Modular Development of Hybrid Systems for Verification in *Coq*

Milad Niqui[*] and Olga Tveretina[**]

Institute for Computing and Information Sciences,
Radboud University Nijmegen, The Netherlands
{M.Niqui,O.Tveretina}@cs.ru.nl

Abstract. In this paper we present a formalization of the theory of hybrid automata and algorithms for building trajectory trees using module types and functors in the *Coq* proof assistant.

1 Preliminaries

Hybrid systems are systems in which there is a significant interaction between the continuous and discrete parts. Many of the applications of hybrid systems are *safety critical* and require the guarantee of a safe operation. The problem of safety verification seeks an answer to the *reachability* problem: is there a potentially unsafe state reachable from an initial state?

The notion of a hybrid automaton was introduced in order to extend verification methods towards the systems with continuous and discrete dynamics [1]. A Hybrid automaton can be defined as a tuple $\mathcal{H} = (\mathcal{DS}, n, \mathcal{S}_0, \mathcal{I}, \phi, \mathcal{G}, \mathcal{R})$ with the following components: $\mathcal{DS}$ is a finite set of discrete locations; $n \geq 0$ is the dimension of $\mathcal{H}$. The state space of $\mathcal{H}$ is $\mathcal{S} := \mathcal{DS} \times \mathbb{R}^n$. Each state has thus the form (l, x), where $l \in \mathcal{DS}$ and $x \in \mathbb{R}^n$. $\mathcal{S}_0 \subseteq \mathcal{S}$ is a set of initial states. $\mathcal{I} : \mathcal{DS} \to \mathcal{P}(\mathbb{R}^n)$ assigns to each location l an invariant set $\mathcal{I}(l) \subseteq \mathbb{R}^n$. $\phi : (\mathcal{DS} \times \mathbb{R} \times \mathbb{R}_{\geq 0})^n \to \mathbb{R}^n$ defines the flow of a system in a discrete location with an initial condition $\phi(l, x_0, 0) = x_0$. ϕ is a vector of n functions $f_i : \mathcal{DS} \times \mathbb{R} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ such that for each i exists $g_i : \mathcal{DS} \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}_{\geq 0}$ such that for all $d \in \mathcal{DS}$, $x, y \in \mathbb{R}$, $t \in \mathbb{R}_{\geq 0}$ if $f_i(d, x, t) = y$ then $g_i(d, x, y) = t$. $\mathcal{G} : \mathcal{DS} \times \mathcal{DS} \to \mathbb{R}^n$ describes a guard condition. $\mathcal{R} : \mathcal{DS} \times \mathcal{DS} \times \mathbb{R}^n \to \mathbb{R}^n$ is a reset function. The semantics of a hybrid automaton is given by the transition system [2].

Our method is based on decomposing the continuous state space according to an n-dimensional rectangular grid. We denote by χ an abstract state, by S^a the set of all abstract states, and by S^a_0 the set of initial abstract states.

Definition 1 (Strict Abstract Transition System – *SATS*). *A hybrid automaton $\mathcal{H} = (\mathcal{DS}, n, \mathcal{S}_0, \mathcal{I}, \phi, \mathcal{G}, \mathcal{R})$ and an abstract state space S^a generate the* strict abstract transition system $\mathcal{T}^{sats} = \{\mathcal{S}^a, \rightsquigarrow_c, \rightsquigarrow_d, S^a_0\}$ *with*

[*] Research supported by the Netherlands Organisation for Scientific Research (NWO).
[**] Research supported by the BRICKS/FOCUS project 642.000.501.

– *the set of initial abstract states* $\mathcal{S}_0^a$*: an abstract state* $(l,\chi) \in \mathcal{S}_0^a$ *if there is* $(l,x) \in \mathcal{S}_0$ *such that* $x \in \chi$;
– $(l,\chi) \rightsquigarrow_c (l,\chi') \Leftrightarrow \exists t \geq 0, x \in \chi, x' \in \chi', l' \in \mathcal{DS},\ f_i(l,x_i,t) = x'_i \wedge$
$(f_i(l,x_1,t),\ldots,f_i(l,x_n,t)) \in \mathcal{G}(l,l') \wedge$
$(\forall \bar{t} \in [0,t]\ ,(f_i(l,x_1,\bar{t}),\ldots,f_i(l,x_n,\bar{t})) \in \mathcal{I}(l))$,
where $x = (x_1,\ldots,x_n), x' = (x'_1,\ldots,x'_n)$;
$(l,\chi) \rightsquigarrow_d (l',\chi') \Leftrightarrow \exists x \in \chi, x' \in \chi', (l,x) \in \mathcal{G}(l,l') \wedge x' = \mathcal{R}(l,l',x)$.

2 Formalization of Hybrid Automaton in *Coq*

Coq [3] is an interactive theorem prover based on constructive type theory. Among the many ingredients of *Coq*, what is of interest in the present work is the ability to axiomatize theories as *modules*. In short we have *module types*, that we will use for formalizing the general theory of hybrid automata; and the *module implementations* that are basically concrete hybrid automata. Implementing a module then means that we should provide parameters and *prove* that they satisfy the axioms.

Another characteristic aspect of *Coq* that is relevant for our work is the presence of *dependent types*. Central in our work is the dependent type of vectors to model the n-dimensional space of continuous states.

In our formalization of $\mathcal{H}$ we need some basic data-types that are used in this project. Most of these (eg. natural and real numbers, booleans) are defined in the standard library of *Coq* [3] in a straightforward way. Some use inductive types — a generalization of more familiar algebraic types— which are the main building blocks of *Coq* and its logic *Calculus of Inductive Constructions*. We use the inductive type of `List` of polymorphic finite lists and `Vector` of representing the n-dimensional vectors of elements of a set A, (i.e. elements of A^n).

```
Inductive Vector (A:Set): ℕ → Type :=
    | ∅̆ : Vector A 0
    | ⁙  : ∀ (a:A) (n:ℕ), Vector A n → Vector A n+1.
```

Thus, a 0-dimensional vector is $\breve{\varnothing}$ (the empty vector), and an n-dimensional vector $a ⁙ v$ is obtained by “pairing” an element $a \in A$ with an $n-1$-dimensional vector v. One can consider vectors as lists of length n where n is coded in their type. The type of a vector is dependent on its dimension.

The theory of hybrid automata in *Coq* will be an abstract data-type defined as the following *module type*.

```
Module Type 𝓗.
 Parameter 𝓓𝓢: FiniteSetOfNaturals.
 Parameter dim: ℕ.
 Definition ℝ^dim:= Vector ℝ dim.
 Definition 𝓢 := 𝓓𝓢×ℝ^dim.
```

```
  Parameter 𝒮₀ : 𝒮 → Prop.
  Parameter ℐ: 𝒟𝒮 → ℝ^dim → Prop.
  Parameter ϕ: Vector (𝒟𝒮 → ℝ → ℝ≥0 → ℝ) dim.
  Parameter 𝒢: 𝒟𝒮 → 𝒟𝒮 → ℝ^dim → Prop.
  Parameter ℛ: 𝒟𝒮 → 𝒟𝒮 → ℝ^dim → ℝ^dim.
  Axiom ϕ_invertible:is_true_∀_coord _ _ ϕ
                         (λf∃g, ∀d x r t, f d x t = r → g d x r = t).
End ℋ.
```

The Parameters in this definition correspond to those of hybrid automaton, while the sole axiom corresponds to the property that the flow (which is the vector of solutions to differential equations) should contain only invertible functions. In above FiniteSetOfNaturals, a type for finite subsets of $\mathbb{N}$, is defined using a combination of module types and dependent types. The type Prop, the universe of propositions, is used to formalize subsets as predicates on a set.

The function is_true_∀_coord in this axiom is the function that given a vector of elements of some set A (in this case ϕ) and a property P of elements of A checks whether P holds for all coordinates of the vector (underscore (_) denotes the automatically inferable arguments of functions). In our case P is a property of the functions $\mathcal{DS}{\to}\mathbb{R}{\to}\mathbb{R}_{\geq 0}{\to}\mathbb{R}$ and states that

$$P(f) \text{ iff } \exists g \forall dxrt, ((f(d))(x))(t) = r \implies ((g(d))(x))(r) = t \ .$$

The above code lays the basis of the theory of hybrid automata and it can be extended to provide the abstract data types for various transition systems. An $SATS$ is definable as a module type with four parameters that extends the above module:

```
Declare Module H: ℋ.
  Parameter Grid: Vector Partition H.dim.
  Parameter Grid_initial : List Label.
  Parameter ⇝c : Abstract_State → Abstract_State → bool.
  Parameter ⇝d : Abstract_State → Abstract_State → bool.
```

Here Partition is a list of elements of $\mathbb{R}$ that denote the grid in each dimension. Moreover, H.*dim* means that the dimension *dim* should be inherited from the parent theory $\mathcal{H}$ which is declared as H. For Label we first define what a *hyperinterval* is. This is formalized as a record containing a hypercube (vector of intervals) and a property that checks whether the edges of this hypercube correspond to *consecutive* intervals in the Partition. A Label is then a pair of a discrete state and a List of hyperintervals. Finally Abstract_State is a pair containing a discrete state and a hyperinterval in that discrete state.

After defining the module type extension for $SATS$ we can develop a theory by defining functions and proving the lemmas that hold for *every* instance of $SATS$. In particular we can formalize an algorithm BuildTreeOATPS that builds the tree of trajectories in each $SATS$:

```
Definition BuildTree0ATPS (d0:Label): MTree Label :=
  gist_BuildHistoryTree0ATPS
   (BuildHistoryTree0ATPS
     (Build_Label_ext (fst d0) (snd d0)
                     (λd:H.𝒟𝒮, if d=(fst d0) then snd d0
                                                 else ∅)
                     false MAX)
    ).
```

Next we can instantiate the theory with the thermostat in [2].

```
Module Thermostat_as_ℋ <: ℋ.
 Definition 𝒟𝒮:= { 1, 2, 3}.
 Definition Heat:=1.
 Definition Cool:=2.
 Definition Check:=3.
 Definition dim:=2.
 Definition clock (v:ℝ^dim) : ℝ := Vhead _ 1 v.
 Definition temperature (v:ℝ^dim) : ℝ := Vhead _ 0 (Vtail _ 1 v).
 Definition coordinates (v:ℝ^dim) :=(clock v,temperature v).
 Definition 𝒮₀ (s:𝒮) : Prop :=
   let (d,v):=s in d = Heat ∧ clock v=0 ∧ 5≤temperature v≤10.
 Definition ℐ (d:𝒟𝒮) (v:ℝ^dim):Prop:=
   d=Heat∧ (clock v ≤ 3) ∧ (temperature v ≤ 10) ⋁
   d=Cool∧ (5 ≤ temperature v) ⋁
   d=Check ∧ (clock v ≤ 1).
    ⋮
 Lemma ϕ_invertible:is_true_∀_coord _ _ ϕ
                        (λf∃g, ∀d x r t, f d x t = r → g d x r = t).
End Thermostat_as_ℋ.
```

Note that this time instead of an axiom we have to prove a lemma. The proof in this case is easy and boils down to proving simple properties of `exp` and `ln` functions.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.
2. R. Alur, T. Dang, and F. Ivančić. Predicate abstraction for reachability analysis of hybrid systems. *ACM transactions on embedded computing systems (TECS)*, 5(1):152–199, 2006.
3. The Coq Development Team. *The Coq Proof Assistant Reference Manual, Version 8.1.* LogiCal Project, Dec. 2007. `http://coq.inria.fr/V8.1pl3/refman/index.html`, [cited 8 Jan. 2008].