Automatic Abstraction for Model Checking Software Systems with Interrelated Numeric Constraints*

Yunja Choi
Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455
yuchoi@cs.umn.edu

Sanjai Rayadurgam
Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455
rsanjai@cs.umn.edu

Mats P. E. Heimdahl
Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455
heimdahl@cs.umn.edu

ABSTRACT

Model checking techniques have not been effective in important classes of software systems characterized by large (or infinite) input domains with interrelated linear and non-linear constraints over the input variables. Various model abstraction techniques have been proposed to address this problem. In this paper, we wish to propose *domain abstraction* based on data equivalence and trajectory reduction as an alternative and complement to other abstraction techniques. Our technique applies the abstraction to the input domain (environment) instead of the model and is applicable to *constraint-free* and deterministic *constrained* data transition system. Our technique is automatable with some minor restrictions.

Categories and Subject Descriptors

D.2.4 [**Software/Program Verification**]: Model Checking

General Terms

Verification

Keywords

Model checking software systems, domain abstraction, numeric constraints

1. INTRODUCTION

Model checking (both explicit state and symbolic) has been successfully used to verify properties of various finite state systems, such as communications protocols [4], concurrent systems [21], hardware designs [7], and some software systems [2, 8, 19]. Nevertheless, the usefulness in verifying properties of software systems has been limited since important classes of software systems involve large input domains (e.g., unbounded integer variables) as well as interrelated numeric constraints over the variables in the input domain—characteristics that severely limit the usefulness of model checking. To address this problem, various techniques for *abstracting the model* to a simplified model that can be model checked is an active research area. Here we propose *domain abstraction* based on data equivalence and trajectory reduction as an alternative and complement to other abstraction techniques. This technique can be fully automated and applies the abstractions to the input domain instead of the constraints on transitions in the model itself.

Important classes of software systems can be viewed as consisting of a finite *control component* and a (typically infinite or very large) *data component*; examples include many safety-critical control systems. In such systems, the variables in the data component represent the input quantities to the system, for example, in an avionics system, inputs may be altitude, range, bearing and various pilot selectable thresholds. The transitions in the system are guarded by various linear and non-linear constraints on these variables. Model checking systems with these characteristics poses serious challenges.

To address this problem, various techniques to abstract the model to a finite domain have been proposed, for example, predicate abstraction [14]. Other researchers have attempted to combine constraint solving with model checking [9]. Both approaches have achieved some success. These related efforts are discussed in some detail in Section 2.

As an alternative and a complement, we have investigated abstractions over the *input domain* of the systems rather than the system itself. *Domain abstraction* based on data equivalence uses the transition conditions on the input variables to partition the infinite input domain into a finite set of partitions from which one representative input is selected. For systems where there are no data constraints, the abstracted system bisimulates the original system; but, if there are data constraints, the abstracted system simulates the original system. In systems with data constraints, *trajectory reduction* maps a possibly infinite set of input variable trajectories through the state space to a single representative trajectory in a finite domain. This reduced domain is

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

computed based on the numeric constraints on the input variables in conjunction with a fix-point computation for the variables whose trajectories in which we are interested.

In the next section we provide an overview of related work. After introducing our system model in Section 3, Section 4 discusses domain abstraction for constraint free systems and provides a proof that it is a bi-simulation relation. In Section 5, we present trajectory reduction and provide a proof that it is a simulation relation for systems with deterministic data constraints. We illustrate with examples, both abstraction techniques applied to an infinite state-space system. Section 6 discusses the conclusions and future work.

2. RELATED WORK

State space explosion is a serious problem when verifying systems using model-checkers. This problem is especially pronounced in the case of software system specifications, which often have large or infinite data domains. To deal effectively with the state space explosion problem, one should be able to derive meaningful conclusions about the reachable states of the system without actually exploring those states individually. There has been extensive work in addressing state space explosion problem in model-checkers [6, 12, 13, 15, 16, 22, 27, 28], and many techniques are being successfully used, e.g., symbolic model-checking, partial order reduction, symmetry, induction, abstraction and compositional reasoning. See [10] for a comprehensive treatment of the subject. Here, we discuss a few specific works that are closely related to ours.

Heitmeyer *et al.*[19] describe automatic abstraction techniques for specifications written in SCR. Given a property to verify, they describe an abstraction method to automatically *remove irrelevant data variables* and *remove detailed monitored variables*. The former is similar to slicing [17, 18], where dependency information from the model is used to remove parts that do not have an effect on the property of interest. In our proposed method, irrelevant data variables would get replaced with a single representative value, effectively removing it. The latter is a data abstraction technique which is achieved in our method by choosing representative elements from the data equivalence classes of the variable.

Chan *et al.* [9] describe a technique for model-checking certain classes of systems in which data constraints across transitions can be separated as pre-state and post-state conditions connected by boolean operators. Such systems can be model-checked by representing the conditions as boolean variables and constraints as values of the variables before and after the transition. During model-checking, a constraint solver is used to eliminate infeasible combinations of conditions. This technique can handle systems with both linear and non-linear numeric conditions, but is limited to data-memoryless or data invariant system which corresponds to our notion of constraint-free system described later. Our proposed method, when applied to such systems, would statically find representative values in the domain of the data variables for each feasible combination of the conditions before model-checking the system.

When data constraints cannot be separated as above, one may have to approximate the system behavior using abstract data. For example, consider a transition of the form $[a \longrightarrow b \text{ when } y > 50]$ with the constraint $y' = y + 1$. If we are only interested in global properties that check if $y > 50$, we may be able to approximate the system behavior using *Predicate abstraction* [26, 14]. A specific predicate of interest is modeled as an abstract boolean variable. In this case $y > 50$ can be represented as p. The values of the data-variables appearing in the predicate and constraints involving them are then mapped to values of and constraints involving the abstract variable. Thus, the domain of the non-negative integer y is implicitly mapped as $y = [0..50] \rightarrow p = false$ and $y = (50..\infty) \rightarrow p = true$. $y' = y + 1$ would then be transformed to $p' = true$ if $(p = true)$ and $p' =$ one of $true$ or $false$ if $(p = false)$. This is a conservative abstraction since it introduces more behavior. When mapped back to the concrete domain, the corresponding transitions in the concrete system can be seen as $y' \in [0..\infty)$ when $y \leq 50$ and $y' \in [51..\infty)$ when $y > 50$. This is a superset of the transitions represented by $y' = y+1$. However, by this conservative abstraction, we have now reduced a large domain $[0..\infty]$ to a two-element domain $true, false$. In the process, we also had to modify the data-constraint to map it to the abstract domain. Such an abstraction, however, may not be able to capture path properties that depend on the data-constraints in the original system.

An alternative approach would be to retain the data-constraints as they are, but *bound* the domain of the data variables to make model-checking feasible. However, one should be careful not to remove existing behavior in the process of bounding the domains, for then the resulting system may satisfy some properties that were not true in the original system (e.g., restricting y to [0..10], would result in $AG(y < 50)$) becoming true). Such under-approximations could be useful when the goal is to show the existence of an error. In such a case, one could presumably retain a subset of the behavior of the original system and yet successfully exhibit the presence of errors. However, in the context of verification, the abstraction must ensure that the data that is excluded by bounding the domain does not have any unique behavior that is not captured by the data included within the bound. We pursue this alternative approach by suggesting such a method for bounding data domains. This method ensures that satisfaction of a property expressed in $\forall CTL^*$ by the abstract system guarantees satisfaction of the property in the original system.

It must be noted here that there are other approaches for model checking infinite state systems with data transition constraints in the domain of hybrid systems and concurrent systems [1, 5]. These approaches are based on building special-purpose tools (e.g., [20]) to model-check such systems. In contrast, we apply abstraction techniques statically and use a traditional model-checking tool like SMV [23, 24] to verify properties on the abstract system.

3. SYSTEM MODEL AND DEFINITIONS

We first define some terms and formalisms that are used in the remainder of the paper.

3.1 System Model

Our system model is a tuple $(N, N_0, v, D, D_0, \Delta, C, \Psi)$ where:

N is a finite set of control nodes;

N_0 is a subset of N containing initial control nodes;

v is a finite vector $[v_1, \ldots, v_k]$ of data variables;

D is the domain of v obtained as a cross-product of the domains of the components v_i as $D_1 \times \ldots \times D_n$;

D_0 is a subset of D describing the initial values of data variables;

C is a finite set of conditions on data variables of the form $c(v) \bowtie 0$ where $\bowtie \in \{<, \leq, =, \neq, \geq, >\}$ and $c : D \longrightarrow \Re$;

Ψ is a finite set of data constraints of the form $\psi(v, v') \bowtie 0$ where v' is a vector of data variables identical to v and $\psi : D \times D \longrightarrow \Re$; and,

Δ is a relation between $N \times N$ and $b(C) \times b(C) \times b(\Psi)$, where $b(C)$ and $b(\Psi)$ are sets of finite boolean combinations of data-conditions C and data-constraints Ψ, respectively.

Data conditions in C define data regions of the state-space in terms of numeric inequalities while the data constraints in Ψ capture joint constraints involving data values in the pre-sate and post-state. This system model can be used to represent a wide range of reactive systems. For our purposes, we require some restrictions on the type of data-constraints ψ, which are presented later in this section.

This system model can be considered to be a basic transition system $M = (S, S_0, R, L, AP)$, where:

- $S = N \times D$; the state has control and data parts.
- $S_0 = N_0 \times D_0$; the initial states are given by the initial values for control and data parts.
- $R((m, x), (n, y)) \iff \exists \alpha, \beta \in b(C), \exists \gamma \in b(\Psi) : \Delta((m, n), (\alpha, \beta, \gamma)) \wedge \alpha(x) \wedge \beta(y) \wedge \gamma(x, y)$; i.e., there is a transition between two states exactly when the system model has a transition between the corresponding control and data parts as defined by Δ.
- $AP = N \cup C$; i.e., propositions are control nodes and data conditions.
- $L(n, x) = \{n\} \cup \{\alpha \in C \mid \alpha(x)\}$; i.e., states are labeled by their control nodes and data conditions.

In this paper, when we refer to a basic transition system, it is assumed that there is an underlying system model, which is viewed as the basic transition system. 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.

We define some common notations below that we use in this paper.

1. For $s \in S, R(s) = \{t \in S \mid R(s, t)\}$ and for $A \subseteq S, R(A) = \bigcup_{s \in A} R(s)$.
2. $R^0(s) = R(s)$ and $R^{n+1}(s) = R(R^n(s))$ for $n \geq 0$.
3. $s|_N = n$, $s|_D = d$, when $s = (n, d) \in S$,
4. $R|_D = \{(x, y) \mid R((m, x), (n, y))$ for some $m, n \in N\}$.

$R|_D$ denotes the projection of the transition relation R onto the data component, and $s|_N$, $s|_D$ denote respectively, the control and data components of state s. We would use D_i instead of D, if the projection to the i^{th} data variable is required.

3.2 Basic Definitions

Following basic definitions will be used throughout this paper.

DEFINITION 1. *Two data points $x, y \in D$ are **data equivalent**, written $x \equiv y$, if $\forall c \in C : c(x) = c(y)$.*

$D/_{\equiv}$ denotes the set of equivalence classes induced by $\equiv$ on D, e_i denotes the i^{th} data equivalence class, and $rep(D/_{\equiv})$ denotes a set of representatives from D, one per equivalence class e_i. Intuitively, x is *data equivalent* to y if and only if x and y have the same valuations (i.e., same truth values) for all data conditions. A data equivalence class e_i is a set of data values that have same valuations for all data conditions.

DEFINITION 2. *Two states $s, s' \in S$ are **state equivalent**, written $s \simeq s'$, if $L(s) = L(s')$, i.e., $s|_N = s'|_N \wedge s|_D \equiv s'|_D$.*

Two states are state equivalent when they have the same control node and also their data nodes have same valuations for all data conditions. Note that two states are equivalent if and only if they have the same labels according to our system model. Data nodes from two equivalent states are data equivalent, but the reverse case is not necessarily true. $S/_{\simeq}$ denotes the set of equivalence classes induced by $\simeq$ on S and E_i denotes the i^{th} state equivalence class. A state equivalence class E_i is a set of states that are state equivalent to each other, i.e., a set of states with the same label.

We consider transition systems according to two categories: constraint free data transition systems, and constrained data transition systems.

DEFINITION 3. *A transition system M is a **constraint-free data transition system** if $R(s, t)$ for some s, t implies $R(s', t')$ for all $s' \simeq s$, $t' \simeq t$.*

In a *constraint-free data transition system*, existence of a transition between two states (s and t) implies the existence of a transition between any pair of states (s' and t') from their corresponding state-equivalence classes. In other words, the systems does not impose any constraints on how data variables change value.

When a system is not a constraint-free data transition system, the transition relation constrains how the data variables may change values. In general, the data constraints

that determine how the data variables can change value can be an arbitrary relation. In the work presented here we consider only relations that are functions over the data domain that satisfy certain restrictions. These restrictions are formally defined in the following paragraphs.

DEFINITION 4. *$R|_{D_i}$ is said to be a **constrained transition** if for each state-equivalence class E_j, there is a unique function $f_{E_j} : D_i \longrightarrow D_i$ such that, $t|_{D_i} = f_{E_j}(s|_{D_i})$, whenever $s \in E_j$ and $R(s,t)$ holds for states $s, t \in S$.*

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 are 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. Thus, whether or not a transition exists between two states is determined by the data transition function as well as the state labels.

When the transition functions are the same for all state equivalence classes, i.e., $f_{E_i} = f_{E_j}$ for all $E_i, E_j \in S/_{\simeq}$, we say that the data transition is a *global data transition.* Otherwise, we call it a *local data transition.* Global data transitions are defined uniformly over D while local data transitions are defined depending on each state label. Typically, global data transitions are used for imposing environmental constraints, such as physical laws, on data variables. Local data transitions are used for data maintained by the system such as counters and flags.

DEFINITION 5. *A transition system M is a **constrained data transition system** if $R|_{D_i}$ is a local or a global data transition for some i.*

A data variable v_i is said to be *globally constrained* if $R\,|_{D_i}$ is a global transition, and *locally constrained* if $R\,|_{D_i}$ is a local transition, and *unconstrained* if there is no data transition constraint imposed on the transition.

3.3 Bisimulation and Simulation Relations

We use definitions of bisimulation and simulation relations borrowed from [10] for formal proofs presented in the next two sections.

DEFINITION 6. *Let $M = (S, S_0, R, L, AP)$ and $M' = (S', S'_0, R', L', AP)$ be two structures with the same set of atomic propositions AP. A relation $B \subseteq S \times S'$ is a **bisimulation relation** between M and M' if and only if for all s and s', if $B(s,s')$ then following conditions hold:*

1. *$L(s) = L'(s')$*
2. *For every state s_1 such that $R(s,s_1)$ there is s'_1 such that $R'(s',s'_1)$ and $B(s_1,s'_1)$.*
3. *For every state s'_1 such that $R'(s',s'_1)$ there is s_1 such that $R(s,s_1)$ and $B(s_1,s'_1)$.*

DEFINITION 7. *The structures M and M' are **bisimulation equivalent** if there exists a bisimulation relation B such that for every initial state $s_0 \in S_0$ in M there is an initial state $s'_0 \in S'_0$ in M' such that $B(s_0,s'_0)$. In addition, for every initial state $s'_0 \in S'_0$ in M' there is an initial state $s_0 \in S_0$ in M such that $B(s_0,s'_0)$.*

Bisimulation equivalence guarantees that M and M' satisfy the same set of CTL^* formulas constructed using the atomic propositions in AP. However, bisimulation is a strong requirement and for many systems it may not yield a significant reduction in the state-space. In other to achieve a greater reduction, the notion of a *simulation relation* is introduced.

DEFINITION 8. *Given two structures M and M' with $AP' \subseteq AP$, a relation $H \subseteq S \times S'$ is a **simulation relation** between M and M' if and only if for all s and s', if $H(s,s')$ then the following conditions hold.*

1. *$L(s) \cap AP' = L'(s')$.*
2. *For every state s_1 such that $R(s,s_1)$, there is a state s'_1 such that $R'(s',s'_1)$ and $H(s_1,s'_1)$.*

DEFINITION 9. *M' **simulates** M if there exists a simulation relation H such that for every initial state s_0 in M there is an initial state s'_0 in M' such that $H(s_0,s'_0)$.*

When M' simulates a structure M, every behavior of M is also a behavior of M'. However, the abstracted structure M' may have behaviors that are not possible in the original structure M. It is shown that if M' simulates M then every $\forall CTL^*$ formula satisfied by M' is also satisfied by M [10].

4. CONSTRAINT-FREE SYSTEMS

When the system does not have data constraints, one could bisimulate it with a structure that has one representative from each equivalence class. This is formally established here.

4.1 Abstraction

For a given constraint-free data transition system $M = (S, S_0, R, L, AP)$, let $D' = rep(D/_{\equiv})$ and $M' = (S', S'_0, R', L', AP)$ where $S' = N \times D'$, $R' = R \cap (S' \times S')$, $S'_0 = S_0 \cap S'$ and $L' : S' \longrightarrow 2^{AP}$.

THEOREM 1. *The state equivalence relation $\simeq$ is a bisimulation relation between M and M'.*

PROOF. Suppose $s \simeq s'$ where $s \in S, s' \in S'$.

1. $L(s) = L(s') = L'(s')$ since s and s' have the same control label and the same valuation for all data conditions.

Figure 1: The ASW system

2. Suppose $R(s,t)$ holds for some $t \in S$. Let e_i be the data-equivalence class of t, i.e., $t|_D \in e_i$. By the definition of D', there is some $r_i \in D'$ such that $r_i \in e_i$. Let $t' = (t|_N, r_i)$. $t \simeq t'$ is obvious. Since M is a constraint-free data transition system, by definition, $R(s,t) \wedge s \simeq s' \wedge t \simeq t'$ implies $R(s',t')$. Therefore $R'(s',t')$ holds since $R' = R \cap (S' \times S')$ and $s', t' \in S'$.

3. Suppose $R'(s',t')$ holds for some $t' \in S'$. By the definition of R', $R(s',t')$ holds in M where $t' \in S' \subseteq S$. Let $t = t'$. From the fact that M is a constraint-free data transition system, $R(s',t') \wedge s' \simeq s \wedge t' \simeq t$ implies $R(s,t)$.

□

THEOREM 2. *The structures M and M' are bisimulation equivalent.*

PROOF. Given an initial state $s_0 \in S_0$, let $e_i \in D/_{\equiv}$ be the equivalence class to which s_0 belongs. By definition, D' contains a representative r_i of the class e_i. Let $s'_0 = (s_0|_N, r_i)$. Then, $s'_0 \in S'$ and $s_0 \simeq s'_0$. The converse is trivial. □

As a result, we can reduce the number of states of the abstracted system by choosing one representative from each data equivalence class without affecting the system behavior. The resulting system M' is an exact abstraction of the concrete system M. The idea behind this abstraction technique is similar to that of partition testing. When the input data space can be partitioned in a way such that the system behavior is determined completely by identifying the partition to which a given input data belongs irrespective of the actual data values, one could completely test the system by selecting one representative from each input partition [3].

4.2 Example: Constraint Free Altitude Switch

Now, we show a practical application of the abstraction technique for a constraint-free system. Here, an integer valued representative of each equivalence class is effectively computed by finding real-valued representatives and obtaining the closest integer in the class—a problem of polynomial complexity.

Our example is drawn from the the avionics domain: the Altitude Switch (ASW) [25]. The ASW is a hypothetical device that turns on another subsystem (the Device-Of-Interest–DOI), whenever the aircraft descends below a threshold altitude and turns the power back off again when the aircraft ascends above the threshold (plus a hysteresis factor). The hysteresis factor is defined in terms of the threshold—the ASW turns off the DOI when the aircraft ascends above $threshold + \frac{threshold}{50}$. Other systems in the cockpit may independently turn the DOI off and on. The input variable *DOIval* indicates the status of the DOI. There are no initial values or data constraints in the system — this is a constraint-free system. Figure 1 shows the ASW state transition diagram.

In this setting we may want to check properties such as "the command will not be issued if *DOIval=Off* and altitude is above threshold". This property can be specified in CTL^* as:
safety1: $AG((DOIval = Off) \wedge (altitude > threshold) \Rightarrow AX\neg command)$

Model checking properties such as this without abstraction is infeasible because of the large numeric variables in this

eq_class	min(a)	max(a)	min(t)	representative (a, t)
e_1	0	35000	2000 when $a = 0$	(0, 2000)
e_2	2000	35000	2000 when $a = 2000$	(2000, 2000)
e_3	2040	40000	2000 when $a = 2040$	(2040,2000)
e_4	2001	35700	2000 when $a = 2001$	(2001,2000)

Table 1: Selection of a representative from each equivalence class

system; *altitude: 0..40,000* and *threshold: 2,000..35,000.*

To compute the equivalence classes needed for abstraction, the numeric conditions are extracted from the transition relation and the property in question. In this example, *altitude* $<$ *threshold*, *altitude* $\geq$ *threshold*, *altitude* $\geq$ *threshold* $+\frac{threshold}{50}$ are the data conditions extracted from the transition constraints and *altitude* $>$ *threshold* is extracted from the property *safety1.* Note that this condition is not part of the system description (i.e., initial state and transition conditions) but is rather an environment condition specified in the property for verification. These conditions define the equivalence classes.

In general, if we have n conditions, we have 2^n possible equivalence classes—in this case 16. In actuality, the number of satisfiable equivalence classes is much smaller. There are ways of reducing the number of equations we need to solve to find the equivalence classes, but this problem is beyond the scope of this report and will not be discussed further here. In the ASW, there are only four satisfiable equivalence classes: (below, a represents *altitude*, t represents *threshold*)

$$e_1 \ : \ a < t \ \wedge \ a \leq t \ \wedge \ a < t + \frac{t}{50}$$
$$e_2 \ : \ a \geq t \ \wedge \ a \leq t \ \wedge \ a < t + \frac{t}{50}$$
$$e_3 \ : \ a \geq t \ \wedge \ a > t \ \wedge \ a \geq t + \frac{t}{50}$$
$$e_4 \ : \ a \geq t \ \wedge \ a > t \ \wedge \ a < t + \frac{t}{50}$$

Using a constraint solver, the maximum and the minimum value in the class are identified and one of them is used for selecting the representative (Table 1). The constraint solver $CLP(q, r)$ [11] is used for this purpose. The last column shows the *(altitude, threshold)* pair representing each equivalence class—this reduces the input domain to *altitude:* {*0, 2000, 2001, 2040*} and *threshold:* {*2000*}. With this abstraction, the property *safety1* is verified to be true using the model checker NuSMV [24] in seconds.

Since the DOI may be a safety system that is only turned on as the aircraft is descending, a desired property may be that the ASW shall never turn the DOI on while ascending. One could formulate the property that the DOI is not turned on, using the following CTL^* property:
safety2: $AG((DOIval = OFF \wedge \neg(DOI\_Status = On)) \Rightarrow AX\neg(DOI\_Status = On))$.
The abstraction for constraint-free Altitude Switch described above is not sufficient to express the fact that the aircraft is ascending without imposing data constraints on altitude. In the system as described above, the altitude may change unconstrained (thus we have no concept of the aircraft climbing) or the threshold may be changed at any time so that in one step the aircraft is above the threshold and in the next it is below because of an increase in threshold. In actuality, the environment of the ASW is constrained—the threshold is always *fixed* before flight and constant thereafter, and the altitude can only change so much between two steps. This gives us a data constrained system where *threshold' = threshold*, and the altitude can be constrained to be in an ascent, for example, *altitude' = altitude + 10.* Such a system is a constrained data transition system – specifically, global data transition system – and will be discussed in the next section.

5. CONSTRAINED SYSTEMS

When a system has constrained data variables, domain reduction based on state equivalence still produces a conservative abstraction, but it is not sufficient to preserve path properties in the concrete system. A more refined abstraction is required to preserve interesting path properties while at the same time reducing the size of the data domain.

To get such a refined data abstraction for constrained data systems, we define a reduced domain D' such that it includes every possible data path in terms of data equivalence. The approach can be conceptually viewed as follows: For each path in M, find the sequence of data equivalence classes as given by the states in the path in order. Find a shortest path in M that passes through the same sequence of data equivalence classes. The reduced domain D' contains all the data values in the shortest path. As we will show later, with some restrictions, the abstract system obtained by such a domain abstraction simulates the concrete system.

5.1 Definitions and Assumptions

First, we introduce some necessary definitions and two assumptions that are used to establish the simulation relation.

DEFINITION 10. *A* ***path*** *in the structure M from a state s is an 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.*

For the purposes of the abstraction approach described here, the constrained data transition systems that we consider are assumed to satisfy the following assumptions.

ASSUMPTION 1. *Constrained and unconstrained data are exclusive in a system, i.e., a data condition cannot refer to both unconstrained and constrained data variables.*

ASSUMPTION 2. *Transition relations between control nodes are deterministic on labels and data transition constraints; i.e.,* $R(s,t) \wedge s \simeq s' \wedge t \simeq t' \wedge (s'|_D, t'|_D) \in R|_D$ *implies* $R(s',t')$.

The first assumption is to separate out non-determinism in the transition system. Based on the first assumption, we can separate numeric conditions with constrained data variables from numeric conditions with unconstrained data variables. For data equivalence classes partitioned by numeric conditions with unconstrained data variables, applying the abstraction technique described in the previous section suffices. Thus in this section, without loss of generality, we assume that all data variables are constrained.

By the second assumption, the control node transition is deterministic on state labels and the data transition constraint. In other words, given a valid path (refer to Definition 10), any sequence of states that are state-equivalent to the corresponding states in the given path, is also a valid path, provided that the data nodes in the sequence satisfy the same data transition constraints as those in the given path, in order.

DEFINITION 11. *A* ***trace*** $T[\pi(s_0)]$ *of a path* $\pi(s_0)$ *is a sequence of data equivalence classes* $e_0e_1e_2\dots$ *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* $s_{i+1}|_D \in e_{j+1}$.

In other words, a trace $T[\pi(s_0)]$ is a permutation of a subset of data equivalence classes of D that are reachable from s_0 in order.

DEFINITION 12. *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.

Nodes of change are defined with respect to a given path. These are states along the path at which there is a change in data equivalence class as one traverses the path in order starting from s_0.

DEFINITION 13. *The* ***segment length sequence*** $N[\pi(s_0)]$ *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* $s_p|_D \in e_i$, $s_q|_D \in e_{i+1}$, $p = \Sigma_{j=0}^{i-1} n_j$ *and there is no node of change between* s_p *and* s_q.

Thus, $N[\pi(s_0)] = \{n_1, n_2, \dots\}$ is an ordered set of natural numbers that represents the minimum 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.

DEFINITION 14. *A path* $\pi(s_0)$ *is a* ***minimal path*** *if and only if* $N[\pi(s_0')] \not\prec N[\pi(s_0)]$ *for any path* $\pi'(s_0')$ *with* $T[\pi(s_0)] = T[\pi'(s_0')]$ *and* $s_0 \simeq s_0'$.

We call s_0 a minimal state when $\pi(s_0)$ is a minimal path. Intuitively, a minimal path is a path with minimal segment lengths among all paths with the same trace.

Our reduced domain for data variables is composed of data nodes on the minimal paths.

DEFINITION 15. D' *is a subset of* D *including* $\{R^n|_D(s_0) \mid 0 \leq n \leq \Sigma n_j,\ n_j \in N[\pi(s_0)],\ s_0$ *: minimal state,* $\pi(s_0)$ *: minimal path*$\}$.

D' is the reduced domain for abstraction including one trace representative per minimal path up to the last node of change. It includes every data node on a minimal path obtained by repeatedly applying the data transition relation from each minimal state until it reaches the last node of change.

5.2 Abstraction

We first introduce a data identity transition R_{id}^D defined as $((n,x),(n',x)) \in R_{id}^D$ if $((n,x),(n',x')) \in R$ 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 stuttering of data nodes.

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 $AP' = AP$, $S' = N \times D'$, $S_0' = N_0 \times D_0'$, $L' : S' \longrightarrow 2^{AP'}$, and $R' = (R \cup R_{id}^D) \cap (S' \times S')$. Here, D' is the reduced bound defined in Definition 15 and $D_0' = \{d_0' \in D' \mid d_0' \equiv d_0$ for some $d_0 \in D_0\}$.

THEOREM 3. *For any path* $\pi = s_0s_1\dots s_n\dots$ *in* M, *there exists an* equivalent path $\pi' = s_0's_1'\dots s_n'\dots$ *in* M' *such that* $s_i \simeq s_i'$ *for all* i.

PROOF. By definition of D', there exists a minimal path $\tilde{\pi}(t_0)$ in M' such that $T[\pi] = T[\tilde{\pi}]$ and $N[\tilde{\pi}] \preceq N[\pi]$ where $t_0 \simeq s_0$. Define $\pi' = s_0's_1'\dots s_n'\dots$ as follows:
(1) $\forall i,\ s_i'|_N = s_i|_N$.
(2)

$$s_i'|_D = \begin{cases} t_0|_D & \text{if } i = 0 \\ x = R|_D(s_{i-1}') & \text{if } x \equiv s_i|_D \text{ and } x \in D' \\ s_{i-1}'|_D & \text{otherwise} \end{cases}$$

$s_i \simeq s_i'$ is guaranteed by the fact that $T[\pi] = T[\tilde{\pi}]$ and $N[\tilde{\pi}] \preceq N[\pi]$. $s_i' \in S'$ for all i since any node of change along the path $\tilde{\pi}$ is included in D'. π' is a path in M' since $R'(s_i', s_{i+1}')$ holds either by $R(s_i', s_{i+1}')$ or by the data identity transition. □

By Theorem 3, for any path in M there is an equivalent path in M' with equal length with data stuttering.

DEFINITION 16. *$s' \in S'$ is* on the same trajectory *as $s \in S$ if and only if*

1. *$s \simeq s'$ and*
2. Existence of trajectory reduction :
   Given path $\pi(s) = ss_1s_2\ldots$ in M, $\exists\pi'(s') = s's'_1s'_2\ldots$ in M' such that
   (a) trace equivalence : $T[\pi(s)] = T[\pi'(s')]$ *and*
   (b) path equivalence of each trace segment : $s_i \mid_D \equiv s'_i \mid_D$ *implies* $s_i \simeq s'_i$ *for all i, and*
   (c) trajectory reduction : $N[\pi'(s')] \preceq N[\pi(s)]$.

Intuitively, a state $s' \in M'$ is on the same trajectory as $s \in M$ if a path $\pi'(s')$ has the same trace as a path $\pi(s)$ and it reduces the number of steps needed to reach each data equivalence class along the path $\pi(s)$. Note that if $\pi'(s'_0)$ in M' is an equivalent path of $\pi(s_0)$ then s'_0 is on the same trajectory as s_0 when M is a system with deterministic data transition constraints.

Now, we define a relation between M and M' to prove that M' simulates M.

DEFINITION 17. *$H \subseteq S \times S'$ is a relation on $S \times S'$ such that $H(s, s')$ if and only if s' is on the same trajectory as s.*

THEOREM 4. *H is a simulation relation between M and M'.*

PROOF. 1. Given $H(s, s')$, $L(s) = L(s') = L'(s')$ from $s \simeq s'$ and by the definition of the labeling function. $L(s) \cap AP' = L'(s')$ is obvious since $AP = AP'$

2. Suppose $R(s, t)$ for some $t \in S$. By the determinism of control node transition and data transition constraints, the path $\pi(st)$ is uniquely defined and there is a path $\pi'(s'k)$ in M' such that $T[\pi(st)] = T[\pi'(s'k)]$ and $N[\pi'(s'k)] \preceq N[\pi(st)]$ by $H(s, s')$.
   (a) Case 1 : $k \simeq t$ when $k \mid_D \equiv t \mid_D$
   Let $t' = k$ and $\pi'(t') = \pi'(s'k) - \{s'\}$. $t' \simeq t$ is clear and $R'(s', t')$ is a regal transition in M' by $R'(s', k)$. $H(t, t')$ directly follows from $H(s, s')$.
   (b) Case 2 : $t \simeq s \simeq s'$ when $k \mid_D \not\equiv t \mid_D$.
   Let $t' = s'$ and $\pi'(t') = \pi'(s'k)$. $t' \simeq t$ and $R'(s', t')$ holds by identity transition. $\pi'(t')$ satisfies the trace equivalence and path equivalence condition. To see that the trajectory reduction holds, let $N[\pi'(s'k)] = \{n_0, n_1, n_2, \ldots\}$ and $N[\pi(st)] = \{m_0, m_1, m_2, \ldots\}$. We can infer that $n_0 < m_0$ and $\forall i, n_i \leq m_i$ from $N[\pi'(s'k)] \preceq N[\pi(st)] \wedge t \simeq s \wedge k \not\simeq t$. Note that $N[\pi(t)] = \{m_0 - 1, m_1, m_2, \ldots\}$, i.e., $m'_0 = m_0 - 1$ and $\forall i,\ m'_i = m_i$ for $m'_i \in N[\pi(t)]$. $N[\pi'(s'k)] = N[\pi'(t')] \preceq N[\pi(t)]$ follows.

Therefore, t' is on the same trajectory as t and $H(t, t')$ holds. □

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

Figure 2: Trace and minimal state generation Algorithm

THEOREM 5. *M' simulates M.*

PROOF. Directly follows from Theorem 3. □

We conclude that domain abstraction based on trajectory reduction is conservative.

5.3 Example: Constrained Altitude Switch

In this section, we show a possible approach for automated domain reduction for constrained data transition systems using the constrained Altitude Switch example discussed in Section 4.2.

For constrained data transition systems, constructing a reduced data domain is more involved; it is necessary to identify data values along a minimal path for each unique trace. There is no guarantee that the abstraction will produce a reduced domain for all systems and we may end up with the original domain D in the worst case. Nevertheless, in practice, this method results in domain reduction for many systems. For example, domain reduction can be automated for systems that satisfy following assumptions: (see Figure 2).

1. Data transition constraints are independent of control nodes.
2. Data transition constraints are not cyclic. A data transition constraint $R|_D$ is cyclic if starting from a state it is possible to reach the same equivalence class of D more than once by applying the data constraints.
3. Each numeric function appearing in the numeric conditions is continuous and grows within a polynomial bound. This guarantees that the numeric conditions are not periodic. Together with Assumption 2, this guarantees that the trace of each path is finite since the number of equivalence classes is finite.

By these assumptions, we know that the number of reachable data equivalence classes is finite. Therefore, for any

trace	step	k	$SAT \wedge Target$	satisfy?	d	continue?	(a,t)
$e_1 \rightarrow e_3$	0	1	$a < t$ & $a < t + t/50$ & $a + 10 \geq t$ & $a + 10 \geq t + t/50$	no	30	yes	
		2	$a < t$ & $a < t + t/50$ & $a + 20 \geq t$ & $a + 20 \geq t + t/50$	no	20	yes	
		3	$a < t$ & $a < t + t/50$ & $a + 30 \geq t$ & $a + 30 \geq t + t/50$	no	10	yes	
		4	$a < t$ & $a < t + t/50$ & $a + 40 \geq t$ & $a + 40 \geq t + t/50$	no	0	yes	
		5	$a < t$ & $a < t + t/50$ & $a + 50 \geq t$ & $a + 50 \geq t + t/50$	yes		finish	$a = 1990$, $t = 2000$
$e_1 \rightarrow e_3 \rightarrow e_2$	5	1	$a < t$ & $a < t + t/50$ & $a + 50 \geq t$ & $a + 50 \geq t + t/50$ & $a + 60 \geq t$ & $a + 60 < t + t/50$	no	10	yes	
		2	$a < t$ & $a < t + t/50$ & $a + 50 \geq t$ & $a + 50 \geq t + t/50$ & $a + 70 \geq t$ & $a + 70 < t + t/50$	no	20	no	

Table 2: Sample trace computation for Altitude Switch

given equivalence class that may have the initial data nodes, we can compute a sequence of reachable equivalence classes with minimum segment lengths based on the data transition constraints with the aid of a constraint solver. Figure 2 outlines a prototype minimal state generation algorithm.

Initially, $step = 0$, SAT is an equivalence class e_i and $REACH = \{e_1, e_2, e_3, e_4\} - \{e_i\}$. The algorithm recursively computes the satisfiability of $SAT \wedge (f^m(e_j))$ by exercising all permutations of the set of equivalence classes until there are no more equivalence classes to be added in the trace SAT. We use the Euclidean distance between SAT and $f^m(e_j)$ as a progress measure. When there are no more satisfiable equivalence classes to be added to SAT, we then pick a vector x from the region of SAT. x generates a sequence of reachable equivalence classes in a minimum number of steps.

Note that the constrained altitude switch example satisfies the above assumptions: since the data constraint is global, it is independent of control nodes; the data transition constraint function is linear and so the second assumption is satisfied; and, the numeric conditions are also linear, and so the third assumption is satisfied.

Recall that the property we want to check is $safety2$: $AG((DOIval = OFF \wedge \neg(DOI\_Status = On)) \Rightarrow AX\neg(DOI\_Status = On))$, assuming the system models an ascending aircraft. Again, the data conditions $altitude < threshold$, $altitude \geq threshold$, and $altitude \geq threshold + \frac{threshold}{50}$ extracted from the transition constraints are used to define data equivalence classes (refer to Section 4.2). The other data condition $altitude > threshold$ is not used since $safety2$ does not contain it. This produces slightly different data equivalence classes from the previous case.

$$e_1 \ : \ a < t \ \wedge \ a < t + \frac{t}{50}$$
$$e_2 \ : \ a \geq t \ \wedge \ a < t + \frac{t}{50}$$
$$e_3 \ : \ a \geq t \ \wedge \ a \geq t + \frac{t}{50}$$

Using the algorithm, we compute a data node for a minimal state and corresponding trace based on the set of equivalence classes $\{e_1, e_2, e_3\}$ as follows. Starting from an equivalence class e_i, we check the satisfiability of $e_i \wedge f(e_j)$ where $f(e_j)$ is a transformation of each numeric condition in e_j by applying the data transition constraint. For example, the data condition $a < t$ that helps define e_1 would be transformed to $a + 10 < t$ since the data constraints force $t' = t$ and $a' = a + 10$. If $e_i \wedge f(e_j)$ is satisfiable, it means that e_j is reachable from e_i in one step. Otherwise, there are two cases; e_j is not reachable from e_i or is reachable from e_i in more than one step. We measure progress by computing the Euclidean distance between the two regions e_i and $f^n(e_j)$ from $n = 1$—if the distance increases in an iteration, given our restrictions, we know the equivalence class is not reachable and we can terminate this computation. If the distance is shrinking, we keep applying the transformation to e_j until $e_i \wedge f^n(e_j)$ is satisfied. When $e_i \wedge f^n(e_j)$ is satisfiable, then n is the minimum number of steps required to move from e_i to e_j. The constraint solver $CLP(q, r)$ [11] is used for the satisfiability check in the algorithm.

Table 2 shows a sample trace generation starting from the equivalence class e_1. The domain constraints $0 \leq a \leq 40000$ and $2000 \leq t \leq 35000$ are assumed in the table. The table shows that e_3 is reachable from e_1 in five steps and there are no other reachable classes further. This produces the trace $e_1 \rightarrow e_3$.

Two traces $\{e_1 \rightarrow e_3\}$, $\{e_1 \rightarrow e_2 \rightarrow e_3\}$ are generated by the application of the algorithm to e_1, e_2, and e_3 in order. Both traces required 5 steps. We identified the initial points of the traces to be $(a, t) = (1990, 2000)$ in the region that satisfies $\{e_1 \rightarrow e_3\}$ and $(a, t) = (1991, 2000)$ in the region that satisfies $\{e_1 \rightarrow e_2 \rightarrow e_3\}$. With these initial points, we can generate the required domain for *altitude* (a) and *threshold* (t) by iterating the data constraints five times (one for each trajectory—in this case they were both five steps long). This yields the following domains:
altitude: {*1990, 1991, 2000, 2001, 2010, 2011, 2020, 2021, 2030, 2031, 2040, 2041* }
threshold: { *2000* }

Using the reduced domains, the property $safety2$ is verified in seconds with NuSMV [24].

6. DISCUSSION

In this paper we described *domain abstraction* as a complement and alternative to other abstraction techniques, such as predicate abstraction. The abstraction technique yields a bisimulation for constraint free systems with large integer inputs and polynomial transition constraints over the data domain. We also show how the technique is extended to data constrained systems with deterministic constraints, but in this case the abstraction yields a simulation relation. We provided proofs for both these claims.

We are aware that the limitation to deterministic data constraints is quite severe. Most realistic applications require at least limited non-determinism in the data constraints, for example, we would like to model the altitude to change at most 300 *up or down* each step. We believe that the technique will extend to systems with non-deterministic data constraints—we have to date, however, been unable to complete a proof for this case. This is a topic currently under investigation.

In our case studies, the technique has been remarkable effective and we believe that in conjunction with other reduction techniques, such as slicing, we will be able to extend the reach of model checking into the domain of software specification for a class of systems of particular interest to us, namely safety-critical control systems.

7. 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] J. Atlee and J. Gannon. State-based model checking of event-driven system requirements. In *Proceedings of the ACM SIGSOFT '91 Conference on Software for Critical Systems. Software Engineering Notes. Volume 16 Number 5*, 1991.

[3] B. Beizer. *Software testing techniques*. Van Nostrand Reinhold, New York, 2nd edition, 1990.

[4] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using qdds. *Formal Methods and System Design*, 14(3):237–255, May 1999.

[5] T. Bultan, R. Gerber, and C. League. Verifying systems with integer constraints and boolean predicates: A composite approach. In *Proceedings of ISSTA'98*, pages 113–123, march 1998.

[6] T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables: Symbolic representations, approximations, and experimental results. *ACM Transactions on Programming Languages and Systems*, 21(4):747–789, July 1999.

[7] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, April 1994.

[8] W. Chan, R. Anderson, P. Beame, S. Burns, F. Modugno, D. Notkin, and J. Reese. Model checking large software specifications. *IEEE Transactions on Software Engineering*, 24(7):498–520, July 1998.

[9] W. Chan, R. Anderson, P. Beame, and D. Notkin. Combining constraint solving and symbolic model checking for a class of systems with non-linear constraints. In *Computer Aided Verification*, pages 316–327. Springer Verlag, 1997.

[10] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[11] Constraint logic programming over rational or real. Available at http://www.ai.univie.ac.at/clpqr/.

[12] M. Colon and T. Uribe. Generating finite-state abstractions of reactive systems using decision procedures. In *Computer Aided Verification*, pages 293–304. Springer, 1998.

[13] D. Dams, O.Grumberg, and R. Gerth. Generation of reduced models for checking fragments of CTL. In *Computer Aided Verification*, pages 479–490. Springer Verlag, 1993.

[14] S. Das, D. L. Dill, and S. Park. Experience with predicate abstraction. In N. Halbwachs and D. Peled, editors, *Proceedings of CAV '99*, volume 1633 of *LNCS*, pages 160–171. Springer-Verlag, July 1999.

[15] 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.

[16] O. Grumberg and D.E.Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, May 1994.

[17] J. Hatcliff, M. B. Dwyer, and H. Zheng. Slicing software for model construction. *Higher-Order and Symbolic Computation*, 13(4):315–353, December 2000.

[18] M. P. Heimdahl, J. M. Thompson, and M. W. Whalen. On the effectiveness of slicing hierarchical state machines: A case study. In *Proceedings of the Twenty-fourth EUROMICRO Conference*, volume 1, pages 435–444, 1998.

[19] C. Heitmeyer, J. Kirby, B. Labaw, M. Archer, and R. Bharadwaj. Using abstraction and model checking to detect safety violations in requirements specifications. *IEEE Transactions on Software Engineering*, 24(11):927–948, November 1998.

[20] T. A. Henzinger and P.-H. Ho. HyTech: The Cornell Hybrid Technology Tool. In *Hybrid Systems II*, volume 999 of *Lecture Notes in Computer Science*, pages 265–294. Springer-Verlag, 1995.

[21] G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, pages 279–295, May 1997.

[22] Z. Manna, M. Colon, B. Finkbeiner, H. Sipma, and T. Uribe. Abstraction and modular verification of infinite-state reactive systems. In *Requirements Targeting Software and Systems Engineering (RTSE), LNCS*. Springer Verlag, 1998.

[23] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[24] NuSMV: A New Symbolic Model Checking. Available at http://http://nusmv.irst.itc.it/.

[25] J. M. Thompson, M. P. Heimdahl, and S. P. Miller. Specification based prototyping for embedded systems. In *Seventh ACM SIGSOFT Symposium on the Foundations on Software Engineering*, volume 1687 of *Lecture Notes in Computer Science*, pages 163–179, September 1999.

[26] W. Visser, S. Park, and J. Penix. Using predicate abstraction to reduce object-oriented programs for model checking. In *Proceedings of the Third ACM Workshop on Formal Methods in Software Practice*, pages 3–12, August 2000.

[27] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *Computer Aided Verification*, pages 88–97. Springer, 1998.

[28] P. Wolper and P. Godefroid. Partial-order methods for temporal verification. In *CONCUR'93*, volume 715 of *LNCS*, pages 233–246. Springer-Verlag, 1993.