Dynamically-Driven Timed Automaton Abstractions for Proving Liveness of Continuous Systems*

Rebekah Carter and Eva M. Navarro-López

School of Computer Science,
The University of Manchester, UK
{carterr,eva.navarro}@cs.man.ac.uk

Abstract. We look at the problem of proving inevitability of continuous dynamical systems. An inevitability property says that a region of the state space will eventually be reached: this is a type of liveness property from the computer science viewpoint, and is related to attractivity of sets in dynamical systems. We consider a method of Maler and Batt to make an abstraction of a continuous dynamical system to a timed automaton, and show that a potentially infinite number of splits will be made if the splitting of the state space is made arbitrarily. To solve this problem, we define a method which creates a finite-sized timed automaton abstraction for a class of linear dynamical systems, and show that this timed abstraction proves inevitability.

Keywords: Continuous-time systems, Abstraction, Automated verification, Liveness properties, Timed automata.

1 Introduction

Dynamical systems can have very complex behavior patterns, and over the years the mathematics and control communities have developed a lot of theory to analyze their behavior. In particular, Lyapunov stability theory [8] is important for proving properties about continuous dynamical systems. One of the elements of stability theory is the notion of *attractivity*, which says that trajectories of a system tend toward a set or a point as time goes to infinity.

From the computer science viewpoint, the related notion to attractivity ensures that a set is reached in finite time, and is expressed as the property "we eventually reach some set in the state space". This is an *inevitability* property, a type of liveness property. In this paper we address the problem of proving such liveness properties, as there are only a few methods currently defined [5,9,14].

In the computer science community, particularly the hybrid systems community, a lot of effort has gone into trying to verify dynamical systems, mostly

* This paper has been made under the framework of the EPSRC-funded project "DYVERSE: A New Kind of Control for Hybrid Systems" (EP/I001689/1). The second author is also grateful for the support of the RCUK (EP/E50048/1).

M. Jurdziński and D. Ničković (Eds.): FORMATS 2012, LNCS 7595, pp. 59–74, 2012.

through using model checking methods [3,10,13,16]. One class of such methods involves *abstraction* of the system to a discrete system, usually a finite-state automaton, in order to be able to use discrete model checkers on the abstraction to prove properties about the original system (for example, see [3]). The properties that can be proved are typically safety properties, which say that something is always true. In fact, proving liveness properties in dynamical systems, where we want something to eventually be true, is not possible by means of a purely discrete abstraction as there is no guarantee of progress of time.

In order to prove inevitability properties, we need to transfer some information about the times at which events occur to the discrete system, and so we turn to using *timed automata* (TA) for our abstraction. Reachability of TA is decidable, and so they are a sensible candidate for an abstraction. There are various provers available for proving properties of TA, the most widely used one being Uppaal [2].

There are a few methods which propose how to abstract a dynamical system to a timed automaton, including [4,9,11,14,15]. We consider the method of Maler and Batt [9], who did not specify how the state space of a system should be split to create the TA, but stated that the accuracy of the model could be improved indefinitely by increasing the number of splits. However, in most systems arbitrary splitting will not be very good at capturing the dynamics of the system, and we may well be required to make a very large number of splits to verify the system.

In this work, we advocate *dynamically-driven* splitting of the state space, using known properties of the flow of the system to decide how to make the splits. We identify a terminating splitting method for a class of upper triangular linear systems which ensures that the resulting timed automaton will always prove the inevitability property. Our method is most closely related to that of [14], which abstracts continuous systems to timed automata using the idea of the method of [9], but the authors in that work use Lyapunov functions to define the slices considered, whereas we use the original idea of constant variable slices.

2 Overview of Systems and Method Being Considered

In this section, we consider all autonomous n-dimensional continuous dynamical systems, of the form

$$\dot{x} = f(x), \tag{1}$$

with $x = [x_1, \ldots, x_n]^T \in \mathbb{R}^n$ and $f(x) = [f_1(x), \ldots, f_n(x)]^T$, with $f : \mathbb{R}^n \to \mathbb{R}^n$ smooth. The state space of the system is assumed to be a finite box, which is defined by the limits $x \in [s_1^-, s_1^+) \times \ldots \times [s_n^-, s_n^+) = S$.

The method we consider is from [9]. It is an approximation method defined to abstract a continuous system to a TA, and is based on minimum and maximum velocities of a system defining bounds on the time taken to cross a certain distance in the system. The resulting TA is an over-approximation of the system, in the sense that every trajectory in the system is matched in time and space by one in the abstraction, but additional trajectories may be allowed in the abstraction (see [9] for more discussion of this).

The basic idea of [9] is to split the state space into slices by making splits along lines of the form $x_i = C$, where C is constant. These slices also define hyper-rectangles (which we refer to as *boxes*) of the space by the intersection of a slice in each dimension (see Fig. 1 for the 2-dimensional (2-D) case with $x \in \mathbb{R}^2$).

A slice is a part of the state space, restricted only in one dimension. In this work we allow slices to be of differing widths, defined by a vector of split points $Verts_i$ in each dimension i.[1] Let v_i be an index which indicates which slice we are considering in dimension i, and then the slice is given by

$$X_{i,v_i} = \left[s_1^-, s_1^+\right) \times \ldots \times \left[Verts_i(v_i), Verts_i(v_i+1)\right) \times \ldots \times \left[s_n^-, s_n^+\right) \subseteq S. \quad (2)$$

Slices are right-open so that they do not intersect, and so the set of slices for each i will form a partition of the state space S (this is why S was defined as being right-open). Similarly, let the index for a box be defined by $v = [v_1, \ldots, v_n]$, then the box is defined as $X_v = \bigcap_{i=1}^n X_{i,v_i} \subseteq S$. The set of all boxes in the partitioning of S is denoted by V. To find possible crossings between these boxes we consider the sign of the velocity on each face between adjacent boxes.

Definition 1 (Automaton Abstraction [9]). *The automaton $A = (V, \delta)$ is an abstraction of the system if δ consists of the pairs of boxes with indices $v = [v_1, \ldots, v_i, \ldots, v_n]$ and $v' = [v_1, \ldots, v_i + 1, \ldots, v_n]$ where f_i can take a positive value on the face between them, or pairs of v with $v' = [v_1, \ldots, v_i - 1, \ldots, v_n]$ where f_i can take a negative value on the face between them.* ■

To make the timed automaton abstraction, we define clocks to keep track of the times at which crossings are made in each dimension. Within any box X_v, let d_i be the width of this box in dimension i, then the maximal time that it can take to leave this box is over-approximated by the *box time*, defined as

$$\overline{t}_v = \min_{1 \le i \le n} \left(\frac{d_i}{\min(|f_i|) \text{ in box } X_v} \right). \quad (3)$$

If $\min(|f_i|) = 0$ in box X_v, then we define $\overline{t}_v = \infty$.

The times spent in slices of the space can also be limited, both above and below, in the positive and negative directions. Let d_i be the size of the dimension i slice we are considering, and let $\underline{f_i}$ be the minimum velocity in this slice, and $\overline{f_i}$ be the maximum velocity. Then the minimum ($\underline{t}$) and maximum ($\overline{t}$) times that can be spent in this slice in the positive ($+$) and negative ($-$) directions are given in Table 1 (*slice times*).

If we enter a slice from the lower face in dimension i, then the minimum time we can take to leave by the opposite face is $\underline{t}^+$, and the maximum time to leave by the opposite face is $\overline{t}^+$, and similarly for entering from the upper face with $\underline{t}^-$ and $\overline{t}^-$. We use these values to bound timed automaton clocks within slices of the space: there are two clocks per dimension, z_i^+ which bounds positive direction movements using $\underline{t}^+$ and $\overline{t}^+$, and z_i^- which bounds the negative

[1] Note that the first and last elements of $Verts_i$ are s_i^- and s_i^+ respectively.

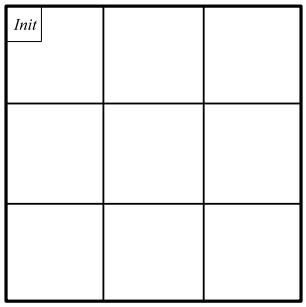


Fig. 1. Partition into boxes (rectangles in the 2-D case).

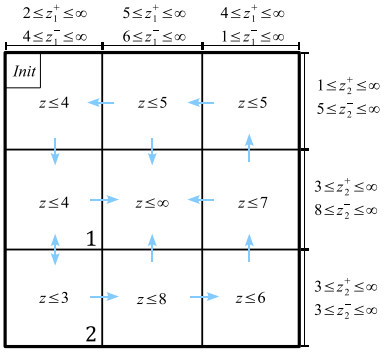


Fig. 2. Calculate (1) which transitions exist (denoted by arrows), and (2) the min/max times on the clocks $z, z_1^+, z_1^- \ldots$.

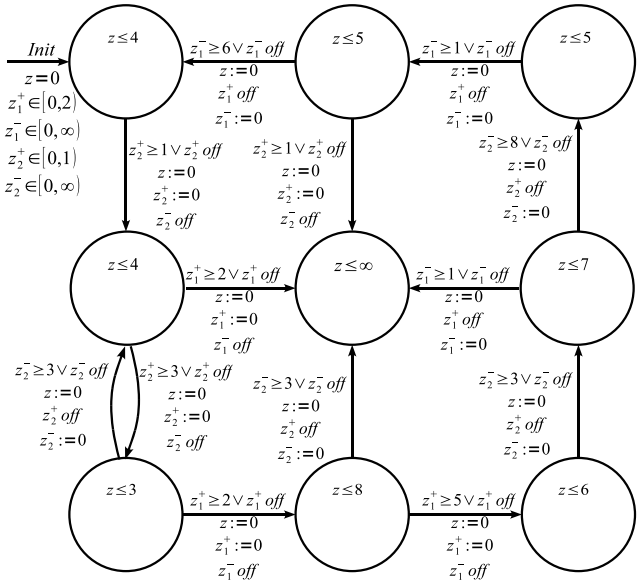


Fig. 3. The TA resulting from the method of [9] applied to the dynamical system. In this case, no slice clock has an upper limit, so they do not appear inside locations.

Table 1. Minimum and maximum times that can be spent in slice i in the positive and negative directions (respectively)

	$\underline{t}^+$	$\overline{t}^+$	$\underline{t}^-$	$\overline{t}^-$
$0 < \underline{f_i} < \overline{f_i}$	$d_i/\overline{f_i}$	$d_i/\underline{f_i}$	∞	∞
$\underline{f_i} < \overline{f_i} < 0$	∞	∞	$-d_i/\underline{f_i}$	$-d_i/\overline{f_i}$
$\underline{f_i} < 0 < \overline{f_i}$	$d_i/\overline{f_i}$	∞	$-d_i/\underline{f_i}$	∞

direction movements. We also use a box clock z in the TA to satisfy the conditions on how long we can stay in each box, using (3).

Figures 1–3 illustrate the abstraction process for an arbitrary 2-D system. This overview of the method involved should be enough to understand the content of this paper, but for further details see [9].

3 Problems for General Continuous Systems

There are various reasons why this method does not work well for general continuous systems, some of which we highlight in this section. One very fundamental reason can be that the trajectories of the system do not fit well with splitting based on crossing constant variable lines. A particular example of this is for 2-D linear systems with complex eigenvalues, where the trajectories of the system are spirals. Even when the real parts of the eigenvalues are negative and the trajectories go inward, the timed automaton can allow flow round the edge of the split space without forcing us to move closer to the center of the spiral (see for example Fig. 3, where the center location should be reached, but the TA allows flow through the eight locations round the edge indefinitely).

Part of the problem with this abstraction for general continuous dynamical systems is the fact that the discrete automaton abstraction can allow pairs of automaton locations where the trajectories can go both ways across the shared face (see locations marked 1 and 2 in Fig. 2). The timed automaton does not restrict when these transitions can be taken, as box times are not directional and slice times only limit the time to reach the *opposite* face, so these pairs of locations introduce Zeno behaviour into the abstraction [7]. This kind of behaviour prevents every trace of the abstraction from reaching the desired final location, so the inevitability property cannot be proved.

Another potential problem with this method is that it does not guarantee to calculate *finite* values of the box times $\overline{t}_v$, due to the fact that if a zero velocity occurs in the box v in every dimension, then $d_i / \min(|f_i|) = \infty$ in every dimension. If there is one box in the abstraction which has an infinite box time, then any trace which reaches this box will never be forced to leave this box even if the actual trajectories of the system would all leave it in finite time.

Some of the above problems can be remedied by choosing an appropriate method for splitting the state space. The method of [9] does not specify how we should choose the splitting, but just tells us the properties of the resulting TA when a choice has been made. As we see it, there are two ways to do such a splitting: either we make the splitting arbitrarily (systematically but not based on the system's dynamics) and rely on refinement to eventually capture enough information about a system, or we can use properties of the dynamics to choose where to split the system. The pros and cons of these are discussed below.

Arbitrary splitting. This approach does not rely on knowledge about the structure of the dynamical system, and so it can be used for complex systems no matter where the complexity comes from. The TA created by the abstraction method of Maler and Batt does approach the actual dynamics

(theoretically) as more and more splits are made (see [9]). However, due to the two issues (1) that transitions both ways between pairs of automaton locations can exist and (2) that there can be an infinite over-approximation of the time it takes to get across a box, the number of splits required is often very large, if not infinite, and so we obtain a huge number of locations in the timed automaton.

Using system properties for splitting. Here we use the dynamics of the system we are looking at to automatically split the state space in a way which removes or reduces some of the problems associated with arbitrary splitting. In specific systems, it should be possible to make a splitting which can be proved to satisfy desirable properties, for instance that the liveness property is automatically satisfied. Even in systems where we cannot prove the liveness property immediately, it may be possible to at least have a much better starting point for refining the abstraction. The main problems with this idea are that a splitting method will only work for a certain class of systems, and that there are no automatic splitting methods in existence already; methods need to be designed for many different types of systems.

Given the considerations above, in this paper we wish to start the process of finding dynamically-driven automatic methods to create splittings. We will work from a theoretical basis to show that certain types of linear systems have a splitting which proves inevitability by the TA abstraction, with the idea that future work can extend these methods to be useful for more general systems (nonlinear, piecewise continuous, or hybrid).

4 Inevitability for Upper Triangular Linear Systems

In this section we consider the class of upper triangular linear dynamical systems, as a first step to using this method for general systems. We assume that a splitting has been created which removes some of the problems highlighted in the previous section. We then show that the timed automaton abstraction created from this splitting proves an inevitability property. In Sect. 5 we will identify a method for a sub-class of these systems which creates a splitting with the desired properties. These two parts together will prove that the sub-class considered in Sect. 5 can always have an inevitability-proving TA abstraction.

We consider upper triangular linear systems, with no input vector:

$$\dot{x} = Ax = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ 0 & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{n,n} \end{pmatrix} x. \tag{4}$$

We assume that the eigenvalues (the diagonal entries of A) are negative. This type of system has a unique equilibrium point at zero,[2] and the negative eigenvalues mean that this point is stable and attractive, which is asymptotic stability.

[2] The equilibrium point x_e satisfies the equation $\dot{x} = 0$, that is $Ax_e = 0$. Invertibility of A means the only equilibrium point is at $x_e = 0$.

This means that some non-empty region L containing the equilibrium point must be reached within a finite time. From a dynamical systems perspective we know this property is true, but we want to prove the same property computationally — the long-term goal of this work is to prove inevitability properties of dynamical systems which cannot be proved by dynamical theory (due to complexity). In linear temporal logic (LTL), this inevitability property is written $\Diamond(x \in L)$, where $\Diamond$ is the temporal logic operator meaning "eventually".[3] In this paper, we will say the system is *live* if this LTL condition is satisfied for all trajectories.

We note that the only real restriction on the class of linear systems is the existence of real negative eigenvalues: for any linear system of the form $\dot{x} = Bx + b$ with B having real negative eigenvalues and b any real vector, it can be transformed to $\dot{x} = Bx$ without loss of generality, and can then be transformed to an equivalent upper triangular system by making the Schur decomposition.

4.1 The Assumptions

There are four assumptions we make about the splitting, each with various levels of difficulty in achieving them. In Sect. 5 we will define a method which can satisfy these assumptions for a subclass of the systems under study, which shows these are not unreasonable assumptions to make.

The first assumption is about where the equilibrium point occurs in relation to the splitting, and is here to give us one (and only one) box that we are interested in reaching for the inevitability property.

Assumption 1. There is exactly one box L in the splitting which contains the equilibrium point,[4] and the equilibrium is not on the boundary of L. We will call L the *live box*.

The second assumption specifies that there are a *finite* number of boxes in the abstraction, which is necessary for a useful abstraction.

Assumption 2. The automaton abstraction (Def. 1) of the system has a finite number of discrete states.

The next assumption has to do with how transitions are allowed in the abstracted system. We do not want it to be possible to keep transitioning between a pair of discrete states in the timed automaton, as discussed in Sect. 3.

Assumption 3. The continuous flow across any box face only occurs in one direction. That is, if the box face is $x_j = C$ with the other x_i's within the box limits, then the velocity $\dot{x}_j$ across this face will either be always $\dot{x}_j \geq 0$ or always $\dot{x}_j \leq 0$.

The fourth assumption is related to the timing constraints in the TA, ensuring we leave every box on a trace within a finite time.

Assumption 4. In each box v, except the live box L, the box time $\overline{t}_v$ is finite.

[3] See [12] for more information about LTL.

[4] This is always true for linear systems, where there is only one equilibrium point.

4.2 The Theorems

We wish to show that every system of form (4) is proved to satisfy the inevitability property by any splitting satisfying the assumptions. To do this we will prove that the discrete automaton abstraction (Def. 1) of the continuous system only has finite traces, and that the only location with no outgoing edges (hence the only possible final location) corresponds to the live box L. We then use the assumption of finite box time to ensure that L is reached in finite time.

Theorem 1. *Assume we have an n-dimensional system of the form of (4) with negative entries on the diagonal, and an automaton abstraction created by the method of [9] satisfying Assumptions 1–4. Then the automaton abstraction of the continuous system only has finite traces.*

Proof. Assume (for a contradiction) that we can find a trace of infinite length in the automaton. As the automaton abstraction must have a finite number of locations by Assumption 2, any infinite trace must go through at least one discrete location infinitely often. Hence, for any move in the infinite trace that is made in the k-th dimension in the positive direction, we must be able to find a corresponding move in the k-th dimension in the negative direction, and vice-versa. We now prove, by induction, that this requirement is not satisfied in the system.

Base Case. In the n-th dimension, the dynamics of the system is $\dot{x}_n = a_{n,n}x_n$ with $a_{n,n} < 0$. Hence, across any slice boundary in the n-th dimension, if $x_n > 0$ then $\dot{x}_n < 0$, and if $x_n < 0$ then $\dot{x}_n > 0$. Therefore crossing any n-th dimensional slice boundary can only be done in one direction, and so cannot be reversed as is necessary for this type of crossing to be present in the infinite trace. Hence n-th dimensional crossings are not involved in the infinite trace.

Inductive Part. Assume that $n-k+1, \ldots, n$ dimensional crossings are not involved in the infinite trace, and so the $x_{n-k+1}, \ldots, x_n$ variables are within one slice each for this trace. The $\dot{x}_{n-k}$ equation only depends on $x_{n-k}, \ldots, x_n$. Assumption 3 does not allow the sign of $\dot{x}_{n-k}$ to change over the course of a box face, but since $x_{n-k+1}, \ldots, x_n$ are within one slice each, any constant value of x_{n-k} makes the whole slice face the same sign as one of the component box face. Hence $\dot{x}_{n-k}$ is either completely non-negative or completely non-positive for a constant value of x_{n-k}. This means it is not possible to reverse a crossing of a constant x_{n-k} surface in the infinite trace, and so $(n-k)$-th dimensional crossings cannot be present in the infinite trace.

Hence, by base case and inductive part, the infinite trace through the automaton abstraction of the system cannot involve transitions in any dimension, which contradicts the existence of an infinite trace. So the automaton abstraction of the n-dimensional system under the assumptions only has finite traces. □

Theorem 2. *Assume we have a system of form (4) with negative diagonal entries, and an automaton abstraction created by the method of [9] satisfying Assumptions 1–4. Then the only location with no outgoing edges in the automaton abstraction corresponds to the box L containing the equilibrium point $x_e = 0$.*

Proof. Assume there is a location of the automaton abstraction which has no transitions out of it. Then all the existing transitions are inwards. It is not possible for whole box sides to have zero flow across them, due to not allowing splitting at $x_i = 0$ surfaces (by Ass. 1). Therefore the two opposite sides of a box in dimension i have opposite (inwards) flows, so by continuity of the flow, the zero surface $\dot{x_i} = 0$ must pass through this box. As this is the case for all dimensions, then the linearity of the system means the equilibrium point x_e must be in the corresponding box of the partition of the state space. By Assumption 1 the equilibrium point is wholly within one box L, and so the only location with no outgoing edges is box L. (Note that box L is an invariant set.) □

Theorem 3. *Assume we have a system of form (4) with negative diagonal entries, and an automaton abstraction created by the method of [9] satisfying Assumptions 1–4. Then all trajectories of the system from all initial boxes get to the live box L within finite time.*

Proof. Theorems 1 and 2 together imply that all traces of the TA lead to the box L containing the equilibrium point in a finite number of steps. Assumption 4 says that each cube on this route is left within a finite time, and so the total time for any TA trace to reach the box L is finite. By the over-approximation of number of trajectories, all trajectories of the original system reach the box L within finite time (and stay there as L is invariant). □

5 Dynamically-Driven Splitting Method

In this section we define a method to split the state space of a continuous system such that the TA abstraction created from it satisfies the four assumptions for a subset of the upper triangular linear systems. Together with the previous section this proves that we can automatically create a TA abstraction of such systems which proves inevitability. The method is shown to terminate for this particular class of systems, and the number of locations in the resulting abstraction is analyzed.

5.1 The Class of Systems Considered

The class we now consider are a subclass of upper triangular linear systems of form (4) with two conditions on them:

- All of the entries on the main diagonal are negative.
- In each row, a maximum of one other non-zero entry is allowed.

These conditions mean that x_i's differential equation is in one of two forms, for each $i = 1, \ldots, n$, either

$$\dot{x_i} = a_{i,i}x_i \qquad \text{for } a_{i,i} \text{ negative, or} \tag{5}$$

$$\dot{x_i} = a_{i,i}x_i + a_{i,j}x_j \quad \text{for } a_{i,i} \text{ negative}, a_{i,j} \neq 0 \text{ and } i < j \leq n. \tag{6}$$

This class of systems does have restrictions, but allows various special classes of systems. In particular, all 2-D systems with negative real eigenvalues can be transformed by the Schur decomposition to an equivalent dynamical system of this form. The verification results for such 2-D systems (with state space limits and live box limits suitably transformed) will prove the desired properties about the original system. Higher dimensional systems which can be of this form include systems modelling chains of behaviour, where each x_i's evolution only depends on itself and the element next in the chain. For example, simplified models of biological cascades can be expressed in this form [6]. There are also some piecewise linear models of such cascades, where each element depends only on itself and the element before it: these can be modelled in the form (5) or (6) for each x_i and each region of dynamics. The method proposed is easily extendible to piecewise systems of this form provided the divides between different dynamics in the system occur at constant values of x_i (this is the case in [1], for example).

If row i of the matrix has the form (5), this means that we always have the same value of $\dot{x}_i$ for any constant value of x_i. So Assumption 3 is always true on all box faces in dimension i. On the other hand, if row i has the second form (6), this means that when we try to separate $\dot{x}_i > 0$ from $\dot{x}_i < 0$ on a particular face $x_i = C$, we get a constant value of $x_j = -\frac{a_{i,i}C}{a_{i,j}}$ where the $\dot{x}_i = 0$ surface occurs through this face. Therefore, we can choose to split at this point in the j-th dimension to make sure of separating the positive and negative velocities in the i-th dimension. This ease of selecting where to split is not available to us for general upper triangular systems, and is what makes this special class better for automatic splitting.

5.2 The Splitting Method

The method we propose for splitting systems of this new form is described in Algorithm 1. The idea is to originally split based on the live box boundaries which creates a box satisfying Assumption 1 (Step 1), and then to split based on where a $\dot{x}_j = 0$ surface crosses any constant x_j surface (Step 2). After these two steps the resulting TA abstraction satisfies Assumptions 1 and 3.

Step 3 of the algorithm then finds and divides the boxes where infinite time has been found, whilst still keeping these other two assumptions intact, in order to satisfy Assumption 4. It works by finding the intervals of existence of $\dot{x}_i = 0$ surfaces, and splitting between two intervals calculated in dimension i if they do not intersect. This method is demonstrated on a 2-D example in Fig. 4.

We will now show that this algorithm terminates and quantify the size of the resulting abstraction, then we will give an overview of the proof of why the abstraction satisfies Assumptions 1–4.

Termination and Abstraction Size. Firstly we show that FollowSplits terminates. FollowSplits consists of two for loops, the first clearly has a finite number of executions $(n-1)$. The second iterates over a finite number of elements of the list *List* for $i = 1$, and each iteration removes an element from the current

Algorithm 1. Automatic splitting algorithm

Input: Linear dynamical systems with each x_i's dynamics of the form of (5) or (6), with state space $[s_1^-, s_1^+) \times \ldots \times [s_n^-, s_n^+)$, and live box $L = [l_1^-, l_1^+) \times \ldots \times [l_n^-, l_n^+)$.

Output: A splitting of the system such that the TA abstraction proves inevitability of the live box.

add splits at $x_i = l_i^-$ and $x_i = l_i^+$ for each i ▷ Step 1

call FollowSplits ▷ Step 2

Calculate box times ▷ Step 3
$B \leftarrow$ list of boxes with infinite box time (except L)
while B is non-empty **do**
  for $k = 1, \ldots,$ length(B) **do**
    $V \leftarrow$ get vertices of box $B(k)$
    $Z \leftarrow V$ (initialize the region where zero surfaces occur: let Z_i^- be lower bound and Z_i^+ be upper bound in dimension i)
    for $i = n, n-1, \ldots, 1$ **do**
      if $\dot{x}_i$ has an off-diagonal entry **then**
        $j \leftarrow$ position of the off-diagonal entry in row i of A
        $\begin{bmatrix} c_i^- & c_i^+ \\ c_j^- & c_j^+ \end{bmatrix} \leftarrow$ limits of the surface $\dot{x}_i = 0$ in the box
        if $c_j^- > Z_j^+$ **then**
          add split at $x_j = (c_j^- + Z_j^+)/2$ (or nearby if this is $x_j = 0$)
          break loop
        else if $c_j^+ > Z_j^-$ **then**
          add split at $x_j = (c_j^+ + Z_j^-)/2$ (or nearby if this is $x_j = 0$)
          break loop
        else if $c_i^- > Z_i^+$ **then**
          add split at $x_i = (c_i^- + Z_i^+)/2$ (or nearby if this is $x_i = 0$)
          break loop
        else if $c_i^+ > Z_i^-$ **then**
          add split at $x_i = (c_i^+ + Z_i^-)/2$ (or nearby if this is $x_i = 0$)
          break loop
        end if
      else(row i does not have an off-diagonal entry)
        if $Z_i^+ < 0$ **then**
          add split at $Z_i^+/2$
          break loop
        else if $Z_i^- > 0$ **then**
          add split at $Z_i^-/2$
          break loop
        end if
      end if
    end for
  end for
  call FollowSplits
  $B \leftarrow$ new list of boxes with infinite box time (except L)
end while

Algorithm 2. FollowSplits sub-algorithm

Input: Current splitting state of the system (including the newly made splits), with a finite list *List* of newly made splits that need to be followed down the dimensions.
Output: A new splitting with only one direction of flow across the faces of the boxes.

```
1: for i = 1, ..., n − 1 do
2:     f_i = the sublist of List of constant values for x_i (splits in dimension i)
3:     for all k ∈ f_i do
4:         v ← find ẋ_i velocity at vertices of the splitting surface x_i = k
5:         if all(v ≥ 0) or all(v ≤ 0) then
6:             remove x_i = k from List
7:         else
8:             solve ẋ_i = 0 when x_i = k giving x_j = d, for some i < j ≤ n.
9:             add split at x_j = d
10:            add x_j = d to List
11:            remove x_i = k from List
12:        end if
13:    end for
14: end for
```

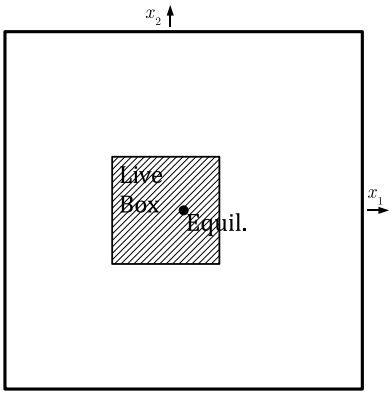
dimension list and possibly adds one to a lower dimension's list. Since this is done a finite number of times, each iteration of the inner loop is only done a finite number of times, and so FollowSplits terminates. The number of splits that can be added by this is dependent on the size of the current splitting in each direction (say this size is c_i for $i = 1 \ldots, n$), and also dependent on the size of the newly made splits list (n_i for $i = 1, \ldots, n$). The worst case is when each dimension causes a split in the dimension immediately after it, as the effect of these splits builds up, so the maximum number of splits (overall) after FollowSplits in each dimension $i = 1, \ldots, n$ is

$$c_i + \sum_{j=1}^{i-1} n_j. \tag{7}$$

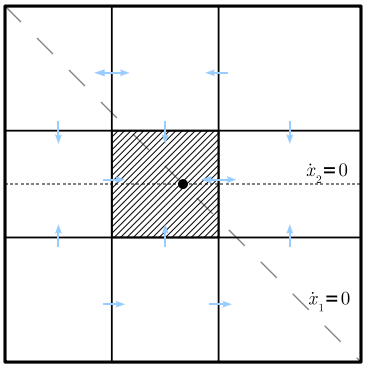
Now consider Algorithm 1 as a whole. Clearly line 1 of Algorithm 1 performs a finite number of splits (2 in each dimension), so terminates. Then, line 2 simply calls FollowSplits on these initial splits, which terminates, with maximum of $2, 4, \ldots, 2n$ splits in each of the $1, 2, \ldots, n$-th directions respectively.

For Step 3, lines 3–39, we must consider the while loop and the two for loops inside it. Both for loops iterate over only a finite number of values, so the combination of the two must terminate (given that all individual lines terminate). So we now need to show that the while loop terminates, which occurs when we have removed the infinite box time on all boxes except the live box.

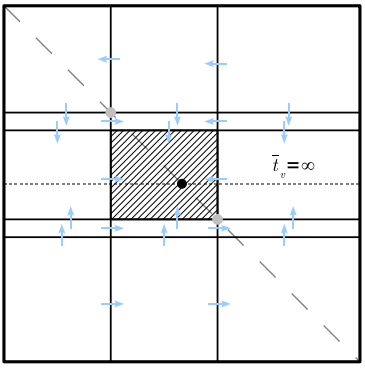
The proof that all infinite-time boxes will be removed is a little more involved, and we need to understand how this algorithm splits the state space of the system. First we will show that all boxes with infinite time at the start of step 3 must touch the live box L (along an edge or at a corner). Assume not, then there



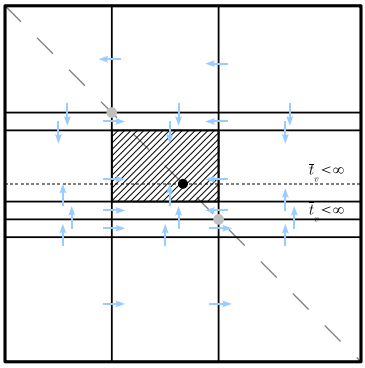
(1) Initial Setup: Given live box L containing the equilibrium point.



(2) After Step 1: Splits made based on the boundaries of the live box.



(3) After Step 2: Light grey dots indicate where $\dot{x}_1 = 0$ on x_1 constant lines. Splits made at these points.



(4) After Step 3: An extra split is added above the lower dot, removing infinite time in the box marked in (3).

Fig. 4. Applying the splitting algorithm to the example $\dot{x}_1 = -x_1 - x_2$ and $\dot{x}_2 = -x_2$, with the given live box defined slightly off centre around the equilibrium point (equil). Arrows indicate the allowed directions of flow across box boundaries, and the shaded region indicates the live box L as it changes size.

is a box B_1 which is $p > 1$ steps away from L in some dimension i. Then, letting j be the other dimension which occurs in the equation $\dot{x}_i = a_{i,i}x_i + a_{i,j}x_j = 0$, we take the projection of this $n-1$-dimensional surface to a line in the x_i-x_j plane. So if the box B_1 has infinite time, this line $\dot{x}_i = 0$ must pass through a corner of the edge nearest to L, by step 2 and linearity. But then, as this line also passes through the equilibrium point in the middle of box L, it must have passed through the middle of an edge $x_i = C$, contradicting that step 2 has been completed. So all infinite-time boxes must be a maximum of one step away from L in any dimension, which means that *every infinite-time box touches* L.

The rest of the proof is too complex to explain in detail, but is an inductive argument on the dimensions. Roughly, assume we have a box B_1 with infinite time which cannot be split on the first run through the while loop of Algorithm 1, and this box is offset from L's slices in the dimensions $i_1, i_2, \ldots, i_k$ (where these are in order smallest to largest). Then, if equation $\dot{x}_{i_k}$ depends on x_{i_k} and x_j, then there must be a box B_2 with infinite box time which shares a dimension j edge with B_1. If B_2 is split by the algorithm, this makes B_1 splittable on the next run through the while loop (after FollowSplits). If B_2 is not split, then it too must have a neighboring box in dimension dependent on the $\dot{x}_j$ equation, and so on. Eventually we reach a splittable box (at dimension n if not before), which makes the previous box splittable after FollowSplits is completed, and so at most $n-1$ runs through the while loop are necessary to remove all infinite time boxes, hence Algorithm 1 terminates.

Algorithm 1 creates at most one split for each infinite-time box, with this split being, in dimension one greater than the largest dimension in which the box is offset from the live box L (in the worst case). Now, in one dimensional systems, there are clearly a maximum of two boxes in dimension 1 which are 'next to' the box L. In two dimensions, there are an extra 6, all with their maximal offsets in dimension 2, and 2 (as before) only offset in dimension 1. By induction we can find that there are $2 \times 3^{i-1}$ boxes with a highest dimensional offset in dimension i. Infinite time is not possible with any offset from the central slice in dimension n, so we only need consider the splits made in dimensions 1 to $n-1$.

As each of these possible infinite-time boxes can create a maximum of one split in the next dimension, this creates $2 \times 3^{i-2}$ splits in each of the $i = 2, \ldots, n$ dimensions. When FollowSplits is done, using the formula in (7) and the initial number of splits $2i$ for each dimension $i = 1, \ldots, n$, we can compute the number of splits as: for $i = 1$, 2 splits, and for $i = 2, \ldots, n$, $2i + 3^{i-1} - 1$ splits. This makes the maximum number of slices 3 for $i = 1$ and $2i + 3^{i-1}$ for $i = 2, \ldots, n$. The total number of boxes is the product of the slices in each dimension, so

$$NumBoxes = 3 \times \prod_{j=2}^{n} (2j + 3^{i-1}). \tag{8}$$

Satisfying the Assumptions. We will now give an outline of the proof of why this algorithm creates an abstraction which satisfies the assumptions.

Assumption 1. Step 1 creates one box containing the equilibrium $x_e = 0$, under the original specification that the live box should include x_e (not on the boundary). Step 2 can change the size of the box containing the equilibrium, but cannot add splits exactly at $x_i = 0$ for any i (because of linearity of dynamics), so the box containing x_e does not have it on the boundary. Step 3 similarly can make splits which affect the box L, but again they are chosen not to be at the equilibrium.

Assumption 2. There are a finite number of boxes in this splitting, which we have already quantified.

Assumption 3. After Step 2 has happened, Assumption 3 is satisfied due to splitting *at* the zero values, and then after each change in Step 3 the "FollowSplits" function is used, which again makes the TA satisfy this assumption.

Assumption 4. We showed that Algorithm 1 terminates, and in the process showed that it only terminates when all infinite times on boxes are removed (apart from L). Hence, this assumption is satisfied.

5.3 The Main Result

The above properties result in the statement that if a TA is created by the method of [9] using the splitting of Alg. 1, then the proof of inevitability of L on the TA abstraction will prove the inevitability of L for the original system. The proof follows from the method of Alg. 1 with the assumptions of Sect. 4.1 and the theorems of Sect. 4.2.

Corollary 1. *Given a continuous dynamical system with each x_i's dynamics of the form (5) or (6), then, by considering the TA abstraction, all possible trajectories of the linear system will reach box L containing $x_e = 0$.* ■

6 Conclusions and Future Work

In this work we have defined a method for a class of linear systems which creates a splitting of the state space. When using this splitting to create a timed automaton by the method of [9], we have shown that certain properties are true of the timed automaton. Together these properties mean that the timed automaton will prove inevitability of the original system reaching a set L around the equilibrium point x_e. The method is easily extendible to a related class of piecewise linear systems.

Our future goal is to extend this method to more general dynamical systems, be they linear, nonlinear, piecewise, or hybrid systems. There are various problems to be overcome with these systems, one of which will be the termination of the splitting method, as the current method only terminates because of the special dynamics involved. Hence, part of the future work will be to revise the splitting method to be more useful for more general systems. For piecewise and hybrid systems, we aim to develop dynamically-driven splitting methods for considering the guards/resets (changes between areas of different dynamics), so that general guards can be considered by the splitting.

References

1. Batt, G., Belta, C., Weiss, R.: Model Checking Liveness Properties of Genetic Regulatory Networks. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 323–338. Springer, Heidelberg (2007)
2. Behrmann, G., David, A., Larsen, K.G.: A Tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)

3. Clarke, E.M., Fehnker, A., Han, Z., Krogh, B., Ouaknine, J., Stursberg, O., Theobald, M.: Abstraction and Counterexample-Guided Refinement in Model Checking of Hybrid Systems. International Journal of Foundations of Computer Science 14(4), 583–604 (2003)
4. D'Innocenzo, A., Julius, A.A., Di Benedetto, M.D., Pappas, G.J.: Approximate timed abstractions of hybrid automata. In: 46th IEEE Conference on Decision and Control, pp. 4045–4050 (2007)
5. Duggirala, P.S., Mitra, S.: Lyapunov Abstractions for Inevitability of Hybrid Systems. In: Hybrid Systems: Computation and Control (HSCC), pp. 115–123 (2012)
6. Heinrich, R., Neel, B.G., Rapoport, T.A.: Mathematical Models of Protein Kinase Signal Transduction. Molecular Cell 9(5), 957–970 (2002)
7. Johansson, K.H., Egerstedt, M., Lygeros, J., Sastry, S.: On the regularization of Zeno hybrid automata. Systems & Control Letters 38(3), 141–150 (1999)
8. Lyapunov, A.M.: The general problem of the stability of motion. PhD thesis, Moscow University (1892); Reprinted in English in the International Journal of Control 55(3) (1992)
9. Maler, O., Batt, G.: Approximating Continuous Systems by Timed Automata. In: Fisher, J. (ed.) FMSB 2008. LNCS (LNBI), vol. 5054, pp. 77–89. Springer, Heidelberg (2008)
10. Mitchell, I., Tomlin, C.J.: Level Set Methods for Computation in Hybrid Systems. In: Lynch, N.A., Krogh, B.H. (eds.) HSCC 2000. LNCS, vol. 1790, pp. 310–323. Springer, Heidelberg (2000)
11. Olivero, A., Sifakis, J., Yovine, S.: Using Abstractions for the Verification of Linear Hybrid Systems. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 81–94. Springer, Heidelberg (1994)
12. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th International Symposium on the Foundations of Computer Science, pp. 46–57 (1977)
13. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation-based abstraction refinement. ACM Transactions on Embedded Computing Systems 6(1), 573–589 (2007)
14. Sloth, C., Wisniewski, R.: Verification of continuous dynamical systems by timed automata. Formal Methods in System Design 39(1), 47–82 (2011)
15. Stursberg, O., Kowalewski, S., Engell, S.: On the Generation of Timed Discrete Approximations for Continuous Systems. Mathematical and Computer Modelling of Dynamical Systems: Methods, Tools and Applications in Engineering and Related Sciences 6(1), 51–70 (2000)
16. Tiwari, A.: Abstractions for hybrid systems. Formal Methods in System Design 32(1), 57–83 (2008)