Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 02-013

Domain Reduction Abstraction

Yunja Choi, Mats P. Heimdahl, and Sanjai Rayadurgam

April 03, 2002

Domain Reduction Abstraction *

Yunja Choi, Mats Heimdahl, Sanjai Rayadurgam

Computer Science and Engineering, University of Minnesota,
Minneapolis, MN 55455, USA
{yuchoi, heimdahl, rsanjai}@cs.umn.edu

Abstract. We suggest *domain reduction abstraction* for model checking systems with numeric guarding conditions and data transition constraints. The technique abstracts the domain of data variables using *data equivalence* and *trajectory reduction* in order to reduce the (possibly infinite) state space.
Earlier work introduced the technique for systems with no data constraints or deterministic data constraints. Here, we extend the work to *non-deterministic constrained data transition systems*. We provide a formal definition and proof of the soundness of the technique, and illustrate the abstraction technique with a small example.

Keyword: Domain abstraction, model checking software systems, constrained data transition systems, numeric constraints

1 Introduction

Model checking has been successful in checking properties of various finite state systems, but the usefulness is limited by the size of the system state space. Especially, the capability to verify software systems has been limited since software systems are likely to have infinite, or finite but very large, state spaces.

Important classes of software systems can be viewed as consisting of a finite *control component* and a (typically infinite or very large) *data component*. In such systems, the transitions between control variables are guarded by various linear and non-linear data conditions, and the transitions between data values may be subject to various constraints which can be non-deterministic. For example, a temperature control system can have the guarding condition $temp < 10$ for a control transition and $temp' = temp + [-\alpha, \beta]$ as a constraint for the temperature change.

Constraint-free data transition systems allow random change of data variables while *constrained data transition systems* constrain the way data variables change their value between pre-state and post-state, such as the data constraint $x' = x + 1$. In a previous paper [5] we presented domain reduction abstraction based on *data equivalence* and *trajectory reduction* for constraint-free data transition systems and *deterministic* constrained data transition systems. In this paper we extend the technique to systems with limited non-deterministic constrained data transitions, such as systems with data constraints of the type $temp' = temp + [-\alpha, \beta]$. We provide a formal foundation for the domain reduction abstraction for constrained data transition systems. Our technique abstracts the domain of data variables in order to reduce the state space, leaving the control part of the system unchanged.

Trajectory reduction abstracts data trajectories to shorter data trajectories with the same characteristics (except for trajectory length). By introducing *data stuttering* in the abstract

* This work has been partially supported by NASA grant NAG-1-224 and NASA contract NCC-01-001.

system, we restore trajectory lengths and provide a simulation of the original system. Trajectory reduction may dramatically reduce the domain of the data variables, while retaining the essential behavior of the system, and make model checking feasible. The abstraction technique is conservative but precise enough to not introduce many spurious behaviors. In addition, the technique applies to systems with both linear and non-linear data conditions.

The remainder of this paper is organized as follows; after discussing some closely related works below, Section 2 formalizes the type of systems covered by our approach and introduces some basic definitions. Section 3 shows the theory behind the abstraction and a formal proof that the abstraction is a simulation of the original system. A practical application of the technique is illustrated in Section 4. Section 5 concludes with a discussion.

Related Work: Among the various abstraction techniques suggested for model checking systems with data variables [1, 3, 4, 7, 9, 10, 2], few address constrained data transition systems.

Predicate abstraction [10] encodes each data condition (predicate) with a boolean variable reducing the possibly infinite state space to a finite one. Predicate abstraction is capable of handling constrained data transition systems with the manual aid of decision support tools such as PVS, but produces a rather coarse abstraction. Namjoshi and Kurshan [12] suggest an algorithm to automate the process by using syntatic analysis. It allows bounded nondeterminism in the system by exploiting each deterministic action individually, which can be very expensive for systems with non-deterministic data transition constraints.

Chan *et al.* [4] describe a technique for model-checking systems with non-linear data conditions using an approach similar to predicate abstraction. In their work, a constraint solver is used to eliminate infeasible combinations of conditions. The technique is capable of model checking systems with complex non-linear data conditions, but limited to constraint-free data transition systems and a small subset of constrained data transition systems that allow only identity data transitions $x' = x$.

Alur *et al.* [1] and Bultan *et al.* [3] use symbolic approximation based on abstract interpretation for model checking systems that are similiar to our notion of constrained data transition systems. Their approaches are different from others in that they directly model check infinite systems using symbolic approximation without constructing an abstract system. The work of Alur *et al.* is of particular intererst since it deals with systems with non-deterministic data transition constraints though it applies in the hybrid systems domain. Both techniques are limited to linear data conditions and linear data constraints, and suffer from non-termination problems when exact approximation is required.

The notion of *domain reduction* is not entirely new; among several earlier works, data path abstraction for hardware systems [11] and the finite domain method for verifying equality formulas [14] are examples that attempted to reduce the variable domains for limited types of systems. Our work attempts to generalize *domain reduction abstraction* for software systems to the largest extent.

2 System Model and Basic Definitions

2.1 System Model

We use the same system model introduced in [4] as a basis to classify our systems of interest.

Our system model is a tuple $(N, N_0, v, D, \Delta, C)$ where

- N is a finite set of control nodes and $N_0 \subseteq N$ is a set of initial control nodes.

- v is a finite vector of data variables over D where D is the Cartesian product of the domains of the data variables.
- Δ is a mapping from N^2 to $2^{D \times D}$.
- C is a finite set of conditions on v of the form $c := \alpha(v) \bowtie 0$ where $\bowtie \in \{<, \leq, =, \neq, \geq, >\}$ and $\alpha : D \longrightarrow \Re$.

The system model defines a basic transition system $M = (S, S_0, R, AP, L)$ [4] where $S = N \times D$ is a set of states, $S_0 = N_0 \times D$ is a set of initial states, $AP = N \cup C$ is a set of atomic propositions, $L(n, v) = \{n\} \cup \{\alpha \in C \mid \alpha(v)\}$ labels each state with atomic propositions in AP, and R is a transition relation defined on $S \times S$ so that $R((m, x), (n, y))$ iff $(x, y) \in \Delta((m, n))$.

Note that data values are only distinguishable through the conditions in C in this type of system. Also, we use the term *data condition* for $c \in C$ instead of the term *constraint* in order to distinguish it from our notion of *data constraint* which will be introduced shortly.

For notational convenience we write $(s, t) \in R$ as $R(s, t)$ and call s and t the pre-state and post-state respectively. For a state $s = (n, d) \in S$, we use $s \mid_N = n$, $s \mid_D = d$ to represent the control node and the data node respectively. We would use D_i instead of D, if the projection to the i^{th} data variable is required. A *path* in the system M from a state s is a (finite or infinite) sequence of states $\pi = s_0 s_1 s_2 \ldots$ such that $s_0 = s$ and $R(s_i, s_{i+1})$ holds for all $i \geq 0$. $\pi(s_0)$ denotes a path with initial state s_0.

2.2 Classification

We classify the basic transition system defined above into two categories: *constraint free data transition systems*, and *constrained data transition systems* after introducing some basic definitions.

Definition 1 ***Data and state equivalence***

1. *$x, y \in D$ are **data equivalent**, written $x \equiv y$, iff $\forall c \in C : c(x) = c(y)$.*
2. *Two states $s, s' \in S$ are **state equivalent**, written $s \simeq s'$, iff $L(s) = L(s')$, i.e., $s|_N = s'|_N \ \wedge \ s|_D \equiv s'|_D$.*

We denote $D/_{\equiv}$ ($S/_{\simeq}$) for the set of equivalence classes induced by $\equiv$ ($\simeq$) on D (S) and $e_i(E_i)$ for the i^{th} data (state) equivalence class.

Definition 2 *R is said to be a **constrained transition** for D_i if for each state-equivalence class E_j, there is a finite set of data transition functions $F^i_{E_j} = \{f_i \mid f_i : D_i \longrightarrow D_i\}$ such that*

1. *$\{f_i(x) \mid f_i \in F^i_{E_j}\} \neq D_i$ for some $x \in D_i$, and*
2. *$R(s, t)$ for $s \in E_j$ iff $(s|_D, t|_D) \in \Delta((s|_N, t|_N))$ and $t|_{D_i} = f_i(s|_{D_i})$ for some $f_i \in F^i_{E_j}$.*

A constrained data transition means that the transition relation imposes constraints on the specific data values of the pre-state and the post-state. The type of constraints we consider here is a finite set of functions—i.e., the data in the post-state is an application of a function to the data in the pre-state. The specific function to be applied may be dependent on the label of the pre-state, dependent on each data equivalence class, or globally defined.

When $F^i_{E_j}$ has more than one function element, the data transition is taken by non-deterministic choice among the several data transition functions. In this way, we allow finite non-determinism in the system.

A data variable v_i is said to be *constrained* if R is a constrained transition for D_i, and *unconstrained* otherwise (i.e., the variable change values randomly).

Definition 3 *A transition system M is a* ***constrained data transition system*** *(constrained system for short) if there exists a data variable v_i which is constrained. It is a* ***constraint-free data transition system*** *if v_i is unconstrained for every i.*

The following property can be drived from the definition; if there is a transition between a pair of states (s, t), any other pair of states (s', t') with the same labels as s, t respectively satisfies a legal transition between them if they satisfy data transition constraints.

Property 1. Let $M = (S, S_0, R, L, AP)$ be a constrained system where some data variables $v_1, v_2, \ldots, v_n$ are constrained with $F_{E_j} = F^1_{E_j} \times F^2_{E_j} \times \ldots \times F^n_{E_j}$ for each E_j. Then, for $s \in E_j$ and $(f_1, f_2, \ldots, f_n) \in F_{E_j}$, $R(s,t) \wedge s \simeq s' \wedge t \simeq t' \wedge f_i(s'|_{D_i}) = t'|_{D_i}$, $1 \leq \forall i \leq n$, implies $R(s', t')$.

In general, there may be transition systems which do not fall in the two categories. Such systems are not considered here. In this paper, when we refer to a transition system, it is assumed to be either a constrained data transition system or a constraint-free data transition system.

3 Domain Reduction Abstraction for Constrained Systems

For constraint-free data transition systems, the domain reduction abstraction technique reduces the original domain to a set of data representatives from each data equivalence class. We have proved that the abstract system with the reduced domain bisimulates the original system [5]. Here, we discuss the domain reduction for constrained data transition systems in depth.

Domain reduction abstraction for constrained systems is based on the selection of representative data values for the abstract system–a data value v can be a representative of other data value v' if for any data trajectory from v' there is a data trajectory from v that moves through the same data equivalence classes in fewer steps. The reduced domain includes each representative data value v, and all the data values on the *minimal path* from v upto the last *node of change*. The notion of *minimal path* and *node of change* will be discussed shortly.

By introducing *data stuttering*, an abstract system based on the reduced domain simulates the original system. This technique works for constrained transition systems with a finite state space, or with infinite state systems that have at least one data trajectory per state that passes through finitely many number of data equivalence classes.

3.1 Basic Definitions

The following definitions were originally introduced in [5], but we include them here since they are essential to understand the remainder of this paper.

Definition 4 ***Trace and segment length sequence***

1. *A* ***trace*** *$T[\pi(s_0)]$ of a path $\pi(s_0)$ is a sequence of data equivalence classes $e_0 e_1 e_2 \ldots$ with $e_i \neq e_{i+1}$ for all i such that $s_0|_D \in e_0$ and for all $i, j \geq 0$, $s_i|_D \in e_j$ implies $s_{i+1}|_D \in e_j$ or $s_{i+1}|_D \in e_{j+1}$.*
2. *A* ***node of change*** *in a path $\pi(s_0)$ is either s_0 or a state s_i such that $s_i|_D \in e_j$ and $s_{i-1}|_D \notin e_j$ for some data equivalence class e_j.*
3. *The* ***segment length sequence*** *$N[\pi(s_0)] = [n_1, n_2, \ldots]$ of a path $\pi(s_0)$ is an ordered sequence of natural numbers where $n_i = q - p$, s_p and s_q are nodes of change such that $s_p|_D \in e_{i-1}$, $s_q|_D \in e_i$ for two adjacent data equivalence classes e_{i-1}, e_i in $T[\pi(s_0)]$.*

Intuitively, a trace $T[\pi(s_0)]$ is a sequence of the data equivalence classes of D that are reachable from s_0 in order, nodes of change are states along the path at which there is a change in data equivalence class, and the segment length sequence is a sequence of natural numbers that represents the number of steps to reach from one equivalence class to another along the path $\pi(s_0)$.

We say $N[\pi(s_0)] \preceq N[\pi'(s'_0)]$ if $\forall i,\ n_i \leq m_i$ where $n_i \in N[\pi(s_0)]$ and $m_i \in N[\pi'(s'_0)]$, $N[\pi(s_0)] \prec N[\pi'(s'_0)]$ when $N[\pi(s_0)] \preceq N[\pi'(s'_0)]$ and $n_i < m_i$ for some i. We also say that π' is a *trajectory reduction* of π if $T[\pi] = T[\pi']$ and $N[\pi'] \preceq N[\pi]$.

Definition 5 ***Minimal state and minimal path***

1. *A state s is a minimal state if and only if $\forall s' \simeq s,\ \exists \pi(s)$ such that for all path $\pi(s')$, $T[\pi(s')] = T[\pi(s)]$ implies $N[\pi(s')] \not\prec N[\pi(s)]$.*
2. *A path $\pi_{min}(s)$ is a minimal path from s if and only if $\forall \pi(s)$, $T[\pi(s)] = T[\pi_{min}(s)]$ implies $N[\pi(s)] \not\prec N[\pi_{min}(s)]$. $\pi^c_{min}(s)$ denotes the subpath of $\pi_{min}(s)$ up to the last node of change.*

When a state s is a minimal state, for any given state s' that has the same label as s, there exists a path $\pi(s)$ that either reduces the segment lengths for all the paths $\pi(s')$ with the same trace, or has a unique trace. From the possibly non-deterministic data transition constraints, a state can be an initial state for several different paths with the same trace. Among them, a minimal path is a trajectory reduction of all other comparable paths from the same initial state with the same trace.

The $\preceq$ relation defines a partial order on the set of paths, and thus, there can be incomparable paths using the $\preceq$ relation. The above definitions ensures that those incomparable paths are all considered in our reduced domain.

The following proposition can be easily derived from the definition of a minimal state.

Proposition 1. *s is not a minimal state if and only if $\exists s' \simeq s$, $\forall \pi(s)$ such that for some $\pi(s')$, $T[\pi(s')] = T[\pi(s)] \wedge N[\pi(s')] \prec N[\pi(s)]$.*

3.2 Abstraction Theory

A minimal state that reduces trajectories of all paths from another state is a representative state to be included in a reduced domain. The following lemma and corollary show the existence of such a minimal state for any state in the system under certain assumption discussed before.

Lemma 1. *Suppose $\forall s \in S$, $\exists \pi(s)$ with a finite trace. Then, for any state $s \in S$, there exists a minimal state $s' \in S$ such that $s' \simeq s$ and $\forall \pi(s)$, $\exists \pi(s')$ that satisfies $T[\pi(s)] = T[\pi(s')]$ and $N[\pi(s')] \preceq N[\pi(s)]$.*

Proof. The proof is obvious when s itself is a minimal state. Suppose s is not a minimal state. By Proposition 1, $\exists s_1 \simeq s$, $\forall \pi(s)$ such that $T[\pi(s_1)] = T[\pi(s)] \ \wedge \ N[\pi(s_1)] \prec N[\pi(s)]$ for some $\pi(s_1)$. If s_1 is a minimal state, then let $s' = s_1$. Otherwise, again $\exists s_2 \simeq s$, $\forall \pi(s_1)$ such that $T[\pi(s_2)] = T[\pi(s_1)] \ \wedge \ N[\pi(s_2)] \prec N[\pi(s_1)]$ for some $\pi(s_2)$. In this way we can identify a sequence of states $s_1 s_2 \ldots$ such that $\forall \pi(s), \exists \pi(s_1), \pi(s_2), \ldots$ with $T[\pi(s)] = T[\pi(s_1)] = T[\pi(s_2)] \ldots$ and

$$\ldots \prec \ldots \prec N[\pi(s_i)] \prec N[\pi(s_{i-1})] \prec \ldots \prec N[\pi(s_1)] \prec N[\pi(s)]$$

Note that the sequence cannot be infinite when some of the trace $T[\pi(s)]$ is finite and each element of $N[\pi(s)]$ is a finite natural number. Therefore, there exists a finite sequence of states $s_1 s_2 \ldots s_n$ with a minimal state s_n such that $\forall \pi(s)$, $\exists \pi(s_1), \pi(s_2), \ldots \pi(s_n)$ where

$$T[\pi(s_n)] = T[\pi(s_{n-1})] = \ldots = T[\pi(s)]$$
$$N[\pi(s_n)] \prec N[\pi(s_{n-1})] \prec \ldots \prec N[\pi(s)].$$

Let $s' = s_n$. s' is the minimal state that satisfies the claim. □

The following corollary can be proved in a similar way.

Corollary 1. *Let S be finite. Then, for any state $s \in S$, there exists a minimal state $s \in S$ such that $\forall \pi(s)$ in M, $\exists \pi(s')$ in M that satisfies $T[\pi(s)] = T[\pi(s')]$ and $N[\pi(s')] \preceq N[\pi(s)]$.*

As a result, the existence of the representative minimal state is guaranteed for finite (no matter how large they are) state systems, and infinite state systems with at least one finite trace per state. In the following sections we assume this assumption holds.

3.3 Domain reduction abstraction

Now, the reduced domain for an abstract system of M is defined to include a data value of a minimal state and all data values on its minimal data paths up to the last node of change.

Definition 6 *D' is the smallest subset of D including*
$\bigcup_{s_0 \in min\_states} \{s_i|_D \mid s_i : i_{th}$ *state on* $\pi^c_{min}(s_0)\}$, *where min_states is a set of all minimal states in S.*

Next, we introduce a data identity transition R^D_{id} defined as $((n,x),(n',x)) \in R^D_{id}$ if $((n,x),(n',x')) \in R$ and $x \equiv x'$ for any control nodes n, n' and data nodes x, x'. Intuitively, the data identity transition allows the system to stay in the same data node while the control node changes according to the transition relation R. This allows for the stuttering of data nodes within the same data equivalence class.

For a given constrained data transition system $M = (S, S_0, R, L, AP)$, let $M' = (S', S'_0, R', L', AP')$ be an abstracted transition system of M where $AP' = AP$, $S' = N \times D'$, $S'_0 = N_0 \times D'$, $L' = S' \lhd^1 L$, and $R' = (R \cup R^D_{id}) \cap (S' \times S')$. Here, D' is the reduced domain defined in Definition 6.

The following lemma shows that for any state $s \in S$, there is a state $s' \in S'$ that has an equivalent path $\pi'(s')$ in M' for any path $\pi(s)$ in M. A proof that M' simulates M follows.

Lemma 2. *$\forall s_0 \in S$, $\exists s'_0 \in S'$ such that for any $\pi(s_0) = s_0 s_1 \ldots$ in M, $\exists \pi(s'_0) = s'_0 s'_1 \ldots$ in M' that satisfies $s_i \simeq s'_i$ for all i.*

Proof. If s_0 is a minimal state, then $s_0 \in S'$ and there exists $\pi^c_{min}(s_0)$ in M' such that $T[\pi(s_0)] = T[\pi^c_{min}(s_0)] \ \wedge \ N[\pi^c_{min}(s_0)] \preceq N[\pi(s_0)]$ from the definition of M'. Otherwise, there exists a minimal state $t_0 \simeq s_0$ in S and a minimal path $\pi_{min}(t_0)$ in M such that $T[\pi(s_0)] = T[\pi_{min}(t_0)] \wedge N[\pi_{min}(t_0)] \preceq N[\pi(s_0)]$ from Lemma 1 or Corollary 1. Note that $t_0 \in S'$ and the corresponding $\pi^c_{min}(t_0)$ with $T[\pi^c_{min}(t_0)] = T[\pi_{min}(t_0)]$ is in M' by definition of S' and M'.

[1] $\lhd$ is the notation for domain restriction in Z. $S \lhd R$ of a relation R to a set S relates x to y iff R relates x to y and x is a member of S [15]

For $N[\pi(s_0)] = \{n_1, n_2, \ldots\}$ and $N[\pi^c_{min}(t_0)] = \{m_1, m_2, \ldots\}$, let $s_{idx} = \{p_1, p_2, \ldots, \}$ and $t_{idx} = \{q_1, q_2, \ldots, \}$ be sequences of indices of nodes of change in $\pi(s_0)$ and $\pi^c_{min}(t_0)$ respectively. That is, $p_i = \Sigma^i_{j=1} n_j$ and $q_i = \Sigma^i_{j=1} m_j$.

A path $\pi(s')$ in M' can be defined as follows.

1. $\forall i, s'_i|_N = s_i|_N$,
2.

$$s'_i|_D = \begin{cases} t_0|_D \text{ if } i = 0 \\ t_k|_D \text{ where } \begin{cases} k = i & \text{if } i < q_1 \\ k = q_1 - 1 & \text{if } q_1 \le i < p_1 \\ k = i - p_j + q_j & \text{if } p_j \le i < p_{j+1} \text{ and } i - p_j < m_{j+1} \\ k = q_{j+1} - 1 & \text{if } p_j \le i < p_{j+1} \text{ and } i - p_j \ge m_{j+1} \\ k = q_l & \text{if } i > p_l \text{ where } p_l \text{ is the largest.} \end{cases} \end{cases}$$

Then, $s'_i \simeq s_i$ for all i. Note that $(s'_i, s'_{i+1}) \in R'$ for all i, since either (1) $(s'_i, s'_{i+1}) \in R$ by Property 1 or (2) $(s'_i, s'_{i+1}) \in R^D_{id}$. Case 1 implies $R'(s'_i, s'_{i+1})$ by the definition of R' and the fact that $s'_i, s'_{i+1} \in S'$. Case 2 is a legal transition relation in M'. Therefore, $\pi(s'_0)$ is a path in M'. □

We say a path $\pi(s'_0) = s'_0 s'_1 \ldots$ is equivalent to a path $\pi(s_0) = s_0 s_1 \ldots$ if and only if $s'_i \simeq s_i$ for all i.

Now, we define a relation between the original system and the abstract system in terms of the path equivalence as follows.

Definition 7 *$H \subseteq S \times S'$ is a relation on $S \times S'$ such that $H(s, s')$ if and only if $\forall \pi(s), \exists \pi(s')$ such that $\pi(s')$ is equivalent to $\pi(s)$.*

Theorem 1. *H is a simulation relation [2] between M and M'.*

Proof. Let $H(s, s')$. $L(s) = L(s') = L'(s')$ from $s \simeq s'$ and by the definition of the labeling function. $L(s) \cap AP' = L'(s')$ is obvious by $AP = AP'$

Let $R(s, t)$ for some $t \in S$. For any path $\pi(st)$ there exists a path $\pi(s'\hat{t})$ in M' that is equivalent to $\pi(st)$ by Lemma 2. Therefore, $t \simeq \hat{t}$ guaranteeing the existence of $\hat{t} \in S'$ that satisfies $R'(s', \hat{t}) \wedge \hat{t} \simeq t$. Suppose for all such $\hat{t}$ ($t \simeq \hat{t} \wedge R'(s', \hat{t})$), $\exists \pi(t)$ in M such that $\forall \pi(\hat{t})$ in M', $t_i \not\simeq \hat{t}_i$ for some i. Consider a path $\pi(st) = s \mid \pi(t)$ and $\pi(s'\hat{t}) = s' \mid \pi(\hat{t})$. Clearly, $\pi(s'\hat{t})$ is not equivalent to $\pi(st)$ for any $\hat{t}$ with $\hat{t} \simeq t \wedge R'(s', \hat{t})$, which contradicts to $H(s, s')$. Therefore, $\exists \hat{t} \in S'$ such that $\forall \pi(t)$ in M, $\exists \pi(\hat{t})$ in M' with $t_i \simeq \hat{t}_i$ for all i and $R'(s', \hat{t})$. $H(t, \hat{t}) \wedge R'(s', \hat{t})$ follows. □

The fact that M' simulates M directly follows from Lemma 2.

Theorem 2. *M' simulates [3] M.*

Proof. From Lemma 2, for any initial state $s_0 \in S_0$, $\exists s'_0 \in S'$ such that $H(s_0, s'_0)$ holds. Since $s'_0 \simeq s_0$, $s'_0 \in S'_0$ by definition of S'_0. □

We conclude that M' is a conservative abstraction of M, and thus, any $ACTL^*$ formula satisfied by M' is also satisfied by M.

[2] We use the same definition of simulation relation defined in [6].

[3] We use the same notion defined in [6].

4 Application

In a previous report [5], we defined an algorithm to compute a reduced data domain for systems with *deterministic* data transition constraints. We can use the same algorithm in the *non-deterministic* case discussed in this paper. If we can put an upper and lower limit on the rate of change of the data variables, we can use these deterministic limits to over-approximate the reduced domain using the algorithm for the deterministic case.

4.1 Treatment of Non-deterministic Data Constraints

Our approach is based on over-approximating the reduced domain by computing the worst case change of the data variables, that is, we compute reduced domains using the upper limit for the data growth rate and using the lower limit of the growth rate. Our reduced domain for the nondeterministic system is the union of the reduced domains for the deterministic approximations.

For the deterministic cases, we compute every possible *maximal trace* and its corresponding *minimal data trajectory*. A maximal trace t starting in equivalence e_1 is a trace that is not a subtrace of any other trace starting in e_1. The minimal data trajectory is the trajectory with minimal segment lengths that follow trace t. The initial data value of each minimal data trajectory can be a data value of a minimal state. Our reduced domain is an interval including each initial data value v and all the data values of a minimal data trajectory starting from v and up to the last node of change.

As an example, suppose we have a constrained system with two data equivalence classes $e_1 = \{x \mid x < 10\}$, $e_2 = \{x \mid x \geq 10\}$ with non-deterministic data transition functions defined over each e_i, $F_{e_1} = \{f \mid f(x) = x + \alpha,\ \alpha \in [-1..2],\ x \in e_1\}$, $F_{e_2} = \{f \mid f(x) = x + \beta,\ \beta \in [-3..2],\ x \in e_2\}$. Then, every finite or infinite trace of a trajectory is one of $(e_1, (e_2, e_1)^*)$,$(e_2, (e_1, e_2)^*)$, $(e_1, e_2)^*$, and $(e_2, e_1)^*$. Among all possible data transition functions for x, we can choose one with maximum positive change $f^+(x) = x + 2$ and one with maximum non-positive change $f^-(x) = x - 3$ and compute minimal data trajectories for the maximal finite traces (e_1, e_2) and $(e_2, e_1,)$ respectively.

Note that we need two assumptions to reduce the non-deterministic case to a deterministic case; (1) it is possible to provide an upper and lower bound of the growth rate of the data constraints, and (2) every maximal trace is finite for deterministic data constraints. We limit our discussion of automation to systems with the following conditions that assure that these two assumptions hold. We are currently investigating how we can relax these assumptions to provide an algorithm for the more general cases.

1. Each data transition constraint function $f \in F_{E_i}$ for a variable v_k is in the form of $f(v_k) = av_k + b$, where $a \in \{0, 1\}$ and b is an integer constant.
2. Each numeric function $\alpha(v)$ appearing in the set of data conditions is a polynomial.

Since there is a finite number of data constraint functions, the one with the maximum (or minimum) change can be identified by identifying the maximum and minimum values for b. The finiteness of the maximal traces are guaranteed since (1) the number of equivalence classes is finite, (2) the data values do not change signs periodically, and (3) the data conditions $c(v)$ are not periodic.

4.2 Automation

Given the upper and lower limits of the data constraint functions, we can derive a reduced domain for each case and use the union as the reduced domain for the non-deterministic

```
/* initially, SAT = e_i, Trace = {e_i}, step = 0, REACH = D/≡ − {e_i}. */

min_trace(e_i, SAT, Trace, step, REACH)
    while REACH ≠ ∅
        pick e_j ∈ REACH, REACH := REACH − {e_j}
        k := 1;, distance := ∞
        while(1)
            Target := f^{k+step}(e_j)
            if SAT ∧ Target is satisfiable
              SAT := SAT ∧ Target, step := step + k, Trace := Trace + {e_j}
              min_trace(e_i, SAT, Trace, step, REACH)
            else
                d := distance_test(SAT, Target)
                if d < distance
                  distance := d ; k^{++};
                else break;
    add (SAT, Trace, step) to the output list.
```

Fig. 1. Maximal trace and minimal segment length sequence generation

system. We first discuss systems with all constrained variables. The approach for systems with both constrained and unconstrained data variables is then briefly covered.

Trace computation. The algorithm in Figure 1, which was originally presented in [5], illustrates a process to identify a maximal trace with a minimal segment length sequence. A data value of a minimal state is identified by computing a possible maximal finite trace and a minimal number of steps required for a data trajectory to follow the trace. For a given set of data equivalence classes $\{e_1, e_2, \ldots, e_n\}$, the existence of a trace from e_i to e_j can be computed by recursively checking the satisfiability of $SAT \wedge f^{k_1}(e_j)$, where $f^{k_1}(e_j)$ denotes a k_1 times symbolic transformation of e_j using f. SAT is initially a data equivalence class $e_i (\neq e_j)$. By repeating this process now using $SAT := SAT \wedge f^{k_1}(e_j)$ we can investigate the reachability of the other equivalence classes. We repeat this process until a maximal satisfiable formula SAT is identified. The ordered equivalence classes in the resulting SAT formula is a maximal trace and the $\{k_i\}$ sequence is the segment length sequence for a possible minimal path.

Whether or not there exists a k that satisfies $SAT \wedge f^k(e_j)$ can be determined by checking the distance between SAT and $f^k(e_j)$ as k grows. Given our assumptions, the distance can be defined as $\mid x - f^k(y) \mid$ for a pair of random initial values $x \in SAT$ and $y \in e_j$. As y is transformed, the computation stops if the distance grows concluding that e_j is not reachable from the data region represented by SAT.

The algorithm as stated in Figure 1 applies to the case where the data conditions are linear. The same algorithm applies to systems with non-linear data conditions with a minor modification; an equivalence class defined by non-linear data conditions may appear more than once on a maximal trace. For example, in a system with two data equivalence classes $e_1 : x^2 > y$, $e_2 : x^2 \leq y$, and a data transition constraints $x' = x + 1$, $y' = y + 1$, a possible maximal trace is $e_1 e_2 e_1$. Therefore, the initial $REACH$ must include several copies of data equivalence classes—the required number of copies of data equivalence classes is determined by the degree of the data conditions.

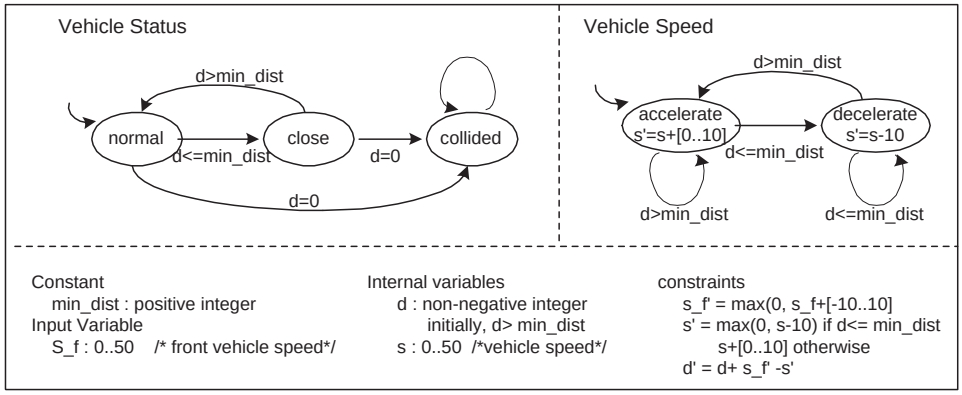

Fig. 2. Cruise Control System

Domain Reduction. For each constraint function with a maximum change f_i, the algorithm generates as its result a set of tuples $(\{(SAT_{i_k}, Trace_{i_k}, Step_{i_k})\}$ where $Trace_{i_k}$ is a possible trace, SAT_{i_k} is the region from which the trace can start, and $Step_{i_k}$ is the number of steps required to for a minimal trajectory to follow the trace. Note that any value $v_{i_k} \in SAT_{i_k}$ is a data value of a possible minimal state. Our reduced domain D' is initially the union of all such values ($\bigcup\{v_{i_k}\}$) and expanded by including the data values of all nodes of change as follows:

1. $D' := D' \cup \{f_i^{Step_{i_k}}(v_{i_k})\}$.
2. $D' := D' \cup \{f_j^t(v_{i_k}) \mid v_{i_k} \in e_1 \wedge f_j^t(e_m) \wedge SAT_{i_k},$ where $Trace_{j_k} = e_1 e_2 \ldots e_m\}$

The first step expands D' so that for each v_{i_k}, D' includes the data value of the last node of change for the trajectories following the trace $Trace_{i_k}$. If v_{i_k} belongs to an initial data equivalence class of some other trace ($Trace_{j_k}$) generated by some other maximum constraint function (f_j), we want to include the last node of change for the trajectory starting in v_{i_k} and following trace $Trace_{j_k}$, if there is one,—this node of change is computed by $f_j^t(v_{i_k})$ where t is the number of steps for SAT_{i_k} to reach the last equivalence class on the other trace $Trace_{j_k}$. The second step ensures that D' includes the data value of the last node of change for every possible minimal data trajectory from v_{i_k}. Our final reduced domain is the smallest interval that includes all data values identified from these two steps.

4.3 Systems with both Unconstrained and Constrained Data Variables

When a variable is unconstrained in an equivalence class while other variables are constrained, the same algorithm can be applied by introducing auxiliary variables over the same domain as the unconstrained variables. For example, suppose a system has two equivalence classes $e_1 : x < y$, $e_2 : x \geq y$, and x is constrained by $x' = x + 1$ while y is unconstrained. Then the symbolic transformation will be in the form of $e_1 \wedge f(e_2) = x < y \ \wedge x + 1 \geq z$. We have to introduce the new unconstrained variable z to capture that there is no relationship between the y that satisfies e_1 and the y that satisfies $f(e_2)$. For each identified maximal formula SAT, data values for possible minimal states consist of (1) a random value for each unconstrained variable that satisfy SAT and (2) values for constrained variables that satisfy SAT after determining the random values for unconstrained variables.

4.4 Example: The Cruise Control System

We illustrate the application of domain reduction abstraction with a toy cruise control system. The cruise control system(Figure 2) receives the speed of the front vehicle $s_f \in [0..50]$ as an input and issues the change of speed $s \in [0..50]$ based on the distance d from the front vehicle; if the distance is greater than the pre-defined minimum distance min_dist, it maintains or increases the speed. Otherwise, it decreases the speed using the maximum deceleration. The speed can change by at most 10 mph in a step and vehicles do not go backward.

The safety property of interest is that this system never allows the vehicle to be in the *collided* state, which can be expressed in $ACTL^*$ as $AG\ !(vehicle\_status = collided)$. Note that the domain of d is unbounded which requires an abstraction.

The system model generates three equivalence classes, $e_1 : d > min\_dist$, $e_2 : 0 < d \leq min\_dist$, and $e_3 : d = 0$. We also identify the deterministic data constraint functions that provide an upper and lower bound for the non-deterministic constrains in the model. The upper bound f^+_{max} can be defined with the set of data constrains $f^+_{max} = \{d + (s_f + 10) - (s - 10), s - 10, s_f + 10\}$. The lower bound f^-_{max} is identified to be $f^-_{max} = \{d + (s_f - 10) - (s + 10), s + 10, s_f - 10\}$.

trace	step	k	$SAT \wedge Target$	satisfy?	d	continue?
$[e_1, e_2]$	0	1	$d > 150 \ \wedge 20 < d + s_f - s \leq 170 \wedge 0 \leq s \leq 50$ $\wedge\ 0 \leq s_f \leq 50 \wedge s_f \geq 10$	yes	∞	yes
$[e_1, e_2, e_3]$	1	1	$d > 150 \ \wedge \ 20 < d + s_f - s \leq 170 \wedge 0 \leq s \leq 50$ $\wedge\ d + 2s_f - 2s - 60 = 0 \ \wedge 0 \leq s_f \leq 50 \wedge s_f \geq 20$	no	150	yes
		2	$d > 150 \ \wedge \ 20 < d + s_f - s \leq 170 \wedge 0 \leq s \leq 50$ $\wedge\ d + 3s_f - 3s - 120 = 0 \ \wedge 0 \leq s_f \leq 50 \wedge s_f \geq 30$	yes	.	finish
	3		$SAT = d > 150 \ \wedge \ 20 < d + s_f - s \leq 170 \wedge 0 \leq s \leq 50$ $\wedge\ d + 3s_f - 3s - 120 = 0 \ \wedge 0 \leq s_f \leq 50 \wedge s_f \geq 30$			

Table 1. Sample trace computation

Table 1 shows a sample trace computation using f^-_{max} and $min\_dist = 150$. From the resulting satisfiable formula SAT, the minimum and maximum value pair for each variable is identified as $(d_{max}, d_{min}) = (151, 195)$, $(s_{f_{max}}, s_{f_{min}}) = (30, 40)$, and $(s_{max}, s_{min}) = (40, 50)$ using a constraint solver. In our case studies we have used the constraint solver $CLP(q, r)$ [8] to check the satisfiability and identify the minimum and maximum values. After applying f^-_{max} $step = 3$ times to the boundary value $(d, s_f, s) = (151, 30, 50)$, the interval $d = [0, 195]$, $s_f = [0, 40]$, $s = [40, 50]$ is included in the reduced domain. In a similar way, f^+_{max} identifies a trace (e_3, e_2, e_1) producing the intervals $d = [0, 190]$, $s_f = [40, 50]$, $s = [0, 40]$. The resulting reduced domain includes the union of the domain identified by the lower and upper bounds of the constraint functions—the final domain is $d = [0, 195]$, $s_f = [0, 50]$ and $s = [0, 50]$.

If we use $min\_dist = 100$, the algorithm generates a reduced domain $d : [0, 120]$, $s_f : [0, 50]$, and $s : [0, 50]$. The symbolic model checker $NuSMV$ [13] generates a counter example for the property in seconds. Using the algorithm with $min\_dist = 150$ (as in the detailed discussion above) generates a reduced domain $d : [0, 195]$, $s_f : [0, 50]$, and $s : [0, 50]$ and $NuSMV$ proves the property true.

5 Discussion

We provide a formal foundation for domain reduction abstraction for systems with (linear or non-linear) data conditions and data constraints. We are particularly interested in systems

with non-deterministic data transition constraints since most environmental constraints are defined in such a way.

To make the technique practical, an automated way of computing data values for minimal states is necessary. We suggest an algorithmic way of computing those values for a limited type of systems. The computational complexity of algorithm is exponential in terms of the number of data variables with non-deterministic data constraints and the number of data equivalence classes. Nevertheless, we expect that those numbers are generally small in practice. We are currently pursuing some empirical data to see if this assumption is merited.

The technique does not guarantee that state space generated from the reduced domain will be small enough to be model checked. Nevertheless, our initial case studies are promising and we believe this technique has the potential to be practical for many practical cases, thus, extending the capability of model checking of models of software systems.

References

1. R. Alur, T. Henzinger, and P. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, March 1996.
2. Ramesh Bharadwaj and Constance Heitmeyer. Model checking complete requirements specifications using abstraction. In *First ACM SIGPLAN Workshop on Automatic Analysis of Software*, 1997.
3. T. Bultan, R. Gerber, and W. Pugh. Symbolic model checking of infinite state systems using presburger arithmetic. In *Computer Aided Verification*. Springer Varlag, 1997.
4. William Chan, Richard Anderson, Paul Beame, and David Notkin. Combining constraint solving and symbolic model checking for a class of systems with non-linear constraints. In *Proc. of CAV'97, LNCS 1254*, pages 316–327. Springer, June 1997.
5. Yunja Choi, Sanjai Rayadurgam, and Mats Heimdahl. Automatic abstraction for model checking software systems with interrelated numeric constraints. In *Proceedings of the 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9)*, pages 164–174, Vienna, Austria, September 2001.
6. E. M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.
7. E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. *ACM Transaction on Programming Languages and Systems*, 16(5):1512–1542, September 1994.
8. Constraint logic programming over rational or real. Available at http://www.ai.univie.ac.at/clpqr/.
9. E. Emerson and K. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Thirteenth Annual IEEE Symposium on Logics in Computer Science*, pages 70–80, 1998.
10. Susanne Graf and Hassen Saidi. Construction of abstract state graphs with PVS. In *Proceedings of the Computer Aided Verification(CAV 1997)*, 1997.
11. R. Hojati and R.K. Brayton. Automaic datapath abstraction of hardware systems. In *7th International Conference, CAV1995*, July 1995.
12. Kedar S. Namjoshi and Robert P. Kurshan. Syntatic program transformations for automatic abstraction. In *12th International Conference, CAV2000*, pages 435–449, July 2000.
13. NuSMV: A New Symbolic Model Checking. Available at http://http://nusmv.irst.itc.it/.
14. Amir Pnueli, Yoav Rodeh, Ofer Shtrichman, and Michael Siegel. Deciding equality formulas by small domains instantiations. In *11th International Conference, CAV1999*, pages 455–469, July 1999.
15. J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, 1992.