Symbolic Reachability Analysis of Lazy Linear Hybrid Automata*

Susmit Jha, Bryan A. Brady, and Sanjit A. Seshia

EECS Department, UC Berkeley
{jha,bbrady,sseshia}@eecs.berkeley.edu

Abstract. Lazy linear hybrid automata (LLHA) model the discrete time behavior of control systems containing finite-precision sensors and actuators interacting with their environment under bounded inertial delays. In this paper, we present a symbolic technique for reachability analysis of lazy linear hybrid automata. The model permits invariants and guards to be nonlinear predicates but requires flow values to be constants. Assuming finite precision, flows represented by uniform linear predicates can be reduced to those containing values from a finite set of constants. We present an abstraction hierarchy for LLHA. Our verification technique is based on bounded model checking and k-induction for reachability analysis at different levels of the abstraction hierarchy within an abstraction-refinement framework. The counterexamples obtained during BMC are used to construct refinements in each iteration. Our technique is practical and compares favorably with state-of-the-art tools, as demonstrated on examples that include the Air Traffic Alert and Collision Avoidance System (TCAS).

1 Introduction

A hybrid system is a dynamical system which exhibits both discrete and continuous behavior. Hybrid automata [4] have proved to be useful mathematical structures for modeling systems comprising discrete transition systems interacting with continuous dynamical systems. However, it is clear that in any implementation of a hybrid automaton, the state of the dynamical system reported to the discrete controller is digitized with finite precision by sensors, and the output signals of the controller transmitted to its actuators are also of finite precision. Further, the controller can only observe continuous state variables at discrete time points. Hence, it is somewhat unrealistic to assume that the controller can interact with its environment continuously and with infinite precision.

The inherent discrete nature of a controller of a hybrid system has led to recent efforts [17, 2, 3, 1] towards studying the discrete time behavior of hybrid systems. A similar argument in favor of focusing on discrete time behavior is presented by Henzinger and Kopke [12]. *Lazy linear hybrid automata* (LLHA) [2, 3] model the discrete time behavior of hybrid systems having finite precision

* Supported in part by SRC contract 1355.001, NSF grants CNS-0644436 & CNS-0627734, and Microsoft Research. The first author was also supported by the Berkeley Fellowship for Graduate Studies from UC Berkeley.

and bounded delays in actuation and sensing. Further, their definition of LLHA allows nonlinear invariants and guards. However, the discrete behavior in this model depends on the sampling frequency of the controller as well as the precision of variables, and hence, the discretized representations are very large and any enumerative analysis would not be feasible for systems of appreciable size.

In this paper, we present a symbolic technique for reachability analysis of *lazy linear hybrid automata*. We make the following novel contributions:

1. On the theoretical side, we present an abstraction hierarchy for LLHA that can be used for reachability analysis within a counterexample-guided abstraction-refinement framework.
2. We give an implementation of a symbolic model checker for LLHA based on bounded model checking and k-induction that operates at any level of abstraction.
3. We demonstrate the scalability of our methods in comparison to other state-of-the-art tools on examples such as Automated Highway Control System (AHS) and the Air Traffic Alert and Collision Avoidance System (TCAS).

Related Work PHAver (Polyhedral Hybrid Automaton Verifyer) [11] is a tool for verifying safety properties of hybrid systems. It uses on-the-fly over-approximation to handle affine flows by iterative partitioning of the state space. PHAver considers a continuous time model unlike the discrete time semantics of LLHA. Our work is much more closer to the HYSDEL tool [17]. The discrete hybrid automata underlying the HYSDEL tool is formed by the connection of a finite state machine with a switched affine system through an interface. Our work is similar to HYSDEL in its considering an inertial interface between the digital and the continuous components of the hybrid system. Unlike our symbolic approach, HYSDEL uses numerical simulation for analysis. Further, our technique allows guards and invariants that use any computable function. HSolver [18, 8] also allows general constraints over variables as invariants and guards. It uses interval arithmetic to check whether trajectories can move over the boundaries in a rectangular grid. Our technique uses SAT-based decision procedures for finite-precision arithmetic to do a symbolic analysis instead of an enumerative analysis. Another closely related tool is HybridSAL [21, 20], which constructs discrete finite state abstractions for hybrid systems using predicate abstraction. The tool uses decision procedures and the SAL explicit state model checker. Our approach performs abstraction over the domain of variables, and uses symbolic model checking based on bit-vector decision procedures.

The examples used in this paper have been well-studied; for details on previous case studies, we refer the reader to the relevant references on TCAS [16, 15] and AHS [9, 14].

2 Lazy Linear Hybrid Automata

Definition 1. *A finite precision lazy linear hybrid automaton(LLHA) [3] is a tuple* $(X, V, init, flow, inv, E, jump, D, \epsilon, B, P)$. *The components of LLHA are as follows:*

- *Variables : A finite ordered set* $X = \{x_1, x_2, \ldots, x_n\}$ *of continuous variables.*

- *Control modes : A finite set V of control modes.*
- *Initial conditions : A labeling function init that assigns an initial condition to each control mode $v \in V$. The initial condition is a predicate over the variables in X.*
- *Flow: The possible values of rate of change of any variable in a control mode form a finite set of constant values. Let the set representing the legal flow values for variable x_i be denoted by $\dot{X}_i$. The predicate $flow(v) \equiv (\dot{x}_1 \in \dot{X}_1) \wedge (\dot{x}_2 \in \dot{X}_2) \ldots \wedge (\dot{x}_n \in \dot{X}_n)$ represents legal flows at location $v \in V$.*
- *Invariant condition : A labeling function inv that assigns an invariant condition to each control mode $v \in V$. The invariant condition $inv(v)$ is a convex predicate over the variables in X.*
- *Control switches : A set E of edges (v, v') from a source mode $v \in V$ to a target mode $v' \in V$. A function "$update_{(v,v')}$" associates a variable assignment to each control switch (v, v').*
- *Jump conditions : A labeling function jump that assigns a jump condition to each control switch $e \in E$. A jump condition from the control mode v to v', $\psi_{(v,v')}$ is a predicate over the variables in X.*
- *Delay parameters : $D = \{g, \delta_g, h, \delta_h\}$ is the set of delay parameters such that $0 \leq g \leq g + \delta_g < h \leq h + \delta_h \leq P$, where h denotes the sensing delay, g denotes the actuation delay and P is the sampling interval of the controller.*
- *Precision : ϵ_i is the precision of measurement of variable x_i.*
- *Range : $B_i = [B_{i_{min}}, B_{i_{max}}]$ is the range of the variable x_i.*
- *Period P represents the time period associated with the discrete controller i.e. control mode switches take place at times $T_0, T_1, T_2, \ldots$ where $T_{k+1} = T_k + P$.*

The lazy semantics of hybrid automata [2, 3] means that if a control mode switch took place at time T_k, then the delay in actuating a change in $flow$ lies between $[T_k + g, T_k + g + \delta_g]$. Similarly, a control decision made at time T_{k+1} is based on the values of variables read by the controller at some time in the interval $[T_k + h, T_k + h + \delta_h]$. The parameters δ_g and δ_h represent the bounded uncertainty in actuation and sensing delay respectively. Since the sampling frequency of any implementation of a hybrid automata is always finite, this model focuses on the discrete time behavior of the hybrid automata.

The precision ϵ_i depends on the accuracy of the sensors measuring x_i from the continuous dynamical system. Guards and state invariants are evaluated on the values of the x_i variables that have been rounded using the value of ϵ_i. The parameter B reflects the range of values which can be taken by a state variable associated with a fixed width register. Unlike the conventional definition of linear hybrid automata [12], invariants and guards in LLHA can be nonlinear.

The flows in linear hybrid automata are represented using convex linear predicates over *only* the rates of change of variables (also called uniform linear predicates [13]). Under the assumption of finite precision, such flows can be considered as set of constant values of rate of change of different continuous variables. Thus, LLHA can be used for representing hybrid systems with convex linear flows. (This point is further discussed in the Appendix.) Note that the above model is same as the one formulated by Agrawal and Thiagarajan [2, 3].

Definition 2. *A configuration of a hybrid automaton, with n continuous variables, is a n+1-tuple, $c = (s, x_1, x_2, \ldots, x_n)$ where $s \in V$ is the control mode,*

$x_1, x_2, \ldots, x_n$ *is the valuation of the continuous variables of the hybrid automaton.*

The semantics of a hybrid automaton describes its evolution in terms of change in configuration. We use the notation $c + \alpha$ to denote the configuration in which continuous state variables are incremented by α. Also, we extend the order relation on the continuous variables to configurations. We say that $c \leq c'$ if we know that $x_i \leq x_i'$ for each x_i in c and the corresponding x_i' in c'.

We define a symbolic collection of configurations as a state of the hybrid automaton and describe the evolution of the hybrid automaton in terms of change in its state. This definition is used in Section 4 to present the bounded model checking algorithm.

Definition 3. *A state of the hybrid automaton is a pair (v, ϕ) consisting of a control mode $v \in V$ and a predicate ϕ over the variables X. We identify that the state of a hybrid automaton can change in two ways - flow or jump.*

- *flow: The changed state of a hybrid automata due to flow at control mode v for time T is (v, ϕ^T), where*
  $\phi^T = \exists X_1 \exists \dot{X} \{(\phi \wedge inv(v))[X \leftarrow X_1] \wedge X = X_1 + \dot{X}T \wedge (\dot{X} \models flow(v)) \wedge inv(v)\}$.
- *jump: If (v, ϕ) is state of a system, and (v, v') is a control switch such that $\phi \models jump(v, v')$, then the state of the system can change to (v', ϕ') such that if $update_{(v,v')}$ was the update function over $Y \subseteq X$,*
  $\phi' = \exists Y_1 \{(\phi \wedge inv(v) \wedge \psi_{(v,v')})[Y \leftarrow Y_1] \wedge Y = update_{(v,v')}(Y_1)\}$,
  where $update_{(v,v')}$ is the update function over $Y \subseteq X$.

A state $s_2 = (v, \phi_2)$ is reachable from $s_1 = (u, \phi_1)$ *if and only if* there is a sequence of flow or jump transitions from s_1 to s_2.

3 Hierarchical Abstraction

We detail the theory underlying our hierarchical abstraction technique below. For brevity, proofs of some theorems have been omitted.

Agrawal and Thiagarajan [2, 3] use two fundamental quantities in their analysis. The *fundamental time interval* is Δ = G.C.D of $\{P, g, \delta_g, h, \delta_h\}$. The corresponding abstraction quantum is Γ = G.C.D of $\bigcup_i \{\epsilon_i/2, B_i^{min}, B_i^{max}, V_i^{in}, \dot{x}_i\Delta\}$.

Abstraction. We begin with basic definitions on how abstraction is performed. For ease of presentation, all variables are abstracted in the same way; the theory can be easily extended to a non-uniform abstraction.

Definition 4. *Q_Π is a surjection over the continuous variables using* abstraction quantum *$\Pi = 2^k\Gamma$ for some integer k. That is,*
$Q_\Pi : \mathbb{R} \to \mathbb{R}$, and $Q_\Pi(x_i) = k_i\Pi$ iff $x_i = k_i\Pi + \pi_i$, where $k_i \in \mathbb{Z}$ and $0 \leq \pi_i < \Pi$.

Abstract Configuration: A configuration $c^d = (s^d, x_1^d, x_2^d, \ldots, x_n^d)$ is a Π-abstraction of a concrete configuration $c = (s, x_1, x_2, \ldots, x_n)$ iff $s^d = s$ and $x_i^d = Q_\Pi(x_i)$.

Abstract Transition: Transitions are abstracted by abstracting jump and flow conditions. This must be done in order to ensure that transitions that are feasible in the concrete LLHA continue to be feasible in the abstract transition system, at the possible cost of introducing additional (spurious) behaviors.

1. The intuition behind the following definition of abstract guards and invariants is to relax the atomic constraints so that if $\Phi(x_1, x_2, \ldots, x_n)$ denotes a state invariant or guard, then the corresponding abstracted invariant or guard is $\Phi^a(x_1, x_2, \ldots, x_n)$ such that $\Phi(x_1, x_2, \ldots, x_n) \implies \Phi^a(x_1, x_2, \ldots, x_n)$.
2. The set of flow values are abstracted to overapproximate the reachable configurations. If the flow value in a set $\dot{X}$ is $\dot{x}$, it is abstracted by including flow values $\dot{x}^a$ and $\dot{x}^b$ in its place, where $\dot{x}^a \leq \dot{x} \leq \dot{x}^b$ (details given below).

We first describe how invariants and guards are abstracted, and then describe the over-approximation of flow.

Abstraction of invariants and guards. Invariants or guards can be expressed as a Boolean combination of atomic predicates in negation normal form (NNF), where each predicate is of the form $f(x_1, x_2 \ldots, x_n) \leq b$ where $b \in \mathbb{Q}$. If Φ is an invariant or guard, then $\Phi = f_{bool}(c_1, c_2, \ldots, c_n)$ where the constraint c_i is $f_i \leq b_i$ and where f_{bool} represents an NNF Boolean combination of its arguments.

Each predicate in the invariant or guard can be abstracted using the monotonicity of f with respect to each variable x_i, that is, $f_{x_i} = \frac{\delta f}{\delta x_i}$ is of the same sign over the range of interest. In particular, all polynomials which are linear in each variable, are always monotonic with respect to each variable.

In order to define abstract state invariants and guards, we first describe how to construct abstract inequalities using the above observation about invariants and guards. Without loss of generality, let us assume that $f(x_1, x_2 \ldots, x_n) \leq b$ is an inequality whose partial derivative f_{x_i} with respect to each variable x_i is of the same sign over the range of interest $[Q_\Pi(x_i), Q_\Pi(x_i + \Pi)]$. Then, its (conservative) abstraction is the *relaxed* inequality c'_i defined below:

$$c'_i \equiv f(k_1, k_2 \ldots, k_n) \leq b'$$
$$\text{where } b' = Q_\Pi(b + \Pi) \qquad \text{and} \qquad \begin{aligned} k_i &= Q_\Pi(x_i) \text{ if } f_{x_i} \geq 0 \\ &= Q_\Pi(x_i + \Pi) \text{ if } f_{x_i} < 0 \end{aligned}$$

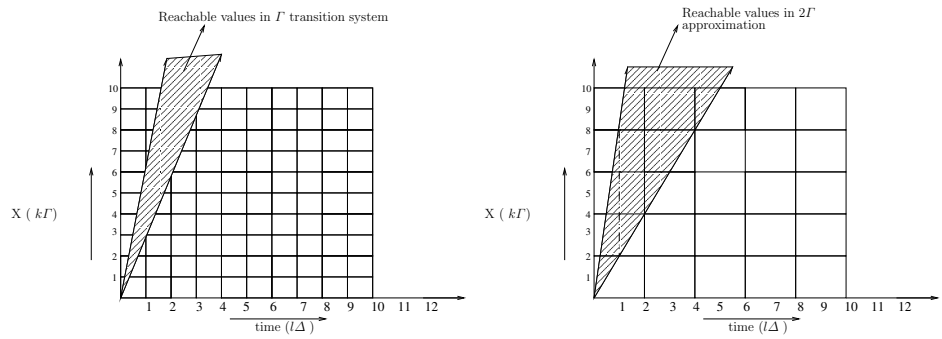
This abstraction rounds up or down each variable to the nearest multiple of Π depending on whether the function f decreases or increases with increase in the variable. The constant b is always rounded up. All assignments to the variables which satisfied the earlier constraint also satisfy the *relaxed* constraint. Hence, this is an overapproximation of the original constraint.

If $\Phi(x_1, x_2, \ldots, x_n) = f_{bool}(f_1 \leq b_1, \ldots, f_n \leq b_n)$ is the invariant or guard, the abstract state invariant or guard is defined as
$\Phi^a(k_1, k_2, \ldots, k_n) = f_{bool}(c'_1, c'_2, \ldots, c'_n)$
where the relaxed inequalities c'_i are obtained from $f_i \leq b_i$ as described above.

Thus, this relaxation results into an upper approximation of the behavior of the hybrid automaton.

Abstraction of flow conditions. If $\dot{x}$ is a rate of change allowed by $flow(s)$ for some location s, then the following two rates of change represent its abstraction $\lfloor(\frac{\dot{x}}{\Pi})\rfloor\Pi$ and $\lceil(\frac{\dot{x}}{\Pi})\rceil\Pi$. Figures 1(a) and 1(b) illustrate how flow conditions

are abstracted. The abstraction of flow with 2Γ leads to an overapproximation of the dynamics of the LLHA: originally $\dot{x} \in \{3, 4, 5, 6\}$, but in the 2Γ-abstraction $\dot{x} \in \{2, 4, 6, 8\}$.



(a) 0-abstraction (b) 1-abstraction

Fig. 1. Illustration of flow abstraction

Definition 5. *A k-abstraction ($k \geq 1$) of a lazy linear hybrid automaton is an abstraction of LLHA obtained using the above explained abstraction of configurations and transitions such that $\Pi = 2^k\Gamma$. The 0-abstraction is called the Γ-transition system as the quantization is done with respect to Γ.*

We define a partial order relation $\preceq$ between transition systems below.

Definition 6. *Let TS and TS' be two transition systems such that every state of TS is mapped to some state of TS'. If every state of TS reachable from some initial state of TS has its corresponding state in TS' also reachable from an initial state of TS', then $TS \preceq TS'$.*

Prior Results. Our model of LLHA is the same as that of Agrawal and Thiagarajan [3], who initially consider a model with constant flow rate and linear invariants, and later extend the result to invariants and guards which are any "reasonable computable function". The main result of theirs which we utilize is summarized in Theorem 1.

Theorem 1. *Let a configuration of hybrid automata be $c = (s, x_1, x_2, \ldots, x_n)$ and its Γ-abstract configuration be $c^d = (s, Q_\Gamma(x_1), Q_\Gamma(x_2), \ldots, Q_\Gamma(x_n))$. A configuration c' is reachable from c iff $Q_\Gamma(c') = c'^d$ where c'^d is reachable from c^d in Γ-transition system.*

Let x_{max} and x_{min} be the maximum and minimum values that can be attained by any continuous variable and m be the number of control modes. The state space size of the Γ-transition system is $O(m^4 2^{2n}(\frac{x_{max}-x_{min}}{\Gamma})^{3n})$ [3], that is, exponential in the number of continuous variables. This huge state space makes it impractical to do any enumerative reachability analysis.

Our Results. The main result is that the *k-abstraction* of LLHA simulates the original LLHA. Further, for increasing values of k, we obtain coarser overapproximations of the LLHA which form a hierarchy of sound abstractions. Figure 2 illustrates the meaning of Theorem 2.

Theorem 2. *Let a configuration of hybrid automata be* $c = (s, x_1, x_2, \ldots, x_n)$ *and its abstraction be* $c^d = (s, Q_\Pi(x_1), Q_\Pi(x_2), \ldots, Q_\Pi(x_n))$, *where* $\Pi = 2^k\Gamma$. *If a configuration*

c' is reachable from c in time $T = l\Delta$ and $Q_\Pi(c') = c'^d$, then c'^d is reachable from c^d in the k-abstraction.

Proof. For configuration $c = (s, x_1, x_2, \ldots, x_n)$, let $\dot{x}_1, \dot{x}_2 \ldots, \dot{x}_n$ be the rates of change of continuous variables satisfying $flow(s)$ and $\widehat{\dot{x}_1}, \widehat{\dot{x}_2}, \ldots, \widehat{\dot{x}_n}$ be the rates of change of continuous variables satisfying $flow(\widehat{s})$ where $\widehat{s}$ is a predecessor state of s, that is, $(\widehat{s}, s) \in E$. Let c' be a configuration reachable from c. In case of change due to reset of variables at jump, the above theorem follows due to the adjustment to guards and invariants. We prove the above theorem for the case where the change is effected due to flow evolution.

Since, the relation $\leq$ for configurations is defined in terms of the ordering of individual variable, we consider an arbitrary variable in the rest of the proof below. If x_i is the value of the variable in c and x'_i is the value in c' after time T such that the flow rate switched after an actuation delay of t, then
$x'_i = x_i + \widehat{\dot{x}_i}t + \dot{x}_i(T - t)$
Using the definition of Γ and Δ,
$x_i = (m2^k + n)\Gamma + \gamma_i$, $\widehat{\dot{x}_i}\Delta = (2^k p' + q')\Gamma$, and $\dot{x}_i\Delta = (2^k p + q)\Gamma$,
where $0 \leq n < 2^k$, $0 \leq \gamma_i < \Gamma$, $0 \leq q' < 2^k$, $0 \leq q < 2^k$.

So, $x'_i = (m2^k + n)\Gamma + \gamma_i + (2^k p' + q')\frac{\Gamma}{\Delta}t + (2^k p + q)\frac{\Gamma}{\Delta}(l\Delta - t)$
$= (m2^k + n)\Gamma + \gamma_i + (2^k(p' - p) + (q' - q))\frac{\Gamma}{\Delta}t + (2^k p + q)l\Gamma$
Thus, $x'_i = (m + pl)2^k\Gamma + (n + ql)\Gamma + \gamma_i + (2^k(p' - p) + (q' - q))\frac{\Gamma}{\Delta}t$.

Since $0 \leq t < T$ in the above equation and $2^k\Gamma = \Pi$, x'_i lies in the interval

- $[(m + pl)\Pi, (m + (p' + 1)l + 1)\Pi)$ if $\dot{x}_i \geq \widehat{\dot{x}_i}$
- $[(m + p'l)\Pi, (m + (p + 1)l + 1)\Pi)$ if $\widehat{\dot{x}_i} \geq \dot{x}_i$

So, $Q_\Pi(x'_i)$ lies in the interval

- $[(m + pl)\Pi, (m + (p' + 1)l)\Pi]$ if $\dot{x}_i \geq \widehat{\dot{x}_i}$
- $[(m + p'l)\Pi, (m + (p + 1)l)\Pi]$ if $\widehat{\dot{x}_i} \geq \dot{x}_i$

The value of i^{th} variable in any configuration c'^d reachable from c^d in the k-abstraction, x'^d_i lies in

- $[(m + pl)\Pi, (m + (p' + 1)l)\Pi]$ if $\dot{x}_i \geq \widehat{\dot{x}_i}$
- $[(m + p'l)\Pi, (m + (p + 1)l)\Pi]$ if $\widehat{\dot{x}_i} \geq \dot{x}_i$

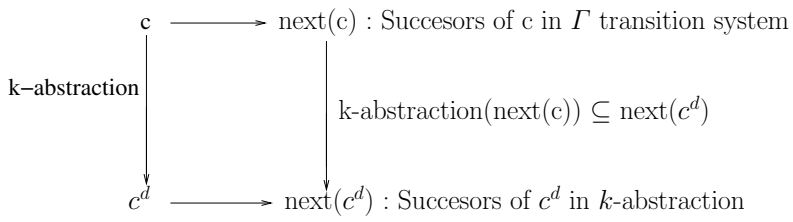


Fig. 2. Simulation by k-abstraction

Thus, for any x'_i, there exists x'^d_i such that $x'^d_i = Q_{\Pi}(x'_i)$. Using the same argument for each variable independently, the theorem immediately follows. □

Using reasoning exactly similar to the one used in Theorem 2, we can prove the hierarchy of k-abstractions presented below.

Lemma 1. *Let a configuration of k-abstraction be*
$c = (s, Q_{\Pi}(x_1), Q_{\Pi}(x_2), \ldots, Q_{\Pi}(x_n))$, *where* $\Pi = 2^k\Gamma$.
Its abstraction in $\widetilde{k}$-abstraction, where $\widetilde{k} \geq k$
$\widetilde{c} = (s, Q_{\widetilde{\Pi}}(x_1), Q_{\widetilde{\Pi}}(x_2), \ldots, Q_{\widetilde{\Pi}}(x_n))$ *where* $\widetilde{\Pi} = 2^{\widetilde{k}}\Gamma$.

If a configuration c' is reachable from c in k-abstraction, then

- $Q_{\Pi'}(c') = \widetilde{c}'$ *where* $\Pi' = 2^{\widetilde{k}-k}\Gamma$
- $\widetilde{c}'$ *is reachable from $\widetilde{c}$ in $\widetilde{k}$-abstraction.*

The Hierarchy Theorem 3 follows from Lemma 1 and Theorem 1.

Theorem 3. *k-abstraction $\preceq$ k'-abstraction if $0 \leq k < k'$. Thus, k-abstractions, where $k \geq 0$, form an hierarchical abstractions of the lazy linear hybrid automata. Further, 0-abstraction is the Γ-abstract transition system which bisimulates the original lazy linear hybrid automaton.*

Theorem 3 provides a framework for use of progressive abstraction of lazy linear hybrid automata to develop a sound and complete abstraction-refinement paradigm for reachability analysis of LLHA. Theorem 4 presents the relative reduction in state space size with k.

Theorem 4. *Let S_k be the state space size of k-abstraction and S'_k of k'-abstraction where $k' > k$. Then $log_2(S'_k/S_k) = 3n(k - k')$ where n is the number of continuous variables.*

4 Model Checking *k*-abstractions of LLHA

Our implementation of a symbolic verifier of LLHA is based on three techniques: bounded model checking, "k-induction", and an overall counterexample-guided abstraction-refinement [7] framework. We describe each of these below.

We first present a symbolic representation of k-abstraction of the hybrid automata as stated in Definition 5. The continuous variables $X = (x_1, x_2, \ldots, x_n)$ are symbolically represented with integer variables $K = (k_1, k_2, \ldots, k_n)$ with the intended mapping being $Q_{\Pi}(x_i) = k_i\Pi$, where $\Pi = 2^k\Gamma$.

In the discussion below, we use the three components - guards (Ψ_{ij}), invariants (inv_i) and the flow conditions $flow(i)$ of the k-abstraction to define a symbolic transition relation TR. This is then used to describe the bounded model checking and inductive verification techniques.

Bounded model checking: We describe how the BMC formula is constructed, starting with a useful definition.

Definition 7. *A frame (F) is a tuple (K, t_1, t_2, t, l) where $K = (k_1, k_2 \ldots k_n)$ represent the variables; t_1 is the sensing delay; t_2 is the actuation delay t_2; t is the time before transition to next frame; l denotes the control mode.*

The initial state of the hybrid automata is the predicate $Init(F_0) \equiv (l = v_{start}) \wedge \phi_0(K)$, where v_{start} denotes the initial control mode and ϕ_0 the initial predicate over continuous variables.

The transition TR is defined as a predicate over the previous frame (F_{m-1}) and the present frame (F_m). It is a disjunction of all possible state switches (G_{ij}) and flow evolutions (E_i).

$$TR(F_{m-1}, F_m) \equiv \bigvee_{(i,j)\in E} G_{ij}(F_{m-1}, F_m) \vee \bigvee_{i\in V} E_i(F_{m-1}, F_m)$$

The switch predicates G_{ij} and the time evolution predicates E_i are defined in terms of three other quantities: I_i is a predicate that tests satisfiability of state invariant inv_i at control mode i, predicate g_{ij} tests satisfiability of guard ψ_{ij}, and e_{hi} deals with time evolution in control mode i with predecessor mode h.

Let us consider two functions - *compensated for sensing delay (csd)* and *compensated for actuation delay (cad)*. These map a set of valuations of the continuous variables (K) to a set of possible corresponding valuations obtained after compensating for sensing and actuation delay respectively.

$csd(K, i, t_1) = \{(k_1 - \dot{k}_1 t_1, \ldots, k_n - \dot{k}_n t_1) \mid (\dot{k}_1, \dot{k}_2, \ldots \dot{k}_n) \models flow(i)\}$.

$cad(K, h, i, t_2, t) = \{(k_1 + (\dot{k}_{1h} - \dot{k}_{1i})t_2 + \dot{k}_{1i}t, \ \ldots, \ k_n + (\dot{k}_{nh} - \dot{k}_{ni})t_2 + \dot{k}_{ni}t)$
$\mid (\dot{k}_{1h}, \dot{k}_{2h}, \ldots \dot{k}_{nh}) \models flow(h)$ and $(\dot{k}_{1i}, \dot{k}_{2i}, \ldots \dot{k}_{ni}) \models flow(i)\}$.

Let the current frame be $F_m = (K^m, t_1^m, t_2^m, t^m, l^m)$ and the previous frame be $F_{m-1} = (K^{m-1}, t_1^{m-1}, t_2^{m-1}, t^{m-1}, l^{m-1})$.

$$\begin{aligned} I_i(F_m) &\equiv (i = l^m) \wedge \exists K'[K' \in csd(K^m, l^m, t_1^m) \wedge inv_i(K')] \\ e_{hi}(F_{m-1}, F_m) &\equiv (i = l^{m-1} \wedge i = l^m) \wedge K^m \in cad(K^{m-1}, h, i, t_2^{m-1}, t^m) \\ g_{ij}(F_{m-1}, F_m) &\equiv (i = l^{m-1} \wedge j = l^m) \wedge \exists K'[K' \in csd(K^{m-1}, l^{m-1}, t_1^{m-1}) \wedge \psi_{ij}(K')] \end{aligned}$$

Note that the existential quantification over K' in the above identities simply reduces to a disjunction over possible flow values (see the description of *cad* and *csd* functions).

The switch and evolution predicates can now be defined as follows:

$$G_{ij}(F_{m-1}, F_m) \equiv I_i(F_{m-1}) \wedge I_j(F_m) \wedge g_{ij}(F_{m-1}, F_m) \wedge [K^m = update_{ij}(K^{m-1})]$$

$E_i(F_{m-1}, F_m) \equiv I_i(F_{m-1}) \wedge I_i(F_m) \wedge [\bigvee_{h \in pred(i)} e_{hi}(F_{m-1}, F_m)]$
where $pred(i)$ denotes the set of predecessor locations of i.

This completes the definition of the transition predicate.

Let the state to be checked for reachability be (s_r, ϕ_r). If reachability analysis is used to check safety properties, then (S_r, ϕ_r) would be the error state violating the safety property, Then, the predicate $unsafe(F) \equiv (l = s_r \wedge \phi_r(K))$ represents the error state, that is the *target* state for reachability analysis.

If d is the number of steps to which we want to check the k-abstraction for reachability of (s_r, ϕ_r), we need to check for the satisfiability of

$$BMC^d \equiv Init(F_0) \wedge \bigwedge_{n=1}^{d} (TR(F_{n-1}, F_n)) \wedge unsafe(F_d).$$

If BMC^d is satisfied, then the target state $(s_r, \phi_r(K))$ is reachable in k-abstraction and the frames $F_0, F_1, \ldots, F_d$ gives a trace from the start state to the target state.

Further, it is sufficient to do BMC for p steps to prove that a target state is not reachable where p is the diameter of the transition system. If BMC^j is unsatisfiable for all $j \leq p$, then the target state can not be reached in the transition system. Since the number of reachable states of the transition system provides an over-estimate of the diameter, it is sufficient (though unrealistic) to do BMC for number of steps equal to the state space size of the k-abstraction.

Induction: We now describe an induction procedure to guarantee the unreachability of a state in a model. This can be used to prove the satisfaction of a safety property which can be expressed as a reachability query.

If N steps of BMC are found to be not satisfiable, that is, BMC^N is unsatisfiable, then we test the satisfiability of

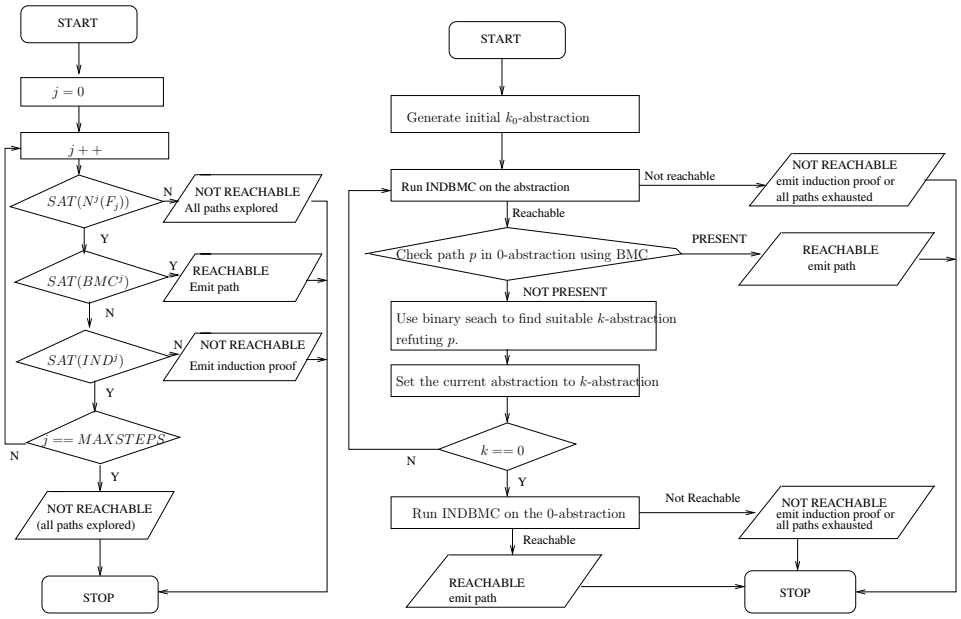
$$\neg unsafe(F_0) \wedge \bigwedge_{k=1}^{N+1} (TR(F_{k-1}, F_k)) \wedge unsafe(F_{N+1}).$$

If the above is unsatisfiable, no further bounded model checking is required and all the states of the model are guaranteed to satisfy the property. Based on this, we present below a BMC algorithm along with use of induction to check for safety properties in a LLHA. We define the following predicates to be used in the algorithm.

$N^j(F_j) \equiv Init(F_0) \wedge \bigwedge_{k=1}^{j} TR(F_{k-1}, F_k)$ and $S^{j+1}(F_{j+1}) \equiv \neg unsafe(F_0) \wedge \bigwedge_{k=1}^{j+1} TR(F_{k-1}, F_k)$

If at any step of the BMC, we find that $N^j(F_j)$ is not satisfiable, it means that there does not exist a path of length j or more, and hence we can terminate with the output that the model satisfies the safety property.

The bounded model checking predicate and the induction step predicate are $BMC^j \equiv N^j(F_j) \wedge unsafe(F_j)$ and $IND^j \equiv S^{j+1}(F_{j+1}) \wedge unsafe(F_{j+1})$



(a) INDBMC module (b) Iterative refinement

Fig. 3. Symbolic reachability analysis based on BMC and induction

The sub-routine INDBMC is presented in Figure 3(a). The technique is sound and complete due to the results of the preceding section; we present a detailed discussion of the abstraction-refinement framework in the next section.

Counterexample guided refinement of k-abstractions: We now describe an automated CEGAR [7] technique presented in Figure 3(b) which exploits the linear abstraction hierarchy presented in section 3. An initial coarse abstraction can be arbitrary chosen as k_0-abstraction depending on the size of the state space. In case the target state is not reachable in k_0-abstraction, the target state is also not reachable in the LLHA by Theorem 2. In case the target state is reachable in LLHA, then BMC will yield a path p_0 from the initial state to the target state in the k_0-abstraction. This needs to be validated with respect to the 0-abstraction. If the abstract path p_0 found in k_0-abstraction is present in 0-abstraction, then the target state is reachable in the LLHA too. If it is not present in 0-abstraction, then we select a more finer refinement k_i-abstraction which refutes the abstract spurious path. The same technique is repeated for progressively finer abstractions until the target states is shown to be unreachable or a valid path to the target sate is found. The key components of this technique are counter-example validation technique and automated refinement step. The path obtained at any iteration is a satisfying assignment to BMC^j,

it can be validated on 0-abstraction by doing a BMC on BMC^j to identify the first spurious transition. If BMC^j has a satisfying assignment for 0-abstraction too, then the path is valid. The hierarchy of abstractions allows the use of binary search to find the smallest value of l such that the l-abstraction refutes the path identified in the coarser abstraction. The complete technique for reachability analysis of LLHA based on iterative refinement and bounded model checking is presented as a flowchart in Figure 3(b). The soundness of this technique is ensured by Theorem 2 and 1. Since, we start with some initial k_0-abstraction and every step involves a progress in refinement, we take at most k_0 iterations before terminating. For the iteration considering k-abstraction, $MAXSTEPS$ would be the diameter of the k-abstraction. In worst case, this algorithm needs to consider 0-abstraction which can have a very large diameter and the BMC of this transition system can be unrealistic, but the completeness of our technique is guaranteed.

Theorem 5. *The iterative abstraction refinement technique presented in Figure 3(b) is sound and complete.*

5 Experimental Results

In this section, we present the results of experiments on two case studies. [1] All experiments were performed on a workstation with Intel Xeon 3.06 GHz processors and 4GB RAM. UCLID bit-vector decision procedure [5] was used with MiniSat as the underlying SAT engine. Any other bit-vector decision procedure could alternatively be used as the verification engine in our technique.

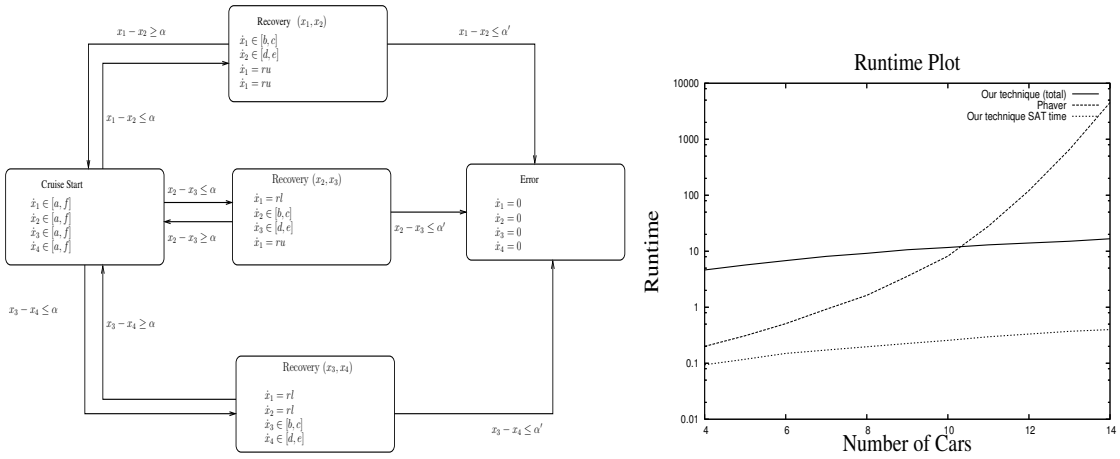
Automated Highway Control System:

AHS (Figure 4(a)) is an arbiter which ensures that there is no collision between cars running on a highway by imposing legal speed ranges. This example has being widely used in literature [9, 14]. We use the description by Jha et al [14] and extend it to handle inertial delays. The number of cars is used as a parameter to scale the example.

A set of legal parameter values is:
(All distance measures are in km, time is in hr and all speeds are in km/hr)
$\alpha = .002$, $\alpha' = .0005$, $a = 10, rl = 20, b = 30, c = 40, d = 50, e = 60, ru = 70, f = 100$
$\epsilon = 10^{-5}$, $g = 10^{-3}$, $h = 5 \times 10^{-4}$, $\delta_g = 5 \times 10^{-4}, \delta_h = 5 \times 10^{-5}$ and $P = .01$.
Correspondingly, quantization factors are $\Delta = 5 \times 10^{-5}$ and $\Gamma = 5 \times 10^{-5}$.

The safety property to be verified was that the control mode is never the "error" mode. Figure 4(b) compares the runtime of our technique and that of Phaver on this example for different number of cars. It shows that our approach is more scalable than Phaver. Our technique could handle large instances with 150 cars in less than 2 minutes while Phaver took more than 10 hours to analyze model with 15 cars. For this example, *we did not do any abstraction.* Γ-abstraction for AHS with even large number of cars could be easily handled by our BMC+induction technique and did not necessitate any abstraction-refinement iteration as shown by the runtime plot in Figure 4(b). Further, the bulk of the run-time taken by

[1] A complete set of UCLID, Phaver or HSolver modules as well as data pertaining to run-time and memory requirements can be obtained from the first author's webpage.



(a) Model (b) Runtime plot

Fig. 4. Automated Highway Control System with 4 vehicles

our technique is used up in building the model. The time taken to solve the corresponding SAT problems for BMC and Induction are a very small percentage of total run time.

Air Traffic Alert and Collision Avoidance System:

TCAS is a predictive warning system used for avoiding collision of aircrafts using a sequence of preventive and corrective resolution advisories. The model for TCAS resolution used here is similar to the one used by Pappas et al [16]. We make a few changes to the model to make it more realistic. The TCAS specification [6] uses *expected time to collision* for detecting collision threats and not distance between aircrafts used in Pappas et al example [16]. The max in the constraint avoids division by zero. The k/x_r term ensures that slow approaches are avoided by triggering threat if x_r is small. This makes the problem harder since these invariants are non-linear. Hence, LHA tools like Phaver can not be used for this example.We allow the input for speed of aircrafts to be an interval. It is realistic to expect the speed of aircrafts to be in a range rather than assuming them to be a constant input. We also allow inertial delays in actuation and sensing.

The parameters used in the experiment were taken from the specifications in TCAS 2, Version 7 documentation [10] and TCAS-201 simulator [19] specifications. The time-zone considered for advisory is $30 - 120$ seconds (t_{near} and t_{far}, respectively). The distance d is taken to be 15 nautical miles (that is, 27.78 kms.). The range of speed for aircraft is allowed to range between 100 knots to 510 knots (nearly 200 km/hr to 1000 km/hr). It may be noted that the maximum

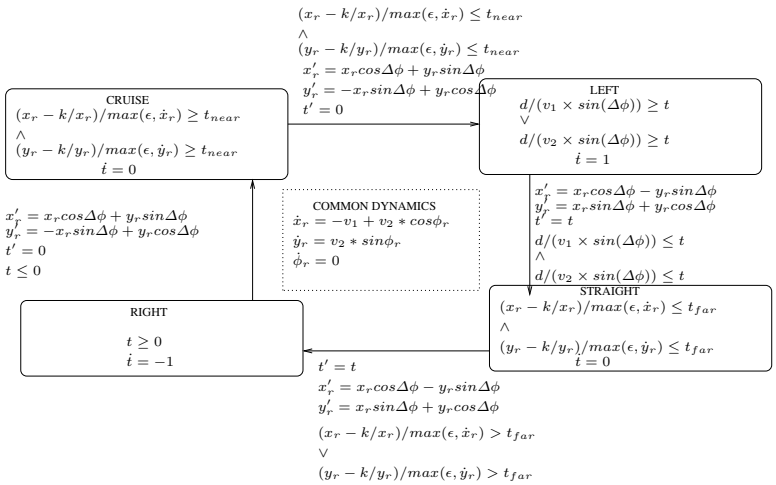
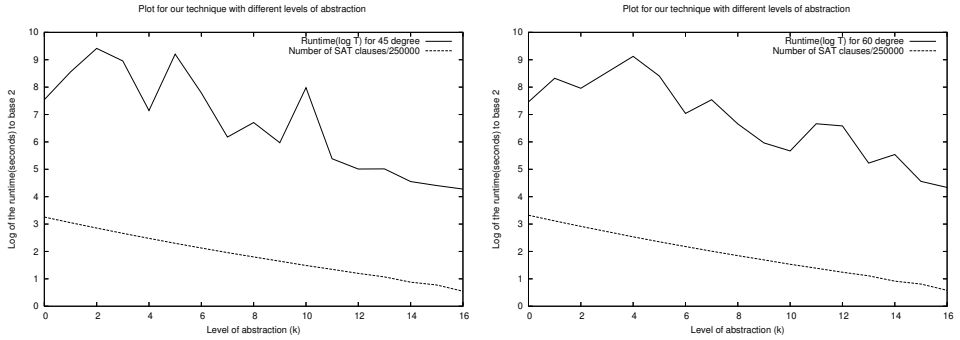


Fig. 5. Air Traffic Alert and Collision Avoidance System

speed of Airbus 380 is Mach 0.88 (nearly 505 knots). The LLHA parameters used in our example are $128\mu s \leq g, h \leq 256\mu s$ and $\epsilon = 2^{-15}$ nautical miles [19].



(a) $\Delta\phi = 45$ degree (b) $\Delta\phi = 60$ degree

Fig. 6. Plot comparing runtimes of our technique for different levels of abstraction

Phaver cannot handle this example due to non-linear invariants and guards. We modeled TCAS example using same LLHA parameter values in HSolver [8]. It did not terminate in 180 minutes with any answer. This further stresses the *hardness* of this example and underlines the significance of our technique's scalability. Figure 6 depicts how the run-time of our tool and state space size vary for different levels of abstraction (the x-axis gives the value of k for k-abstraction).

There is an initial increase due to addition of extra flows but for larger abstractions, the time taken is much less compared to the actual model. Since no refinement is needed for any value of k, the points in the graph represent runtime for only a particular abstraction level. The analysis of the 16-abstraction of the original model allows us to conclude in less than 20 seconds that the model is safe, about 10 times faster than analyzing the original model (0-abstraction).

References

1. M. Agrawal, F. Stephan, P. S. Thiagarajan, and S. Yang. Behavioural approximations for restricted linear differential hybrid automata. In J. P. Hespanha and A. Tiwari, editors, *HSCC*, volume 3927 of *LNCS*, pages 4–18. Springer, 2006.
2. M. Agrawal and P. S. Thiagarajan. Lazy rectangular hybrid automata. In R. Alur and G. J. Pappas, editors, *HSCC*, volume 2993 of *LNCS*, pages 1–15. Springer, 2004.
3. M. Agrawal and P. S. Thiagarajan. The discrete time behavior of lazy linear hybrid automata. In M. Morari and L. Thiele, editors, *HSCC*, volume 3414 of *LNCS*, pages 55–69. Springer, 2005.
4. R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems I*, LNCS 736, pages 209–229, 1992.
5. R. E. Bryant, D. Kroening, J. Ouaknine, S. A. Seshia, O. Strichman, and B. Brady. Deciding bit-vector arithmetic with abstraction. In *Proceedings of TACAS 2007*, volume 4424 of *LNCS*, pages 358–372. Springer Verlang, 2007.
6. W. Chan, R. Anderson, P. Beame, and D. Notkin. Combining constraint solving and symbolic model checking for a class of a systems with non-linear constraints. In *CAV*, pages 316–327, 1997.
7. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV '00*, pages 154–169, London, UK, 2000. Springer-Verlag.
8. W. Damm, G. Pinto, and S. Ratschan. Guaranteed termination in the verification of LTL properties of non-linear robust discrete time hybrid systems. In *ATVA*, pages 99–113, 2005.
9. A. Deshpande, D. N. Godbole, A. G, and P. Varaiya. Design and evaluation tools for automated highway systems. In *Hybrid Systems*, pages 138–148, 1995.
10. Federal Aviation Administration. Introduction to TCAS II Version 7, November, 2000. http://www.arinc.com/downloads/tcas/tcas.pdf.
11. G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In *HSCC*, pages 258–273, 2005.
12. T. A. Henzinger and P. W. Kopke. Discrete-time control for rectangular hybrid automata. *TCS*, 221(1–2):369–392, 1999.
13. P.-H. Ho. *Automatic analysis of hybrid systems*. PhD thesis, Cornell Univ., 1995.
14. S. K. Jha, B. H. Krogh, J. Weimer, and E. M. Clarke. Iterative relaxation abstraction (IRA) for linear hybrid automata. In *HSCC'07*, LNCS, 2007.
15. C. Livadas, J. Lygeros, and N. A. Lynch. High-level modeling and analysis of tcas. In *RTSS '99*, page 115, Washington, DC, USA, 1999. IEEE Computer Society.
16. G. Pappas, C. Tomlin, and S. Sastry. Conflict resolution for multi-agent hybrid systems. In *CDC*, pages 1184–1189, 1996.
17. B. Potocnik, A. Bemporad, F. Torrisi, G. Music, and B. Zupancic. Hysdel Modeling and Simulation of Hybrid Dynamical Systems. In *MATHMOD Conference*, Vienna, Austria, Feb. 2003.

18. S. Ratschan and Z. She. Constraints for continuous reachability in the verification of hybrid systems. In *AISC*, pages 196–210, 2006.
19. TCAS201 Specification Datasheet. http://www.aeroflex.com/products/avionics/rf/datasheets/tcas201.pdf.
20. A. Tiwari. Approximate reachability for linear systems. In *HSCC*, volume 2623 of *LNCS*, pages 514–525. Springer, Apr. 2003.
21. A. Tiwari and G. Khanna. Series of abstractions for hybrid automata. In C. Tomlin and M. R. Greenstreet, editors, *HSCC*, volume 2289 of *LNCS*, pages 465–478. Springer, 2002.

APPENDIX

Constant Differential Inclusion We provide an intuitive explanation of how constant differential inclusion of uniform linear predicates [13] are used in literature to model linear flows for reachability analysis. Such a transformation is sufficient for determining reachability as we do not reason about the time taken to reach a configuration.

This is illustrated in Figure 7. The convex polygon represents the rate of change of variables (x, y) such that is (r_x, r_y) is a permitted flow, then $(r_x t, r_y t)$ lies in the interior of the convex polygon, where t is one time unit.. The polygon is convex as we only consider convex linear flows. Also, the polygon is constant and does not change with change in configurations (x, y) as the flow condition is uniform and does not depend on (x, y). The configurations (x, y) reachable in k time units would be $(x_0 + k(r_x t), y_0 + k(r_y t))$, where (x_0, y_0) is initial configuration, $(r_x t, r_y t)$ is a point in the convex polygon. Thus, the two rays with (x_0, y_0) as origin and touching the angular extremes of the convex polygon represent the possible reachable configurations. If we form a rectangle using the vertices of the polygon that touch the rays, the set of configurations reachable for flow values lying in the rectangle is same as the previously reachable sets. The bound constraints arising from the rectangle can be, thus, used in place of the uniform linear predicate for the purpose of reachability analysis.

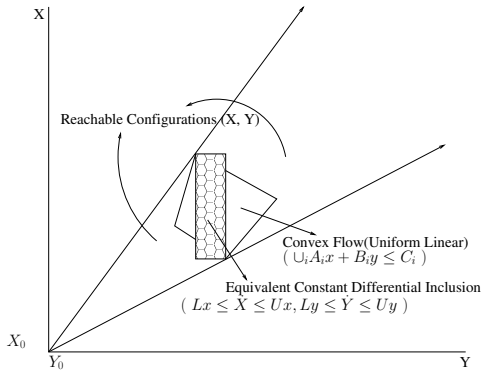


Fig. 7. The flows given as uniform convex linear predicates can be represented using constant differential inclusion for the purpose of reachability.