2011 32nd IEEE Real-Time Systems Symposium

On the Feasibility of Linear Discrete-Time Systems of the Green Scheduling Problem

Zheng Li, Pei-Chi Huang, Aloysius K. Mok
Department of Computer Science
The University of Texas at Austin
{zhengli, peggy, mok}@cs.utexas.edu

Truong Nghiem, Madhur Behl, George Pappas, Rahul Mangharam
Department of Electrical and Systems Engineering
University of Pennsylvania
{nghiem, mbehl, pappasg, rahulm}@seas.upenn.edu

***Abstract*—Peak power consumption of buildings in large facilities like hospitals and universities becomes a big issue because peak prices are much higher than normal rates. During a power demand surge an automated power controller of a building may need to schedule ON and OFF different environment actuators such as heaters and air quality control while maintaining the state variables such as temperature or air quality of any room within comfortable ranges. The green scheduling problem asks whether a scheduling policy is possible for a system and what is the necessary and sufficient condition for systems to be feasible. In this paper we study the feasibility of the green scheduling problem for HVAC(Heating, Ventilating, and Air Conditioning) systems which are approximated by a discrete-time model with constant increasing and decreasing rates of the state variables. We first investigate the systems consisting of two tasks and find the analytical form of the necessary and sufficient conditions for such systems to be feasible under certain assumptions. Then we present our algorithmic solution for general systems of more than 2 tasks. Given the increasing and decreasing rates of the tasks, our algorithm returns a subset of the state space such that the system is feasible *if and only if* the initial state is in this subset. With the knowledge of that subset, a scheduling policy can be computed on the fly as the system runs, with the flexibility to add power-saving, priority-based or fair sub-policies.**

***Keywords*-green scheduling; feasibility;**

I. Introduction

Peak power consumption of buildings in large facilities like hospitals and universities becomes a big issue because peak prices are much higher than normal rates. Thus peak power consumption is directly related to the energy bill. [1]'s study on the power market data in the Pennsylvania-New Jersey-Maryland territory in 2006 suggests that if the peak load is reduced by 4.8% on average then the total expense would be cut by 3.5% or $\$1.2$ billion. During a power demand surge an automated power controller of a building may need to schedule ON and OFF different environment actuators such as heaters and air quality control while maintaining the state variables such as temperature or air quality of any room within comfortable ranges.

However, some systems may not be feasible by any scheduling policy. For instance, consider the scenario where a controller in a building can power ON at most one heater any time and there are two rooms (with two separate heaters) whose temperatures are so low that both will drop below their lower thresholds in one time unit if the rooms are not heated. In this scenario the system is deemed to fail after one time unit. [2] formularizes this kind of scheduling problems, referred to *green* scheduling, and it presents a necessary and sufficient condition in continuous time domain for any system to be feasible. However, when time is discretized additional conditions are required.

In this paper, we present our research on linear discrete-time green scheduling systems. We built our work on the previous problem formulation by [2]–[4] and the geometric interpretation of the problem by [2], [3]. We derive an analytical form of the necessary and sufficient conditions for 2D systems to be feasible under certain assumptions. We also designed an algorithmic solution for general systems of n tasks ($n \geq 2$). Given the increasing and decreasing rates of the tasks, our algorithm returns a subset of state space such that the system is feasible if and only if its initial state is in this subset. With the knowledge of that subset, a scheduling policy can be computed on the fly as the system runs, with the flexibility to add power-saving, priority-based or fair sub-policies.

The paper is organized as follows. Section (II) gives the formulation of the green scheduling problem. In Section (III) we explain our work on systems of two tasks and introduce some key concepts and ideas for solving general systems of n tasks. In Section (IV) we present our algorithm for general systems of n tasks. Our simulation results are shown in Section (V). In Section (VI) we note the related work. In the last section, we conclude.

II. Task System

In this section we rephrase the formulation of the green scheduling problem by [2]–[4] and the state space (or geometric) interpretation of the problem given by [2], [3].

A. Task Model

Consider a control system that controls temperature of a room within a certain range $[l, h)$ with a heater. Temperature is called the *state variable* of the control system. The control system could switch the heater ON or OFF. When the heater is ON, the temperature would change according to the dynamics of the system. For simplicity, here we assume the temperature increases linearly with rate a. Similarly, the temperature decreases linearly with rate b when the heater is OFF. In this paper a "linear" system means the state variable increases or decreases linearly (constant slope) with respect to time. It is different from a "linear" system in control theory, where a state

1052-8725/11 $26.00 © 2011 IEEE
DOI 10.1109/RTSS.2011.34

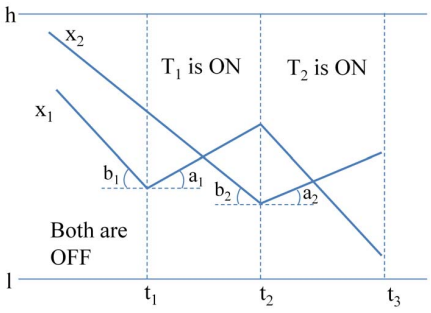


Figure 1. Example: scheduling of two tasks.

variable's dynamics is characterized by a linear differential equation.

We construct the task model for such a control system as follows. A *task* T consists of a tuple of attributes (x, l, h, $x(0)$, M, a, b).

- x: the state variable; $x \in \Re$.
- l, h: the lower and upper threshold of the state variable; $l, h \in \Re$ and $l < h$.
- $x(0)$: the initial state; $x(0) \in [l, h)$.
- M: the mode of the task; $M \in \{\text{ON}, \text{OFF}\}$.
- a, b: the increasing (decreasing) rate of the state variable when the mode is ON (OFF); $a, b \in (0, +\infty)$.

We assume that the attributes l, h, a and b of a task are fixed with respect to time. The mode of a task can be either ON or OFF at any time. The dynamic equation of the state variable is given by:

$$\dot{x}(t) = \begin{cases} a & \text{if } M(t) = \text{ON} \\ -b & \text{if } M(t) = \text{OFF} \end{cases}.$$

In [2] a more general dynamic equation is given, in which the rates are not necessarily constant. In this paper we only consider linear trajectories.

A task is *safe* if and only if $\forall t \geq 0, x(t) \in [l, h)$. A task *fails* if $\exists t \geq 0, x(t) \notin [l, h)$.

B. Scheduling Problem

A task system S is a set of n ($n \geq 1$) tasks $\{T_i\}$ ($i = 1, 2, \cdots, n$). A scheduling policy π on S assigns the mode of each task in S for all $t \geq 0$. S is *schedulable* by a policy π if and only if under π every task in S is safe and at most one task is ON at any time. S is *feasible* if and only if there exists at least one scheduling policy under which S is schedulable. S is *infeasible* if and only if there is no scheduling policy under which S is schedulable. S *fails* at time t if at least one task in S fails at time t.

Fig. 1 shows an example of scheduling a task system of two tasks. For simplicity, $l_1 = l_2 = l$ and $h_1 = h_2 = h$.

For a task system S, we want to find the necessary and sufficient condition on its feasibility.

C. System State Space

For a task T_i, define its *normalized* state variable $\hat{x}_i(t) = \frac{x_i(t) - l_i}{h_i - l_i}$. It follows that T_i is safe if and only if $\hat{x}_i(t) \in [0, 1)$ for all $t \geq 0$. Similarly, define normalized increasing and decreasing rates as $\hat{a}_i = \frac{a_i}{h_i - l_i}$ and $\hat{b}_i = \frac{b_i}{h_i - l_i}$, respectively. The dynamics of T_i becomes

$$\dot{\hat{x}}_i(t) = \begin{cases} \hat{a}_i & \text{if } M_i(t) = \text{ON} \\ -\hat{b}_i & \text{if } M_i(t) = \text{OFF} \end{cases}.$$

Consider a task system S of size n ($n > 1$). Define *system state* $X = \begin{bmatrix} \hat{x}_1 & \hat{x}_2 & \cdots & \hat{x}_n \end{bmatrix}^T$, where T denotes transpose of matrix. Let X_i denote the i-th component of X, i.e., $X_i = \hat{x}_i$. The dynamics of all task states can be reduced to the dynamics of the system state in a n-dimensional state space. For example, suppose during the time interval $[t, t + \Delta t]$, T_1 is ON while other tasks are OFF. The dynamics during the interval is given by $\dot{\hat{x}}_1 = \hat{a}_1$ and $\dot{\hat{x}}_i = -\hat{b}_i$ ($2 \leq i \leq n$). It is equivalent to $\dot{X} = \begin{bmatrix} \hat{a}_1 & -\hat{b}_2 & \cdots - \hat{b}_n \end{bmatrix}^T$. In other words, the system state X moves in the direction $\begin{bmatrix} \hat{a}_1 & -\hat{b}_2 & \cdots - \hat{b}_n \end{bmatrix}^T$ in the state space during the interval. In the following context we also refer a system of n tasks as a *n-dimensional* system.

It follows that the system is safe if and only if the movement of the system state is restricted to a n-dimensional box in the state space for all time $t \geq 0$. We call this n-dimensional box *safety box*, denoted as $SafetyBox$. We use the notation $Interval_1^1 \times Interval_2^2 \times \cdots \times Interval_n^n$ to denote the set of states $\{X | \forall i (1 \leq i \leq n), X_i \in Interval^i\}$. Thus,

$$SafetyBox = [0, 1)_1 \times [0, 1)_2 \times \cdots \times [0, 1)_n.$$

D. Discrete Time Systems

Since practical systems are mostly scheduled and controlled by digital computers, we focus on discrete-time systems from now on. Let the unit time interval be δt, and let $\tilde{a}_i = \hat{a}_i * \delta t$ and $\tilde{b}_i = \hat{b}_i * \delta t$. Then the normalized task state $\hat{x}_i$ either increases by $\tilde{a_i}$ (ON) or decreases by $\tilde{b_i}$ (OFF) during a unit time interval. In other words,

$$\hat{x}_i(k+1) = \begin{cases} \hat{x}_i(k) + \tilde{a}_i & \text{if } M_i(k) = \text{ON} \\ \hat{x}_i(k) - \tilde{b}_i & \text{if } M_i(k) = \text{OFF} \end{cases}.$$

Let

$$g^{(0)} = \begin{bmatrix} -\tilde{b}_1 & -\tilde{b}_2 & \cdots & -\tilde{b}_n \end{bmatrix}^T \quad (1)$$

$$g^{(i)} = \begin{bmatrix} -\tilde{b}_1 & \cdots & \tilde{a}_i & \cdots & -\tilde{b}_n \end{bmatrix}^T \quad (1 \leq i \leq n) \quad (2)$$

The dynamic equation of a discrete time task system is given by

$$X(k+1) = X(k) + g(k), \quad (3)$$

where $g(k) \in \{g^{(0)}, g^{(1)}, \cdots, g^{(n)}\}$ and

$$g(k) = \begin{cases} g^{(0)} & \text{iff all tasks are OFF} \\ g^{(i)} & \text{iff } T_i (1 \leq i \leq n) \text{ is ON while others are OFF} \end{cases}. \quad (4)$$

Fig. 2 illustrates the dynamics of a system state in the 2D safety box. We say that the system takes *movement* $g^{(i)}$ at time k if $g(k) = g^{(i)}$ ($0 \leq i \leq n$).

By (4), a scheduling policy on a task system thus corresponds to a unique infinite sequence of movements the system takes at each time step: $g(0)g(1)\cdots$. The feasibility of the

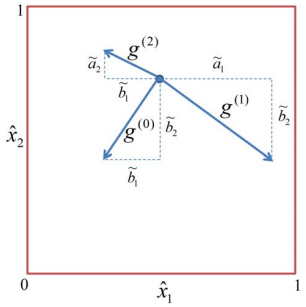
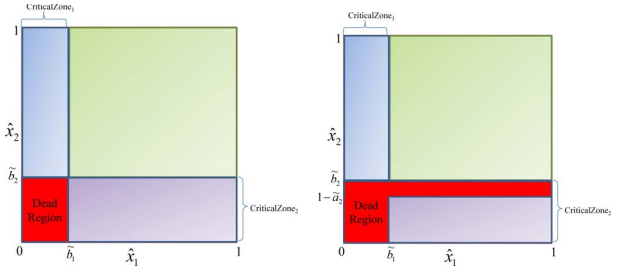


Figure 2. Safety box of a discrete time system consisting of two tasks. $g^{(0)}$, $g^{(1)}$ and $g^{(2)}$ are the possible movements.



(a) $\tilde{a}_1 + \tilde{b}_1 \leq 1$ and $\tilde{a}_2 + \tilde{b}_2 \leq 1$ (b) $\tilde{a}_1 + \tilde{b}_1 \leq 1$ and $\tilde{a}_2 + \tilde{b}_2 > 1$

Figure 3. Dead region and critical zones of 2D systems.

scheduling problem thus reduces to the restricted movement problem of the system state within the n-dimensional safety box.

Theorem 1. *A n-dimensional discrete-time task system S is feasible if and only if given its initial system state $X(0) \in SafetyBox$ and the set of possible movements $G = \{g^{(0)}, g^{(1)}, \cdots, g^{(n)}\}$, there is an infinite sequence of movements $g(0)g(1)\cdots$ such that $\forall k \geq 0, X(k) \in SafetyBox$, provided that the dynamics of X is given by (3).*

A benefit of this state space framework is that it frees us from considering the size of $h_i - l_i$ for each task when addressing the feasibility of a system. The only things mattering are the initial system state and the possible movements. The reason is that from the scheduling point of view a task of larger or smaller $h - l$ is always equivalent to a task of the original $h - l$ but with a scaled initial state and scaled increasing and decreasing rates.

E. Necessary and Sufficient Condition Problem

Because a particular task system with a given scheduling policy and given $X(0)$ and G can be modeled as a timed automaton, timed automata tools such as UPPAAL [5] might be used to verify its safety property. However, our goal in this paper is to find the necessary and sufficient conditions on $X(0)$ for a task system to be feasible, and to synthesize a scheduling policy for it. We first investigate 2-dimensional task systems and introduce some key concepts and ideas that can be applicable to more general n-dimensional task systems. For 2-dimensional task systems we obtain necessary and sufficient conditions on both $X(0)$ and G under certain assumptions. For n-dimensional task systems we design and implement an algorithm that given G computes a set of system states such that the system is feasible if and only if its initial state is in that set.

III. 2D SYSTEMS

Consider a 2-dimensional task system. The safety box is shown in Fig. 3. Define the *critical zone* of T_i as $[0, \tilde{b}_i)_i \times [0, 1)_j$ $(j \neq i)$, denoted as $CriticalZone_i$. Thus, $CriticalZone_1 = [0, \tilde{b}_1)_1 \times [0, 1)_2$ and $CriticalZone_2 = [0, 1)_1 \times [0, \tilde{b}_2)_2$. Once $X(k) \in CriticalZone_i$, for $X(k+1) \in SafetyBox$ we must have $g(k) = g^{(i)}$, otherwise $X_i(k+1) = X_i(k) - \tilde{b}_i < 0$. In other words, T_i must be turned ON in its critical zone, otherwise it will fail at next time step. However, there are some region in the safety box that once the system state is inside the region at time k the system must fail at time $k+1$ no matter what $g(k)$ is. We call such region *dead region*, denoted as $DeadRegion$. Note that we use the word *region* to denote a subset of the safety box.

One obvious dead region for any 2D system is the intersection of the two critical zones, shown in Fig. 3a. We call these dead regions *Type 1* dead regions. That is,

$$DeadRegion(\text{Type 1}) = [0, \tilde{b}_1)_1 \times [0, \tilde{b}_2)_2. \tag{5}$$

However, if $\tilde{a}_i + \tilde{b}_i > 1$, other parts of $CriticalZone_i$ would also form a dead region, namely, $[1 - \tilde{a}_i, \tilde{b}_i)_i \times [0, 1)_j$, because task T_i cannot be turned ON in that region due to its upper threshold limitation (Fig. 3b). We group these dead regions into *Type 2* dead regions. That is,

$$DeadRegion(\text{Type 2}) = [1-\tilde{a}_i, \tilde{b}_i)_i \times [0, 1)_j \text{ for } i, \tilde{a}_i + \tilde{b}_i > 1.$$

In the following discussion on 2D systems, we make the following assumption:

Assumption 1. $\forall i (1 \leq i \leq 2), \tilde{a}_i + \tilde{b}_i \leq 1$.

This assumption guarantees that if T_i is turned ON anywhere in its critical zone at time k, the task itself will not fail at time $k+1$, although it does not guarantee that the whole system will not fail at time $k+1$. The assumption also implies that we would only consider Type 1 dead region in our 2D systems discussion.

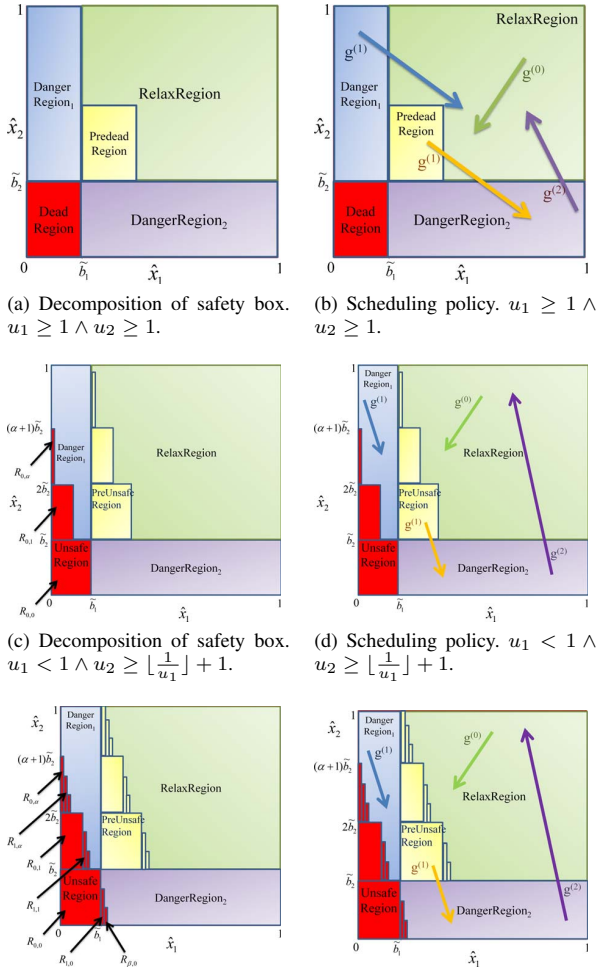
[2] has shown that a necessary condition for a n-dimensional system to be feasible is

$$\sum_{i=1}^{n} \frac{\tilde{b}_i}{\tilde{a}_i + \tilde{b}_i} \leq 1. \tag{6}$$

In 2D system, this is equivalent to

$$\frac{\tilde{a}_1}{\tilde{b}_1} \frac{\tilde{a}_2}{\tilde{b}_2} \geq 1. \tag{7}$$

We will show that this condition plus the condition that the initial state is in certain region would form the necessary and sufficient condition for a 2D system to be feasible.



(a) Decomposition of safety box. $u_1 \geq 1 \wedge u_2 \geq 1$.

(b) Scheduling policy. $u_1 \geq 1 \wedge u_2 \geq 1$.

(c) Decomposition of safety box. $u_1 < 1 \wedge u_2 \geq \lfloor \frac{1}{u_1} \rfloor + 1$.

(d) Scheduling policy. $u_1 < 1 \wedge u_2 \geq \lfloor \frac{1}{u_1} \rfloor + 1$.

(e) Decomposition of safety box. $u_1 < 1 \wedge \frac{1}{u_1} \leq u_2 < \lfloor \frac{1}{u_1} \rfloor + 1$.

(f) Scheduling policy. $u_1 < 1 \wedge \frac{1}{u_1} \leq u_2 < \lfloor \frac{1}{u_1} \rfloor + 1$.

Figure 4. Decomposition of safety box and scheduling policy of 2D systems.

Let $u_1 = \frac{\tilde{a}_1}{\tilde{b}_1}$ and $u_2 = \frac{\tilde{a}_2}{\tilde{b}_2}$. (7) translates to three possible cases: (a) $u_1 \geq 1 \wedge u_2 \geq 1$ (b) $u_1 < 1 \wedge u_2 \geq \frac{1}{u_1}$ (c) $u_1 \geq \frac{1}{u_2} \wedge u_2 < 1$. Without loss of generality, we discuss the first two cases only. We also split the second case into two subcases: $u_2 \geq \lfloor \frac{1}{u_1} \rfloor + 1$ and $\frac{1}{u_1} \leq u_2 < \lfloor \frac{1}{u_1} \rfloor + 1$.

A. Case $u_1 \geq 1 \wedge u_2 \geq 1$

Define $PredeadRegion$ as

$$PredeadRegion = \{X | X \in SafetyBox \text{ and } X + g^{(0)} \in DeadRegion\} \quad (8)$$

Thus, if $X(k) \in PredeadRegion$ and all tasks are OFF at time k, then $X(k+1) \in DeadRegion$. See Fig. 4a for illustration.

Let $ub(R, i)$ denote the tight upper bound of a region in direction $\hat{x}_i$. For example, $ub(DeadRegion, 1) = \tilde{b}_1$ and $ub(PredeadRegion, 1) = 2\tilde{b}_1$.

Assumption 2. $\exists \sigma (1 \leq \sigma \leq 2), ub(PredeadRegion, \sigma) + \tilde{a}_\sigma < 1$.

This assumption guarantees that if $X(k) \in PredeadRegion$ we can always turn ON T_σ and the system will still stay in the $SafetyBox$ at time $k+1$ without surpassing the upper threshold of T_σ.

Define the following regions:

$$\begin{aligned} DangerRegion_i &= CriticalZone_i \setminus DeadRegion \; (i = 1, 2) \\ RelaxRegion &= SafetyBox \setminus \\ &\quad (CriticalZone_1 \cup CriticalZone_2 \\ &\quad \cup PredeadRegion) \end{aligned} \quad (9)$$

Fig. 4a illustrates these regions. It can be shown that the safety box is the union of the following 5 *disjoint* regions:

$$\begin{aligned} SafetyBox = &\; DeadRegion \cup PredeadRegion \\ &\cup DangerRegion_1 \cup DangerRegion_2 \\ &\cup RelaxRegion. \end{aligned} \quad (10)$$

We refer to this fact as *decomposition of the safety box*.

We now present our *premier 2D scheduling policy*.

Premier 2D Scheduling Policy: for all $k \geq 0$

$$g(k) = \begin{cases} g^{(i)} & \text{if } X(k) \in DangerRegion_i \\ g^{(\sigma)} & \text{if } X(k) \in PredeadRegion \\ g^{(0)} & \text{if } X(k) \in RelaxRegion \end{cases}$$

Fig. 4b illustrates the premier 2D scheduling policy by assuming $\sigma = 1$. Scheduling on $DeadRegion$ is omitted because it fails at time $k+1$ anyway.

Proposition 1. *Suppose Assumption 1 and Assumption 2 hold. If $X(k) \in SafetyBox \setminus DeadRegion$ ($k \geq 0$), then under the premier 2D scheduling policy, $X(k+1) \in SafetyBox \setminus DeadRegion$.*

Proof: $X(k)$ must be in one of the other 4 disjoint regions in $SafetyBox$ by the decomposition of the safety box. We now prove the proposition holds for $X(k) \in DangerRegion_1$. Proofs for other regions are similar. By definition $DangerRegion_1 = [0, \tilde{b}_1)_1 \times [\tilde{b}_2, 1)_2$. Under the premier 2D scheduling policy, $g(k) = g^{(1)} = \begin{bmatrix} \tilde{a}_1 & -\tilde{b}_2 \end{bmatrix}^T$. $X(k+1) = X(k) + g^{(1)} \in [\tilde{a}_1, \tilde{b}_1 + \tilde{a}_1)_1 \times [0, 1 - \tilde{b}_2)_2 \equiv D$. Obviously $D \subset SafetyBox$, and $D \cap DeadRegion = \emptyset$ since $\tilde{a}_1 \geq \tilde{b}_1$. Thus, $X(k+1) \in SafetyBox \setminus DeadRegion$. ■

Lemma 1. *Suppose Assumption 1 and Assumption 2 hold. A system satisfying ($u_1 \geq 1 \wedge u_2 \geq 1$) is feasible if and only if $X(0) \in SafetyBox \setminus DeadRegion$.*

Proof: The "only if" part is trivial. For the "if" part, we can apply Proposition 1 to prove the system is schedulable under the premier 2D scheduling policy. ■

B. Case $u_1 < 1 \wedge u_2 \geq \lfloor \frac{1}{u_1} \rfloor + 1$

Since $\tilde{a}_1 < \tilde{b}_1$, some states in $DangerRegion_1$ can reach $DeadRegion$ in one time step by taking $g^{(1)}$, namely the states in $[0, \tilde{b}_1 - \tilde{a}_1)_1 \times [\tilde{b}_2, 2\tilde{b}_2)_2 \equiv R_{0,1} \equiv \{X | X \in$

$SafetyBox$ and $X+g^{(1)} \in DeadRegion\}$. Suppose $X(k) \in R_{0,1}$, then the system must fail by time $k+2$: it either fails at time $k+1$ by taking $g(k) \neq g^{(1)}$ or fails at time $k+2$ by taking $g(k) = g^{(1)}$ and an arbitrary $g(k+1)$.

Let $\alpha = \lfloor \frac{1}{u_1} \rfloor$. Define

$$R_{0,j} = [0, \tilde{b}_1 - j\tilde{a}_1)_1 \times [j\tilde{b}_2, (j+1)\tilde{b}_2)_2 \quad (0 \leq j \leq \alpha), \quad (11)$$

shown in Fig. 4c. Note that $R_{0,0} \equiv DeadRegion$ and $\forall j (1 \leq j \leq \alpha), R_{0,j} \equiv \{X | X \in SafetyBox$ and $X + g^{(1)} \in R_{0,j-1}\}$. Note also that $R_{0,\alpha}$ does not exist if $\lfloor \frac{1}{u_1} \rfloor = \frac{1}{u_1}$. This special case does not affect our discussion below, where we consider the more general case where $\lfloor \frac{1}{u_1} \rfloor \neq \frac{1}{u_1}$.

By induction, we can show that if $X(k) \in R_{0,j}$, then the system must fail by time $k+j+1$. Now we introduce the concept of *unsafe region*. Unsafe region is defined as a region that once the system state is in the region at time k ($k \geq 0$) there exists a finite time step $k' > k$ such that the system must fail by time k' no matter what the sequence of movements $g(k)g(k+1)\cdots g(k'-1)$ is. We denote the unsafe region as $UnsafeRegion$. By definition, $DeadRegion \subseteq UnsafeRegion$ and $k' = k+1$ for the dead region. In the previous case, $UnsafeRegion = DeadRegion$. In this case,

$$UnsafeRegion = \bigcup_{0 \leq j \leq \alpha} R_{0,j}. \quad (12)$$

Accordingly, define *pre-unsafe region* as

$$PreUnsafeRegion = \{X | X \in SafetyBox \text{ and } X + g^{(0)} \in UnsafeRegion\}. \quad (13)$$

We slightly modify Assumption 2 to get

Assumption 3. $\exists \sigma (1 \leq \sigma \leq 2), ub(PreUnsafeRegion, \sigma) + \tilde{a}_\sigma < 1$.

Redefine the following regions:

$$DangerRegion_i = CriticalZone_i \setminus UnsafeRegion \; (i = 1, 2)$$
$$RelaxRegion = SafetyBox \setminus (CriticalZone_1 \cup CriticalZone_2 \cup PreUnsafeRegion) \quad (14)$$

Again the safety box can be decomposed into the following 5 *disjoint* regions (Fig. 4c):

$$SafetyBox = UnsafeRegion \cup PreUnsafeRegion \cup DangerRegion_1 \cup DangerRegion_2 \cup RelaxRegion. \quad (15)$$

We get a more general 2D scheduling policy (Fig. 4d).
General 2D Scheduling Policy: for all $k \geq 0$

$$g(k) = \begin{cases} g^{(i)} & \text{if } X(k) \in DangerRegion_i \\ g^{(\sigma)} & \text{if } X(k) \in PreUnsafeRegion \\ g^{(0)} & \text{if } X(k) \in RelaxRegion \end{cases}$$

Lemma 2. *Suppose Assumption 1 and Assumption 3 hold. A system satisfying ($u_1 < 1 \wedge u_2 \geq \lfloor \frac{1}{u_1} \rfloor + 1$) is feasible if and only if $X(0) \in SafetyBox \setminus UnsafeRegion$. $UnsafeRegion$ is given by (12).*

Proof: Similar to proof of Lemma 1, using the general 2D scheduling policy. The policy is shown in Fig. 4d. ■

C. Case $u_1 < 1$ and $\frac{1}{u_1} \leq u_2 < \lfloor \frac{1}{u_1} \rfloor + 1$

Given a rectangular region $D = [\theta_1, \theta_2)_1 \times [\tau_1, \tau_2)_2$, define its *width* as $width(D) = \theta_2 - \theta_1$ and its *height* as $height(D) = \tau_2 - \tau_1$.

This case is more complicated than the previous case, because unless $R_{0,\alpha}$ does not exist, some states in $DangerRegion_2$ can reach $R_{0,\alpha}$ in one time step by taking $g^{(2)}$, namely the states in $[\tilde{b}_1, \tilde{b}_1 + (\tilde{b}_1 - \alpha\tilde{a}_1))_1 \times [0, \tilde{b}_2 - (\tilde{a}_2 - \alpha\tilde{b}_2))_2 \equiv R_{1,0}$. Thus once $X(k) \in R_{1,0}$, the system must fail by time $k+\alpha+2$. This will in turn bring a series of regions $R_{1,j}$ ($1 \leq j \leq \alpha$) into the unsafe region like in the previous case. Another series of regions $R_{2,j}$ would in turn be inducted from $R_{1,\alpha}$. The process goes on. However, the process must terminate. To see this, let us define these regions first. Define

$$R_{i,j} = [\tilde{b}_1 + (i-1)(\tilde{b}_1 - \alpha\tilde{a}_1) - j\tilde{a}_1, \tilde{b}_1 + i(\tilde{b}_1 - \alpha\tilde{a}_1) - j\tilde{a}_1)_1 \times [j\tilde{b}_2, j\tilde{b}_2 + \tilde{b}_2 - i(\tilde{a}_2 - \alpha\tilde{b}_2))_2$$

for $(1 \leq i \leq \beta, 0 \leq j \leq \alpha)$, where $\beta = \lfloor \frac{\tilde{b}_2}{\tilde{a}_2 - \alpha\tilde{b}_2} \rfloor$. Note that the above definitions and the definitions of $R_{0,j}$ in (11) enforce that $\forall i (1 \leq i \leq \beta), R_{i,0} \equiv \{X | X \in SafetyBox$ and $X + g^{(2)} \in R_{i-1,\alpha}\}$, and $\forall j (1 \leq j \leq \alpha), R_{i,j} \equiv \{X | X \in SafetyBox$ and $X + g^{(1)} \in R_{i,j-1}\}$. It can be seen that $height(R_{i,j}) = \tilde{b}_2 - i(\tilde{a}_2 - \alpha\tilde{b}_2)$. Thus, $height(R_{i+1,j}) < height(R_{i,j})$. But since $height(R_{i,j})$ must be positive, the process must terminate at $i = \beta$. Note also that $\forall i, j (1 \leq i \leq \beta, 1 \leq j \leq \alpha), R_{i,j} \in CriticalZone_1$. To show that, we only need to show $ub(R_{\beta,1}, 1) \leq \tilde{b}_1$.

$$\begin{aligned} ub(R_{\beta,1}, 1) &= \tilde{b}_1 + \beta(\tilde{b}_1 - \alpha\tilde{a}_1) - \tilde{a}_1 \\ &= \tilde{b}_1 + \tilde{a}_1 \lfloor \frac{1}{u_2 - \alpha} \rfloor (\frac{1}{u_1} - \alpha) - \tilde{a}_1 \\ &\leq \tilde{b}_1. \end{aligned}$$

Together with the definition of $R_{0,j}$ by (11), we get the unsafe region in this case:

$$UnsafeRegion = \bigcup_{0 \leq i \leq \beta, 0 \leq j \leq \alpha} R_{i,j}. \quad (16)$$

Define other regions by (13) and (14) as before. We can still decompose the safety box as given by (15).

Lemma 3. *Suppose Assumption 1 and Assumption 3 hold. A system satisfying ($\frac{1}{u_1} \leq u_2 < \lfloor \frac{1}{u_1} \rfloor + 1$) is feasible if and only if $X(0) \in SafetyBox \setminus UnsafeRegion$. $UnsafeRegion$ is given by (16).*

Proof: Similar to proof of Lemma 1, using the general 2D scheduling policy. The policy is shown in Fig. 4f. ■

Combining the three lemmas, we obtain the following theorem.

Theorem 2. *For all 2D systems that satisfy the condition (7), if Assumption 1 holds, then the unsafe region can be analytically determined by (16) and the safety box can be decomposed into*

5 *disjoint regions by (15). If further Assumption 3 holds, then the system is feasible if and only if* $X(0) \in SafetyBox \setminus UnsafeRegion$*, and the system is schedulable by the general 2D scheduling policy.*

IV. Algorithmic Solution to general n-dimensional system

For general n-dimensional systems, analytical forms of necessary and sufficient conditions on $X(0)$ and G for systems to be feasible are difficult to get. However, based on the observation of 2D systems, we design and implement an algorithm that given any G (i.e., $\{\tilde{a}_i, \tilde{b}_i\}$) returns a region $UnsafeRegion$ such that the system is feasible if and only if $X(0) \in SafetyBox \backslash UnsafeRegion$. We will first show how to construct $UnsafeRegion$ algorithmically and then prove it indeed has the above property.

Let $OutSafetyBox$ denote the set of states out of the safety box. That is,

$$OutSafetyBox = \{X | X \notin SafetyBox\}.$$

A. Construct $UnsafeRegion$

For the simplicity of the following discussion, we use the word *rectangle* to refer to any set of the states of the form $[r_{10}, r_{11})_1 \times [r_{20}, r_{21})_2 \times \cdots \times [r_{n0}, r_{n1})_n$. The intersection between two rectangles $R_1 \cap R_2$ is the empty set ϕ or another rectangle, while the set difference $R_2 \setminus R_1$ returns the empty set (when $R_2 \subseteq R_1$) or a set of rectangles. Note that we enforce $R_2 \setminus R_1 = \{R_2\}$ if $R_1 \bigcap R_2 = \emptyset$. Define a function $shift(R, v)$, which returns a rectangle as shifting rectangle R by a vector v. That is,

$$\begin{aligned} shift(R, v) &\equiv \{X + v | X \in R\} \\ &= [r_{10} + v_1, r_{11} + v_1)_1 \times [r_{20} + v_2, r_{21} + v_2)_2 \\ &\times \cdots \times [r_{n0} + v_n, r_{n1} + v_n)_n. \end{aligned}$$

Based on the observation on 2D systems, define the dead region of a n-dimensional system as

Definition 1. $DeadRegion \equiv \{X | X \in SafetyBox$ *and* $\forall i (0 \le i \le n), X + g^{(i)} \in OutSafetyBox\}$.

Accordingly, define $DeadRect$ as a rectangle such that

$DeadRect \subset SafetyBox$ and
$\forall i (0 \le i \le n), shift(DeadRect, g^{(i)}) \subset OutSafetyBox$

There are two types of dead rectangles. Type 1 dead rectangle

$$\begin{aligned} DeadRect(\text{Type 1}) = [0, 1)_1 \times \cdots \times [0, \tilde{b}_i)_i \times \cdots \\ \times [0, \tilde{b}_j)_j \times \cdots \times [0, 1)_n \end{aligned}$$

for $(1 \le i \ne j \le n)$. Type 2 rectangle

$$\begin{aligned} DeadRect(\text{Type 2}) = [0, 1)_1 \times \cdots \times [1 - \tilde{a}_i, \tilde{b}_i)_i \\ \times \cdots \times [0, 1)_n \end{aligned}$$

for any i such that $\tilde{a}_i + \tilde{b}_i > 1$. Let $DeadRectSet$ denote the set of all dead rectangles of both Type 1 and Type 2 of the system. Then the dead region of the whole system is

$$DeadRegion = \bigcup_{R \in DeadRectSet} R. \tag{17}$$

From the discussion of 2D systems, we have learned that we can reduce the safety problem of individual system states to that of a rectangle: if all systems in a rectangle R are unsafe, then we say the rectangle is unsafe.

We denote the set of unsafe rectangles of a system as $UnsafeRectSet$. We initialize the $UnsafeRectSet$ to be $DeadRectSet$. If there exists a rectangle R such that

1) $R \in SafetyBox$
2) $\forall R_{unsafe} \in UnsafeRectSet, R \bigcap R_{unsafe} = \emptyset$
3) $\forall i (0 \le i \le n), shift(R, g^{(i)}) \subset OutSafetyBox \vee (\exists R_{unsafe} \in UnsafeRectSet, shift(R, g^{(i)}) \subseteq R_{unsafe})$, that is, once $X(k) \in R$, then $X(k+1)$ is either out of the safety box or inside of a known unsafe rectangle,

then by induction R is also a unsafe rectangle, and so we append it into the $UnsafeRectSet$. We call the above criteria *unsafety* criteria.

Beginning with $DeadRectSet$, we expand the $UnsafeRectSet$ by looking for R that satisfies the unsafety criteria. Let $R_{unsafe} \in UnsafeRectSet$. Consider the rectangle

$$R_{back} = shift(R_{unsafe}, -g^{(i)}) \bigcap SafetyBox$$

for some $(0 \le i \le n)$. It satisfies that $R_{back} \subset SafetyBox$ and $shift(R_{back}, g^{(i)}) \subseteq R_{unsafe}$. However, it may intersect with other known unsafe rectangles. Thus let

$$workingSet = \emptyset$$

and

$$\begin{aligned} &\forall R'_{unsafe} \in UnsafeRectSet, \\ &workingSet = workingSet \bigcup (R_{back} \setminus R'_{unsafe}). \end{aligned}$$

So $\forall R_{inwork} \in workingSet$, R_{inwork} satisfies the unsafety criteria (1) and (2). Then find any rectangle $R_{newUnsafe} \subseteq R_{inwork}$ that satisfies unsafety criteria (3).

By that end, let

$$\begin{aligned} &OutSafetyBoxSet(R_{inwork}, j) = \\ &\{shift(R', -g^{(j)}) | \forall R' \in shift(R_{inwork}, g^{(j)}) \setminus SafetyBox\}. \end{aligned}$$

It denotes the set of rectangles $\{R''\}$ such that $R'' \subseteq R_{inwork}$ and $shift(R'', g^{(j)}) \subset OutSafetyBox$.

Let

$$\begin{aligned} &UnsafetySet(R_{inwork}, j) = \{shift(R', -g^{(j)}) | \\ &\forall R' \text{ s.t. } R' = shift(R_{inwork}, g^{(j)}) \bigcap R'_{unsafe}, \\ &\text{where } R'_{unsafe} \in UnsafeRectSet, \text{ and } R' \ne \emptyset\} \end{aligned}$$

It denotes the set of rectangles $\{R''\}$ such that $R'' \subseteq R_{inwork}$ and $shift(R'', g^{(j)}) \subseteq R'_{unsafe}$, where R'_{unsafe} is a known unsafe rectangle.

So the set

$$SuspiciousSet(R_{inwork}, j) = OutSafetyBoxSet(R_{inwork}, j) \bigcup UnsafetySet(R_{inwork}, j)$$

is the set of rectangles $\{R''\}$ containing all $R'' \subseteq R_{inwork}$ and $shift(R'', g^{(j)})$ is unsafe.

Let

$$SuspiciousSet(R_{inwork}, j, j') = \bigcup_{R' \in SuspiciousSet(R_{inwork}, j)} SuspiciousSet(R', j')$$

It denotes the set of rectangles $\{R''\}$ containing all $R'' \subseteq R_{inwork}$, and $shift(R'', g^{(j)})$ and $shift(R'', g^{(j')})$ are both unsafe. Thus, we can compute the set $SuspiciousSet(R_{inwork}, 0, 1, \cdots, n)$, which is the set of rectangles $\{R''\}$ containing all $R'' \subseteq R_{inwork}$ and $shift(R'', g^{(j)})$ is unsafe for all $(0 \leq j \leq n)$. Therefore, $\forall R_{newUnsafe} \in SuspiciousSet(R_{inwork}, 0, 1, 2, \cdots, n)$, $R_{newUnsafe}$ satisfies the whole unsafety criteria. We thus attach them into the $UnsafeRectSet$. Then starting with a $R_{newUnsafe}$, repeat the process and find more unsafe rectangles. The process must terminate because $shift(R_{unsafe}, -g^{(j)})$ would shift the rectangle in at least $n-1$ positive directions. Finally the process must terminate when $R_{back} = \emptyset$.

Fig. 5 shows an example illustrating how the algorithm works, although it may not correspond to any real $2D$ system. Algorithm 1, 2 and 3 show the pseudocode of our algorithm of constructing the $UnsafeRectSet$. Once $UnsafeRectSet$ is constructed, the $UnsafeRegion$ is given by:

$$UnsafeRegion = \bigcup_{R \in UnsafeRectSet} R$$

Algorithm 1 `construct_UnsafeRectSet`

$UnsafeRectSet$ = $DeadRectSet$
for $R_{dead} \in DeadRectSet$ **do**
  `recur_sur`(R_{dead})
end for

Algorithm 2 `recur_sur`(R_{unsafe})

for $i = 0$ to n **do**
  s = `sur`(R_{unsafe}, i)
  if $s \neq \emptyset$ **then**
    for $R \in s$ **do**
      $UnsafeRectSet$.append(R)
      `recur_sur`(R)
    end for
  end if
end for

Algorithm 3 `sur`(R_{unsafe}, i)

$R_{back} = shift(R_{unsafe}, -g^{(i)}) \bigcap SafetyBox$
if $R_{back} == \emptyset$ **then**
  return $\emptyset$
end if
$workingSet = \emptyset$
for $R'_{unsafe} \in UnsafeRectSet$ **do**
  $workingSet = workingSet \bigcup (R_{back} \setminus R'_{unsafe})$
end for
for $j = 0$ to n **do**
  if $j == i$ **then**
    continue
  end if
  $newWorkingSet = \emptyset$
  for $R_{inwork} \in workingSet$ **do**

$$newWorkingSet = newWorkingSet \bigcup SuspiciousSet(R_{inwork}, j)$$

  end for
  $workingSet = newWorkingSet$
end for
return $workingSet$

B. Properties of $UnsafeRegion$

Lemma 4. *Any state in $UnsafeRegion$ is infeasible.*

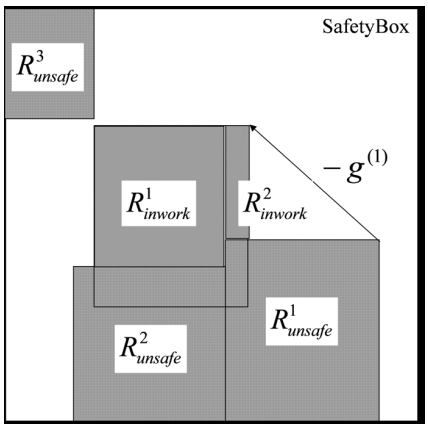
Proof: By definition, $\forall X \in UnsafeRegion, \exists R \in UnsafeRectSet, X \in R$. If $R \in DeadRectSet$, then the system must fail in next time step. If $R \notin DeadRectSet$, then from the unsafety criteria (3) $\exists R' \in UnsafeRectSet$ and corresponding $j (0 \leq j \leq n)$ such that $shift(R, g^{(j)}) \subseteq R'$. Suppose in the process of constructing $UnsafeRectSet$, we attach an incrementing ID to each rectangle added to the $UnsafeRectSet$ starting from those rectanges in $DeadRectSet$. Then R' must have a smaller ID than R. So any states in R must fall into a rectangle that has a smaller ID than R or out of the safety box. Thus by induction those states must eventually reach one of the $DeadRect$s or fail before that. In any case, those states will fail. ■

Let $SafeRegion = SafetyBox \setminus UnsafeRegion$.

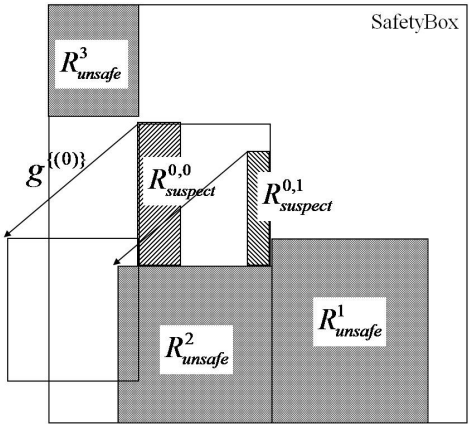
Lemma 5. *Any state in $SafeRegion$ is feasible.*

Proof: $\forall X \in SafeRegion$, since the state space is continuous, there must exist a rectangle R such that $X \in R$. Then R must neither be in $UnsafeRectSet$ nor a part of a rectangle that is in $UnsafeRectSet$. So $\exists j (0 \leq j \leq n), shift(R, g^{(j)}) \subset SafeRegion$. Thus $X + g^{(j)} \in SafeRegion$. So by induction there is an infinite sequence $g^{(j)} g^{(j')} \cdots$ such that $X + g^{(j)} + g^{(j')} + \cdots \in SafeRegion$. Thus X is feasible. ■

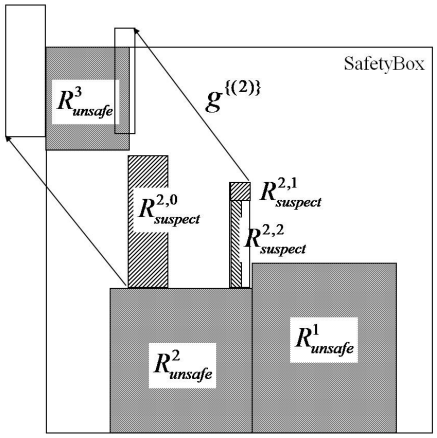
Theorem 3. *A system is feasible if and only if its initial state $X(0) \in SafeRegion$.*



(a) $workingSet = \{R^1_{inwork}, R^2_{inwork}\}$.



(b) Consider R^1_{inwork} now.
$SuspiciousSet(R^1_{inwork}, 0) = \{R^{0,0}_{suspect}, R^{0,1}_{suspect}\}$,
because $OutSafetyBox(R^1_{inwork}, 0) = \{R^{0,0}_{suspect}\}$
and $UnsafetySet(R^1_{inwork}, 0) = \{R^{0,1}_{suspect}\}$.



(c) $SuspiciousSet(R^1_{inwork}, 0, 1) = SuspiciousSet(R^1_{inwork}, 0)$,
because $\forall R \in SuspiciousSet(R^1_{inwork}, 0), shift(R, g^{(1)}) \in R^1_{unsafe}$.

$SuspiciousSet(R^1_{inwork}, 0, 1, 2) = \{R^{2,0}_{suspect}, R^{2,1}_{suspect}, R^{2,2}_{suspect}\}$,
becuase $OutSafetyBox(R^{0,0}_{suspect}, 2) = \{R^{0,0}_{suspect}\}$,
$UnsafetySet(R^{0,0}_{suspect}, 2) = \emptyset$,
$OutSafetyBox(R^{0,1}_{suspect}, 2) = \{R^{2,1}_{suspect}\}$ and
$UnsafetySet(R^{0,1}_{suspect}, 2) = \{R^{2,2}_{suspect}\}$

Figure 5. R^1_{unsafe}, R^2_{unsafe} and R^3_{unsafe} are known unsafe rectangles. Find more unsafe rectangles by examining $R_{back} = shift(R^1_{unsafe}, -g^{(1)}) \bigcap SafetyBox$. Two sub-rectangles of R_{back} are put into the working set: R^1_{inwork} and R^2_{inwork}, whereas others are discarded because they are already included in known unsafe rectangles. Three unsafe rectangles are found in R^1_{inwork}: $R^{2,0}_{suspect}$, $R^{2,1}_{suspect}$ and $R^{2,2}_{suspect}$. Note that there are different ways of partitioning the space of $R^1_{inwork} \cup R^2_{inwork}$ into a $workingSet$. However they are equivalent in producing the final result: $\bigcup_{R_{inwork} \in workingSet} \bigcup_{R_{newUnsafe} \in SuspiciousSet(R_{inwork}, 0, 1, 2)} R_{newUnsafe}$. The reason is that the operations on R_{inwork} to compute $SuspiciousSet(R_{inwork}, 0, 1, 2)$ are shifting, set difference, intersection and union, and all these operations are distributive over set union: shifting is obviously distributive over set union: $shift(A \cup B, v) = shift(A, v) \cup shift(B, v)$, and so is set union itself; set difference and intersection are also distributive over set union [6].

C. Complexity of the algorithm

The recursive call of `recur_sur` resembles a tree structure. From a rectangle R_{dead}, each $g^{(i)} (1 \leq i \leq n)$ may lead to a set of unsafe rectangles. From every one of those rectangles, other unsafe rectangles are found in the same fashion. The third case of 2D system is a good example (Fig. 6), although in this case any finite set of unsafe rectangles returned by `sur` call contains only one element.

Our algorithm performs a depth first search on the tree. The search must terminate because every child rectangle is closer than its parent in at least $n-1$ directions to the point $\begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^T$ in the state space. Eventually, a child R would be too close such that $\forall i (0 \leq i \leq n), shift(R, -g^{(i)}) \subset OutSafetyBox$. Consider the sum of the lower bounds of a rectangle R in the tree in all directions. That is, for $R = [r_{10}, r_{11})_1 \times [r_{20}, r_{21})_2 \times \cdots \times [r_{n0}, r_{n1})_n$, let $S = \sum_{i=1}^{n} r_{i0}$. Also let $\tilde{b}_{min} = min(\tilde{b}_1, \tilde{b}_2, \cdots, \tilde{b}_n)$. From a parent to a child in the tree, S is increased by at least $(n-1)\tilde{b}_{min}$. But $S < n$. So the height of the tree $treeHeight < \frac{n}{(n-1)\tilde{b}_{min}}$. However the number of unsafe rectangles may be exponential to n. Note that from a parent to a child, the size of the rectangle must be decreasing. So there must be a rectangle of the smallest size λ. Since the length of the rectangle in every direction is less than 1, λ is exponential to $-n$. The total number of unsafe rectangles N is in the order of $\frac{1}{\lambda}$, thus N is exponential to n. Indeed, if $\tilde{b}_{min}$ is very small compared to 1, not only the tree could be tall but also the

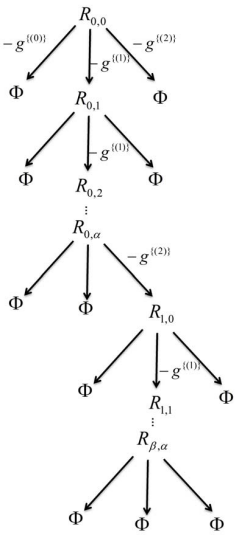


Figure 6. Tree structure of `recur_sur` call on the third case 2D system.

unsafe rectangles tend to be smaller. The double effects may bring a big performance issue to our algorithm.

Let us determine the time complexity of our algorithm more formally. λ should be in the order of $(\tilde{b}_{min})^n$, so N is in the order of $(\frac{1}{\tilde{b}_{min}})^n$. To find a unsafe rectangle, it takes $\mathcal{O}(N)$ time to compute the $workingSet$ and the $workingSet$ is of size $\mathcal{O}(N)$. For each $R_{inwork} \in workingSet$, it takes $\mathcal{O}(nN)$ time to compute the $SuspiciousSet$ for all directions. So it may take $\mathcal{O}(nN^2)$ time to find a unsafe rectangle. The total time is thus $\mathcal{O}(nN^3)$ or $\mathcal{O}(n(\frac{1}{\tilde{b}_{min}})^{3n})$.

D. Scheduling policy

The proof of Lemma 5 suggests a method by the controller that computes the next scheduling decision on the fly: $\forall k \geq 0$, search for $g^{(j)}$ such that $X(k) + g^{(j)} \in SafeRegion$. The searching method may incorporate some priority policy or integrate some optimal decision algorithm chosen by the controller. For example, if $g^{(0)}$ has the highest priority, then the controller always seeks to turn OFF all the tasks whenever possible (power-saving). In fact, the General 2D Scheduling Policy is such a policy. The controller may also maintain frequency counts for all tasks to implement a fair policy.

E. Extensions

Although in the paper we assume at most one task could be ON at any time, our algorithm can be extended to deal with the situation where at most $k(k < n)$ tasks can be ON at any time. By that end, the set of possible movements G must be enlarged to add $\sum_{j=2}^{k} \binom{n}{j}$ directions. Meanwhile, the Type 1 dead rectangles reduce to

$$DeadRect(\text{Type 1}) = [0,1)_1 \times \cdots \times [0,\tilde{b}_{j_1})_{j_1} \times \cdots \times [0,\tilde{b}_{j_{k+1}})_{j_{k+1}} \times \cdots \times [0,1)_n$$

for $(1 \leq j_1 \neq j_2 \neq \cdots \neq j_{k+1} \leq n)$. Although the increase on the number of possible movements might blow up the problem, the second for loop in the algorithm 3 can be parallelized. Also note that this algorithm is only run once to get the safe regions and the subsequent scheduling policy is computed on-the-fly (the choosing of $g^{(i)}$ at any time t can be parallelized too). The computation of safe regions is only needed when the system gets reconfigured.

The state space trajectories considered in this paper are linear. However, our algorithm can be extended to deal with more complex tasks' dynamics as well, for example those characterized by affine differential equations, which are considered in [2]. This extension is beyond the scope of this paper and will be addressed in future papers.

V. Simulation and Results

We perform simulations on different systems. Fig. 7 shows the results on some 2D and 3D systems. The rectangles in gray are the unsafe rectangles. Fig. 7a shows a 2D system that does not satisfy the necessary condition given by (7). All the states in the system are infeasible. The system in Fig. 7b just satisfies the necessary condition. It belongs to the third case of 2D systems we discussed. Fig. 7c shows a 2D system of the second case, while in Fig. 7d a system of the first case is shown. The system in Fig. 7e is not discussed because we assume $\tilde{a}_1 + \tilde{b}_1 \leq 1$ and $\tilde{a}_2 + \tilde{b}_2 \leq 1$ in order to get an analytical form of necessary and sufficient conditions. But our algorithm does not make any assumption on $\{\tilde{a}_i, \tilde{b}_i\}$ thus it can correctly find the unsafe rectangles.

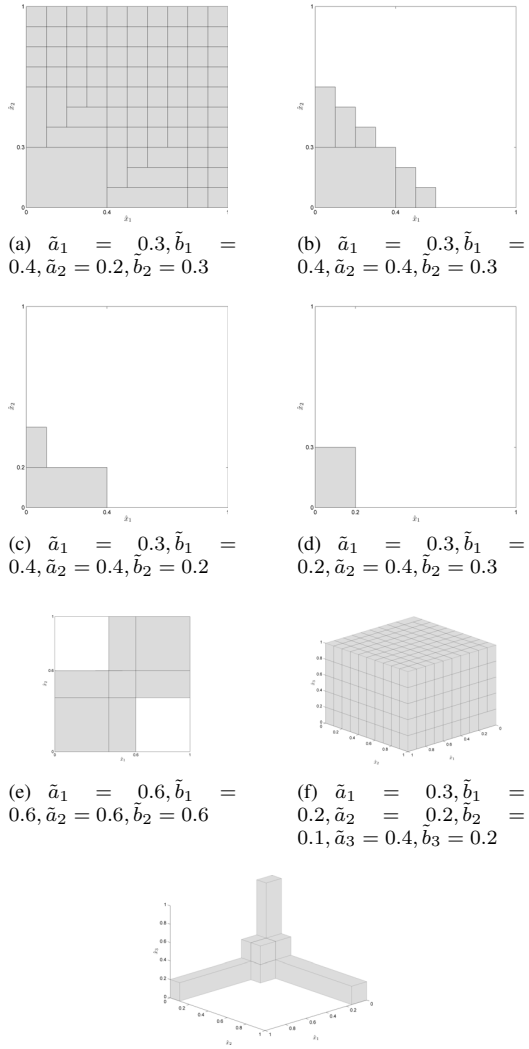
Fig. 7f shows an example of 3D systems that do not satisfy the necessary condition (6). All states are infeasible there. The system in Fig.7g just satisfy the condition, so the system is feasible if and only if the initial state is not in the unsafe region, shown in gray in the figure. The algorithm suggests the *safe* region is the following:

```
[0.3,1.0) X [0.1,1.0) X [0.0,1.0) U
[0.15,0.3) X [0.2,1.0) X [0.0,1.0) U
[0.15,0.3) X [0.1,0.2) X [0.4,1.0) U
[0.3,1.0) X [0.0,0.1) X [0.2,1.0) U
[0.15,0.3) X [0.0,0.1) X [0.4,1.0) U
[0.0,0.15) X [0.2,1.0) X [0.2,1.0) U
[0.0,0.15) X [0.1,0.2) X [0.4,1.0)
```

VI. Related Work

As mentioned, [2] presents a necessary and sufficient condition in continuous time domain for any system to be feasible. However, when time is discretized additional conditions are required. [2] also mentions that systems in the intersection of two critical zones are infeasible but those infeasible regions are not complete. Other regions may also be infeasible. In this paper we present an algorithm to find all the infeasible regions for the discrete-time linear case.

Work has been done towards energy efficient CPU scheduling using Dynamic Voltage Scaling (DVS) and energy aware task allocation ([7], [8] and [9]). The integration of control and scheduling has been covered by [10], [11] and [12], where real-time scheduling approaches are extended to incorporate control task specifications. Our approach and the approach in [2], [3] is focused on energy consuming control systems with a system-wide resource constraint and departs from a CPU-centric view



(a) $\tilde{a}_1 = 0.3, \tilde{b}_1 = 0.4, \tilde{a}_2 = 0.2, \tilde{b}_2 = 0.3$

(b) $\tilde{a}_1 = 0.3, \tilde{b}_1 = 0.4, \tilde{a}_2 = 0.4, \tilde{b}_2 = 0.3$

(c) $\tilde{a}_1 = 0.3, \tilde{b}_1 = 0.4, \tilde{a}_2 = 0.4, \tilde{b}_2 = 0.2$

(d) $\tilde{a}_1 = 0.3, \tilde{b}_1 = 0.2, \tilde{a}_2 = 0.4, \tilde{b}_2 = 0.3$

(e) $\tilde{a}_1 = 0.6, \tilde{b}_1 = 0.6, \tilde{a}_2 = 0.6, \tilde{b}_2 = 0.6$

(f) $\tilde{a}_1 = 0.3, \tilde{b}_1 = 0.2, \tilde{a}_2 = 0.2, \tilde{b}_2 = 0.1, \tilde{a}_3 = 0.4, \tilde{b}_3 = 0.2$

(g) $\tilde{a}_1 = 0.3, \tilde{b}_1 = 0.15, \tilde{a}_2 = 0.2, \tilde{b}_2 = 0.1, \tilde{a}_3 = 0.4, \tilde{b}_3 = 0.2$

Figure 7. $UnsafeRegion$ (in gray) of different systems.

to a PSU-centric (Power Supply Unit) resource allocation. The extension of traditional real time scheduling algorithms like EDF and RMS, for activation of electrical loads is used in [13] and [4] by assuming a periodic task activation model for the physical system. Although the periodic task model allows for the use of traditional scheduling algorithms but the underlying assumption that electrical loads need to switch periodically (and not based on state feedback) makes the system overly constrained and less flexible to changes in system dynamics.

VII. Conclusion

We have studied the feasibility of the discrete-time linear green scheduling problem. For a system of two tasks, if certain assumptions hold, we can determine analytically the necessary and sufficient conditions for the system to be feasible. For more general n-dimensional systems, we design and implement an algorithm which, given the increasing and decreasing rates of the tasks of a system, returns a subset of the state space such that the system is feasible if and only if the initial state is in the subset. Given the subset, a scheduling policy can be computed on the fly as the system runs, with the flexibility to add any power-saving, priority based or fair sub-policies. [1]

References

[1] K. Spees and L. Lave, "Impacts of responsive load in pjm: Load shifting and real time pricing," *The Energy Journal*, vol. 29, no. 2, pp. 101–122, 2008.

[2] T. Nghiem, M. Behl, R. Mangharam, and G. J. Pappas, "Green scheduling of control systems for peak demand reduction," in *50th IEEE Conference on Decision and Control (CDC)*, Dec 2011.

[3] T. Nghiem, M. Behl, G. J. Pappas, and R. Mangharam, "Green scheduling: Scheduling of control systems for peak power reduction," in *2nd IEEE International Green Computing Conference (IGCC)*, July 2011.

[4] M. D. Vedova, M. Ruggeri, and T. Facchinetti, "On real-time physical systems," in *Proceedings of the 18th International Conference on Real-Time and Network Systems (RTNS)*, 2010, pp. 41–49.

[5] UPPAAL, "Website," http://www.uppaal.com.

[6] J. Woodcock and M. Loomes, *Software Engineering Mathematics*. Pitman and Addison-Wesley, 1988.

[7] T. AlEnawy and H. Aydin, "Energy-aware task allocation for rate monotonic scheduling," in *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE*, 2005, pp. 213 – 223.

[8] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *Computers, IEEE Transactions on*, vol. 53, no. 5, pp. 584 – 600, May 2004.

[9] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, ser. SOSP '01, 2001, pp. 89–102.

[10] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "Trade-off analysis of real-time control performance and schedulability*," *Real-Time Systems*, vol. 21, pp. 199–217, November 2001.

[11] A. Cervin and J. Eker, "The control server: A computational model for real-time control tasks," in *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, July 2003, pp. 113–120.

[12] R. Chandra, X. Liu, and L. Sha, "On the scheduling of flexible and reliable real-time control systems," *Real-Time Systems*, vol. 24, pp. 153–169, March 2003.

[13] T. Facchinetti, E. Bini, and M. Bertogna, "Reducing the peak power through real-time scheduling techniques in cyber-physical energy systems," in *Proceedings of the First International Workshop on Energy Aware Design and Analysis of Cyber Physical Systems (WEA-CPS)*, 2010.

[1]This research is partially supported by a grant from the Office of Naval Research under award number N00014-10-1-0359.