Towards Assurance for Open Soft Real-Time Systems*

Chris Gill
Associate Professor of Computer Science and Engineering
Washington University, St. Louis, MO, USA
cdgill@cse.wustl.edu

3rd Layered Assurance Workshop (LAW '09)

August 4 - 5, 2009, San Antonio, TX, USA

*Research supported in part by NSF awards CNS-0716764 (Cybertrust) and CCF-0448562 (CAREER)

*Thanks to the many collaborators who have been involved with different aspects of this work, including: Terry Tidwell, Robert Glaubius, Venkita Subramonian, Huang-Ming Huang, Cesar Sanchez, William D. Smart, Doug Niehaus, Henny Sipma, and Zohar Manna

Motivation: Open Soft Real-Time Systems

- Interact with variable environments
  - » Varying degrees of autonomy
  - » Performance is deadline sensitive
  - » Often involve cyber-physical semantics
- Many activities must run at once
  - » Device interrupt handing, computation
  - » Comm w/ other systems/operators
- Need *assurance* of properties
  - » E.g., how to enforce utilization shares when scheduling shared resources for competing, variable execution times?

Example Systems:
(1) Self-Maintaining Clusters,
(2) Lewis (WUSTL M&M Lab)

Remote Operator Station
(for all but full autonomy)

Wireless
Communication

An Example of What We Want to Assure

- A mobile robot with multiple onboard sensors interacting with a dynamic unknown environment
- Onboard sensors have different but measurable characteristics
  - » Camera - long uncertain capture time (scene dependent)
  - » Laser range finder - shorter more predictable interval
- Scheduling the orientation, activation, and processing of such sensors impacts timely and correct operation of the systems as a whole
  - » Robot can pause (soft real-time consequences of delay)
    - But this is sub-optimal and constitutes an assurance failure
  - » Many forms of resource contention must be considered
    - E.g., CPU utilization by processing threads
    - E.g., shared pan-tilt unit on which sensors are mounted

Overview for this Talk

Background on requirements, platforms, and formal models

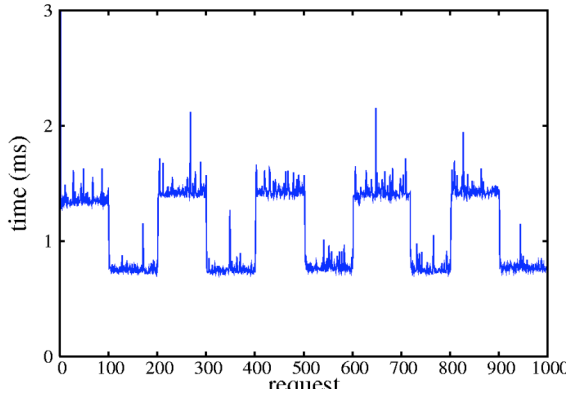
I. Toward assurance in cyber-physical systems (including mixed-criticality systems as an important sub-category)
II. The challenge to evolve platforms - OS and middleware
III. Interactions with the environment and other actors

Towards assurance for open soft real-time systems

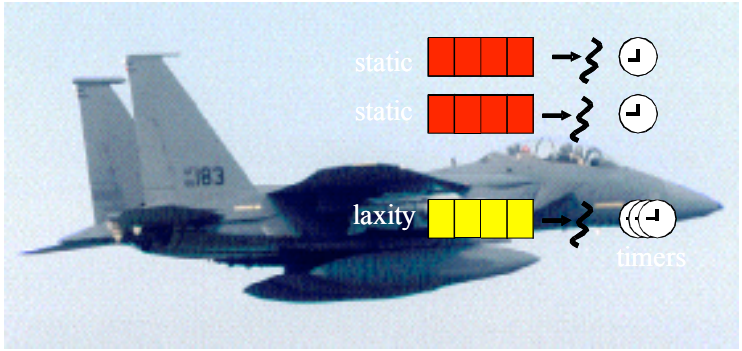
IV. Scheduling policy design in open soft real-time systems
V. State space representations and reductions
VI. Towards (timed) verification in those systems

I. Toward Assurance in Cyber-Physical Systems

- *Inter-dependence* is fundamental, and spans system properties: functionality, distribution, concurrency, etc.
- *Physical world* introduces quantitative (continuous) time and uncertainty/variability in execution/response times: i.e., inter-dependence of cyber and physical properties
- *Mixed criticality* systems are an important sub-category



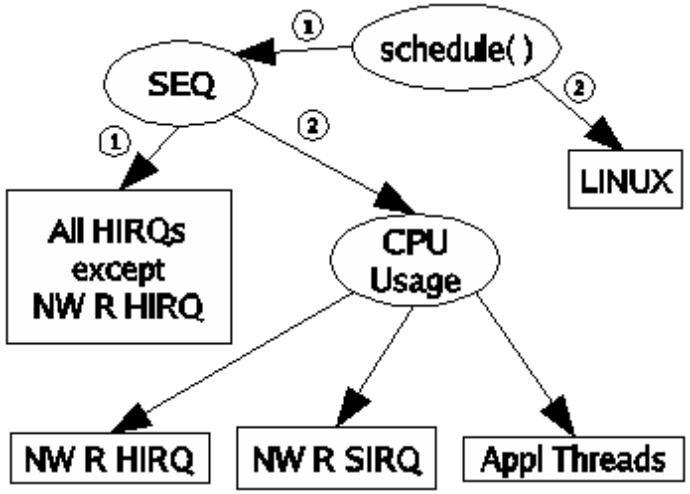
Variable and modal image capture response times due to scene, occlusion



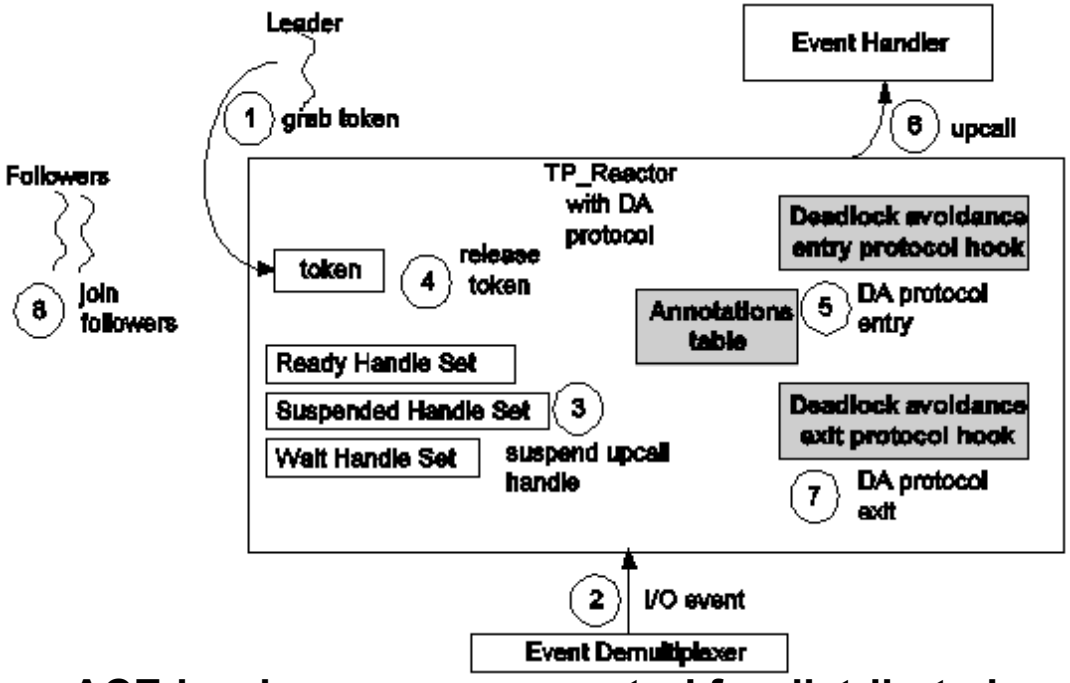
ASFD and WSOA software and flight demonstrations (AFRL, DARPA, Boeing, BBN, Honeywell, WUSTL, OIS)

II. The Challenge to Evolve Platforms

- HW, OS and/or middleware must *enforce* system properties
- *Architecture* may aid or thwart modeling and control of system properties (e.g., event sequencing and timing)
- Even *mechanisms* for following a protocol can be challenging
- Beyond functional plug-n-play to *separation* and *composition*



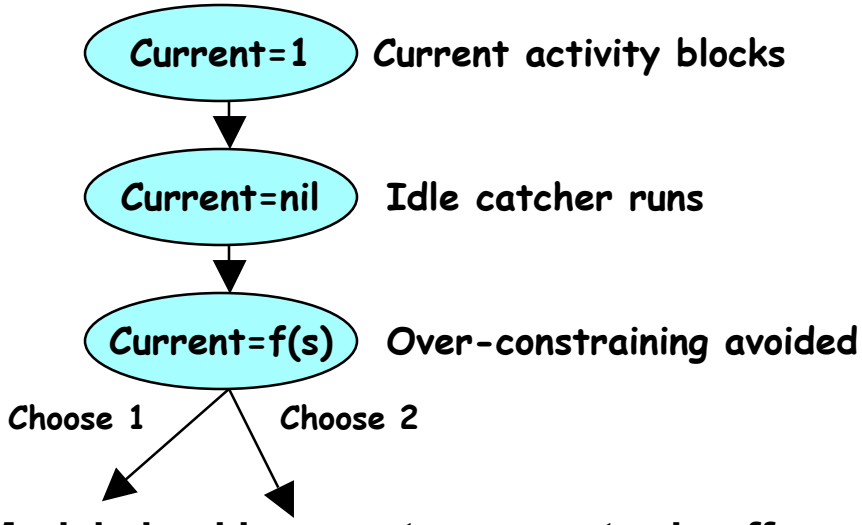
KURT-Linux/Kokyu Group Scheduling (DARPA, U. Kansas, WUSTL)



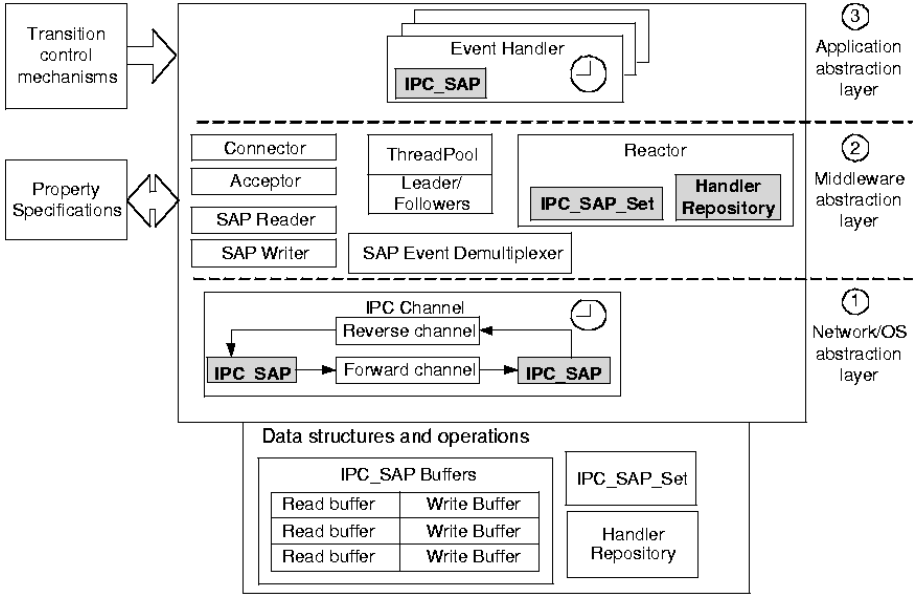
ACE-level concurrency control for distributed deadlock avoidance (NSF, WUSTL, Stanford)

III. Interactions

- Modeling the enforced timing, event ordering, and resource utilization constraints requires comparably rich mechanisms
- System behavior is strongly influenced by interactions among system components, environment, and other actors
- Constraining interaction may be useful, but in many systems some interactions are fundamental (can't design them away)



Model checking must manage trade-offs between under- and over-constraining the system state space (Subramonian)



ACE-level system components modeled by timed automata in IF and UPPAAL (Subramonian)

IV. Scheduling Policy Design Approach

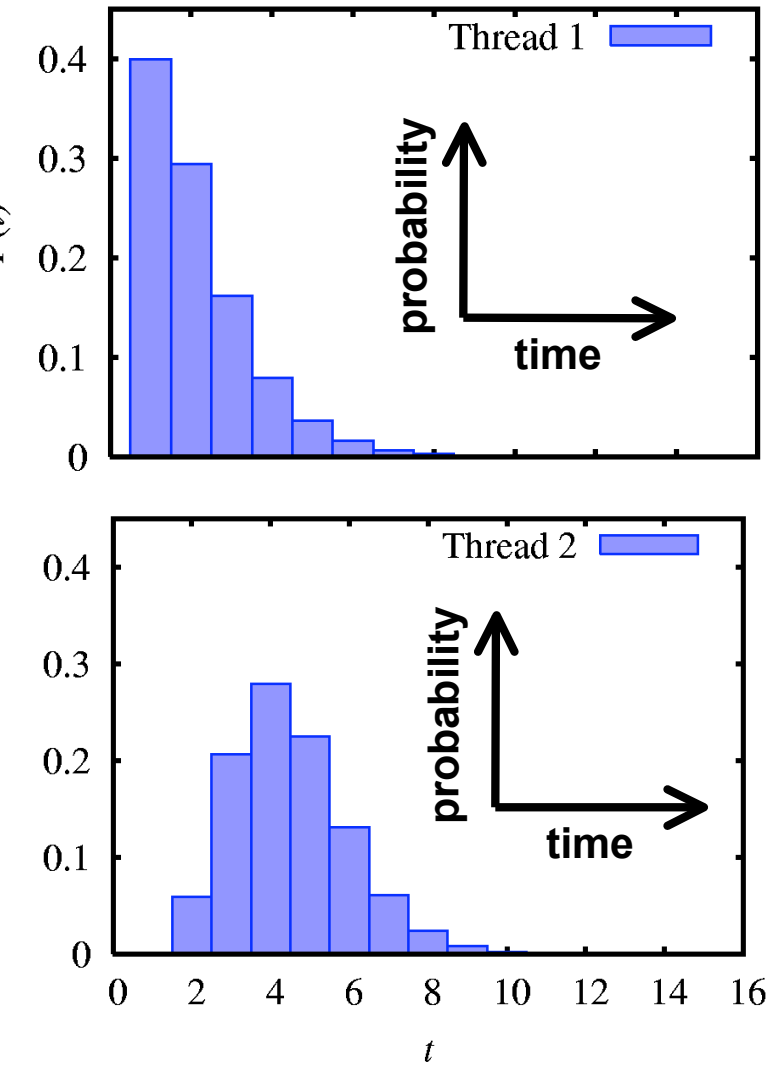
- Our focus is on "messy" details of actual systems
  - Especially, on inter-dependent execution (e.g., due to resource contention introduced in design)
  - Even simple cases are non-trivially challenging to assure
- Goal of the work on which much of the rest of this talk rests is to develop rational scheduling policies
  - Assume uncertain and interdependent execution
  - Still enforce desired properties (e.g., resource shares)
- We leverage techniques from machine learning
  - Represent uncertainty as Markov Decision Process
  - Perform policy iteration to find locally optimal policy

Consider a (Very Simple) System Model

- Separate activities require a shared resource
  - » Require mutually exclusive access to run
  - » E.g., non-preemptive threading, or device sharing, etc.
- Each activity binds the resource when it runs
  - » Binds resource for a duration then releases it
  - » Modeled using discrete variables that count *time quanta*
- Variable execution times with known distributions
  - » We assume that each activity's run-time distribution is known and bounded, and independent of the others
- Non-preemptive scheduler (repeats perpetually)
  - » Scheduler chooses which activity to run (based on *policy*)
  - » Scheduler dispatches activity which runs until it yields
  - » Scheduler waits until the activity releases the resource

Scheduling Policy Design Considerations

- We summarize system state as a vector of integers
  - » Encode resource usage times
- Each usage time comes from a known, bounded distribution
- Scheduling an activity changes the system state
  - » Utilization changes after activity runs, based on actual usage time
  - » State transition probabilities are based on usage time distributions
- This forms a basis for policy design and optimization



From Usage Times to a Scheduling MDP

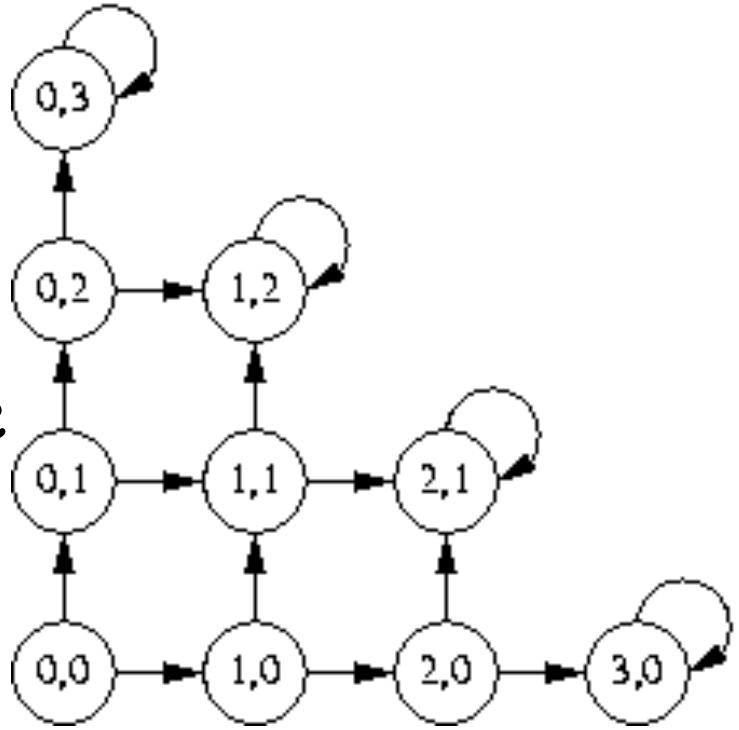
- We model scheduling decisions as a Markov Decision Process (MDP) – e.g., based on thread run times
- The MDP is given by 4-tuple: (X,A,R,T)
  - X: the set of process states
    - Correspond to thread utilization states
  - A: the set of actions
    - I.e., scheduling a particular thread
  - R: reward function for taking an action in a state
    - Expected utility of taking that action
    - Distance of the next state(s) from a desired utilization (vector)
  - T: transition function
    - Encodes the probability of moving from one state to another state for each action
- Solve MDP to obtain a locally optimal policy

Policy Iteration Approach

- Define a cost function $r(x)$ penalizing deviation from target utilization
- Start with some initial policy π_0
- Repeat for t=0,1,2,…
  - » Compute the value $V_t(x)$ -- the accumulated cost of following π_t -- for each state x.
  - » Obtain a new policy, π_{t+1}, by choosing the greedy action at each state.
- Guaranteed to converge to the optimal policy
- Requires storing V_t and π_t in lookup tables.

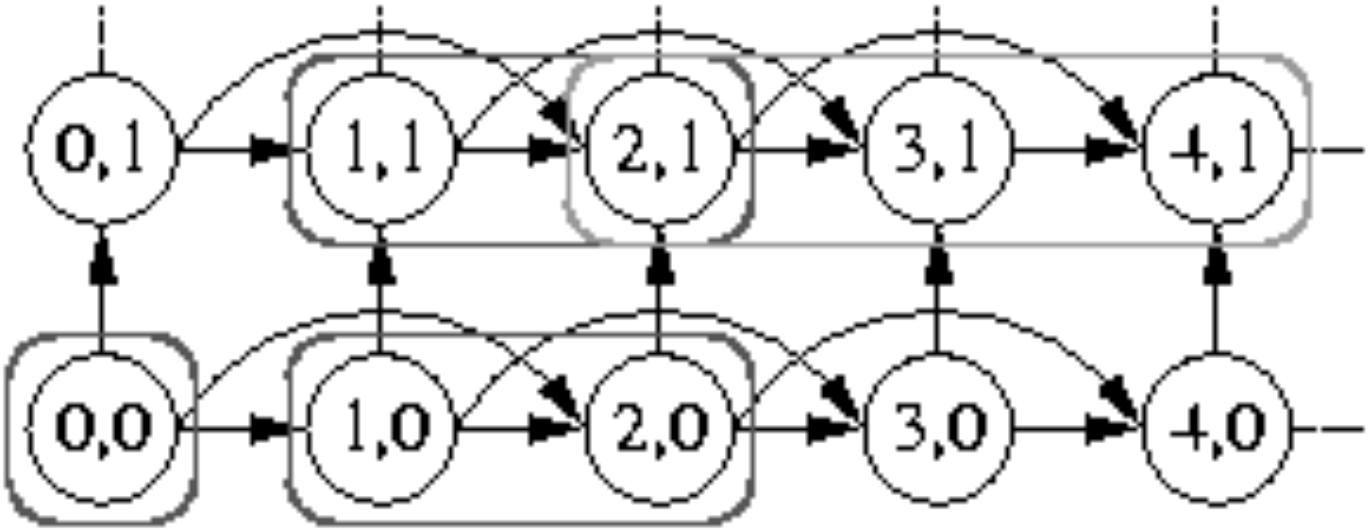
Basic Utilization State Space Structure

- Simple (here unary) expansion of increases in resource use
- To bound the state space, we used a system termination notion in our initial approach
  - Produces absorbing states where utilization stays same
  - E.g., 0,3 and 1,2 etc.
- Drawbacks
  - Artifacts, limited horizon
- A limited but illustrative model
  - Later we exploit state space structure to remove termination



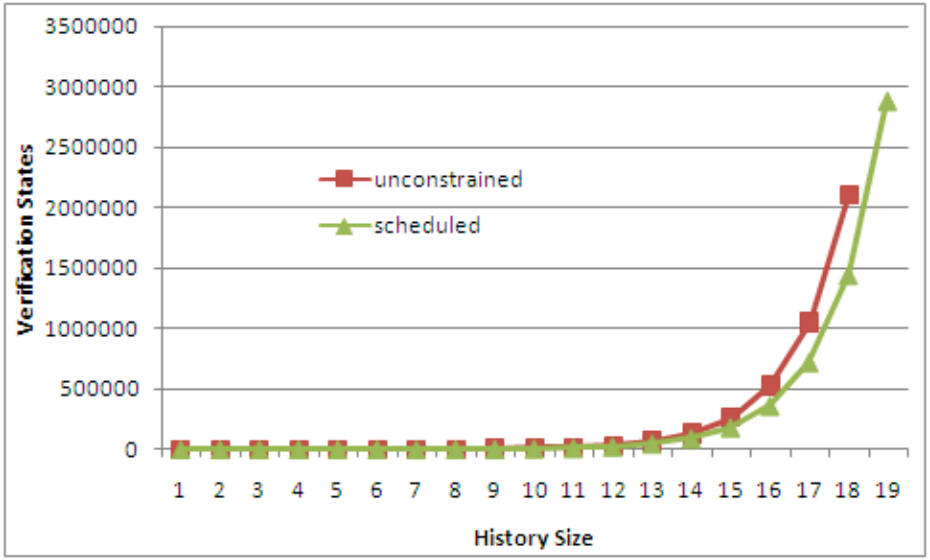
Constructing a Verification State Space

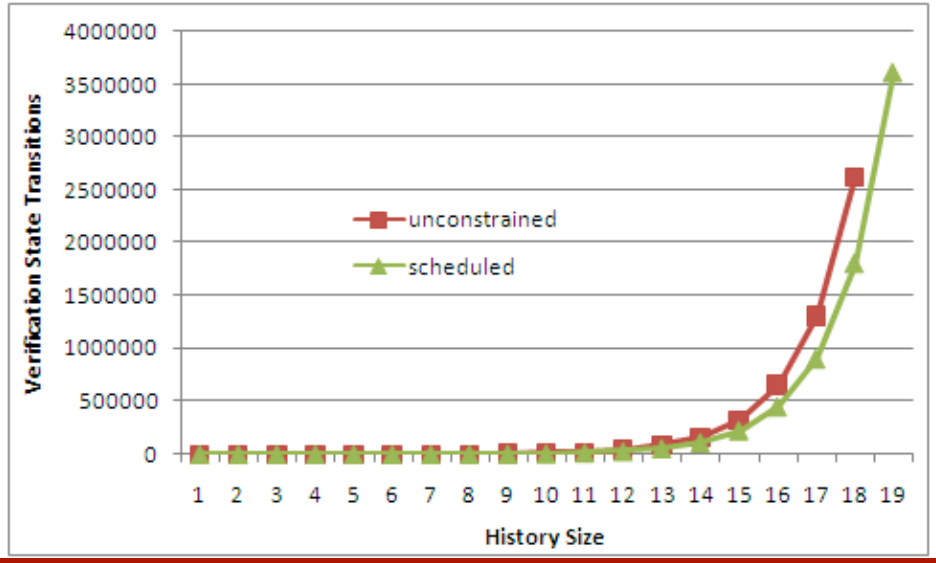
- We can also generate a verification state space
  - A verification state (box) combines utilization state (circle) subsets reachable on a scheduling action
  - Transitions condensed from the utilization state space
- Note that verification states also often overlap
  - E.g., utilization state 2,1 is in two verification states

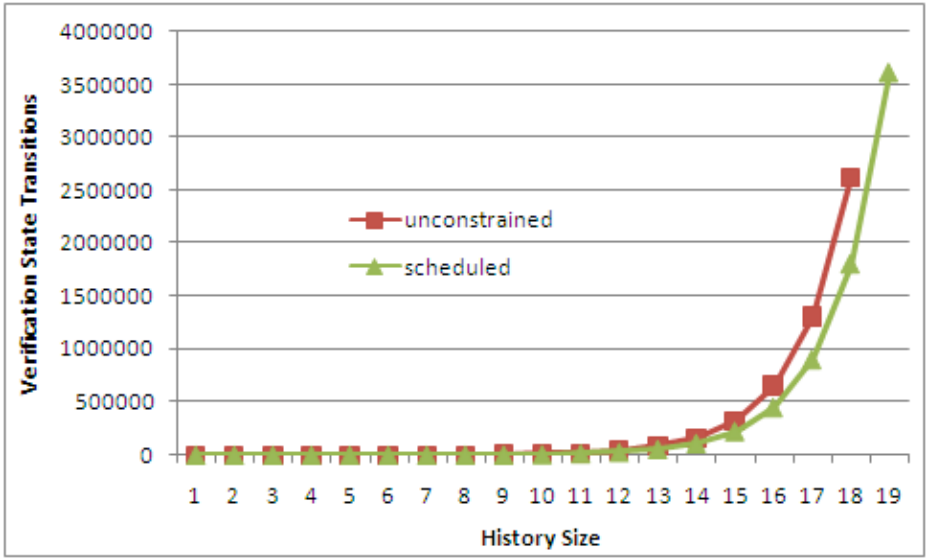


Basic Verification State Space Size and Cost

- State space *exponential* in # of threads *and* time to termination (history)
- Scheduling reduces cost but only delays explosion
- Motivates further work to reduce the state space





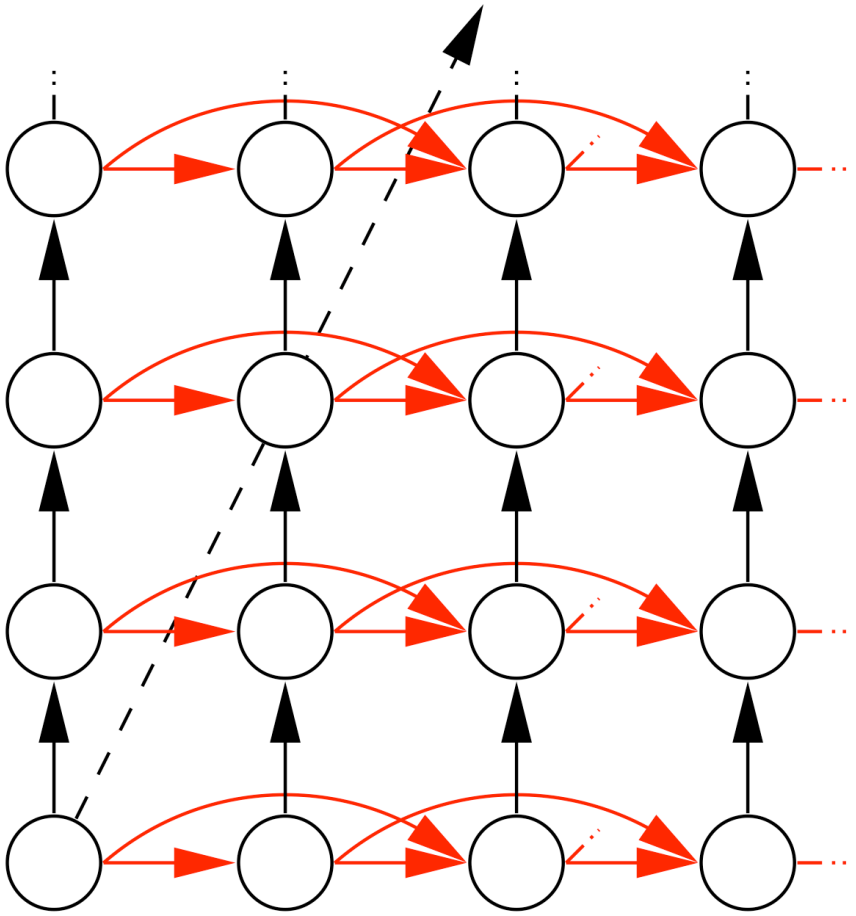


V. State Space Representations and Reductions

- Automated verification introduces fundamental problems of decidability and tractability
  - We restrict our attention to generally decidable representations suitable for model checking (e.g., timed automata and no stronger)
  - However, tractability still remains a major issue
- We look for inherent and induced regularity of state spaces (e.g., quasi-cyclic structure)
  - This in turn allows state space reduction by folding together equivalent states

Limitations of Policy Iteration for Scheduling

- Utilization state space is not fundamentally finite
  - » Termination boundary is an artificial restriction
- Can't apply MDP solution techniques directly to actual state space
  - » Need to reduce the size of the state space in order to solve for some policy.
  - » Our approach: reduce the state space to a collection of equivalence classes.

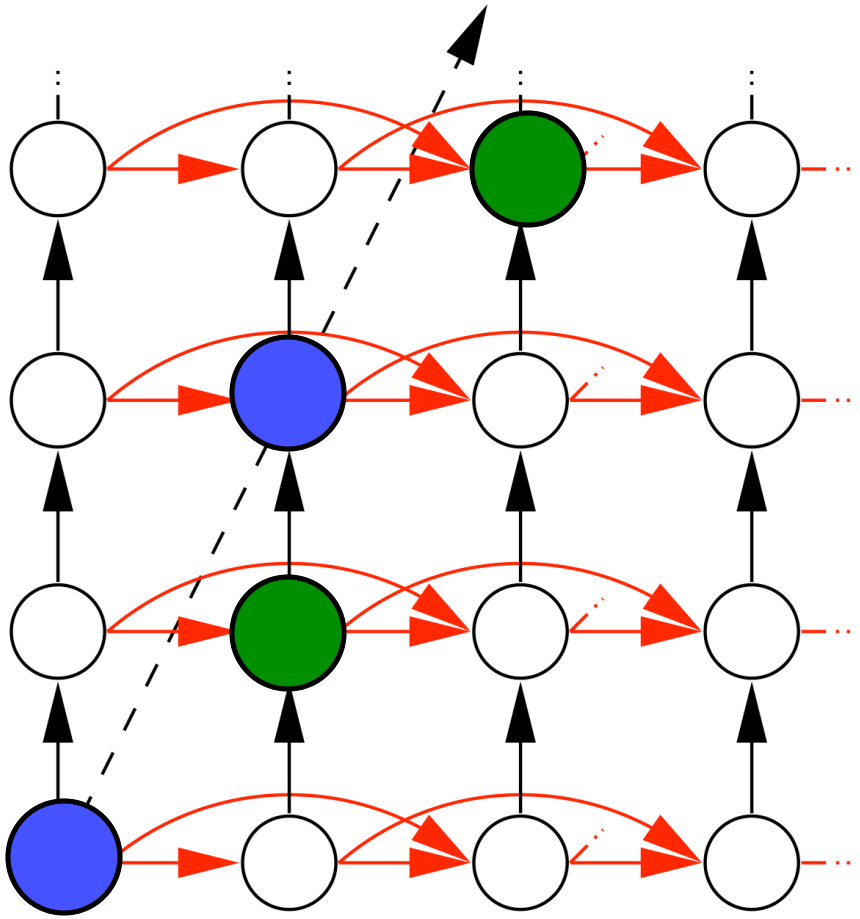


Insight: State Value Equivalence

- Cost function

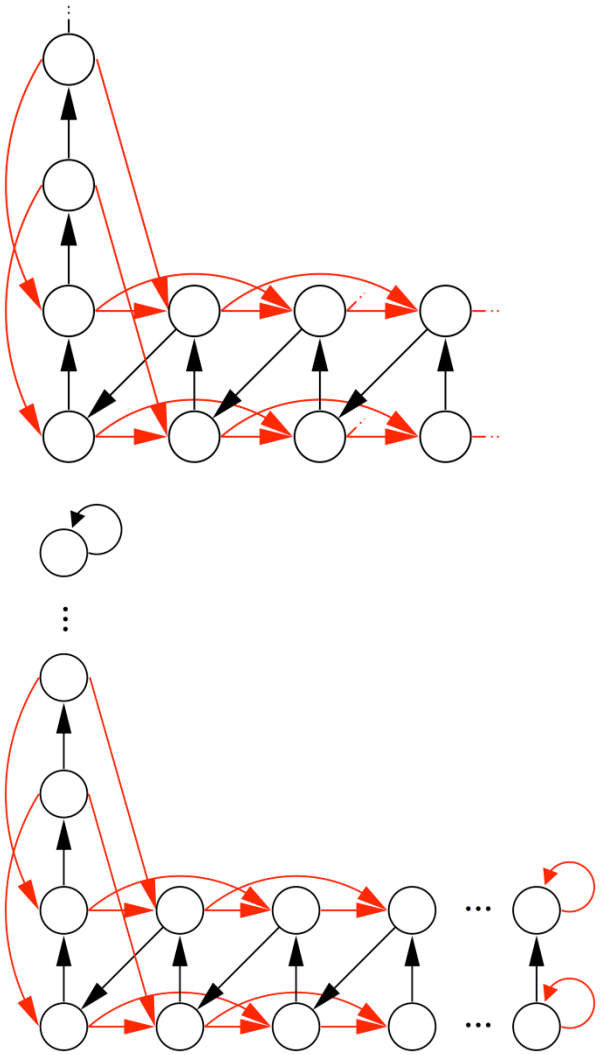
$$r(x) = -\left\| x - \|x\|_1 u \right\|_1$$

- Any two states co-linear along the target utilization ray have the same cost
- Any two states have the same relative distribution over future states
- *Any two states with the same cost have the same optimal value!*



Technique: State Wrapping

- We are thus able to collapse the equivalent states down into a set of exemplar states
- We then bound the number of states by introducing "absorbing" states
  - » Greedy and optimal policies appear to agree at sufficient distance from target utilization
- Can then use policy iteration to obtain a policy



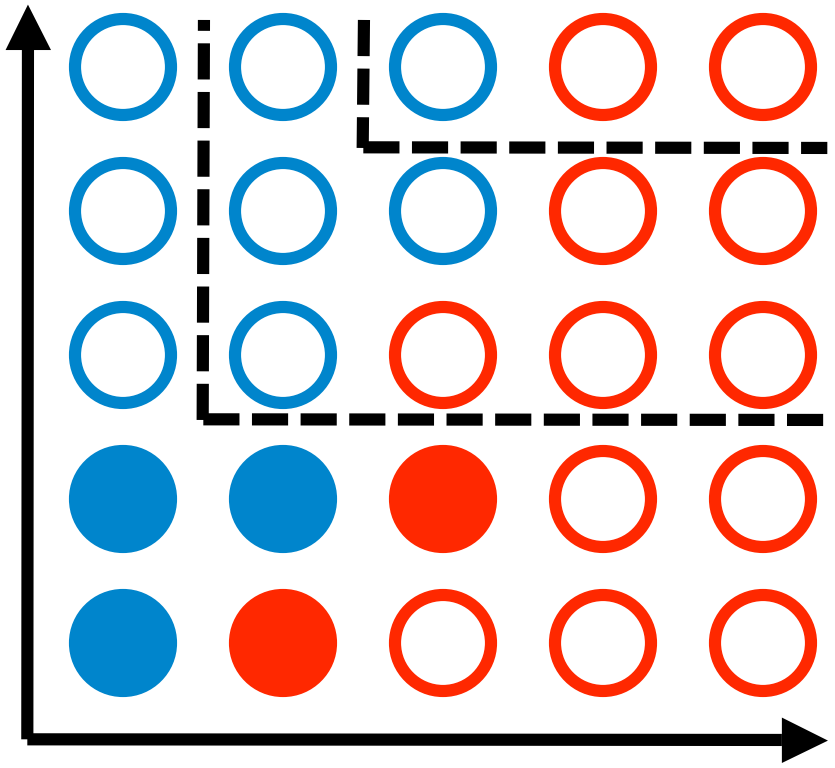
Successor Function for Wrapped State Space

- Successor function computation steps:
  - » Split on policy
  - » Simulate action
  - » Wrap

$$succ(s,a) = \{\{w(x + t\delta_a) \mid w(x) \in s, \pi(w(x)) = a, t \in [\alpha_a, \beta_a]\}\}$$

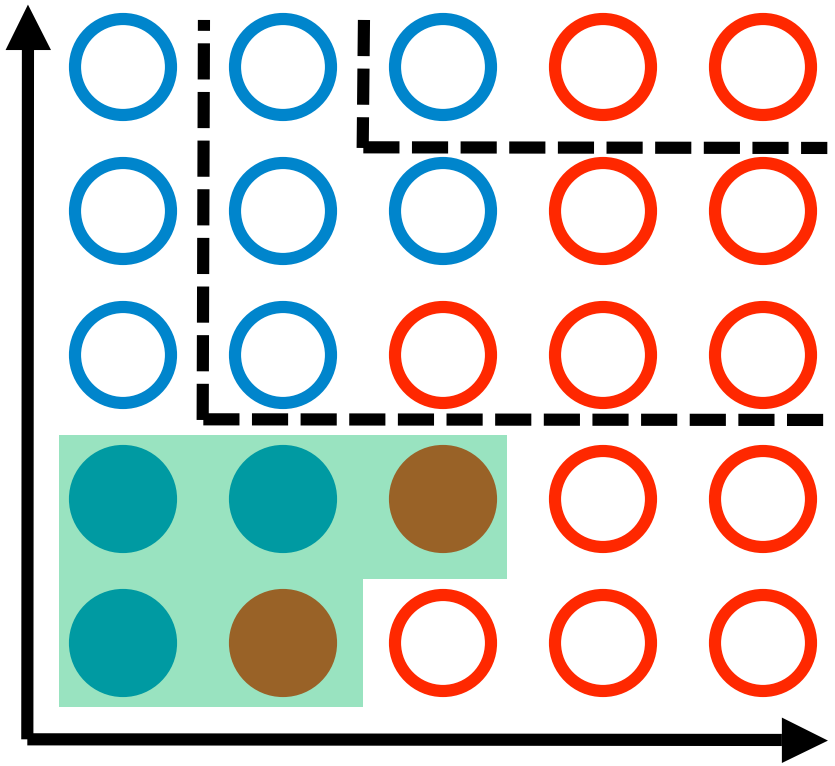
The Successor Function (Illustrated)

$$succ(s,a) = \{\{w(x + t\delta_a) \mid w(x) \in s, \pi(w(x)) = a, t \in [\alpha_a, \beta_a]\}\}$$



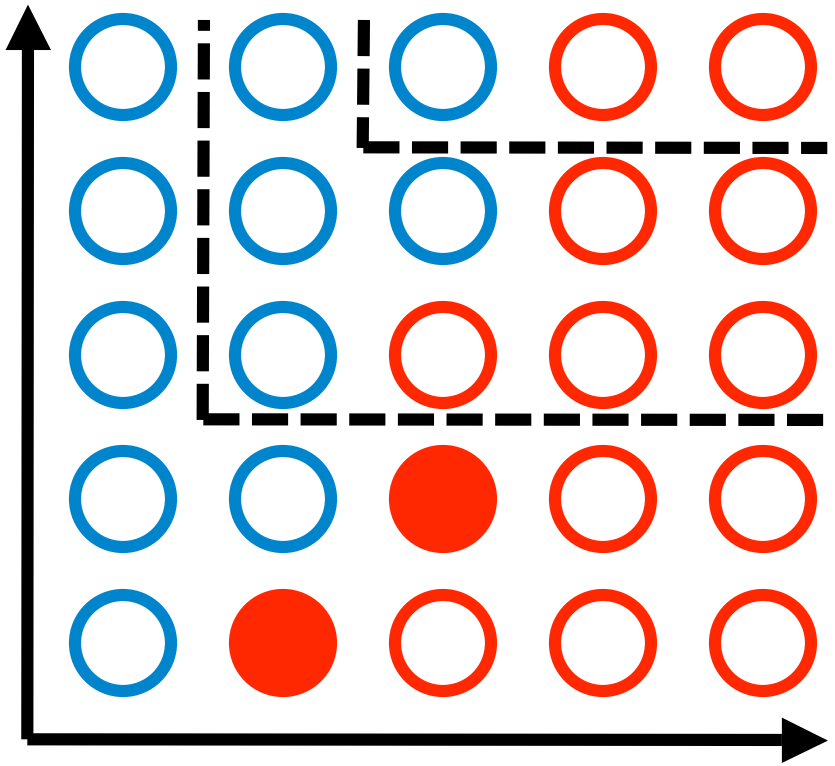
Current States

$$succ(s,a) = \{\{w(x + t\delta_a) \mid w(x) \in s, \pi(w(x)) = a, t \in [\alpha_a, \beta_a]\}\}$$



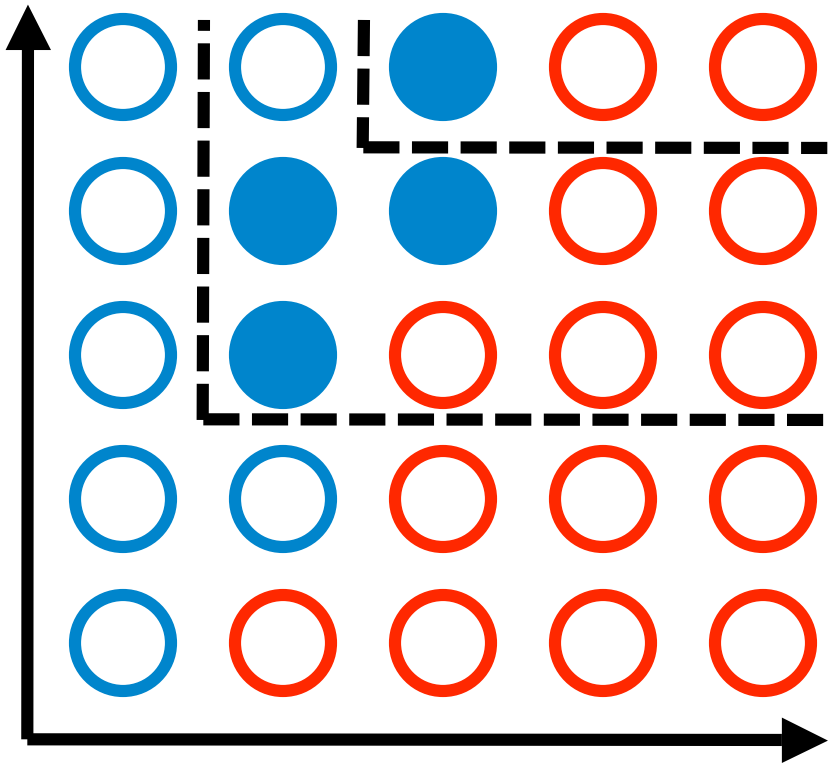
Split on Policy (Red vs Blue)

$$succ(s,a) = \{\{w(x + t\delta_a) \mid w(x) \in s, \pi(w(x)) = a, t \in [\alpha_a, \beta_a]\}\}$$



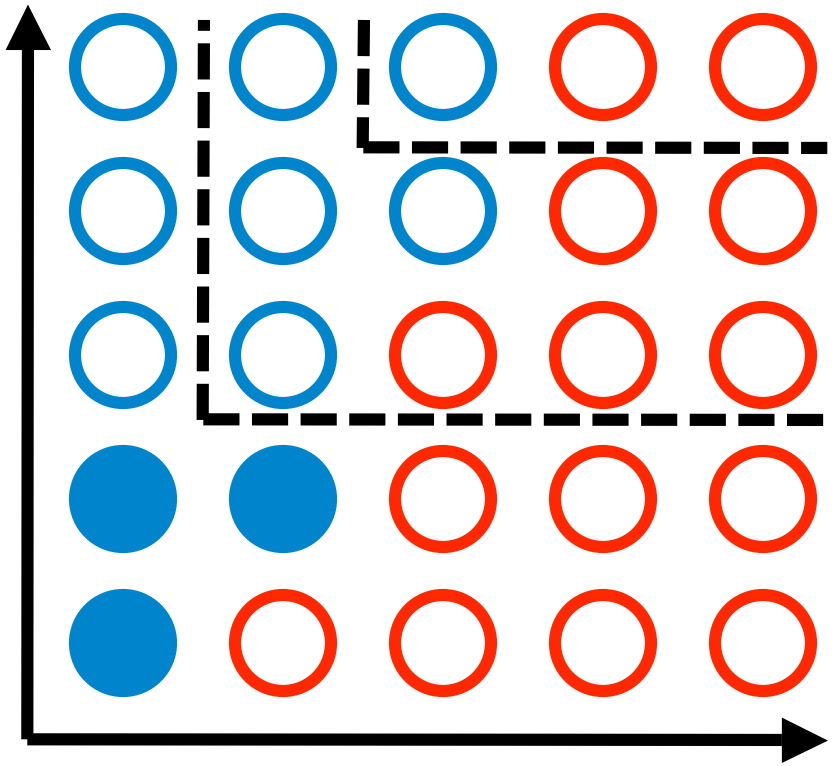
Simulate Action

$$succ(s,a) = \{\{w(x + t\delta_a) \mid w(x) \in s, \pi(w(x)) = a, t \in [\alpha_a, \beta_a]\}\}$$



Apply Wrapping

$$succ(s,a) = \{\{w(x + t\delta_a) \mid w(x) \in s, \pi(w(x)) = a, t \in [\alpha_a, \beta_a]\}\}$$



VI. Towards (Timed) Verification

- Computing successor states is similar to initial approach …
  - … but involves wrapped state space model
- We can also compute difference bound matrices …
  - … and define run and wrap operations over them
  - A basis for timed verification of scheduling properties
- There remain a number of scalability/accuracy trade-offs and other open issues (most recent results under review)
  - E.g., we can represent either unknown modes or unknown distributions (but not yet both at once)
  - State spaces are still very large but polynomial time approximations appear possible and useful (current work in progress)

Support for Model Checking

- Some definitions/assumptions
  - » Each verification state is a set of utilization states
  - » Actions are the same as in the MDP
  - » Initial verification state contains only the utilization state representing zero CPU usage for each thread
  - » We have a successor function over state-action pairs
- Given a model and a property, check whether the property holds in the model
- Build a transition system of reachable states
  - » Repeatedly uses successor operator *succ(s,a)*, to compute the successor state(s) to all previously visited states *s*, with all possible actions *a*, until a fixed point is reached

Potential Improvements to our Representation

- The size of a verification state is proportional to the size of the wrapped utilization state space
  - » Still exponential in the number of threads
- The size of the verification state is also sensitive to the time scale used
  - » The more densely sampled, the larger the representation
- We would prefer a verification state representation that is less affected by these factors

Difference Bound Matrix (DBM) Representation

- Used by timed model checkers
- Compactly represent continuous regions in utilization space
- Each element represents an equation
  - » e.g. $x_1 - x_2 < 3$
- *Quadratic in the number of threads*
- Independent of the time scale

	0	x_1	x_2
0	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$
x_1	$(\infty, \leq)$	$(0, \leq)$	$(\infty, \leq)$
x_2	$(\infty, \leq)$	$(3, <)$	$(0, \leq)$

The Successor Function (Revisited)

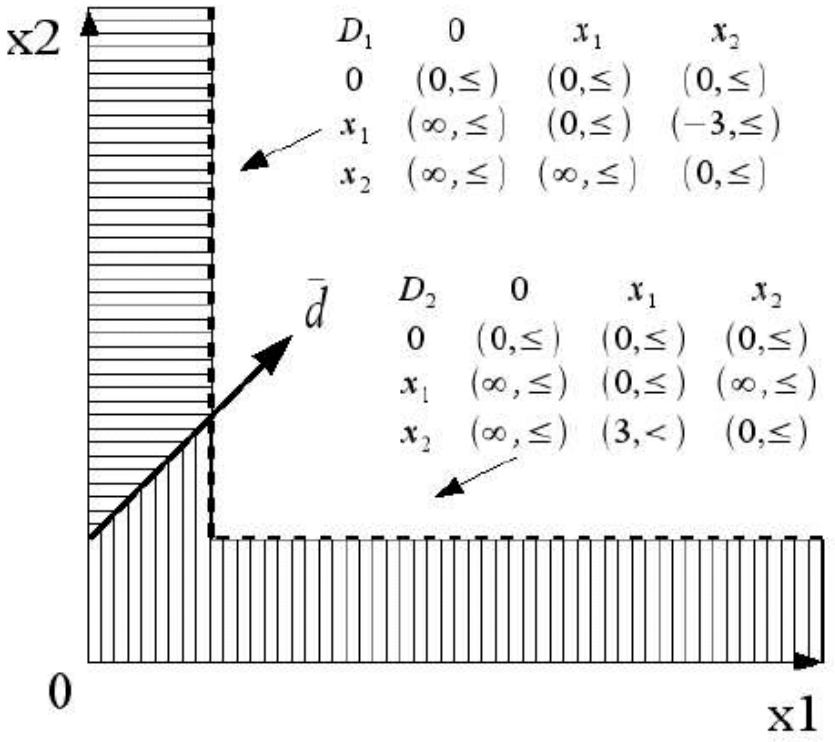
$$succ(s,a) = \{\{w(x + t\delta_a) \mid w(x) \in s, \pi(w(x)) = a, t \in [\alpha_a, \beta_a]\}\}$$

- Successor Function Again Computed in Steps
  - » Split on Policy
  - » Simulate Action
  - » Wrap
- This time we use three DBM operations
  - » Intersection -- canonical DBM operation
  - » Run and wrap -- novel extensions

Split on Policy

$$succ(s,a) = \{\{w(x + t\delta_a) \mid w(x) \in s, \pi(w(x)) = a, t \in [\alpha_a, \beta_a]\}\}$$

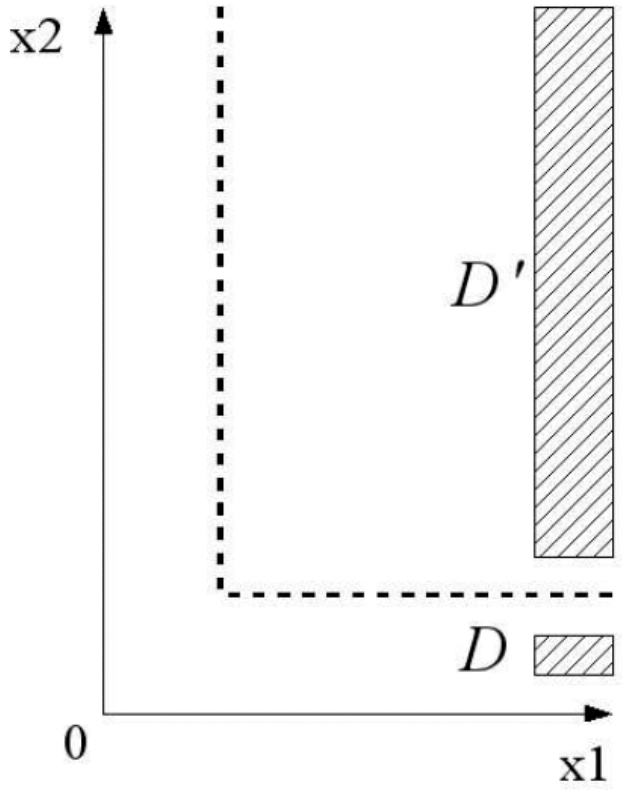
- First, represent the decision boundary as a DBM
- Then *intersect* the DBM representing the verification state with the DBM for the appropriate decision region



Simulate Action

$$succ(s,a) = \{\{w(x + t\delta_a) \mid w(x) \in s, \pi(w(x)) = a, t \in [\alpha_a, \beta_a]\}\}$$

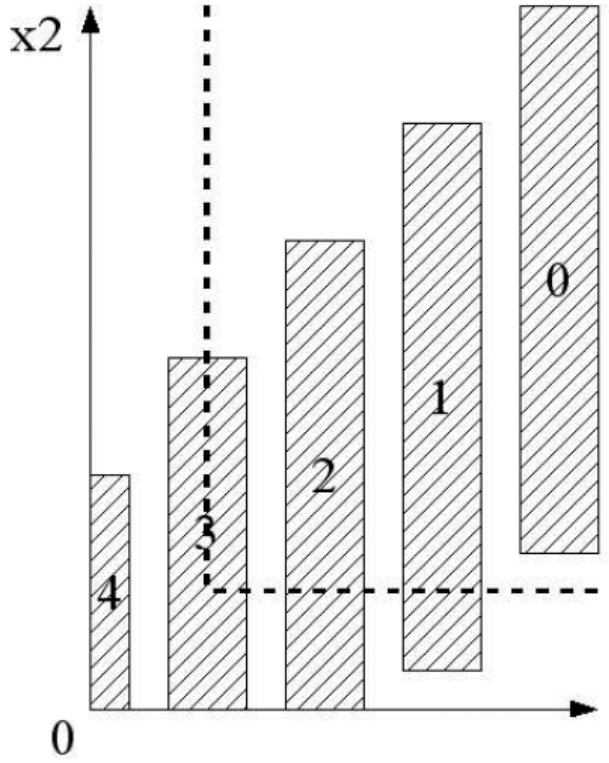
- Apply *Run* operator
  - » Takes a DBM as input
  - » Creates a DBM representing the region that would be produced by running the appropriate thread



Apply Wrapping (1/3)

$$succ(s,a) = \{\{w(x + t\delta_a) \mid w(x) \in s, \pi(w(x)) = a, t \in [\alpha_a, \beta_a]\}\}$$

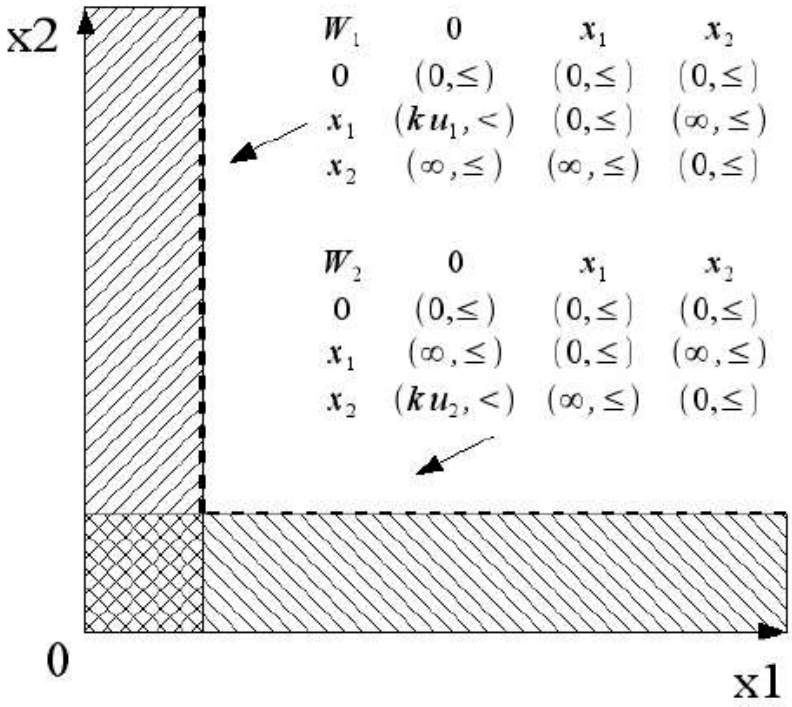
- *Wrap* operator takes a DBM and a non-negative integer
- Invoke repeatedly for successively larger integers until it produces an empty DBM (fixed point)



Apply Wrapping (2/3)

$$succ(s,a) = \{\{w(x + t\delta_a) \mid w(x) \in s, \pi(w(x)) = a, t \in [\alpha_a, \beta_a]\}\}$$

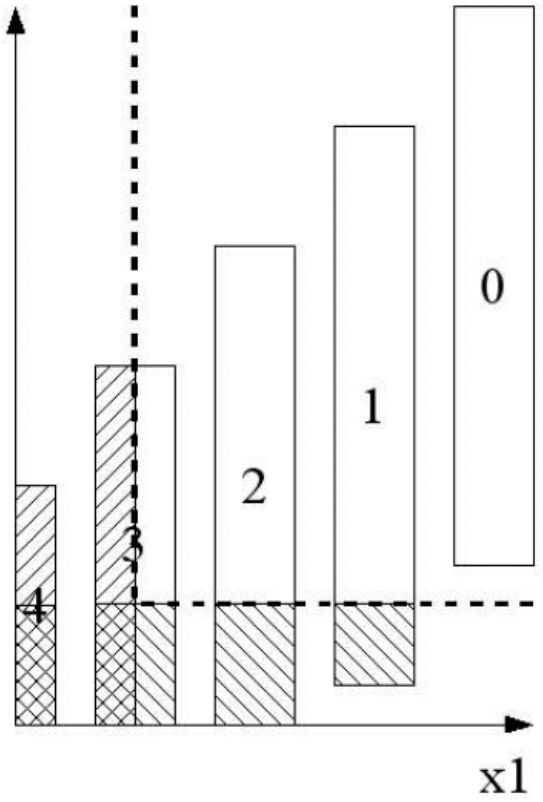
- Confine the DBMs produced by the *Wrap* operator
- Need a DBM per thread to represent the wrapped space



Apply Wrapping (3/3)

$$succ(s,a) = \{\{w(x + t\delta_a) \mid w(x) \in s, \pi(w(x)) = a, t \in [\alpha_a, \beta_a]\}\}$$

- *Intersect* the products of *Wrap* with the constraining DBMs
  - » Gives the set *succ(s,a)*



Now We Can Model Check the Scheduling Policy

- The successor function induces a transition system that models our scheduling policy
- DBM representation is efficient and checkable
- Apply standard model checking tools and techniques to verify properties with this model

Summary

- Our goal has been to design assured scheduling policies for open soft real-time systems
- We use a Markov Decision Process representation to derive scheduling policies
  - » We developed a state wrapping scheme to allow policy iteration
- We exploit state space structure and introduce novel DBM operators to constrain verification complexity
  - » Allows tractable property verification via Model Checking

A Few More Important Pieces of Related Work

- Reference monitor approaches
  - Interposition architectures
    - E.g., Ostia: user/kernel-level (Garfinkel et al.)
  - Separation kernels
    - E.g., ARINC-653, MILS (Vanfleet et al.)
- Scheduling policy design
  - Hierarchical scheduling
    - E.g., HLS and its extensions (Regehr et al.)
    - E.g., Group scheduling (Niehaus et al.)
  - Quasi-cyclic state space reduction
    - E.g., Bogor (Robby et al.)

Concluding Remarks

- MDP approach supports rational scheduling design
  - Even when thread run times vary stochastically
  - Encodes rather than presupposes utilizations
  - Allows policy verification over utilization states
- Ongoing and Future Work
  - Dealing with unknown modes/distributions
  - Polynomial time approximations
  - Empirical evaluation in real-world settings
- Relevant project web pages
  - http://www.cse.wustl.edu/~cdgill/Cybertrust
    - Supported by NSF grant CNS-0716764
  - http://www.cse.wustl.edu/~cdgill/CAREER
    - Supported by NSF grant CCF-0448562