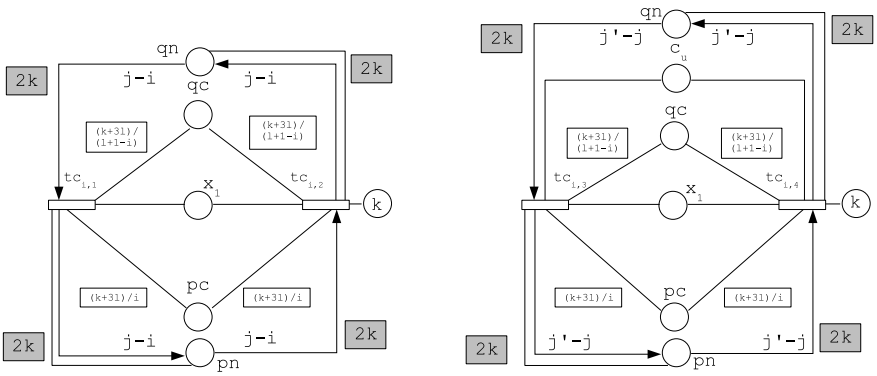

Fig. 7. The first stage of a conditional jump (places are duplicated for readability)

The second stage. This stage is the *difficult* part of this simulation. Due to the fact that the ODE ruling a TDPN is a linear equation inside a configuration, we cannot obtain a precise updating of $\mathbf{m}(pc)$ and $\mathbf{m}(qc)$. Roughly speaking it would require to reach a steady state in finite time which is impossible with linear ODEs. Thus the second stage consists in trying to make the marking of pc as close as possible to j and the one of qc as close as possible to $l+1-j$.

It consists in two transition pairs depending whether the index i of the current instruction is greater or smaller than j. The first case is illustrated in figure 8 (the other case is similar).

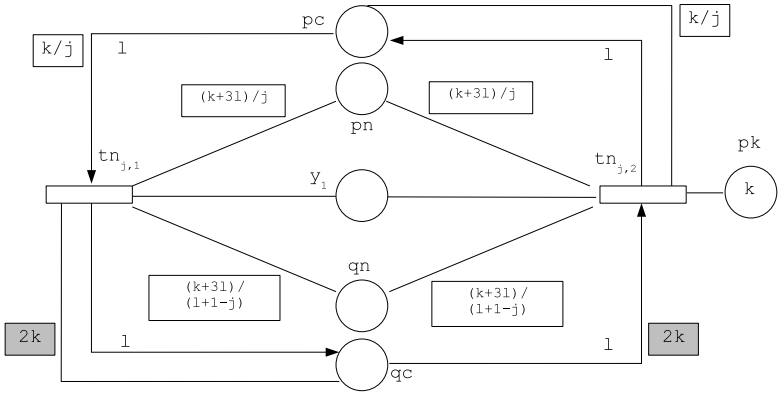

Fig. 8. The second stage

The transition pair $tn_{j,1}, tn_{j,2}$ is activated if both $\mathbf{m}(pn) = j$, $\mathbf{m}(qn) = l+1-j$ and $\mathbf{m}(pc) > j$. If the rate of transition $tn_{j,1}$ was controlled during the whole stage by y_1, pc would loose l tokens. But this means that at the beginning of the stage $\mathbf{m}(pc) > j$ and at the end $\mathbf{m}(pc) \leq j$ which is impossible since $\mathbf{m}(pc)$ must be greater than j in order to trigger the transition pair and thus cannot reach the value j (see our previous remark on linear differential equations). Thus during the second stage *pn must control* the rate of this transition. Since $\mathbf{m}(y_1) \leq k+1$, this means that, at the end of the stage, $(k/j)\mathbf{m}(pc) \leq k+1$ which implies $j \leq m(pc) \leq j+j/k \leq j+l/k$ and consequently $l+1-j-l/k \leq m(qc) \leq l+1-j$ as required by the simulation. The case $i < j$ leads, at the end of the second stage, to $j-l/k \leq m(pc) \leq j$ and consequently $l+1-j \leq m(qc) \leq l+1-j-l/k$.

Theorem 1 is a consequence of our different constructions. The dimension of the associated of ODE is obtained by recalling that the ODE of the clock subnet is 2 and that the following invariants hold in the operating subnet: $\mathbf{m}(pn) + \mathbf{m}(qn) = \mathbf{m}(pc) + \mathbf{m}(qc) = l + 1$. The proof of robustness is omitted.

Theorem 1. *Given a two counter machine $\mathcal{M}$, one can build a TDPN $\mathcal{D}$, with a constant number of places, whose size is linear w.r.t. the machine, whose associated ODE has dimension* 6 *and such that $\mathcal{D}$ robustly simulates $\mathcal{M}$.*

5 A bounded simulation

In this paragraph, we modify our simulation in order to obtain a bounded net. The previous net is unbounded due to the way we model the counters. So we change their management. First we will build a new lazy machine $\mathcal{M}'$ from the original one $\mathcal{M}$. We multiply by 4 the number of intructions, i.e. we create three instructions $\mathtt{A_i}$: goto $\mathtt{B_i}$;, $\mathtt{B_i}$: goto $\mathtt{C_i}$; and $\mathtt{A_i}$: goto $\mathtt{I_i}$; per instruction $\mathtt{I_i}$. Then we modify every label in the original instructions by substituting $\mathtt{A_i}$ to $\mathtt{I_i}$. $\mathcal{M}$ and $\mathcal{M}'$ are equivalent from a simulation point of view since they perform the same computation except that four instructions of $\mathcal{M}'$ do what does a single of instruction of $\mathcal{M}$. We then duplicate the operating subnet ($\mathcal{D}'^{(1)}$ and $\mathcal{D}'^{(2)}$) to simulate $\mathcal{M}$ *via* $\mathcal{M}'$. The only difference between the subnets is that $\mathcal{D}'^{(1)}$ simulates $\mathtt{I_i}$ of $\mathcal{M}$ by simulating $\mathtt{I_i}$ of $\mathcal{M}'$ whereas $\mathcal{N}'^{(2)}$ simulates $\mathtt{I_i}$ of $\mathcal{M}$ by simulating $\mathtt{B_i}$ of $\mathcal{M}'$. This yields a scheduling where one simulation preceedes the other one by two instructions. Superscripts $^{(1)}$ and $^{(2)}$ distinguish between places of the two subnets.

In each subnet, we add three places $pinc_u^{(s)}, pdec_u^{(s)}, d_u^{(s)}$ $(s = 1, 2)$ in addition to $c_u^{(s)}$, to manage the counter $\mathtt{cpt_u}$. The marking of $pinc_u^{(s)}$ (resp. $pdec_u^{(s)}$) is equal to $k+2$ when the current instruction is an incrementation (resp. decrementation) of $\mathtt{cpt_u}$ and $k-1$ otherwise. The transition pairs managing the marking of these places are straightforward to design. Then we change our counter updates in such a way that when a counter $\mathtt{cpt_u}$ is equal to v then $\mathbf{m}(c_u^{(s)}) = (k+2) - 2(1/2)^v$ and $\mathbf{m}(d_u^{(s)}) = (k-1) + 2(1/2)^v$. Hence if $\mathtt{cpt_u} = \mathtt{0}$ then $\mathbf{m}(c_u^{(s)}) = k$ and if $\mathtt{cpt_u} \geq \mathtt{1}$ then $\mathbf{m}(c_u^{(s)}) \geq k+1$ as required for the correctness of the simulation of the conditional jump.

It remains to describe the handling of incrementations and decrementations. Note that the main difficulty is that the decrement (or increment) depends on the current value of the simulated counter. If $\mathtt{cpt_u} = \mathtt{v}$ and we increment the counter, then we must produce (resp. consume) $(1/2)^v$ tokens in $c_u^{(s)}$ (resp. in $d_u^{(s)}$). If $\mathtt{cpt_u} = \mathtt{v}$ and we decrement the counter, then we must consume (resp. produce) $(1/2)^{v+1}$ tokens in $c_u^{(s)}$ (resp. in $d_u^{(s)}$). Let us observe the evolution of marking $pinc_u^{(s)}$ in the simulation (see figure 9) when one simulates the execution of instruction $\mathtt{I_i}$, an incrementation of $\mathtt{cpt_u}$. In the first part of the cycle related to $\mathtt{C_i}$ it raises from $k-1$ to $k+2$, then holds this value during the second part and decreases in the first part of the next cycle to $k-1$. The increasing and the decreasing are not linear but they are *symmetrical*. Thus the mottled area of figure 9 is proportional to the difference between c_u and $k+2$ (equal to the difference between d_u and $k-1$). We emphasize the fact that neither the upper part of this area nor its lower part are proportional to the difference.

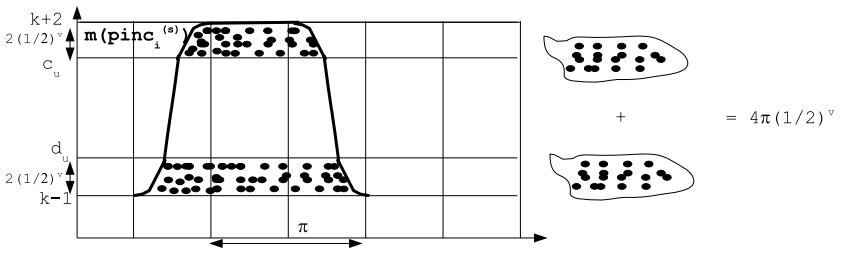

Fig. 9. A way to obtain a "proportional" increment

The subnet of figure 10 exploits this feature to simulate an incrementation of the counter. Let us detail the behaviour of this subnet. This subnet has two transition pairs $inc_{u,1}, inc_{u,2}$ and $inc_{u,3}, inc_{u,4}$. The firing rate of $inc_{u,1}$ is $(1/(4\pi))min(\mathbf{m}(c_u^{(2)}), \mathbf{m}(pinc_u^{(1)}))$ (note again that due to their speed control equal to 1, places $c_u^{(1)}$ and $c_u^{(1)}$ do not determine this rate). The rate of $inc_{u,2}$ is $(1/(4\pi))\mathbf{m}(pinc_u^{(1)})$. Thus they have different speed as long as $\mathbf{m}(pinc_u^{(1)}) > \mathbf{m}(c_u^{(2)})$. So their effect corresponds to the upper part of the mottled area of figure 9. The rate of $inc_{u,3}$ is $(1/(4\pi))min(\mathbf{m}(d_u^{(2)}), \mathbf{m}(pinc_u^{(1)}))$. The rate of $inc_{u,4}$ is $(k-1)/(4\pi)$. So their effect corresponds to the lower part of the mottled area of figure 9. The scaling factor $1/4\pi$ ensures that $1/2(\mathbf{m}(d_u^{(2)}) - (k-1))$ have been transfered from $\mathbf{m}(d_u^{(1)})$ to $\mathbf{m}(c_u^{(1)})$. The subnet managing $\mathbf{m}(d_u^{(2)})$ and $\mathbf{m}(c_u^{(2)})$ behaves similarly except that since $\mathbf{m}(c_u^{(1)})$ and $\mathbf{m}(d_u^{(1)})$ have their new value the scaling factor must be doubled in order to transfer the same amount of tokens from $\mathbf{m}(d_u^{(2)})$ to $\mathbf{m}(c_u^{(2)})$. The decrementation simulation follows a similar pattern.

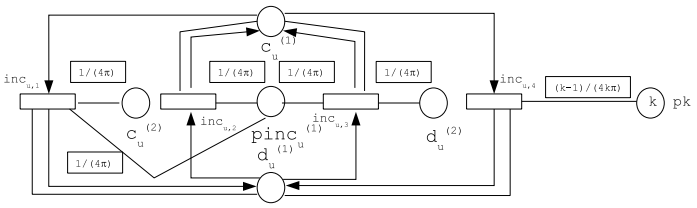

Fig. 10. Incrementing a counter (first stage)

Due to the scheduling, the places of $\mathcal{D}'^{(2)}$ modelling the counter $\mathtt{cpt_u}$ are not modified during the simulation of an instruction in $\mathcal{D}'^{(1)}$ and *vice versa*. Indeed the instruction simulations are translated and surrounded by "no-op" instructions which do not modify the counters. The correctness of this simulation yields the following theorem. The dimension of the ODE is obtained by observing that $\mathbf{m}(c_u^{(s)}) + \mathbf{m}(d_u^{(s)}) = cst$.

Theorem 2. *Given a two counter machine $\mathcal{M}$, one can build a bounded TDPN $\mathcal{D}$ with a constant number of places, whose size is linear w.r.t. the machine and whose associated ODE has dimension* 14 *such that $\mathcal{D}$ simulates $\mathcal{M}$.*

6 Undecidability results

In this section, we apply the simulation results in order to obtain undecidability results. Proofs are omitted. Note that we cannot state the undecidability of the marking reachability problem since in the simulation, places *pc* and *qc* are not required to take precise values. However the steady-state analysis, a kind of ultimate reachability, is undecidable.

Proposition 1 (Coverability and reachability). *Let $\mathcal{D}$ be a (resp. bounded) TDPN whose associated ODE has dimension at least* 6 *(resp.* 14*), $\mathbf{m}_0, \mathbf{m}_1$ be markings, p be a place and $k \in \mathbb{N}$ then:*

– the problem whether there is a τ such that the trajectory starting at $\mathbf{m}_0$ fulfills $\mathbf{m}(\tau)(p) = k$ is undecidable.

– The problem whether there is a τ such that the trajectory starting at $\mathbf{m}_0$ fulfills $\mathbf{m}(\tau)(p) \geq k$ is undecidable.

– The problem whether there is a τ such that the trajectory starting at $\mathbf{m}_0$ fulfills $\mathbf{m}(\tau) \geq \mathbf{m}_1$ is undecidable.

Proposition 2 (Steady-state analysis). *Let $\mathcal{D}$ be a (resp. bounded) TDPN whose associated ODE has dimension at least* 8 *(resp.* 16*), $\mathbf{m}_0$ be a marking. Then the problem whether the trajectory $\mathbf{m}$ starting at $\mathbf{m}_0$ is such that $\lim_{\tau\to\infty} \mathbf{m}(\tau)$ exists, is undecidable.*

7 Conclusion

In this work, we have introduced TDPNs, and we have designed two simulations of counter machines in order to fulfill robustness and boundedness requirements. These simulations are performed by a net with a constant number of places, i.e. whose dimension of associated ODE is constant. We have also proved that marking coverability, submarking reachability and the existence of a steady-state are undecidable. We conjecture that the marking reachability is undecidable and we will try to prove it. In order to obtain decidability results, we also plane to introduce subclasses of TDPNs where the restrictions will be related to both the structure of the net and the associated ODE.

References

1. Rajeev Alur and David Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. E. Asarin, O. Maler, and A. Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138-1:35–65, 1995.
3. Eugene Asarin and Oded Maler. Achilles and the tortoise climbing up the arithmetical hierarchy. *Journal of Computer and System Sciences*, 57(3):389–398, 1998.
4. F. Balduzzi, A. Di Febbraro, A. Giua, and C. Seatzu. Decidability results in first-order hybrid petri nets. *Discrete Event Dynamic Systems*, 11(1 and 2):41–58, 2001.
5. Olivier Bournez. Some bounds on the computational power of piecewise constant derivative systems. *Theory of Computing Systems*, 32(1):35–67, 1999.
6. Michael S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138(1):67–100, 1995.
7. R. David and H. Alla. *Discrete, Continuous, and Hybrid Petri Nets*. Springer-Verlag, 2004.
8. Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.
9. J.E. Hopcroft and J.D. Ullman. *Formal languages and their relation to automata*. Addison-Wesley, 1969.
10. Gerardo Lafferriere, George J. Pappas, and Sergio Yovine. Symbolic reachability computation for families of linear vector fields. *Journal of Symbolic Computation*, 32(3):231–253, 2001.
11. Venkatesh Mysore and Amir Pnueli. Refining the undecidability frontier of hybrid automata. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science 25th International Conference*, volume 3821 of *LNCS*, pages 261–272, Hyderabad, India, 2005. Springer.
12. Hava T. Siegelmann and Eduardo D. Sontag. Analog computation via neural networks. *Theoretical Computer Science*, 131(2):331–360, 1994.
13. M. Silva and L. Recalde. Petri nets and integrality relaxations: A view of continuous Petri nets. *IEEE Trans. on Systems, Man, and Cybernetics*, 32(4):314–327, 2002.