On Under-Determined Dynamical Systems

Oded Maler
CNRS-VERIMAG
University of Grenoble
France
Oded.Maler@imag.fr

ABSTRACT

Under-determined dynamical systems are those that need additional information in order to produce simulation traces. This information may correspond to initial conditions, parameter values or dynamic external influences. The paper discusses this issue and surveys some common approaches to reconcile this fact with the practice of simulation.

Categories and Subject Descriptors

I.6.0 [**Simulation and Modeling**]: General

General Terms

Design, Performance, Verification

Keywords

Hybrid systems, Timed systems, Simulation, Verification

1. INTRODUCTION

The design and analysis of complex systems is often accompanied by mathematical and quasi-mathematical *models* of *dynamical systems*. We use the term in a broad sense which covers all sorts of dynamics and refers to systems that generate *behaviors* which are progressions in time of states (valuation of state variables). In the discrete case these are automata or other discrete-event models generating sequences of states and events, while in the continuous case these can be differential equations generating trajectories in some Euclidean space or manifold. Hybrid systems may generate a combination of both while *timed systems*, which constitute an extremely important level of abstractions which is used *implicitly* in almost any domain, generate discrete events embedded in the real-time axis. The reader is referred to [10] for more details and contemplations.

The state space of a system constructed from several components is a subset of the product of the local state spaces of the subsystems. Thus, for any system admitting a significant number of components one experiences either the *state-explosion problem* (discrete), or the *curse of dimensionality* (continuous) or a combination of both (hybrid). Such systems are not easy to reason about intelligibly by humans and computer-aided *simulation* is often used to generate behaviors of the system, to provide intuition about what is going on and check whether the system behaves as expected or desired.

Simulation may be different in the discrete, timed and continuous domains. Simulation of continuous systems involves many intricate problems associated with the approximation of the mathematical reals by a finite subset of the rationals (floating-point numbers) and the discretization of time. These include all the problems of numerical integration: error estimation, variable steps, forward and backward schemes etc. Discrete simulation need not worry about those but there might still be problems related to the order of simultaneous transitions and combinational loops due to the non-metric structure of the time domain. Hybrid systems simulation needs to solve the problem of event detection and may get into so-called Zeno behaviors where infinitely-many transitions are packed into a bounded time interval.

Abstracting away from these issues, there is one common fact underlying the very practice of simulation: in order to generate a *single* concrete behavior, you need the system to be *fully determined*, either statically or on the fly, during the production of the behavior in question. The next section attempts to make sense out of this somewhat cryptic statement, hopefully resulting in a clear and unified view of *simulation*, *verification*, *random testing* and *parameter-space exploration*, all of which represent different ways to cope with the tension between the *inherent* under-determination of system models and what is needed in order to simulate them. I use the term *under-determined* because the term *non-deterministic* is already loaded by various meanings (sometimes conflicting) in different communities.

The rest of the paper is organized as follows. Section 2 introduces dynamical systems and presents several forms of under-determination ranging from static (punctual) under-determination which corresponds to ignorance concerning initial states or parameters to dynamic under-determination associated with external influences. Then I illustrate two approaches for coping with under-determination in the continuous and hybrid domains: adaptive sampling (Section 3) and reachability computation (Section 4). Section 5 gives a brief overview of under-determination for timed systems.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
EMSOFT'11, October 9–14, 2011, Taipei, Taiwan.
Copyright 2011 ACM 978-1-4503-0714-7/11/10 ...$10.00.

2. UNDER-DETERMINATION

A dynamical system lives in a state space X that we assume to be finite dimensional, say $\mathbb{R}^n$ or $\{0,1\}^n$. A behavior, that we call from now on a (simulation) *trace*, is a sequence $x = x[0], x[1], \ldots$, wher $x[t] \in X$ for every t. We use *discrete time* with fixed steps throughout the paper but we should bear in mind that the time index could be any monotonic sequence $t_0, t_1, \ldots$ of time steps, generated, for example, by variable step numerical integration. For the simulator to produce such a trace it needs:

1. A value for $x[0]$;
2. A procedure to compute $x[t+1]$ from $x[t]$ which has some resemblance to the presumed real-world dynamics that the model represents.

This procedure can be either uniform ("time-invariant") so that $x[t] = x[t']$ implies $x[t+1] = x[t'+1]$, or it might be a particular choice made on the fly each time $x[t+1]$ is computed from $x[t]$. Systems of the former kind are fully deterministic "closed" systems, denoted by (X, f) where $f : X \to X$ is a function such that for every t, $x[t+1] = f(x[t])$. To generate a trace for such systems one needs only to fix $x[0]$ and then apply f successively to produce a trace.

Deterministic systems that admit only one conceivable initial state (these are more common in the discrete domain) produce a unique trace. Otherwise, the initial state gives us the first example of *under-determination*: a piece of information which is missing, an empty slot to be filled in order to produce a trace. We call this under-determination *static* or *punctual* because it requires one state, one point in the state space to be determined. We defer the discussion of the methodological implications of this issue after the introduction of the second type of static under-determination.

As much as one wants to endorse *model-based* design, we should not forget that models, especially continuous models of physical phenomena, are approximations of reality. Hence considering a precise deterministic model of any non-trivial natural phenomenon is an act of epistemological arrogance and self delusion unless one is aware of this fact and takes precautions. A common practice is to compactly store our ignorance concerning the real dynamics in *parameters*. Parameters are more often associated with numerical dynamics where the function f becomes a *template* with some empty slots that have to be filled by parameter values ranging in some parameter space P. Each parameter p instantiates f into a function f_p which is then used to compute $x[t+1]$ from $x[t]$ for every t.

Let us reflect a bit outside the mathematical model and illustrate what type of ignorance can be represented by parametric under-determination.

- Suppose we attempt to model a biochemical reaction inside the cell. In this case f may have a general form derived from *mass action* rules. These rules have kinetic parameters that depend, among other things, on the affinity between the chemical species. These parameters cannot be deduced from first principles but can be measured very roughly by isolated experiments in conditions different from those inside the cell. As a result of all these inaccuracies, one should not assume knowledge of parameter values, only their range;
- Another example is the electrical dynamics of a network of transistors. It is well known that the production of semi-conductor devices is a process of high variability and the characteristics of transistors may change from one production batch to another and even in different areas of the same chip. The problem becomes more acute when circuits have to be designed in parallel with the development of the new material substrate ("technology") in which they will be realized;
- A last example of a more timed and discrete nature is taken from *design-space exploration* for embedded systems. Suppose we want to evaluate the performance of some application software running on a multi-core execution platform. Such an evaluation is based, among other things, on the *execution times* of tasks. When the software is not completely written and the architecture not yet realized, we will have only rough estimates of these numbers.

Although conceptually different, parameters and initial states are mathematically similar. In fact, one can extend the state space and dynamics to include parameters as static state variables that do not change once their value at time 0 is determined, that is, replacing (X, f) by (X', f') where $X' = X \times P$ and $f'(x,p) = (f_p(x), p)$. This can be viewed as if upon initialization the simulator decides which dynamical model f_p to use for producing traces. The difference between a parameter and an initial state is just in the space they are taken from. Sometimes the state-space may be high-dimensional with few parameters, sometimes the state-space is modest and parameters are numerous.

Approaches for handling parameter under-determination range from complete indifference to pedantic anxiety depending on the application domain and its criticality, the mathematical properties of the state-space and dynamics as well as cultural traditions. The most liberal approach to under-determination is to fix some *nominal* values for the parameters and act as if those are the exact values of the parameters. This practice raises the question: what do we learn on the real system by running a simulation with these nominal values? What can we conclude from the properties of the nominal trace about the traces that will be produced by other parameters that may happen to be more faithful to the real dynamics than the nominal ones?

The attitude toward this question depends on many factors. The first one is the *responsibility* of the person that builds the model and performs the simulation. The following, somewhat caricatural, scenario illustrates an extreme case of lack of real responsibility. Imagine a purely scientific context, say in certain type of biological research, where a researcher proposes a model to explain some phenomenon. He or she performs experiments and tunes the model parameters so that the obtained traces resemble observed behaviors. One such a fit has been achieved, a paper can be published. The fact that the behavior can be significantly different with a change of parameters (and the model less robust) will not have any practical implications unless some referee has time to invest the effort. Of course, we exaggerate a bit: the modeler may have both mathematical knowledge and integrity to observe this fact during the tuning process. The implications are completely different when the model has to be combined with other models or be used in a real-life context, for example in pharmacology.

More serious justifications for using nominal values may come from the mathematical properties of the model. To some extent, certain kinds of continuous dynamical systems admit also some continuity in the sensitivity of their qualitative properties to parameter values and in this sense a nominal value p in the parameter space may represent a large neighborhood of parameter values that lead to similar behaviors. There are also specialized mathematical techniques (bifurcation analysis) that can compute the range of parameter variations that preserve the properties of the behavior exhibited nominally. Like everything in the continuous domain, these techniques work better for linear systems and steady-state behaviors and they often break down when switching discontinuity is introduced as in hybrid systems. Yet another reason not to worry too much about parameter variability is in the context of *control* systems. Such systems are designed to cope with external disturbances (dynamic under-determination) and have built-in error-correction mechanism based on feedback so that possible deviations of the system from its nominal behavior are taken into account.

What should one do when there is no a priori reason to be content with one nominal value? Naturally one would like to *cover* as much as possible the non-countable parameter space. There are two basic approaches:

- *Finite sampling*: a finite number of parameter values are chosen and their corresponding traces are generated by the simulator. The sampling method can vary from a fixed grid discretization to random sampling, with or without making probabilistic assumptions and claims. This way or another, the number of sampling points needed to achieve a given coverage will grow exponentially in the corresponding dimension (state space for initial state, parameter space for parameters). Section 3 illustrates a new technique for parameter space exploration that can direct the sampling toward interesting parts of the parameter space.

- *Exhaustive coverage*: this is an idea inspired from discrete formal verification. It is essentially a kind of *set-based* simulation, which goes by the name of *computing reachable sets*. Rather than computing single trajectories, this approach produces tubes of trajectories, also known as *flowpipes*, that constitute an over-approximation of all the states reachable by *all* trajectories emanating from *all* initial states under *all* parameter values. While this may seem magical from the point of view of simulation, it is perhaps best understood as the difference between depth-first and breadth-first search in the space of system trajectories. Section 4 explains the essentials of this approach which works also for dynamic under-determination which is introduced next.

Systems operate in environments that are by themselves *dynamic*. Such environments may influence the behavior of the system that we simulate in a non-uniform manner over time. For example, consider the influence of dynamic temperature variations on the behavior of a transistor or a chemical reaction, the influence of wind on an airplane, or the effect of communication traffic congestions on the response time of an embedded system.

Mathematically, such an open system is defined as (X, V, f) where V is the domain of input variables and $f : X \times V \to X$ is a function such that $x[t+1] = f(x[t], v[t])$ for every t. Such a system exhibits under-determination in each and every step and for each simulation trace we generate we need to provide the simulator with a sequence of input values $v = v[0], v[1], \ldots$ whose length grows with the duration of the trace. Although from the point of view of this paper all inputs are alike, namely, values produced by an *external* source whose behavior is not specified in detail *inside* the model, it should be kept in mind that in practice they can be of different sorts, ranging from distinct stimuli that require a specific reaction to arbitrary disturbances and noise.

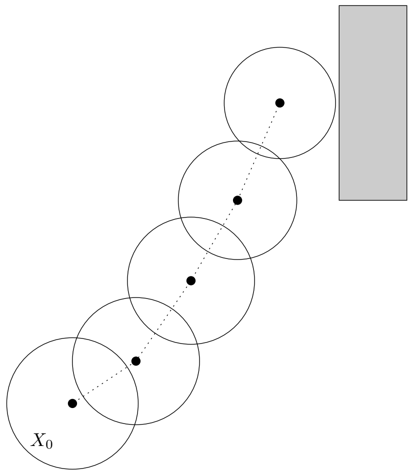

Figure 1: A trajectory decorated with balls.

Like static under-determination, the practice of handling dynamic under-determination is diverse. One may take a nominal input sequence which can be a *constant* sequence (zero or otherwise), a *step* (a sudden change which remains constant thereafter), a *periodic* sinusoid-like input or an arbitrary sequence of elements from V possibly chosen randomly at each step. For the simpler classes of systems, the response to this nominal input may tell us a lot about the response to all inputs. Otherwise, covering the space of inputs by a representative sample is not a trivial matter. The reason is that a parameter space is finite dimensional while the space of dynamic inputs has infinitely many dimensions. Speaking in terms of discrete time, a parameter space $P = [0,1]^m$ can be sampled with ϵ-coverage by $[1/\epsilon]^m$ points while the space of input sequences of length k over $V = [0,1]^m$ needs $[1/\epsilon]^{mk}$ samples for a similar coverage.

3. SENSITIVITY-BASED SAMPLING

This section illustrates an approach due to [4] for adaptive sampling for static under-determination. Suppose we want to verify that all traces of length up to k of a system (X, f) satisfy some safety property (never reach a bad part of the state space), given that the initial state is in some set X_0. Numerical simulators that are used to compute $x[t]$ from $x[t-1]$ can also provide the sensitivity of $x[t]$ to changes in $x[t-1]$. Based on this information one can decorate every computed trace $x[0], \ldots, x[k]$ by a sequence of "balls" $B[0], \ldots, B[k]$ such that each $B[t]$ is centered around $x[t]$ and $B[t]$ contains all the states that can be reached after t steps by traces starting from *any* point in $B[0]$.

The exploration of X_0 starts as follows. First we pick a point x in the center of X_0, let $B[0] = X_0$ and compute the trajectory and associated balls as shown in Figure 1. One

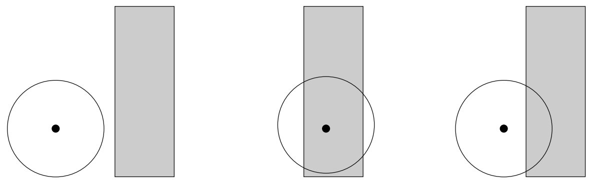

Figure 2: The three cases: safe, unsafe, unknown.

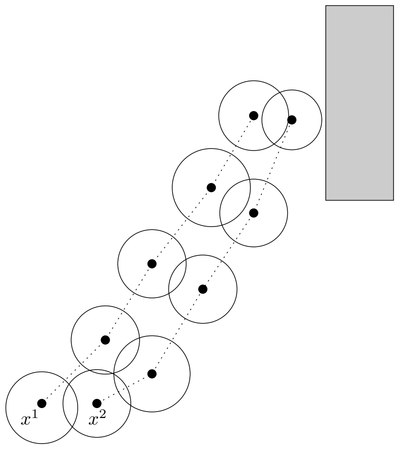

Figure 3: Refining the coverage.

of the following three cases, illustrated in Figure 2, may happen.

1. None of the balls intersects the set of bad states and in this case we are safe because these balls constitute an over-approximation of what may happen starting from any point in X_0;

2. Some $x[t]$ enters the bad region and a concrete counter-example to the safety of the system is found;

3. Some $B[t]$ intersects the bad set but no $x[t]$ does. Here we do not know if this represents a real counter-example or is just a side effect of the over-approximation.

In this last case we backtrack to $B[0]$ and refine our coverage, that is, replace x by two points x^1 and x^2 and two (smaller) balls around them that cover X_0. For each of those points we repeat the process as illustrated in Figure 3. Naturally, this procedure tends to refine the sampling in the suspicious parts of X_0. Except for the pathological case where traces touch the boundary of the bad region without crossing it (which can be resolved by adding some tolerance constant), this procedure can prove bounded-horizon safety for a system admitting a *non-countable* but static under-determination using a *finite* number of simulations. This technique has been implemented into the tool *Breach*[1] and has been applied to the process of parameter-space exploration in several domains such as embedded control systems, analog circuits and biochemical reactions.

[1]`http://www-verimag.imag.fr/~donze/breach_page.html`

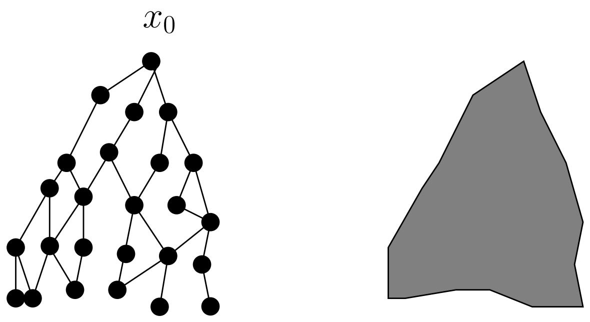

Figure 4: Trajectories induced by inputs from x_0 and the set of reachable states.

4. COMPUTING REACHABLE SETS

In this section I illustrate the principles of computing reachable sets for continuous (and hybrid) systems, which is essentially a set-based extension of numerical integration. We treat here systems with initial-state under-determination (which can be easily extended to parameters) and dynamic input under-determination. Let $\overline{x} \in X^*$ and $\overline{v} \in V^*$ denote sequences of states and inputs, respectively. An initial state x and an input sequence $\overline{v} = \overline{v}[0], \ldots, \overline{v}[t-1]$ yield a trace (trajectory) $\overline{x} = \overline{x}[0], \ldots, \overline{x}[t]$ such that $\overline{x}[0] = x$ and for every i, $\overline{x}[i] = f(\overline{x}[i-1], \overline{v}[i-1])$. Letting $\overline{x}[t] = x'$ we say then that input $\overline{v}$ takes the system from x to x', denoted as $x \xrightarrow{\overline{v}} x'$, or that x' is reachable (via $\overline{v}$) from x within t time:

$$R_t(x, \overline{v}) = \{x'\}.$$

This notion speaks of *one* initial state, *one* input sequence and *one* time instant. Let us generalize it for a *set* of initial states X_0 , for *all* time instants in an interval $I = [0, r]$ and for *all* (prefixes of) sequences in V^*. This yields the definition of the reachable set:

$$R_I(X_0) = \bigcup_{x \in X_0} \bigcup_{t \in I} \bigcup_{\overline{v} \in V^*} R_t(x, \overline{v}).$$

Figure 4 illustrates the induced trajectories and the reachable states for the case where $X_0 = \{x_0\}$.

The previous remark equating the relation between simulation and reachability computation to the relation between depth-first and breadth-first exploration of the space of trajectories corresponds to the commutativity of union:

$$\bigcup_{t \in I} \bigcup_{\overline{v} \in V^*} R_t(x, \overline{v}) = \bigcup_{\overline{v} \in V^*} \bigcup_{t \in I} R_t(x, \overline{v}).$$

Simulation corresponds to the expression on the right-hand side: each time we pick one $\overline{v} \in V^*$ and simulate the system until time t. Reachability computation correspond to the expression on the left: each time we increment time and compute states reachable via all inputs $\overline{v} \in V^t$ from those reachable via inputs in V^{t-1}. This is possible because the reachability operator admits the *semigroup property* with respect to time, namely:

$$R_{[0,t_1+t_2]}(X) = R_{[0,t_2]}(R_{[0,t_1]}(X)).$$

Hence, the computation of $R_I(X)$ for an interval $I = [0, r]$ can be achieved by the following incremental algorithm:[2]

```
ALGORITHM 1 (ABSTRACT REACHABILITY).
Input: A set $X_0 \subset X$
Output: $Q = R_{[0,r]}(X_0)$

$P := Q := X_0$
repeat $i = 1, 2 \dots$
  $P := R_1(P)$
  $Q := Q \cup P$
until $i = r$
```

If we are interested in reachability for *unbounded* time horizon, the termination condition $i = r$ should be replaced by $P \subseteq Q$, that is, the newly-computed reachable states are included in the set of states already computed.

The concrete realization of Algorithm 1 for various types of dynamics, using various classes of geometrical objects to store (approximations of) the sets encountered during the computation, is an adaptation of the basic idea of algorithmic verification (model checking) to the continuous and hybrid domains. This is a challenging problem involving discrete algorithmics, numerical analysis and computational geometry of very high dimensionality, much beyond what one typically encounters in graphics or robotics. I will give a flavor of this domain using systems with linear dynamics.[3]

An open dynamical system (X, V, f) is linear when

$$f(x, v) = Ax + Bv$$

with A and B being matrices of the appropriate dimensions. To simplify notation let B be the unit matrix hence $f(x, v) = Ax + v$. The basic step in the algorithm is the computation of R_1, the f-image of a set P:

$$P' = \{Ax + v : x \in P, v \in V\} = AP \oplus V$$

where $\oplus$ is the Minkowski sum of two sets defined as

$$A \oplus B = \{a + b : a \in A \wedge b \in b\}.$$

Assume P is a convex polytope (a bounded convex polyhedron) represented by its set of *vertices* $\tilde{P}$, which is the minimal set of points for which P is the convex hull, $P = conv(\tilde{P})$. The convex hull of a set $\{x_1, \dots, x_l\}$ is the set of all convex combinations of its elements, that is, points of the form $x = \lambda_1 x_1 + \cdots + \lambda_l x_l$ such that $\sum_{i=1}^{l} \lambda_i = 1$ and $\lambda_i \geq 0$ for every i. To compute $AP = \{Ax : x \in P\}$ we take advantage of the fact that $P = conv(\tilde{P})$ implies $AP = conv(A\tilde{P})$. In other words, applying A to the vertices of P we obtain the vertices of P' as illustrated in Figure 5. This simple idea already solves the problem of exhaustive exploration of a static under-determination space associated with initial states. For dynamic disturbances, the problem is more difficult and we illustrate several approaches for handling it.

One approach is to take the expression $AP \oplus V$ literally and compute the Minkowski sum as illustrated in Figure 6.

[2]We use R_1 as a shorthand for $R_{[1,1]}$. In continuous time P is initially set to $R_{[0,1]}(X_0)$.
[3]To avoid a common confusion, let me stress that I refer to systems which in continuous time are defined by linear differential equations of the form $\dot{x} = Ax$ and in discrete time by recurrence equations of the form $x_i = A' x_{i+1}$ where A' is obtained from A via the matrix exponential associated with the time discretization constant.

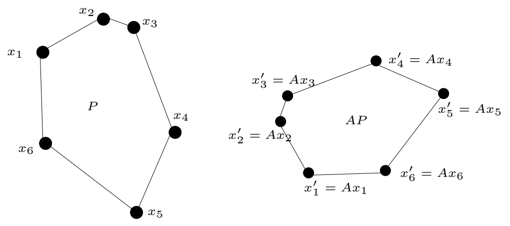

Figure 5: Computing AP from P by applying A to the vertices.

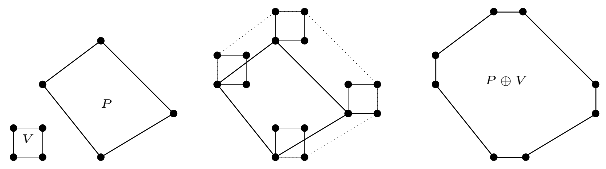

Figure 6: Adding an input polytope V to a polytope P leads to a polytope $P \oplus V$ with more vertices. The phenomenon is more severe in higher dimensions.

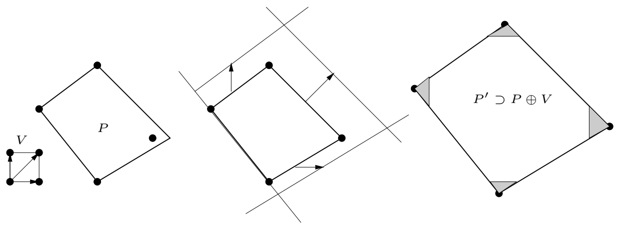

Figure 7: Applying to each face of P the element of V which pushes it outwards to the maximum. The result will typically not have more facets or vertices but it is a superset of $P \oplus V$ (shaded triangles represent the over-approximation error).

The problem is that unlike linear transformations, the sum operation increases the number of vertices. Its successive application will prohibitively increase the number of points to which A is applied. Consequently, methods for reachability under inputs need some compromise between exact computation that leads to explosion and approximations that keep the representation size small but may accumulate errors to the point of becoming useless, a phenomenon known in numerical analysis as the "wrapping effect".

To over-approximate the reachable set while keeping its complexity more or less fixed, assume P to be represented in (or converted into) inequality representation (intersection of halfspaces). For each supporting halfspace H^j defined by $a^j x \leq b^j$, let $v^j \in V$ be the input vector which pushes H^j in the "outermost" way, that is, the one which maximizes the product $v \cdot a^j$ with the normal to H^j. In the discrete time setting described here, v^j is some vertex of V for every j. We then apply to each H^j the transformation $Ax + Bv^j$ and the intersection of the halfspaces thus obtained is an over-approximation of $AP \oplus V$ (see Figure 7).

It turned out to be possible to have *both* accuracy and efficiency due to the following observation. If we look at two consecutive sets P_i and P_{i+1} in the computation, they have the form

$$P_i = A^i P_0 \oplus A^{i-1} V \oplus A^{i-2} V \oplus \ldots \oplus V$$

and

$$P_{i+1} = A^{i+1} P_0 \oplus A^i V \oplus A^{i-1} V \oplus \ldots \oplus V.$$

As one can see, these two sets "share" a lot of common terms that need not be recomputed. Algorithms based on this fact keep a symbolic "lazy" representation of the reachable set on which the transition from i to $i+1$ always involves the same number of linear transformations. While this representation is not very useful for Boolean operations or for visualization, each P_i can be efficiently and tightly over-approximated by a simpler type of set, but this object is *not* used to compute P_{i+1} and hence the wrapping effect is avoided.

This concludes our short excursion into the exhaustive exploration of dynamic under-determination in the context of continuous systems, which can be extended to (and in fact has been motivated by) hybrid systems by intersecting reachable sets with the switching surfaces (transition guards). The idea of applying set-based computation to hybrid systems was among the first contributions of the verification community to hybrid systems research [1] and was initially applied to hybrid automata with very simple dynamics in each state, namely constant derivative which in discrete time can be written as $f(x, v) = x + c + v$ for a constant c. The pioneering tool for computing reachable states for such systems was HyTech [6] which has been superseded after a decade by the more modern Phaver [5]. The approach for linear systems based on polytopes has been developed independently in [2] and [3] leading to the tools *CheckMate* and **d/dt**, respectively. Other approaches for computing reachable sets represent such sets using ellipsoids [7], are inspired by the numerical solution of partial-differential equations [11] or combine reachability with optimization [12]. The efficient yet precise algorithmic scheme for linear reachability is described in detail in [8] where *support functions* [9] are used as a general symbolic representation of convex sets. The tool *SpaceEx: the State-Space Explorer*[4], developed at VERIMAG under the direction of G. Frehse, robustly integrates and improves many of these ideas.

[4]`http://spaceex.imag.fr`

5. TIMED SYSTEMS

In this section I briefly illustrate how these concepts are instantiated in timed systems. Imagine a stream of computational jobs arriving for execution on a platform consisting of several machines. Each job is modeled as a task graph with tasks related by precedence constraints. These tasks are scheduled to execute on the machines that they occupy for their respective durations. For a deterministic setting, assume all jobs are identical, arrive periodically with a fixed period, and are scheduled to execute by a deterministic scheduler. In this case the system can be simulated by a timed discrete-event simulator which maintains the state of the system as a queue of pending events and advances time progressively to the next events (task termination or job arrival). The formulation of such a system as a dynamical system can be done using *timed automata* that combine discrete states and clock variables that measure the time elapsed since the initiation of tasks which are still active in a given state. Practitioners who are content with the simulator "semantics" will not bother to make the dynamical system formulation as they do not see the added value of the formal investment.

Under-determination can enter the picture in many forms. Identity of jobs may differ between instances, arrival rate may have jitter or become completely sporadic, external workload from other unmodeled parts of the system or machine faults may influence machine availability and so on and so forth. Let us focus on one type of under-determination, the duration of tasks, which for each task T_i we consider to be in an interval $[a_i, b_i]$. Static under-determination would mean that once a duration $d_i \in [a_i, b_i]$ has been chosen, all instances of task T_i will have that duration. This is a finite-dimensional parameter space which is a hyper-rectangle corresponding to the product of all duration intervals of the task types. Dynamic under-determination obtains when each instance of a task may choose a different value inside its duration interval, which means that each trace of the system corresponds to a point in an infinite-dimensional rectangle.

Exhaustive coverage can be achieved by zone-based reachability computation for timed automata. This procedure can detect combinations of task durations that lead to bad states such as deadline violation. This approach suffers from two drawbacks: first, it is computationally expensive, having to manipulate high-dimensional polytopes in the clock space (they are simpler than arbitrary polytopes, though) as well as discrete states whose number may grow exponentially with the number of tasks. Secondly, it has a worst-case flavor not compatible with performance evaluation of soft real-time systems: it will always include in its output pessimistic scenarios where each task instance takes the upper bound of its duration interval. To alleviate these problems one may assume a probability distribution over the duration space, conduct random simulations and collect statistics. The second problem can be treated alternatively by generalized (stochastic) reachability where not only reachable states are computed but also their distributions. This, however, does not make the computation easier.

6. DISCUSSION

Although large parts of what is presented here is known, at least implicitly, to those working in model-based design and analysis of systems, it is hoped that the presentation helps to bridge the gap between the practice of simulation and the more advanced but expensive approaches based on formal verification. In particular, it gives some idea about what verification tools for continuous, timed and hybrid systems can do.

Acknowledgement: E. Asarin, D. Nickovic and T. Dang gave some useful comments and advice.

7. REFERENCES

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[2] A. Chutinan. *Hybrid System Verification using Discrete Model Approximations*. PhD thesis, Carnegie Mellon University, 1999.

[3] T. Dang. *Verification and Synthesis of Hybrid Systems*. PhD thesis, Institut National Polytecnique de Grenoble, 2000.

[4] A. Donze. *Trajectory-based Verification and Controller Synthesis for Continuous and Hybrid Systems*. PhD thesis, Université Grenoble 1 – Joseph Fourier, 2007.

[5] G. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. PhD thesis, Radboud Universiteit Nijmegen, 2005.

[6] P.-H. Ho. *Automatic Analysis of Hybrid Systems*. PhD thesis, Cornell University, 1995.

[7] A. A. Kurzhanskyi. *Modeling and Software Tools for Freeway Operational Planning*. PhD thesis, UC Berkeley, 2007.

[8] C. Le Guernic. *Reachability Analysis of Hybrid Systems with Linear Continuous Dynamics*. PhD thesis, Université Grenoble 1 – Joseph Fourier, 2009.

[9] C. Le Guernic and A. Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250 – 262, 2010.

[10] O. Maler. Control from computer science. *Annual Reviews in Control*, 26(2):175–187, 2002.

[11] I. Mitchell. *Application of Level Set Methods to Control and Reachability Problems in Continuous and Hybrid Systems*. PhD thesis, Stanford University, 2002.

[12] F. D. Torrisi. *Modeling and Reach-set Computation for Analysis and Optimal Control of Discrete Hybrid Automata*. PhD thesis, ETH Zurich, 2003.