Abstractions of hybrid systems: Formal languages to describe dynamical behaviour

DOI:
10.3182/20110828-6-IT-1002.01072

Abstractions of hybrid systems: formal languages to describe dynamical behaviour ⋆

Rebekah Carter, Eva M. Navarro-López

School of Computer Science, The University of Manchester
Oxford Road, Manchester, M13 9PL, UK
(e-mails: carterr@cs.man.ac.uk, eva.navarro@cs.man.ac.uk)

Abstract: We show how finite-state automata over finite and infinite words can capture key dynamical properties of hybrid systems. The purpose is to obtain a discrete abstraction of the main complex dynamical behaviour patterns that such systems exhibit. We will form a labelled transition system as an abstraction of the hybrid system, and, in order to differentiate between the key dynamical behaviours, we will associate a set of formal languages to each of the behaviours. We will use both finite regular and infinite ω-regular languages. These formal languages can be accepted by a generalised Muller automaton, which is a novel approach in the specification of dynamical properties of hybrid systems.

Keywords: Hybrid dynamical systems; Hybrid automata; Automata theory; Specification; Abstraction; Formal languages.

1. MOTIVATION

Hybrid dynamical systems combine both real-time continuous-type dynamics and discrete-event occurrences that interact in some way. This interaction can cause complex dynamics, which makes it difficult to know what will happen in the system in general. However, in safety critical systems, it is essential to know how the system will evolve given a certain input.

This paper is devoted to the abstraction of dynamical properties for a class of hybrid systems. The ultimate goal is to differentiate between "good" and "bad" behaviour patterns.

Abstraction is the process of forming a discrete version of the continuous hybrid system model, so that the discrete model captures relevant information about the hybrid system. The most common use of abstraction is in the formal verification of a system, when we find an abstraction of the system that captures enough information to prove a required property. Formal verification is a way of checking whether or not some properties of a system are satisfied. Using an abstraction for verification can greatly decrease the time required to prove a property, since we are considering a much simpler model.

In this work, we consider a procedure for making an abstraction of the behaviours of a hybrid dynamical system, with this abstraction being represented as an automaton. We base the abstraction on the known behaviour patterns of the system, such as equilibrium points and periodic orbits. These behaviour patterns can then be represented as unique regular or ω-regular languages over the finite-state automaton, and can be specified by automaton acceptance conditions, of either classical "final state" type for the regular languages, or Muller type for the ω-regular languages (Muller, 1963).

⋆ Corresponding author: Rebekah Carter. Phone: +44 161 275 6209. Fax: +44 161 275 6204. 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 grateful for the support of the Research Councils United Kingdom (EP/E50048/1).

With the unique languages for the different behaviours, we will distinguish the set of behaviour pattern(s) of interest, and can then set up an abstracted automaton to accept only these behaviour patterns, rejecting the trajectories with a behaviour pattern that is not wanted. In this way, we can differentiate between safe and unsafe trajectories. The contribution of this work is to provide this novel specification framework to distinguish the different behaviour patterns in hybrid systems.

Currently, the approaches to abstraction of hybrid systems are broadly two. The first is the use of *simulation relations*, where a continuous and discrete model are proved to be (bi)similar, so that properties which hold in the discrete model will automatically hold in the continuous one (Henzinger et al., 1998; Tabuada, 2009; Tripakis and Yovine, 2001). This approach needs rigorous proofs of similarity between systems, and so, recently, the concept of *approximate simulation relations* has been introduced (Girard et al., 2008). In approximate simulation, the distance between the actual and abstracted systems is bounded by a certain precision, which means that a much wider class of hybrid systems can use this technique.

The other main approach to abstraction of hybrid systems is the use of abstraction refinement, mainly counterexample-guided abstraction refinement (CEGAR) (Clarke et al., 2003; Ratschan and She, 2007). In abstraction refinement, we start with a basic abstraction of the system, and improve it by using some information about the system. When CEGAR is used for verification, the information used is counterexamples: we search for a trajectory of the abstracted system which starts in the initial region and ends in the unsafe region, and use it to refine the model if it is not a counterexample in the actual system, or we use as a counterexample to prove the property is false in the actual system. We do this until we either prove or disprove the actual property.

Klaedtke et al. (2007) use formal languages to represent the abstracted trajectories of a CEGAR abstraction method. This kind of representation is similar to ours, however it does not

consider infinite length languages, which we do. This paper has been a source of inspiration for our language-based approach, along with Weiss and Alur (2007). This latter paper considers the use of automata on infinite languages as a way of specifying valid schedules for control components. The type of systems treated are piecewise linear discrete-time systems. Here, we treat more general systems and properties.

The rest of this paper is structured as follows. Sections 2 and 3 introduce some background material and definitions. Moreover, in Section 2, a motivating example is considered. Section 4 gives an overview of the method of abstraction, and Section 5 applies the abstraction method for a large class of hybrid systems. Section 6 looks at a case study, and Section 7 considers extensions to the method.

2. THE HYBRID AUTOMATON FRAMEWORK

In this section we introduce the hybrid automaton, and also present a motivating example for the rest of the paper.

2.1 Hybrid automata

Most readers will be familiar with the concept of a hybrid automaton as a representation of the dynamics of a hybrid dynamical system. In this paper we use a definition based on Lygeros et al. (1999, 2003); Johansson et al. (1999), which is reasonably general, but which does not consider inputs or outputs into the system. We do not consider such inputs and outputs in this paper in order to simplify the notation, although they could be added in without too much additional work. The notation used in this definition is similar to that of the model proposed in Navarro-López (2009).

Definition 1. (Hybrid automaton). A hybrid automaton is a collection $H = (Q, \mathscr{X}, \mathscr{F}, Init, Dom, E, G, R)$, where:

- $Q = \{q_1, \ldots, q_m\}$ is the set of discrete locations.
- $\mathscr{X} \subseteq \mathbb{R}^n$ is the continuous state-space.
- $\mathscr{F} = \{f_{q_i}(x) : q_i \in Q\}$ is the collection of vector fields describing the continuous dynamics, such that $f_{q_i} : \mathscr{X} \to \mathscr{X}$. Each $f_{q_i}(x)$ is assumed to be Lipschitz continuous on the location domain for q_i in order to ensure that the solution exists and is unique.
- $Init \subseteq Q \times \mathscr{X}$ is the set of initial hybrid states.
- $Dom : Q \to 2^{\mathscr{X}}$ is the location domain. It assigns a set of continuous states to each discrete location $q_i \in Q$, thus, $Dom(q_i) \subset \mathscr{X}$.
- $E \subseteq Q \times Q$ is a finite set of edges called transitions or events.
- $G : E \to 2^{\mathscr{X}}$ is a guard set. G assigns to each edge a set of continuous states; this set contains the states which enable transition along that edge.
- $R : E \times \mathscr{X} \to 2^{\mathscr{X}}$ is a reset map for the continuous states for each edge. It is assumed to be non-empty, so that the dynamics cannot be destroyed, only changed. ■

We also should consider how we move through this hybrid automaton.

Definition 2. (Execution of a hybrid automaton). An execution of the hybrid automaton H is a sequence of hybrid states with time, $\{(t_0, q^0, x^0), (t_1, q^1, x^1), \ldots, (t_n, q^n, x^n)\}$, where $(q^i, x^i) = (q(t_i), x(t_i))$, such that:

- The run starts in the initial region — $(q^0, x^0) \in Init$.
- Between jump times we evolve continuously — for $t \in [t_i, t_{i+1})$, $\dot{x} = f_{q^i}(x)$.
- At jump points, we take a transition for which we satisfy the guard condition, the continuous state is then reset, and we may change our discrete location. That is, when $t = t_{i+1}$, $q = q^i$ and $x = x^{i,+}$, for any $(q^i, q^{i+1}) \in E$ for which $x \in G((q^i, q^{i+1}))$, the new value of x is defined by $x^{i+1} \in R((q^i, q^{i+1}), x^{i,+})$, and the new discrete location is q^{i+1}. ■

2.2 A motivating example

To motivate the abstraction in this paper, the following discontinuous system, which can be modelled as a hybrid automaton, is considered. It is a simplified oilwell vertical drillstring that exhibits multiple equilibria and periodic oscillations (Navarro-López and Carter, 2010):

$$\begin{aligned} \dot{x}_1 &= \frac{1}{J_\mathrm{r}}\left[-(c_\mathrm{t}+c_\mathrm{r})x_1 - k_\mathrm{t}x_2 + c_\mathrm{t}x_3 + u\right],\ \dot{x}_2 = x_1 - x_3, \\ \dot{x}_3 &= \frac{1}{J_\mathrm{b}}\left[c_\mathrm{t}x_1 + k_\mathrm{t}x_2 - (c_\mathrm{t}+c_\mathrm{b})x_3 - T_{\mathrm{f_b}}(x_3)\right]. \end{aligned} \tag{1}$$

$\mathbf{x} = (x_1, x_2, x_3)^\mathrm{T}$, with x_1 and x_3 the angular velocities of the top-rotary system and the bit, respectively, and x_2 is the difference between the two angular displacements. $u > 0$ and the weight on the bit (W_ob>0) are two varying parameters. The discontinuous friction torque is $T_{\mathrm{f_b}}(x_3) = f_\mathrm{b}(x_3)\mathrm{sign}(x_3)$, and

$$f_\mathrm{b}(x_3) = W_\mathrm{ob}R_\mathrm{b}\left[\mu_{\mathrm{c_b}} + (\mu_{\mathrm{s_b}} - \mu_{\mathrm{c_b}})\exp^{-\frac{\gamma_\mathrm{b}}{v_\mathrm{f}}|x_3|}\right], \tag{2}$$

with $R_\mathrm{b} > 0$ the bit radius; $\mu_{\mathrm{s_b}}$, $\mu_{\mathrm{c_b}} \in (0,1)$ the static and Coulomb friction coefficients associated with the bit; $0 < \gamma_\mathrm{b} < 1$; and $v_\mathrm{f} > 0$. The Coulomb and static friction torque are $T_{\mathrm{c_b}}$ and $T_{\mathrm{s_b}}$, respectively, with $T_{\mathrm{c_b}} = W_\mathrm{ob}R_\mathrm{b}\mu_{\mathrm{c_b}}$, $T_{\mathrm{s_b}} = W_\mathrm{ob}R_\mathrm{b}\mu_{\mathrm{s_b}}$. When $x_3 = 0$, $T_{\mathrm{f_b}}(\mathbf{x}) = T_{s_b} \cdot u_\mathrm{eq}(\mathbf{x}) = c_\mathrm{t}x_1 + k_\mathrm{t}x_2 - (c_\mathrm{t}+c_\mathrm{b})x_3$.

This system is very suitable for our purpose of abstraction and will demonstrate our general methodology well. It exhibits a rich collection of behaviours depending on the values of (u, W_ob), mainly:

- **Positive velocity equilibrium**: the bit velocity x_3, converges to a positive equilibrium value and $x_1 = x_3$.
- **Permanent stuck bit**: the bit stops rotating after some period of time and never starts again.
- **Stick-slip motion**: the bit velocity oscillates between zero and a positive velocity.

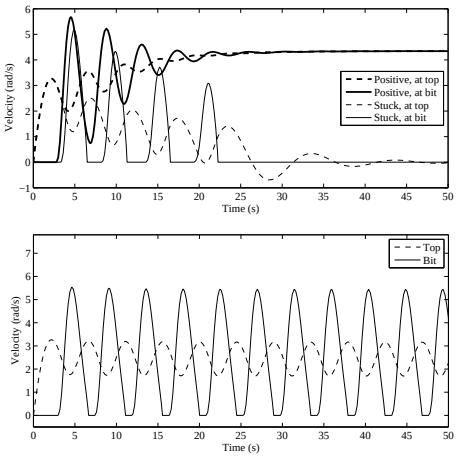

Fig. 1. Top: positive velocity and permanently stuck equilibrium behaviours. Bottom: stick-slip motion. Dotted lines represent x_1, and solid lines represent x_3.

Figure 1 shows these three behaviour patterns. We will return to this system later to illustrate the theory presented.

3. FORMAL LANGUAGE TOOLS

3.1 Finite regular languages and their automata

We define here some theory of the regular languages and automata that is used frequently in computer science. For more details see Hopcroft et al. (2007).

Definition 3. (Finite-state automaton). A finite-state automaton is a quintuple $A = (Q, \Sigma, \delta, Q_0, F)$, where:

- Q is a finite set of discrete states or *locations*.
- Σ is a finite set of input symbols or *events*.
- δ is a *transition function* such that $\delta : Q \times \Sigma \to Q$.
- $Q_0 \subseteq Q$ is the set of *initial states*.
- F is a set of final or *accepting* states; $F \subset Q$. ■

The languages that finite-state automata can accept as input are the regular languages, which can be written as regular expressions. These regular expressions are made up of symbols from the extended alphabet $\Sigma \cup \varepsilon$, where ε is the empty symbol, along with three operators on these symbols:

(1) *Union*: the union of two strings of symbols is either one string or the other. For two strings p and v, the union of p and v is denoted $p + v$.
(2) *Concatenation*: the concatenation of two strings is the first string followed by the second. p concatenated with v is written pv.
(3) *Kleene closure*: the Kleene closure of a string is that string repeated as many times as we want, including none. The Kleene closure of string p is p^*.

3.2 Infinite ω-regular languages and their automata

We look now at infinite languages and an automaton that can accept them. This theory can be found in Thomas (1990). In order to describe the infinite nature of strings, we need an analogue of the Kleene closure operator to describe infinite repetition. For a string p, the ω-closure of p is p^ω; the string p repeated an infinite number of times. For Σ a finite alphabet, Σ^ω is the set of infinite ω-regular words over Σ. For a finite regular language $L \subseteq \Sigma^*$, the ω-closure of L is defined as

$$L^\omega = \{\alpha \in \Sigma^\omega | \alpha = p_0 p_1 p_2 \ldots, \text{ with } p_i \in L \text{ for } i \geq 0\}.$$

Definition 4. (ω-regular language). A language is called ω-regular if it has the form $\bigcup_{i=1}^{n} U_i.(V_i)^\omega$, where U_i and V_i are regular languages, and $n \in \mathbb{N}$. That is, any string in an (infinite) ω-regular language consists of a regular string from one language followed by an infinite number of regular strings from another language. In this sense, ω-regular languages are the infinite extension of regular languages. ■

The only difference between the finite state automaton and automata that accept infinite languages is in the way that the acceptance conditions are defined. That is, we change the definition of "final states" so that infinite languages are captured. In order to define the condition that will accept the infinite languages, we need a piece of notation to describe the set of locations we see infinitely often during a run of an automaton.

Definition 5. (Infinity set). Let Q be the set of locations of a finite-state automaton, and for an infinite string p, let $run(p) = q^0 q^1 \ldots$ be the sequence of locations that p causes the automaton to pass through. Then the *infinity set* of p is defined as:

$inf(p) := \{q \in Q |$ there exist infinitely many n such that $q^n = q\}$, or, informally, the set of states that occur infinitely often during a run of the automaton. ■

We can now introduce the automaton representation that we use in this work to accept infinite ω-regular languages. This is the Muller automaton (Muller, 1963), which accepts an input word if its infinity set is identically equal to one of the sets in a defined family. We use this type of infinite language automata rather than Büchi automata since the deteministic Muller automata accept more languages.

Definition 6. (Muller automaton). A Muller automaton A_M on infinite strings is a quintuple $(Q, \Sigma, \delta, Q_0, \mathscr{F})$ where

- Q is a finite set of discrete states or locations.
- Σ is the input alphabet.
- $\delta : Q \times \Sigma \to 2^Q$ is the transition function.
- $Q_0 \subseteq Q$ is the set of initial states.
- $\mathscr{F} \subseteq 2^Q$ is the *family* of sets of accepting (or recurrent) states.

An infinite string p is accepted by this automaton if $inf(p) = F$ for some $F \in \mathscr{F}$. ■

3.3 Combining finite and infinite languages

In this paper both finite regular languages and infinite ω-regular languages are represented in the same automaton, making use of a definition due to Muller (1963). This definition combines the regular language acceptance condition with a Muller ω-regular language acceptance condition. The combination is written as a formal sum of the two acceptance sets.

Definition 7. (Generalised Muller automaton). A generalised Muller automaton on finite and infinite strings is a quintuple $A_{GM} = (Q, \Sigma, \delta, Q_0, F + \mathscr{F})$ where Q, Σ, δ, Q_0 are the same as in Definition 6, with $F \subseteq Q$ is the set of regular language accepting states, and $\mathscr{F} \subseteq 2^Q$ is the family of sets of states that accept ω-regular strings by Muller acceptance. ■

4. OVERVIEW OF THE ABSTRACTION METHOD

We assume from now on that we have a hybrid dynamical system, which is given as a hybrid automaton H as defined in Definition 1. Given this system, we will specify a discrete automaton with associated languages which describe the possible system behaviour patterns. For this, we will specify a generalised Muller automaton which has different acceptance conditions for each possible behaviour of the system, so that we can use these distinct acceptance conditions to accept or reject behaviours.

A first step to specifying a generalised Muller automaton as an abstraction of our system is to specify how we form the initial abstraction of the system. We form a *labelled transition system* (LTS), which is a finite-state automaton without any acceptance conditions. Informally, this LTS keeps the structure of the hybrid automaton, preserving locations and edges, but labelling the edges with a letter from a labelling set Σ, which represents the guard and reset conditions present on this transition.

Definition 8. (Initial abstraction). Consider a hybrid automaton $H = (Q, \mathscr{X}, \mathscr{F}, Init, Dom, E, G, R)$. The initial abstraction of H is a labelled transition system, $LTS = (Q, \Sigma, \delta, Q_0)$, where:

- Q is identical to the set of locations for H.
- Σ is an alphabet that represents the different possible combinations of guard sets and reset maps in the hybrid automaton. There is a mapping $\mapsto$, such that for each $e \in E$, there is an $a \in \Sigma$ such that $G(e) \times R(e, x) \mapsto a$.

- δ is a transition function which defines the edges in the transition system; for $q_i, q_j \in Q$ and $a \in \Sigma$, we have $\delta(q_i, a) = q_j$ if and only if $(q_i, q_j) = e \in E$ and $G(e) \times R(e,x) \mapsto a$, with $x \in \mathscr{X}$.
- Q_0 is the set of possible starting locations, obtained by projection of the set *Init* onto the set of discrete locations: $q_i \in Q_0$ if and only if there exists a set $X_i \subseteq \mathscr{X}$ such that $q_i \times X_i \subseteq Init$. ■

To specify different acceptance conditions for each behaviour on this initial abstraction, we need to know what the possible behaviour patterns are. This means that we find the equilibrium points, the periodic behaviours, and any other long term behaviour patterns of the hybrid system. When we have these behaviour patterns we work out which discrete locations of the hybrid automaton they exist in: equilibrium points will exist in one location, for example, whereas periodic behaviour could traverse through many different locations, which would give an infinitely switching behaviour in the hybrid automaton.

The way that a particular behaviour pattern moves through the locations of the hybrid automaton can be directly translated to a formal language by comparison with the abstracted LTS that we obtain by using the method above. We then specify automaton acceptance conditions, so that we can have an automaton which is an abstraction of the structure of the discontinuities of the system, but which also represents its behaviour patterns.

To specify the acceptance conditions for the (infinite) ω-regular languages, we may need to make some minor changes to the structure of the abstracted LTS so that we can accept *only* the language we `would` like to accept. These changes can be made in an automated way. We emphasise that it is only the locations and transitions of the automaton that could change; the labelling set Σ cannot change once the abstraction is made. If we need to change the guards/resets to be able to distinguish the behaviours, we must do this before we abstract the system (see Section 7).

5. ABSTRACTION OF A CLASS OF HYBRID SYSTEMS

We now come to the specifics of how this method applies to a class of hybrid systems: discontinuous dynamical systems (DDS) with one surface of discontinuity, which can be modelled by the hybrid automaton in Figure 2. Here we call this automaton the *DDS hybrid automaton*, H_{DDS}: it is a modified version of the one presented in Navarro-López and Carter (2010) as the *extended DDS hybrid automaton*. Here, $s(x) = 0$ is the surface of discontinuity, which we consider to be equivalent to $|s(x)| \leq \delta$, in order to improve numerical stability issues. Note that we do not put any explicit conditions on the initial location here, since we are covering a general class of systems.

We highlight that the class of discontinuous systems considered is general, and includes systems with discontinuous state derivatives and sliding motions. Furthermore, the methodology and the hybrid model can be extended to systems with multiple surfaces of discontinuity by means of the composition of several DDS hybrid automata.

To be able to derive the languages which represent the behaviour of a large class of systems, we need to make some assumptions. Firstly, we assume that there is at most one equilibrium point in each location of the hybrid automaton, and secondly, that any periodic behaviours go through more than one location, hence will have infinite ω-regular languages to represent them. These periodic behaviours must be distinct, in

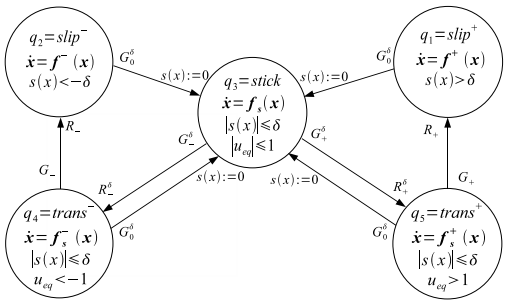

Fig. 2. The DDS hybrid automaton, H_{DDS}, modelling the class of discontinuous dynamical systems with one surface of discontinuity. Inside locations: top line is the location number and name, second line is the equation for the continuous dynamics, and third (and fourth) lines are the domain of that location. Transition markings: guard conditions near the departing location, and resets near the arriving location.

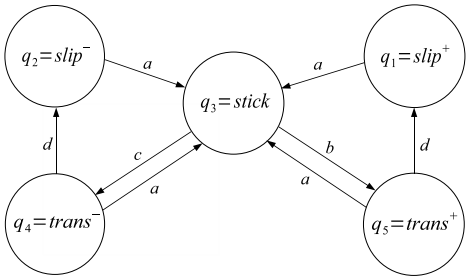

Fig. 3. The initial abstraction of the DDS hybrid automaton.

the sense that no two periodic orbits travel through exactly the same set of locations in the automaton. It is also assumed that the hybrid automaton switches from one location to another a finite number of times in any time interval, so that we do not see Zeno behaviour. We consider how to lift some of these restrictions in Section 7.

With these assumptions in mind, and taking into account some practical considerations about the transition locations (q_4 and q_5), we can list the possible behaviour patterns:

- There are a maximum of 3 equilibrium points, one for each of the locations q_1, q_2, q_3. This means that there are at most three distinct behaviour patterns which converge to an equilibrium point.
- We consider here only a subset of the possible periodic orbits (infinitely switching behaviours). These possible infinite behaviours are 1) to loop through *stick*, *trans*$^+$, *slip*$^+$ and back to *stick* infinitely, or 2) through *stick*, *trans*$^-$, *slip*$^-$ and back to *stick* infinitely, or 3) to alternate these two options so that we do the *slip*$^+$ loop then the *slip*$^-$ loop and repeat this infinitely.

The infinite behaviours that we can distinguish with these restrictions are reasonably broad. They include stick-slip type behaviour, such as occurs in the motivating example (bottom of Figure 1), and also impacting behaviour, such as occurs in the classical bouncing ball example. The alternating *slip*$^+$ and *slip*$^-$ behaviour can model behaviours which oscillate between either side of the discontinuity, as well as behaviours which oscillate, but get stuck in the discontinuity surface at points (a generalisation of the example's stick-slip behaviour).

	Initial location				
Equil. loc.	q_1	q_2	q_3	q_4	q_5
q_1	$\varepsilon+a[(b+c)(\varepsilon+d)a]^*bd$	$a[(b+c)(\varepsilon+d)a]^*bd$	$[(b+c)(\varepsilon+d)a]^*bd$	$(\varepsilon+d)a[(b+c)(\varepsilon+d)a]^*bd$	$d+(\varepsilon+d)a[(b+c)(\varepsilon+d)a]^*bd$
q_2	$a[(b+c)(\varepsilon+d)a]^*cd$	$\varepsilon+a[(b+c)(\varepsilon+d)a]^*cd$	$[(b+c)(\varepsilon+d)a]^*cd$	$d+(\varepsilon+d)a[(b+c)(\varepsilon+d)a]^*cd$	$(\varepsilon+d)a[(b+c)(\varepsilon+d)a]^*cd$
q_3	$a[(b+c)(\varepsilon+d)a]^*$	$a[(b+c)(\varepsilon+d)a]^*$	$[(b+c)(\varepsilon+d)a]^*$	$(\varepsilon+d)a[(b+c)(\varepsilon+d)a]^*$	$(\varepsilon+d)a[(b+c)(\varepsilon+d)a]^*$
Inf. beh.					
$slip^+$	$a[(b+c)(\varepsilon+d)a]^*(bda)^\omega$	$a[(b+c)(\varepsilon+d)a]^*(bda)^\omega$	$[(b+c)(\varepsilon+d)a]^*(bda)^\omega$	$(\varepsilon+d)a[(b+c)(\varepsilon+d)a]^*(bda)^\omega$	$(\varepsilon+d)a[(b+c)(\varepsilon+d)a]^*(bda)^\omega$
$slip^-$	$a[(b+c)(\varepsilon+d)a]^*(cda)^\omega$	$a[(b+c)(\varepsilon+d)a]^*(cda)^\omega$	$[(b+c)(\varepsilon+d)a]^*(cda)^\omega$	$(\varepsilon+d)a[(b+c)(\varepsilon+d)a]^*(cda)^\omega$	$(\varepsilon+d)a[(b+c)(\varepsilon+d)a]^*(cda)^\omega$
Alternating	$a[(b+c)(\varepsilon+d)a]^*(bdacda)^\omega$	$a[(b+c)(\varepsilon+d)a]^*(bdacda)^\omega$	$[(b+c)(\varepsilon+d)a]^*(bdacda)^\omega$	$(\varepsilon+d)a[(b+c)(\varepsilon+d)a]^*(bdacda)^\omega$	$(\varepsilon+d)a[(b+c)(\varepsilon+d)a]^*(bdacda)^\omega$

Table 1. Regular languages representing the behaviour patterns that could be present in H_{DDS}.

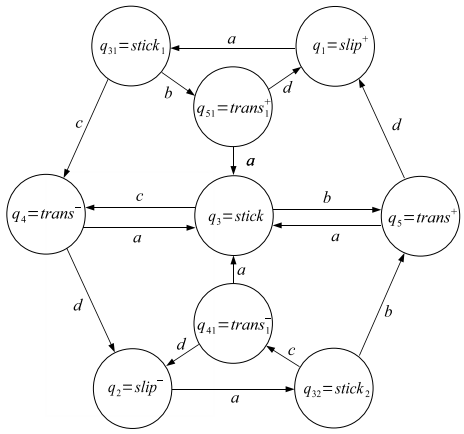

Fig. 4. Muller automaton which can accept the three languages associated to the equilibrium behaviours, as well as the three infinite languages in Table 1, associated to periodic behaviours.

Starting from the hybrid automaton, we can make the initial abstraction of the model to a labelled transition system. This initial abstraction is shown pictorially in Figure 3. Note that each different guard condition and reset map combination has been given a different letter in an alphabet of four symbols, $\Sigma = \{a,b,c,d\}$:

$$a \Leftrightarrow G_0^\delta, \text{ with reset } \equiv \{x \in \mathscr{X} : s(x) := 0\};$$
$$b \Leftrightarrow G_+^\delta, \text{ with reset } R_+^\delta;\ c \Leftrightarrow G_-^\delta, \text{ with reset } R_-^\delta;$$
$$d \Leftrightarrow (G_+ \text{ with reset } R_+) \text{ or } (G_- \text{ with reset } R_-).$$

Note that the assignment of the same letter to the G_+ and G_- transitions is a modelling decision, taken because the structure of the system means this still gives us enough information to distinguish the behaviours.

Using this initial abstraction we can derive the languages which are associated with each of the possible behaviour patterns. For instance, if the initial location is q_3, then the string "$[(b+c)(\varepsilon+d)a]^*bd$" represents the behaviour which converges to the equilibrium point in location q_1. We give the languages for the various behaviours in Table 1, derived with respect to each of the possibilities for the initial locations.

Given these possible behaviour patterns, we now find a generalised Muller automaton which can accept the infinite and finite languages given in Table 1. The result is the automaton in Figure 4. If the initial condition in the initial abstraction contained the location q_i, then in this revised abstraction the initial condition contains the set of locations with related numbers: $\{q_i\}$, $\{q_i, q_{i1}\}$, or $\{q_i, q_{i1}, q_{i2}\}$ (depending on the value of i).

We specify the acceptance conditions on the automaton by:

- q_1 equilibrium. Acceptance location $F = \{q_1\}$.
- q_2 equilibrium. Acceptance location $F = \{q_2\}$.
- q_3 equilibrium. Acceptance set $F = \{q_3, q_{31}, q_{32}\}$.
- $slip^+$ periodic behaviour. Muller acceptance set $\mathscr{F} = \{q_1, q_{31}, q_{51}\}$.
- $slip^-$ periodic behaviour. Muller acceptance set $\mathscr{F} = \{q_2, q_{32}, q_{41}\}$.
- Alternating periodic behaviour. Muller acceptance set $\mathscr{F} = \{q_1, q_{31}, q_4, q_2, q_{32}, q_5\}$.

The important point about this automaton with these acceptance conditions is that, theoretically, we can run the automaton to check what the behaviour of the system is, and check whether it is a good or a bad behaviour, by selecting which of the acceptance conditions correspond to "good" behaviour patterns, and which correspond to "bad".

6. CASE STUDY

Consider the system presented in Section 2.2, which falls into the class we considered in Section 5. Here, $s(\mathbf{x}) = x_3$, where x_3 is the velocity of the bit, and the resets R_+^δ, R_-^δ, R_+, R_- are all identity maps. The guards are defined by

$$G_0^\delta = (|x_3| \le \delta) \wedge (|u_{eq}| \le 1),$$
$$G_+^\delta = (|x_3| \le \delta) \wedge (u_{eq} > 1),$$
$$G_-^\delta = (|x_3| \le \delta) \wedge (u_{eq} < -1),$$
$$G_+ = x_3 > \delta,$$
$$G_- = x_3 < -\delta.$$

The dynamics within the locations are given by equations 1 and 2, with the friction torque defined by

$$T_{f_b} = \begin{cases} f_b(x_3)\text{sign}(x_3) & \text{in locations } q_1 \text{ and } q_2, \\ T_{s_b} \cdot u_{eq}(\mathbf{x}) & \text{in location } q_3, \\ T_{s_b} & \text{in locations } q_4 \text{ and } q_5. \end{cases}$$

In Section 2.2 we said that this system has three behaviour patterns, which are positive velocity on the bit, permanently stuck bit, and positive stick-slip motion. These equate, respectively, to the equilibrium in location q_1, the equilibrium in location q_3, and the periodic behaviour that infinitely repeats the loop $q_3 \to q_5 \to q_1 \to q_3 \ldots$.

In this drillstring model, we usually start in location q_3. For this initial location, we can therefore find the languages which represent the three behaviour patterns present in the system by looking at Table 1:

- Positive velocity equilibrium (q_1): $[(b+c)(\varepsilon+d)a]^*bd$.
- Stuck bit equilibrium (q_3): $[(b+c)(\varepsilon+d)a]^*$.
- Positive stick-slip: $[(b+c)(\varepsilon+d)a]^*(bda)^\omega$.

7. EXTENSIONS FOR MORE GENERAL BEHAVIOURS

In this paper, we have considered a general class of hybrid systems, which are the discontinuous dynamical systems with one surface of discontinuity. We have also considered some languages to describe the behaviour of such systems. The

languages we have considered cover a wide range of possible behaviour patterns, but there are many more possible behaviour patterns that can occur if we remove some of the assumptions we made. Hence, in this section, we will discuss the main issues that arise when extending this theory to more behaviours within the class of DDS with one surface of discontinuity.

The major assumption in Section 5 was to say that two (or more) behaviours cannot occur in the same location, or in the same group of locations. This is a fairly restrictive condition, since even in smooth nonlinear dynamical systems we can have more than one equilibrium point, for instance. However, this problem can be overcome in the following manner.

If we have a DDS which can be modelled by the hybrid automaton in Figure 2, but which has some behaviours which occur in the same location or group of locations, we need to make sure they will be distinguished when we make the abstraction. In order to do this, splitting a location into some new locations by splitting its domain may be appropriate. This splitting should be done so that each behaviour has it's own location (or group of locations) which describe its long-term behaviour in the hybrid automaton.

There are four different types of interactions between equilibria and periodic orbits that we identify which must be separated in order for the initial abstraction of Definition 8 to be able to distinguish them with different languages.

(1) Multiple equilibria exist within one location. This is a problem because they will all have the same finite regular language, so they cannot be distinguished by automaton acceptance conditions.
(2) An equilibrium and a periodic behaviour exist within one location (for the same reason as above).
(3) Two (or more) periodic behaviours occur within one location (for the same reason again).
(4) Two (or more) periodic behaviours occur through exactly the same group of locations. The problem here is that the acceptance condition for these two behaviours will be to infinitely travel through the same group of states in the automaton, and so they will be indistinguishable.

For the multiple equilibria problem, if we have k equilibria, we can separate the languages they will produce by splitting the domain of the location into k parts, with one equilibrium point in each part of the domain. These k new domains then form the basis for k new locations, replacing the one we started with. This splitting can be achieved by use of algorithms for separation of points in Euclidean space (for example, Boland and Urrutia (1995)).

The transitions between these new locations will be only to other locations which have a neighbouring domain, with the guard conditions defined by the domain they will be entering, with no resets (since it is within a continuous section of the dynamics). We can also make calculations for which locations the transitions coming in or going out of the group we have just created will go to, so that we only have as many transitions as are necessary.

In the other three cases, we can apply similar techniques to separate the languages of the behaviours. We advocate approaching these problems in the order specified above, since they increase in computational difficulty as we go down the list, and it may be that some earlier calculations actually separate some behaviours further down, and this makes the more computationally expensive calculations unnecessary.

With the new DDS hybrid automaton created through this process, we will be able to form its initial abstraction, so that we can find distinct languages which represent the behaviours in the DDS. The languages we find will be different depending on what behaviours we have to separate in the initial system, but the process can be automated, minimising human effort.

REFERENCES

Boland, R.P. and Urrutia, J. (1995). Separating collections of points in Euclidean spaces. *Information Processing Letters*, 53, 177–183.

Clarke, E., Fehnker, A., Han, Z., Krogh, B.H., Stursberg, O., and Theobald, M. (2003). Verification of hybrid systems based on counterexample-guided abstraction refinement. In *Proceedings of TACAS 2003*, volume 2619 of *LNCS*, 192–207.

Girard, A., Julius, A.A., and Pappas, G.J. (2008). Approximate simulation relations for hybrid systems. *Discrete Event Dynamic Systems*, 18(2), 163–179.

Henzinger, T.A., Kopke, P.W., Puri, A., and Varaiya, P. (1998). What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57, 94–124.

Hopcroft, J.E., Motwani, R., and Ullman, J.D. (2007). *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 3rd edition.

Johansson, K.H., Egerstedt, M., Lygeros, J., and Sastry, S. (1999). On the regularization of Zeno hybrid automata. *Systems & Control Letters*, 38(3), 141–150.

Klaedtke, F., Ratschan, S., and She, Z. (2007). Language-based abstraction refinement for hybrid system verification. In *Verification, Model Checking, and Abstract Interpretation (VMCAI 2007)*, volume 4349 of *LNCS*, 151–166.

Lygeros, J., Johansson, K.H., Simic, S.N., and Sastry, S.S. (2003). Dynamical properties of hybrid automata. *IEEE Transactions on Automatic Control*, 48(1), 2–17.

Lygeros, J., Tomlin, C., and Sastry, S. (1999). Controllers for reachability specifications for hybrid systems. *Automatica*, 35, 349–370.

Muller, D.E. (1963). Infinite sequences and finite machines. In *Proceedings of the Fourth Annual Symposium on Switching Circuit Theory and Logical Design*, 3–16.

Navarro-López, E.M. (2009). Hybrid modelling of a discontinuous dynamical system including switching control. In *Proceedings of CHAOS09, London, UK*.

Navarro-López, E.M. and Carter, R. (2010). Hybrid automata: An insight into the discrete abstraction of discontinuous systems. *International Journal of Systems Science*. In press.

Ratschan, S. and She, Z. (2007). Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Transactions on Embedded Computing Systems*, 6(1), 573–589.

Tabuada, P. (2009). *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer.

Thomas, W. (1990). *Automata on Infinite Objects*, volume B of *Handbook of Theoretical Computer Science*, Elsevier.

Tripakis, S. and Yovine, S. (2001). Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18, 25–68.

Weiss, G. and Alur, R. (2007). Automata based interfaces for control and scheduling. In *Proceedings of Hybrid Systems: Computation and Control*, volume 4416 of *LNCS*, 601–613.