ICAPS 2012 Tutorial

Recent Advances in Continuous Planning

Scott Sanner

NICTA

THE AUSTRALIAN NATIONAL UNIVERSITY

Tutorial Outline

1. Modeling Continuous Problems
   a) Why continuous?
   b) MDPs and POMDPs
   c) (P)PDDL and RDDL

2. Solving Continuous Problems
   a) Exact dynamic programming
      - Data structures
   b) Open problems
   c) Survey of other solution methods
   d) Connections to control and scheduling

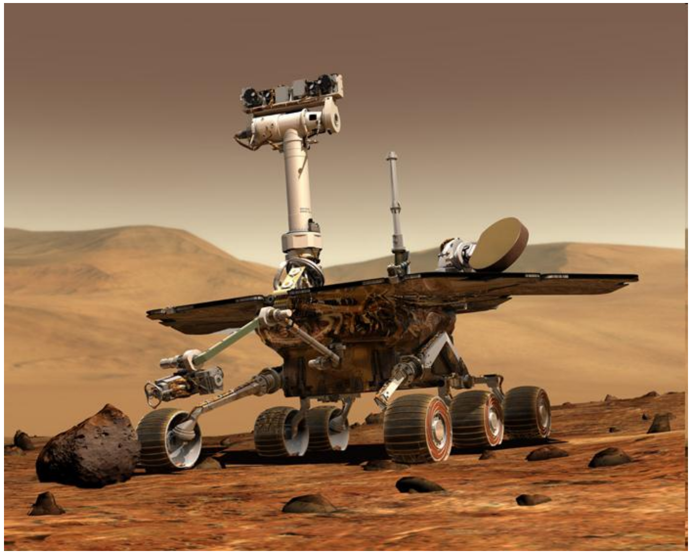
Part 1a: Modeling

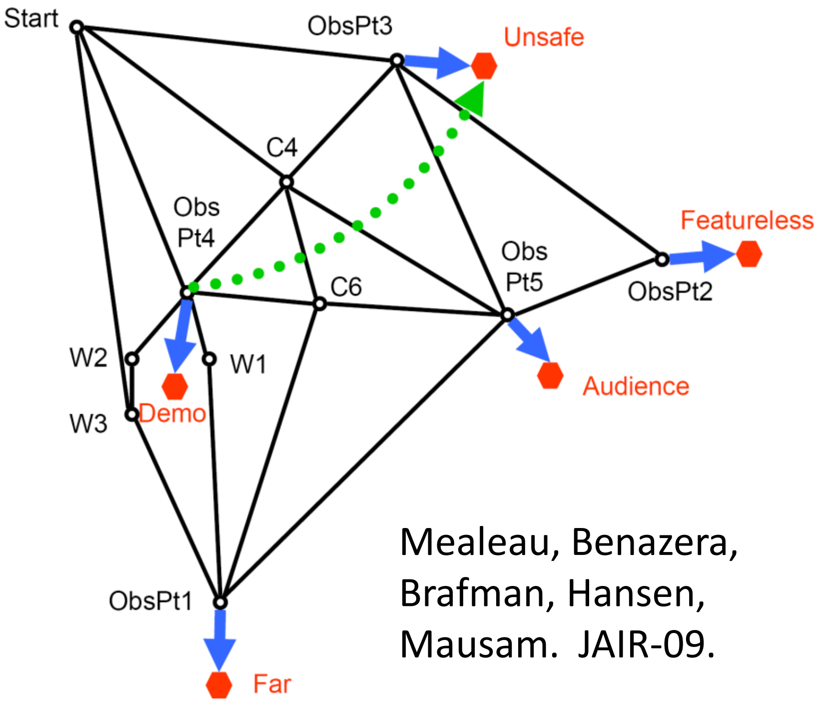
Why continuous?

Why Continuous Planning?

- Many real-world problems have a **continuous** component of **state, action**, or **observations**
  - **Time**
  - **Space and derivatives**
    - Position and angle
    - Velocity, acceleration, …
  - **Resources**
    - Fuel, energy, …
  - **Expected statistics**
    - Traffic volume
    - Density, speed, …

Mars Rovers





Mealeau, Benazera, Brafman, Hansen, Mausam. JAIR-09.

- **Objective?**
  - Carry out actions near places within time windows
- **What's continuous?**
  - Time (t), Energy (e), Robot position (x,y,θ)

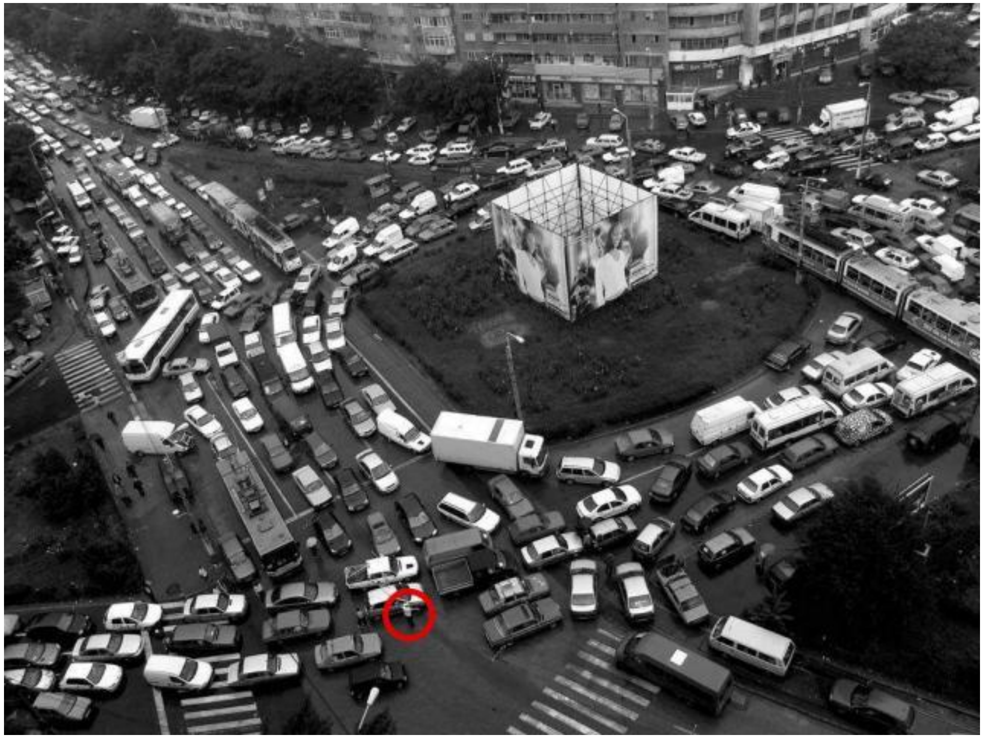
Elevator Control

- **Dynamics**
  - Random arrivals (e.g., Poisson)
- **Objective?**
  - Minimize sum of wait times
- **What's continuous?**
  - Expected people waiting
    - At each floor
    - In elevator
  - Expected time waiting
    - At each floor
    - For each level in elevator

DARPA Grand Challenge

- **Autonomous driving**
  - Real-time, partially observed
- **Objective?**
  - Reach goal, stay on road (wherever it is)
- **What’s continuous?**
  - State: position, velocity
  - Sensing: vision, sonar, laser range finders

Traffic Control



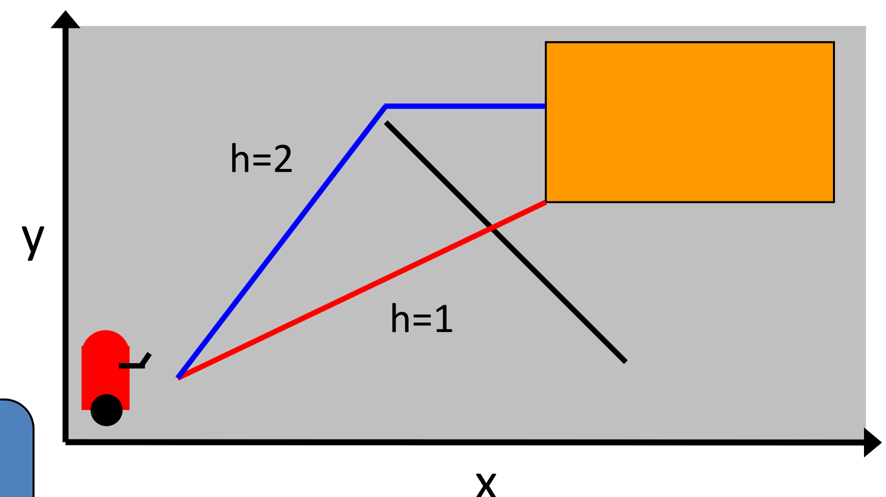
- **Objective?**
  - Minimize congestion, stops, fuel consumption
- **What's continuous?**
  - Expected traffic volume, velocity, wait times

Goal-oriented Path Planning

- **Robotics**
  - Continuous position, joint angles
  - Nonlinear dynamics (sin, cos)

- **Obstacle Navigation**
  - 2D, 3D, 4D (time)
  - Linear dynamics
  - Don't discretize!
    - ~~Grid worlds~~

Solve in 2 steps!



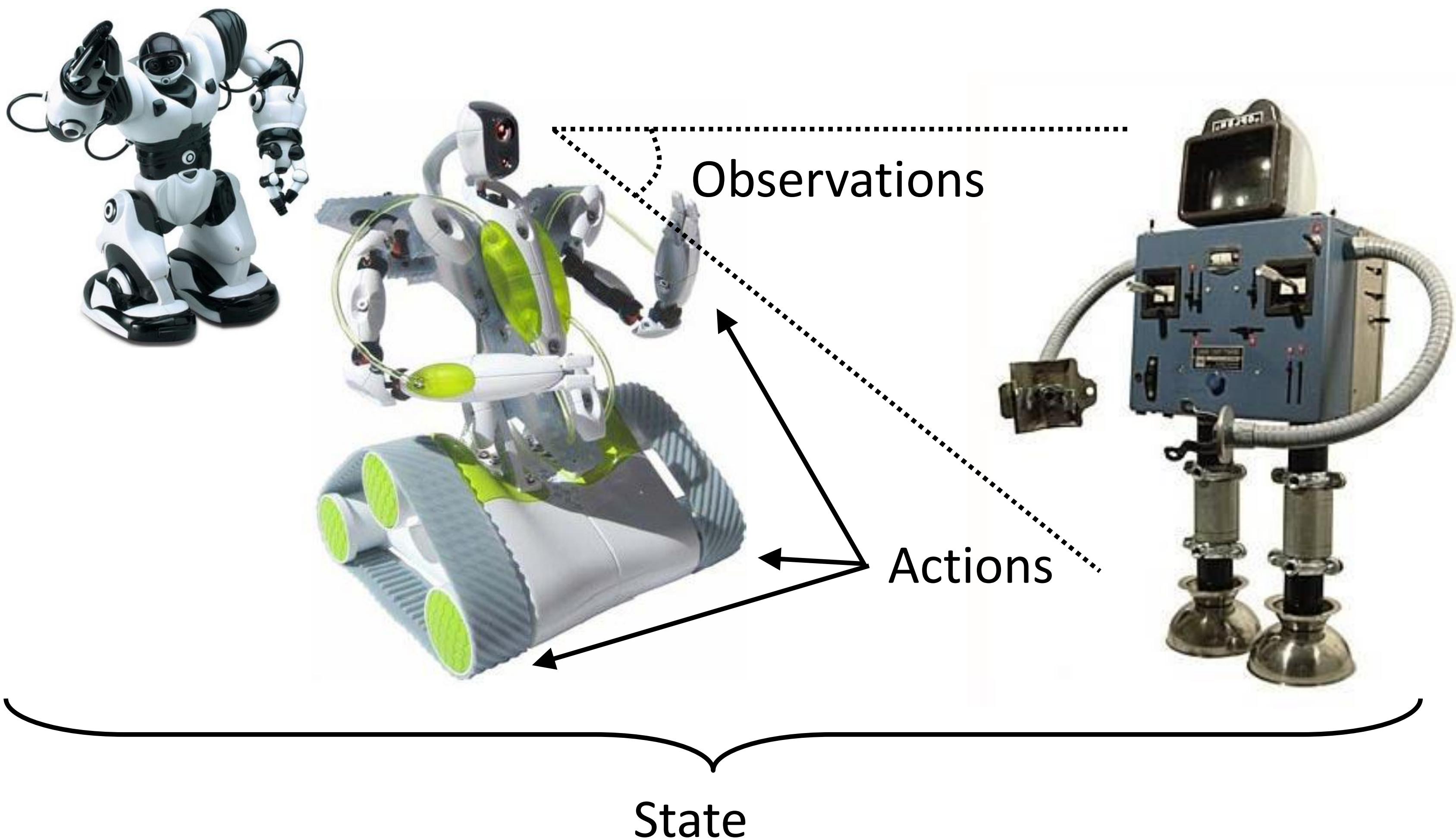
If you can effectively solve any of the previous problems: people will care

But first you have to model them!

Part 1b: Modeling

MDPs and POMDPs

Observations, States, & Actions



Observations

Continuous observations!

- **Observation set O**
  - Perceptions, e.g.,
    - My opponent's bet in Poker
    - Distance from car to edge of road

States

Continuous states!

- **State set S**
  - At any point in time, system is in some state
    - My opponent's hand of cards in Poker
    - Actual distance to edge of road

Agent Actions

Continuous actions!

- **Action set A**
  - Actions could be *concurrent*
  - If k actions, $\mathbf{A} = \mathbf{A_1} \times \ldots \times \mathbf{A_k}$
    - Schedule all deliveries to be made at 10am
    - Set multiple joint angles in robotics

Agent Actions

- **Action set A**
  - All actions need not be under agent control
    - Other agents, e.g.,
      - Alternating turns: Poker, Othello
      - Concurrent turns: Highway Driving, Soccer
    - *Exogenous events* due to *Nature*, e.g.,
      - Random arrival of person waiting for elevator
      - Random failure of equipment

Recap

- So far
  - States (S)
  - Actions (A)
  - Observations (O)

- How to map between
  - Previous states, actions, and future states?
  - States and observations?
  - States, actions and rewards?
  - Sequences of rewards and optimization criteria?

Observation Function

- How to relate states and observations?
  - ***Partially observable***:
    - – Observations provide a belief over possible states
    - – The most realistic world model
      - » E.g., Driving
    - – Solution techniques highly non-trivial
      - » Beyond the scope of this introductory tutorial
  - ***Fully observable***:
    - – **S** $\leftrightarrow$ **O** … the case we focus on!
    - – Assume complete knowledge of state
      - » Inventory Control
    - – Usually OK for “almost fully observable”, e.g.,
      - » Traffic, Path Planning, Elevators, Mars Rover

Transition Function

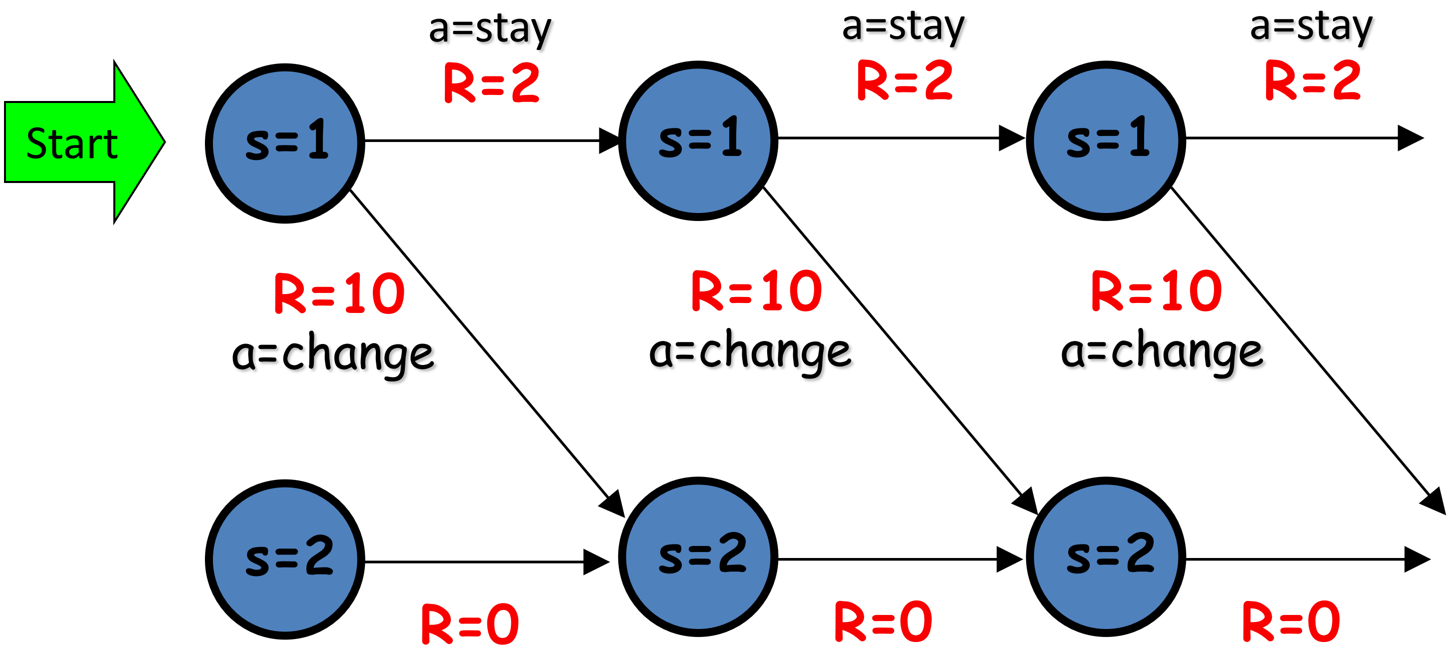
- How do actions take us between states?
  - Some properties
    - *Stationary*: **T** does not change over time
      - » e.g., cannot be controlled adversarial agent
    - *Markovian*:
      - Next state dependent only upon previous state / action
      - If not Markovian, can always augment state description
        - » e.g., elevator traffic model differs throughout day; so encode time in state to make **T** Markovian!

Goals and Rewards

- Goal-oriented rewards
  - Assume maximizing reward…
  - Assign any reward value s.t. $R(success) > R(fail)$
  - Can have negative costs $C(a)$ for action a
- What if multiple (or no) goals?
  - How to specify preferences?
  - R assigns **utilities** to states and actions
    - *E.g., Continuous:* $R(x,y,a) = x^2 + xy$
    - Then *maximize expected reward (utility)*

But, how to trade off rewards over time?

Optimization: Best Action when s=1?



- Must define objective criterion to optimize!
  - How to trade off immediate vs. future reward?
  - E.g., use discount factor γ (try γ=.9 vs. γ=.1)

Trading Off Sequential Rewards

- Sequential-decision making objective
  - Horizon (h)
    - *Finite*: Only care about h-steps into future
    - Infinite: Literally; will act same today as tomorrow
  - How to trade off reward over time?
    - *Expected average cumulative return*
    - *Expected discounted cumulative return*
      - Use discount factor γ
      - Reward t time steps in future discounted by γ^t

Recap

- So far
  - Actions (A)
  - States (S)
  - Observation (O)
  - Transition function (T)
  - Observation function (Z)
  - Reward function (R)
  - Optimization criteria
- But are the above
  - Known or unknown?

Knowledge of Environment

- ***Model-known*:**
  - Know <S,A,T,R> and if partially observed, also <O,Z>
  - Called: **Planning (under uncertainty)** **[Focus of this tutorial]**
    - *Decision-theoretic* planning if maximizing expected utility
- ***Model-free*:**
  - $\geq$1 unknown: <S,A,T,R> and if partially observed, also <O,Z>
  - Called: **Reinforcement learning**
    - Have to *interact with environment to obtain samples*
- **Model-based:**
  - Between model-known and model-free
  - Learn approximate model from samples
  - Permits hybrid planning and learning

Saves expensive interaction!

Important part of AI that is overlooked… learning **relevant** model!

Finally a Formal Model

- **Two main model types:**
  - **MDP:** $\langle$ **S, A, T, R** $\rangle$
  - **POMDP:** $\langle$ **S, A, O, Z, T, R** $\rangle$
  - **Model Known?**
    - **Yes:** (decision-theoretic) planning under uncertainty
    - **No:** reinforcement learning (model-free or model-based)

- **Cannot solve a problem until know objective!**
  - **Single agent (possibly concurrent)**
    - Maximize expected average or discounted sum of rewards

  - **Multi-agent**
    - Solution criteria depends on
      - Alternating vs. concurrent
      - Zero sum vs. general sum
    - Beyond scope of this tutorial

Part 1c: Modeling

(P)PDDL and RDDL

(P)PDDL

Relational
Effects-based Model
for Single Agent MDPs

PDDL – Predicate and Functional Fluents

```
(define (domain test-domain)
  (:requirements :typing :equality :conditional-effects :fluents)
  (:types car box)

  (:predicates (parked ?x - car) (holding ?x - box)
               (in ?x - box ?y - car))
  (:functions (fuel-level ?x - car))

  (:action load  :parameters (?x - box ?y - car)
   :precondition (and (holding ?x) (parked ?y))
   :effect (and (in ?x ?y)
             (forall (?z - car)
               (when (not (= ?z ?y))
                 (not (in ?x ?z))))))

  (:action refuel   :parameters (?x - car)
   :precondition (< (fuel-level ?x) 10)
   :effect (increase (fuel-level ?x) 1)))
```

Boolean and numeric fluents

Boolean fluent action effects

Continuous fluent action effects

Ex. from Younes and Littman, PPDDL 1.0

Probabilistic PDDL – PPDDL

```
(define (domain test-domain)
  (:requirements :typing :equality :conditional-effects :fluents)
  (:types car box)

  (:predicates (parked ?x - car) (holding ?x - box)
               (in ?x - box ?y - car))
  (:functions (fuel-level ?x - car))

  (:action load  :parameters (?x - box ?y - car)
   :precondition (and (holding ?x) (parked ?y))
   :effect (probabilistic 0.7
            (and (in ?x ?y)
              (forall (?z - car)
                (when (not (= ?z ?y))
                  (not (in ?x ?z)))))))

  (:action refuel   :parameters (?x - car)
   :precondition (< (fuel-level ?x) 10)
   :effect (probabilistic 0.3 (increase (fuel-level ?x) 1)
                          0.5 (decrease (fuel-level ?x) 1))))
```

Probabilistic effects

- In absence of effect, assume no change
- Assume effects are consistent (no conflicing assignments)

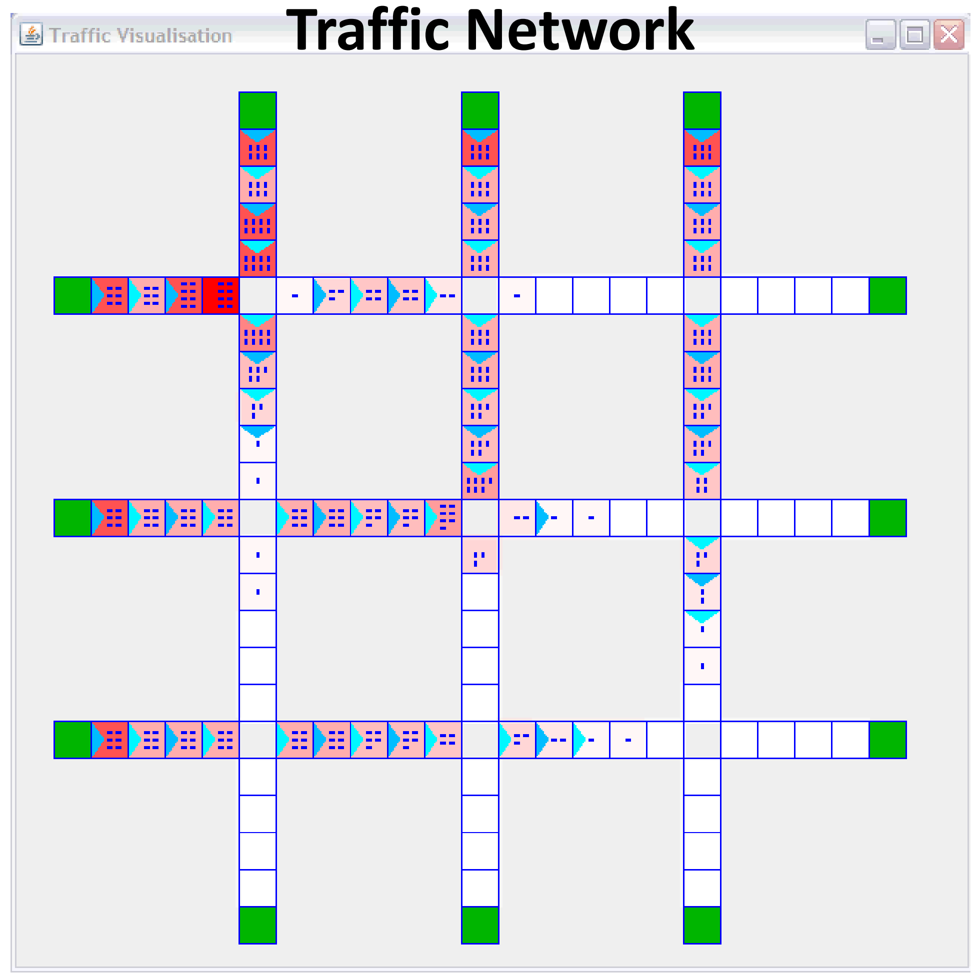
What's missing in PPDDL, Part I

- Continuous effects-based modeling is natural:
  - Can use arithmetic functions for numeric fluent updates
  - But
    - Little provision for state-dependent probabilities
- Multiple Independent Exogenous Events:
  - PPDDL only allows 1 independent event to affect fluent
    - In a stochastic setting, what if cars in a queue change lanes, or brake randomly?

Looking ahead… will need something more like Relational DBN

What’s missing in PPDDL, Part II

- Expressive transition distributions:
  - Stochastic difference equations with arbitrary noise
    - Poisson arrivals
    - Gaussian noise
  - Resolving conflicts of concurrent actions under exogenous events
    - Unprotected traffic turns
- Partial observability:
  - E.g., only observe stopline



What’s missing in PPDDL, Part III

- Distinguish fluents from nonfluents:
  - E.g., topology of traffic network
  - Lifted planners must know this to be efficient!
- Expressive rewards
  - E.g., sums and products over all objects!
  - Function of state (e.g., SysAdmin)
- Global state-action constraints for domain verification:
  - Concurrent domains need *global action* preconditions
    - E.g., two traffic lights cannot go into a given state
  - In logistics, vehicles cannot be in two different locations
    - Regression planners need state constraints!

Is there any hope?

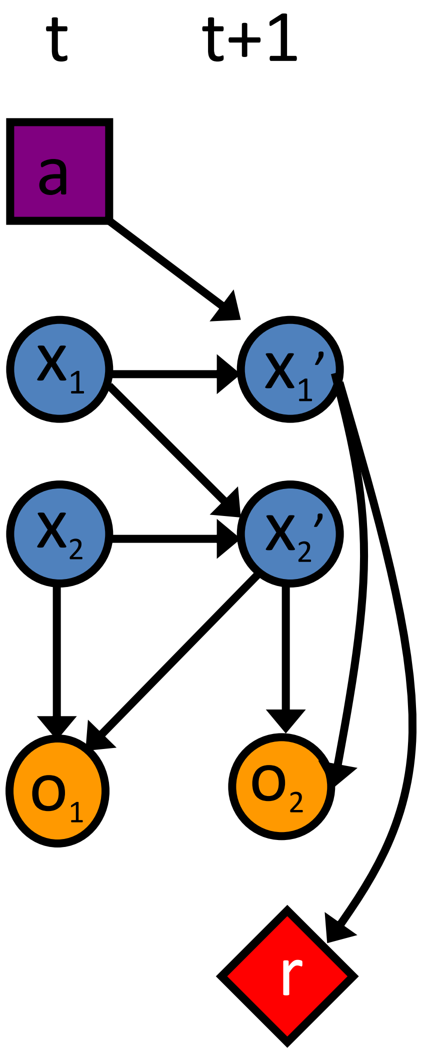
Yes, but we need to borrow from factored MDP / POMDP community…

RDDL

Relational Fluent-oriented Model for Single Agent, Concurrent Action (PO)MDPs

What is RDDL?

- Relational Dynamic Influence Diagram Language
  - Relational [DBN + Influence Diagram]
  - State, action, observations, reward are all variables (fluents)
    - Variables depend on parents in diagram
- Think of it as Relational Factored MDPs and POMDPs
  - SPUDD / Symbolic Perseus



RDDL Principles I

- Everything is a fluent (parameterized variable)
  - State fluents
  - Observation fluents
    - for **partially observed** domains
  - Action fluents
    - supports **factored concurrency**
  - Intermediate fluents
    - **derived predicates, correlated effects**, …
  - **Constant nonfluents** (general constants, topology relations, …)

Regression planners need to know what fluents do not change!

- Flexible fluent types
  - **Binary** (predicate) fluents
  - **Multi-valued** (enumerated) fluents
  - **Integer and continuous** fluents (from PDDL 2.1)

RDDL Principles II

- Semantics is ground DBN / Influence Diagram
  - **DBN** leads to consistent transition semantics
    - Supports **unrestricted concurrency**
      - i.e., concurrent actions may conflict
      - DBN transitions inherently resolve these conflicts
  - Naturally supports **independent exogenous events**
    - E.g., each car in traffic moving autonomously
      - random braking
      - random lane changes

RDDL Principles III

- Expressive transition and rewards
  - Logical expressions ($\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\forall$, $\exists$)
  - Arithmetic expressions (+,−,*,/)
  - In/dis/equality comparison expressions (=, $\neq$, <,>, $\leq$, $\geq$)
  - Conditional expressions (if-then-else, **switch**)
  - **Sum and product over all domain objects: $\sum_x$, $\prod_x$**
  - **General probability distributions**
    - **Bernoulli**
    - **Discrete**
    - **Normal**
    - **Poisson**
    - **Exponential**
    - **…**

Parameters can be function of state and action!

$\sum_x$, $\prod_x$ aggregators over domain objects extremely powerful

RDDL Principles IV

- **Arbitrary state/action constraints**
  - **Joint action preconditions**
    - e.g., two lights cannot be green if they allow crossing traffic
  - **State invariant assertions**
    - e.g., cars can neither be created nor destroyed
    - e.g., a package cannot be in two locations

Interesting problems for ICKEPS community:

- How to generate conflicts?
- Correct domain when conflict arises?
- Correct when solutions don't display expected properties?

Many possible states are illegal –
Important to identify for regression planning

RDDL Principles V

- Goal + General (PO)MDP objectives
  - Arbitrary reward
    - goals, costs, numerical preferences (c.f., PDDL 3.0)
  - Finite horizon
  - Discounted or undiscounted

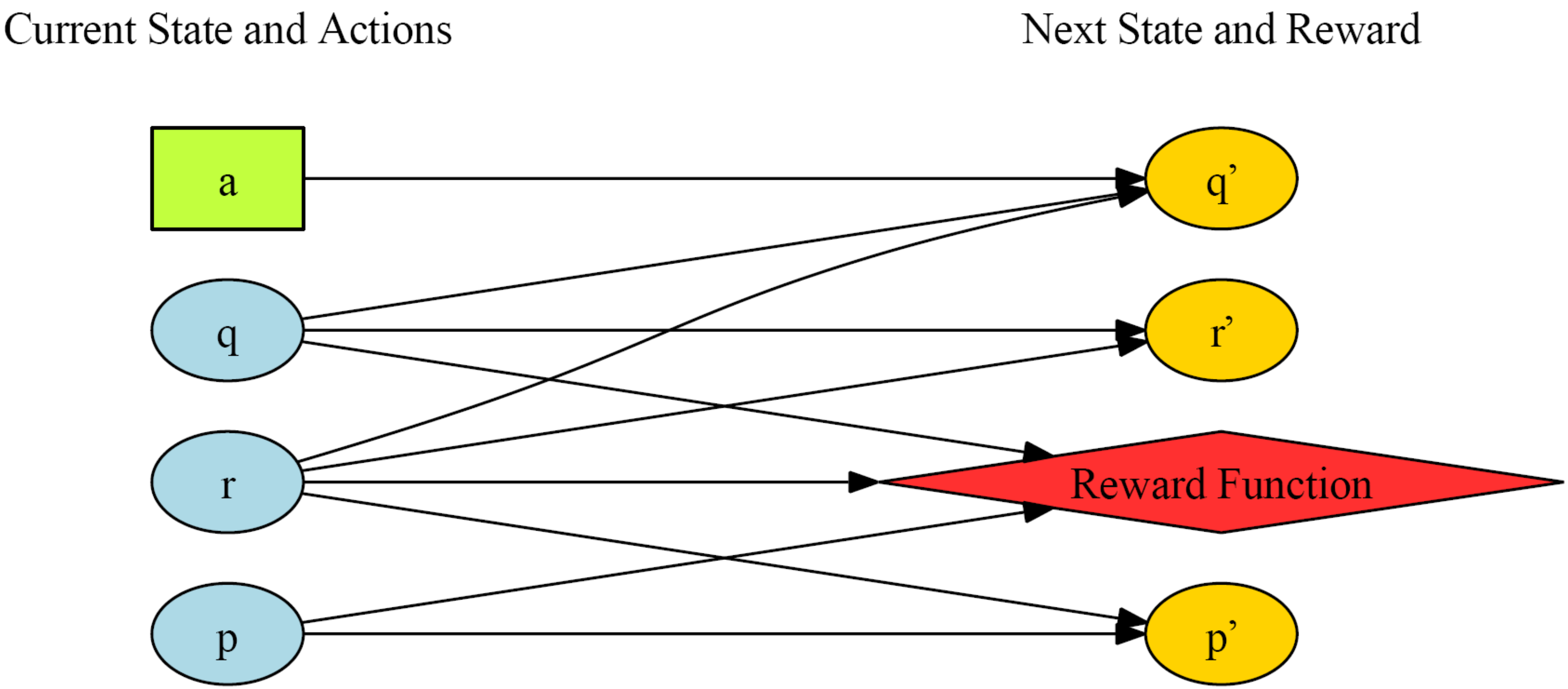
Can use Σ_x, Π_x aggregators here…

e.g., sum of all delivery costs for all packages

RDDL Examples

Easiest to understand
RDDL in use…

How to Represent Factored MDP?



p	r	p'	$P(p'\|p,r)$
true	*true*	*true*	0.9
true	*true*	*false*	0.1
true	*false*	*true*	0.3
true	*false*	*false*	0.7
false	*true*	*true*	0.3
false	*true*	*false*	0.7
false	*false*	*true*	0.3
false	*false*	*false*	0.7

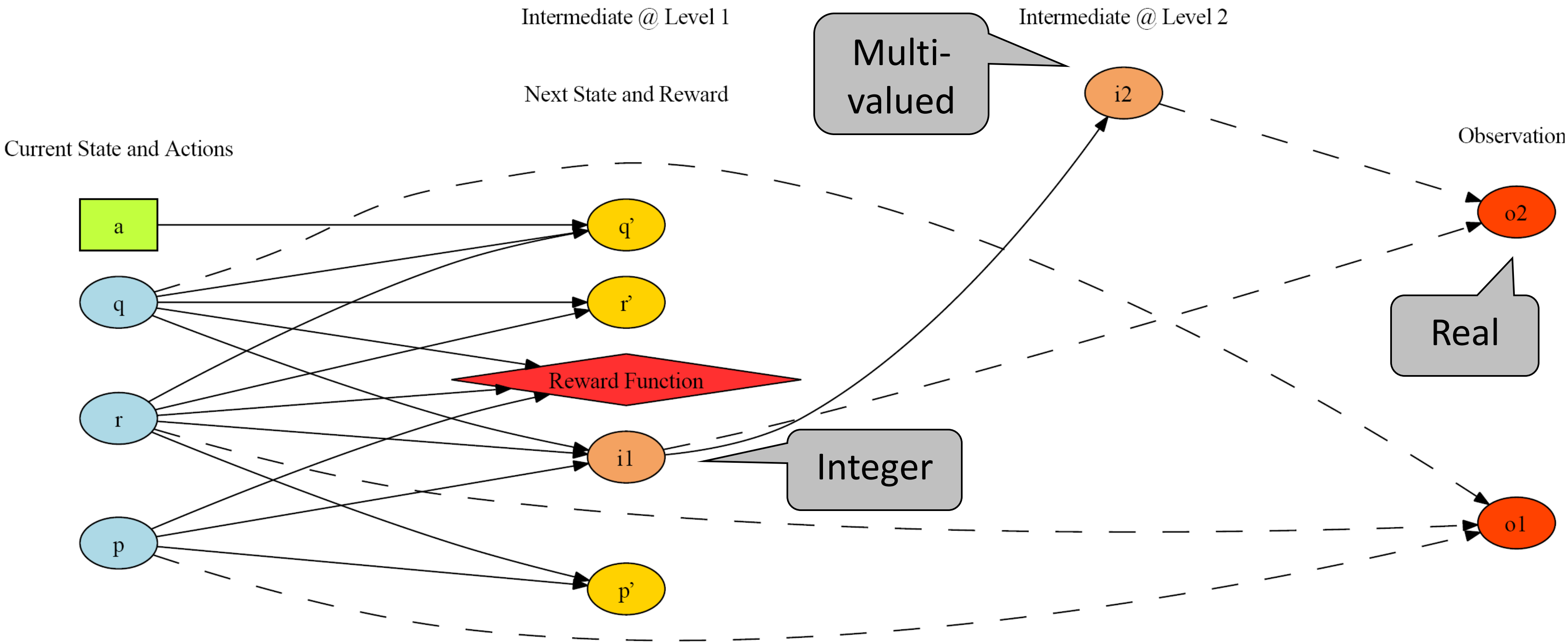
RDDL Equivalent

```
// Define the state and action variables (not parameterized here)
pvariables {
    p : { state-fluent,  bool, default = false };
    q : { state-fluent,  bool, default = false };
    r : { state-fluent,  bool, default = false };
    a : { action-fluent, bool, default = false };
};

// Define the conditional probability function for each
// state variable in terms of previous state and actio
cpfs {
    p' = if (p ^ r) then Bernoulli(.9) else Bernoulli(.3);

    q' = if (q ^ r) then Bernoulli(.9)
                    else if (a) then Bernoulli(.3) else Bernoulli(.8);

    r' = if (~q) then KronDelta(r) else KronDelta(r <=> q);
};

// Define the reward fu          boolean functions are
// treated as 0/1 integ          expressions
reward = p + q - r;
```

Can think of transition distributions as *“sampling instructions”*

Boolean variables are {0,1} so can sum

A Discrete-Continuous POMDP?



A Discrete-Continuous POMDP, Part I

```
// User-defined types
types {
    enum_level : {@low, @medium, @high}; // An enumerated type
};

pvariables {
    p : { state-fluent,  bool, default = false };
    q : { state-fluent,  bool, default = false };
    r : { state-fluent,  bool, default = false };

    i1 : { interm-fluent, int,        level = 1 };
    i2 : { interm-fluent, enum_level, level = 2 };

    o1 : { observ-fluent, bool };
    o2 : { observ-fluent, real };

    a : { action-fluent, bool, default = false };
};

cpfs {

    // Some standard Bernoulli conditional probability tables
    p' = if (p ^ r) then Bernoulli(.9) else Bernoulli(.3);

    q' = if (q ^ r) then Bernoulli(.9)
                    else if (a) then Bernoulli(.3) else Bernoulli(.8);

    // KronDelta is a delta function for a discrete argument
    r' = if (~q) then KronDelta(r) else KronDelta(r <=> q);
```

Intermediate variables – correlated effects, derived predicates

Observation variables

A Discrete-Continuous POMDP, Part II

Integer

```
// Just set i1 to a count of true state variables
i1 = KronDelta(p + q + r);

// Choose a level with given probabilities that sum to 1
i2 = Discrete(enum_level,
                @low : if (i1 >= 2) then 0.5 else 0.2,
                @medium : if (i1 >= 2) then 0.2 else 0.5,
                @high : 0.3
            );

// Note: Bernoulli parameter must be in [0,1]
o1 = Bernoulli( (p + q + r)/3.0 );

// Conditional linear stochastic equation
o2 = switch (i2) {
        case @low    : i1 + 1.0 + Normal(0.0, i1*i1),
        case @medium : i1 + 2.0 + Normal(0.0, i1*i1/2.0),
        case @high   : i1 + 3.0 + Normal(0.0, i1*i1/4.0) };
    };
```

Multi-valued

Real

Mixture of Normals

Variance comes from other previously sampled variables

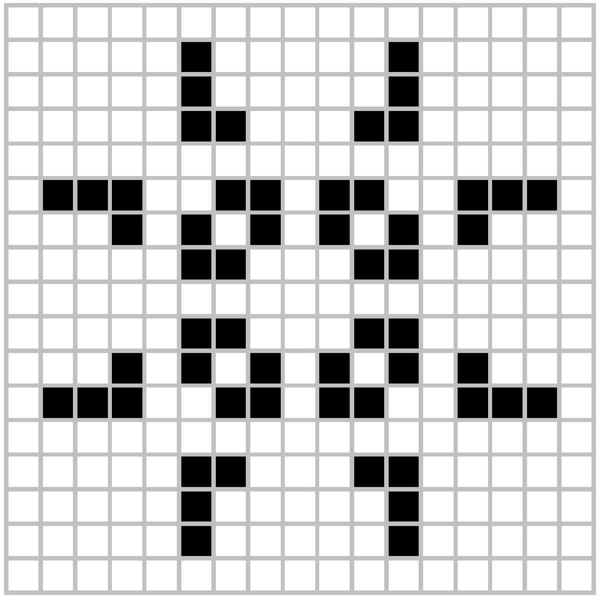
RDDL so far…

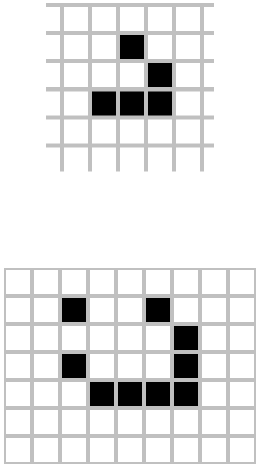
- Mainly SPUDD / Symbolic Perseus with a different syntax ☺
  - A few enhancements
    - concurrency
    - constraints
    - integer / continuous variables
- Real problems (e.g., traffic) **need lifting**
  - An intersection model
  - A vehicle model
    - Specify each intersection / vehicle model once!

Lifting: Conway's Game of Life

(simpler than traffic)

- Cells born, live, die based on neighbors
  - < 2 or > 3 neighbors: cell dies
  - 2 or 3 neighbors: cell lives
  - 3 neighbors → cell birth!
  - Make into MDP
    - Probabilities
    - Actions to turn on cells
    - Maximize number of cells on





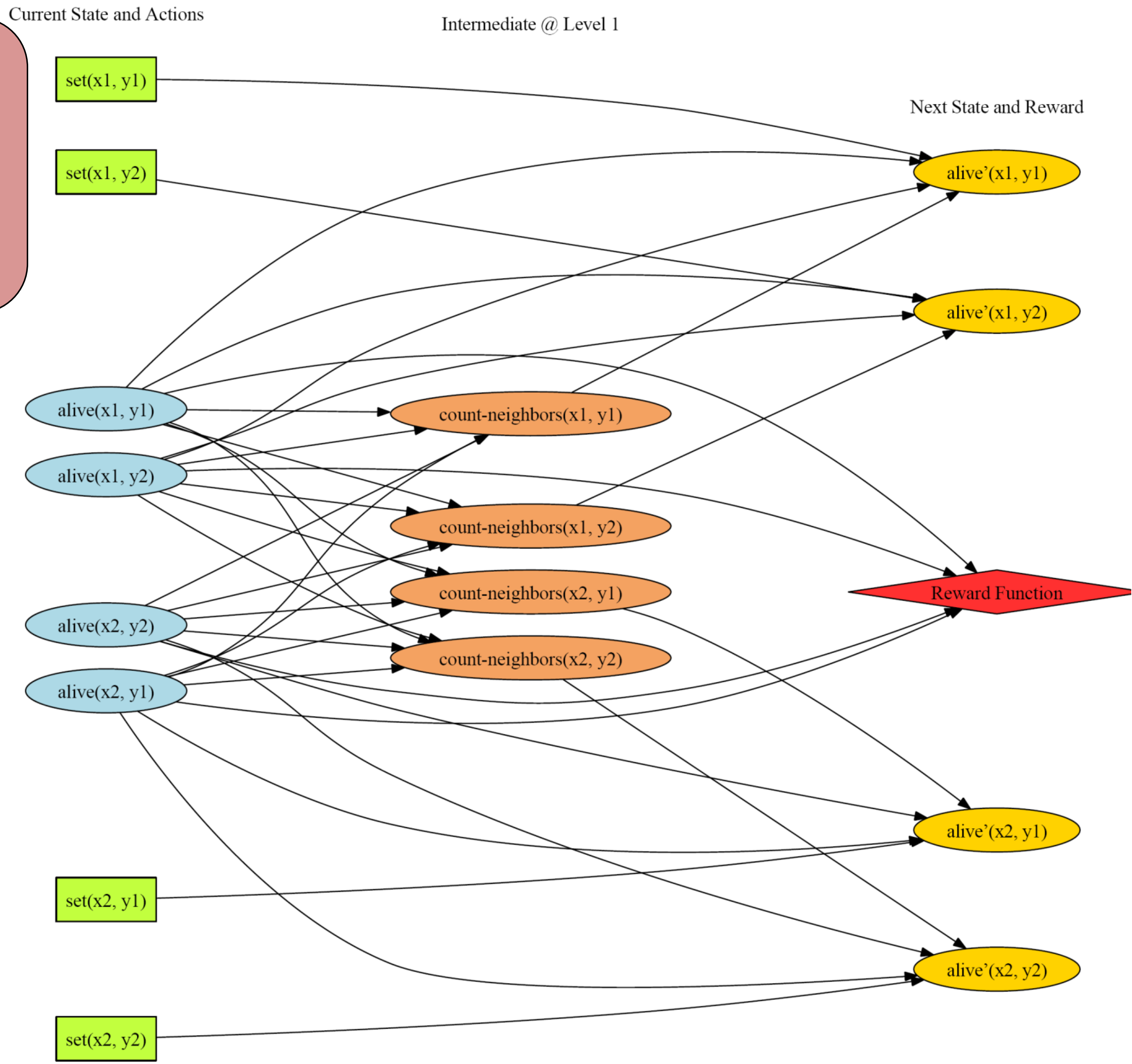
http://en.wikipedia.org/wiki/Conway's_Game_of_Life

- Compact RDDL specification for ***any* grid size**? Relational lifting.

Concurrency as factored action variables

How many possible joint actions here?

Lifted MDP: Game of Life



A Lifted MDP

```
// Store alive-neighbor co
count-neighbors(?x,?y) =
    KronDelta(sum_{?x2 : x_pos, ?y2 : y_pos}
              [NEIGHBOR(?x,?y,?x2,?y2) ^ alive(?x2,?y2)]);

// Determine whether cell (?x,?y) is alive in next state
alive'(?x,?y) = if (forall_{?y2 : y_pos} ~alive(?x,?y2))
                    then Bernoulli(PROB_REGENERATE) // Rule 6
                              ^ (count-neighbors(?x,?y) >= 2)
                              ^ (count-neighbors(?x,?y) <= 3)]
                            | [~alive(?x,?y)
                               ^ (count-neighbors(?x,?y) == 3)]
                            | set(?x,?y))
                    then Bernoulli(PROB_REGENERATE)
                    else Bernoulli(1.0 - PROB_REGENERATE);
 };

 // Reward is number of alive cells
 reward = sum_{?x : x_pos, ?y : y_pos} alive(?x,?y);

 state-action-constraints {
     // Assertion: ensure PROB_REGENERATE is a valid pro
     (PROB_REGENERATE >= 0.0) ^ (PROB_REGENERATE <= 1.0);

     // Precondition: perhaps we should not set a cell if already alive
     forall_{?x : x_pos, ?y : y_pos} alive(?x,?y) => ~set(?x,?y);
 };
```

Intermediate variable: like derived predicate

Using counts to decide next state

Additive reward!

State constraints, preconditions

Nonfluent and Instance Defintion

```
// Define numerical and topological constants
non-fluents game2x2 {
    domain = game_of_life;
    objects {
        x_pos : {x1,x2};
        y_pos : {y1,y2};
    };
    non-fluents {
        PROB_REGENERATE = 0.9; // Numerical constants are just non-fluents
        NEIGHBOR(x1,y1,x1,y2); NEIGHBOR(x1,y1,x2,y1); NEIGHBOR(x1,y1,x2,y2);
        NEIGHBOR(x1,y2,x1,y1); NEIGHBOR(x1,y2,x2,y1); NEIGHBOR(x1,y2,x2,y2);
        NEIGHBOR(x2,y1,x1,y1); NEIGHBOR(x2,y1,x1,y2); NEIGHBOR(x2,y1,x2,y2);
        NEIGHBOR(x2,y2,x1,y1); NEIGHBOR(x2,y2,x1,y2); NEIGHBOR(x2,y2,x2,y1);
    };
}

instance is1 {
    domain = game_of_life;
    non-fluents = game2x2;
    init-state {
        alive(x1,y1);
        alive(x2,y2);
    };
    max-nondef-actions = 3; // Allow up to 3 cells to be set concurrently
    horizon  = 20;
    discount = 0.9;
}
```

Objects that don't change b/w instances

Numerical constant nonfluent

Topologies over these objects

Import a topology

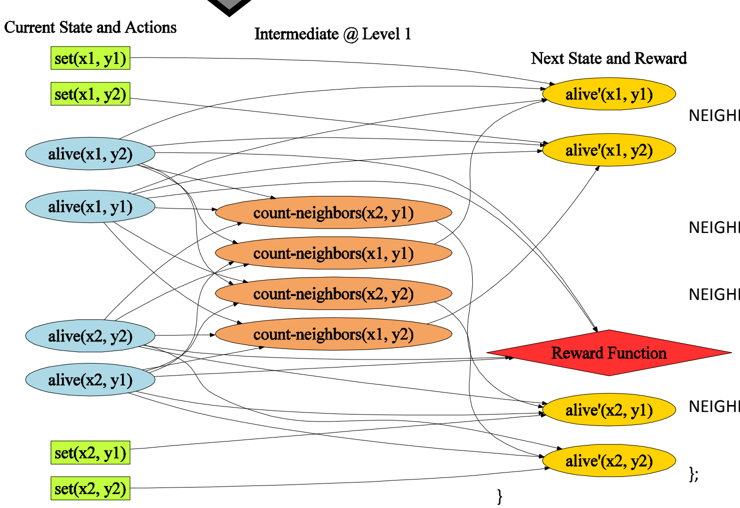
Initial state as usual

Concurrency

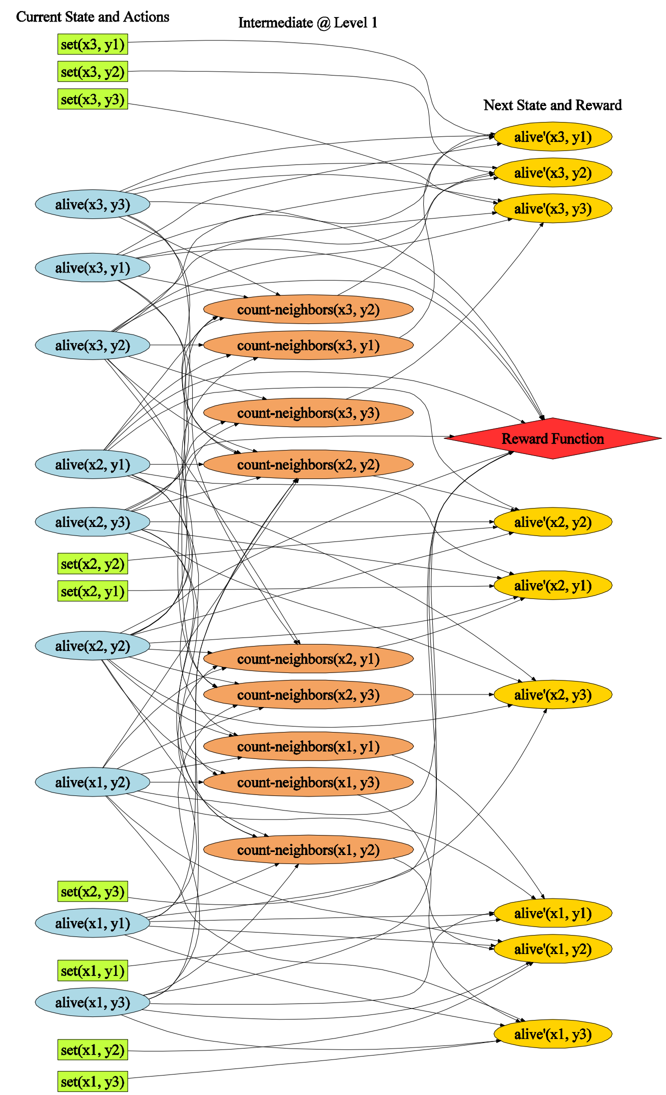
Power of Lifting

Simple domains can generate complex DBNs!

```
non-fluents game2x2 {
        domain = game_of_life;
        objects {
                x_pos : {x1,x2};
                y_pos : {y1,y2};
        };
        non-fluents {
                PROB_REGENERATE = 0.9;

                NEIGHBOR(x1,y1,x1,y2);
                NEIGHBOR(x1,y1,x2,y1);
                NEIGHBOR(x1,y1,x2,y2);

                NEIGHBOR(x1,y2,x1,y1);
                NEIGHBOR(x1,y2,x2,y1);
                NEIGHBOR(x1,y2,x2,y2);

                NEIGHBOR(x2,y1,x1,y1);
                NEIGHBOR(x2,y1,x1,y2);
                NEIGHBOR(x2,y1,x2,y2);

                NEIGHBOR(x2,y2,x1,y1);
                NEIGHBOR(x2,y2,x1,y2);
                NEIGHBOR(x2,y2,x2,y1);
        };
}
```



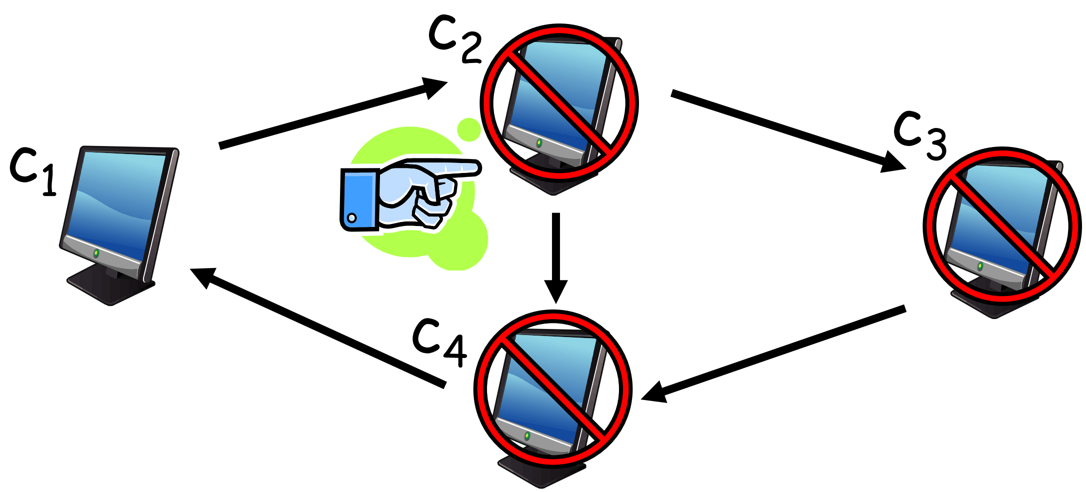

```
non-fluents game3x3 {
        domain = game_of_life;
        objects {
                x_pos : {x1,x2,x3};
                y_pos : {y1,y2,y3};
        };
        non-fluents {
                NEIGHBOR(x1,y1,x1,y2);
                NEIGHBOR(x1,y1,x2,y1);
                NEIGHBOR(x1,y1,x2,y2);
                NEIGHBOR(x1,y2,x1,y1);
                NEIGHBOR(x1,y2,x2,y1);
                NEIGHBOR(x1,y2,x2,y2);
                NEIGHBOR(x1,y2,x2,y3);
                NEIGHBOR(x1,y2,x1,y3);
        NEIGHBOR(x1,y3,x1,y2);
                NEIGHBOR(x1,y3,x2,y2);
                NEIGHBOR(x1,y3,x2,y3);
        NEIGHBOR(x2,y1,x1,y1);
                NEIGHBOR(x2,y1,x1,y2);
                NEIGHBOR(x2,y1,x2,y2);
                NEIGHBOR(x2,y1,x3,y2);
                NEIGHBOR(x2,y1,x3,y1);
        NEIGHBOR(x2,y2,x1,y1);
                NEIGHBOR(x2,y2,x1,y2);
                NEIGHBOR(x2,y2,x1,y3);
                NEIGHBOR(x2,y2,x2,y1);
                NEIGHBOR(x2,y2,x2,y3);
                NEIGHBOR(x2,y2,x3,y1);
                NEIGHBOR(x2,y2,x3,y2);
                NEIGHBOR(x2,y2,x3,y3);
        NEIGHBOR(x2,y3,x1,y3);
                NEIGHBOR(x2,y3,x1,y2);
                NEIGHBOR(x2,y3,x2,y2);
                NEIGHBOR(x2,y3,x3,y2);
                NEIGHBOR(x2,y3,x3,y3);
        NEIGHBOR(x3,y1,x2,y1);
                NEIGHBOR(x3,y1,x2,y2);
                NEIGHBOR(x3,y1,x3,y2);
        NEIGHBOR(x3,y2,x3,y1);
                NEIGHBOR(x3,y2,x2,y1);
                NEIGHBOR(x3,y2,x2,y2);
                NEIGHBOR(x3,y2,x2,y3);
                NEIGHBOR(x3,y2,x3,y3);
        NEIGHBOR(x3,y3,x2,y3);
                NEIGHBOR(x3,y3,x2,y2);
                NEIGHBOR(x3,y3,x3,y2);
        };
}
```



Complex Lifted Transitions: SysAdmin

SysAdmin (Guestrin et al, 2001)

- Have n computers $C = \{c_1, \ldots, c_n\}$ in a network
- **State:** each computer c_i is either "up" or "down"



- **Transition:** computer is "up" proportional to its state and # upstream connections that are "up"
- **Action:** manually reboot one computer
- **Reward:** +1 for every "up" computer

Complex Lifted Transitions

```
pvariables {

    REBOOT-PROB : { non-fluent, real, default = 0.1 };
    REBOOT-PENALTY : { non-fluent, real, default = 0.75 };

    CONNECTED(computer, computer) : { non-fluent, bool, default = false };

    running(computer) : { state-fluent, bool, default = false };

    reboot(computer) : { action-fluent, bool, default = false };
};

cpfs {

  running'(?x) = if (reboot(?x))
     then KronDelta(true)  // if                                    then must be running
     else if (running(?x)) // else                                  network properties
        then Bernoulli(
         .5 + .5*[1 + sum_{?y : computer} (CONNECTED(?y,?x) ^ running(?y))]
                / [1 + sum_{?y : computer} CONNECTED(?y,?x)])
        else Bernoulli(REBOOT-PROB);
};

reward = sum_{?c : computer} [running(?c) - (REBOOT-PENALTY * reboot(?c))];
```

Probability of a computer running depends on ratio of connected computers running!

How to Think About RDDL Distributions

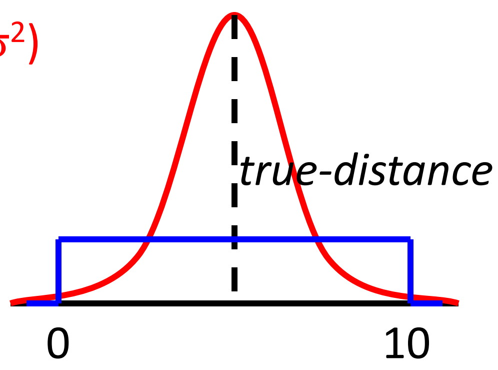
- Transition distribution is **stochastic program**
  - Similar to BLOG (Milch, Russell, et al), IBAL (Pfeffer)
  - Basically just complex conditional distributions
- *S*pecification of generative sampling process
  - E.g., noisy distance measurement in robotics
    - First draw **boolean** *Noise* := sample from Bernoulli (.1)
    - Then draw **real** *Measurement* :=

      If (*Noise* == true)

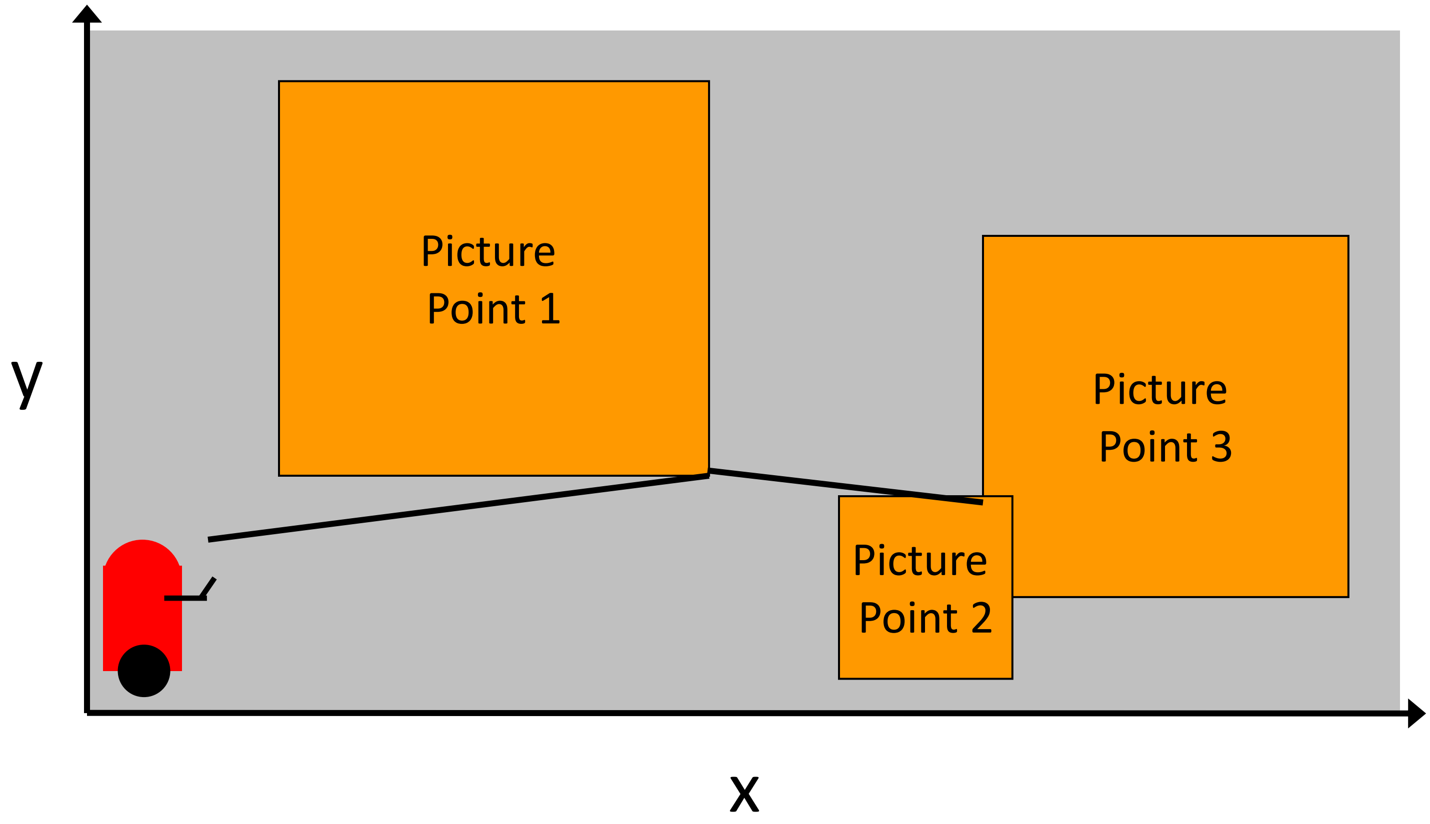
      » Then sample from Uniform(0, 10)

      » Else sample from Normal(true-distance, σ^2)

Convenient way to write complex mixture models and conditional distributions that occur in practice!



Lifted Continuous MDP in RDDL: Simple Mars Rover



Simple Mars Rover: Part I

```
types { picture-point : object; };

pvariables {

    PICT_XPOS(picture-point)   : { non-fluent, real, default = 0.0 };
    PICT_YPOS(picture-point)   : { non-fluent, real, default = 0.0 };
    PICT_VALUE(picture-point)  : { non-fluent, real, default = 1.0 };
    PICT_ERROR_ALLOW(picture-point) : { non-fluent, real, default = 0.5 };

    xPos : { state-fluent, real, default = 0.0 };
    yPos : { state-fluent, real, default = 0.0 };
    time : { state-fluent, real, default = 0.0 };

    xMove       : { action-fluent, real, default = 0.0 };
    yMove       : { action-fluent, real, default = 0.0 };
    snapPicture : { action-fluent, bool, default = false };

};
```

Constant picture points, bounding box

Rover position (only one rover) and time

Rover actions

Question, how to make multi-rover?

Simple Mars Rover: Part II

```
cpfs {

        // Noisy movement update
        xPos' = xPos + xMove + Normal(0.0, MOVE_VARIANCE_MULT*xMove);

        yPos' = yPos + yMove + Normal(0.0, MOVE_VARIANCE_MULT*yMove);

        // Time update
        time' = if (snapPicture)
                then DiracDelta(time + 0.25)
                else DiracDelta(time +
                        [if (xMove > 0) then xMove else -xMove] +
                        [if (yMove > 0) then yMove else -yMove]);

};
```

White noise, variance proportional to distance moved

Fixed time for picture

Time proportional to distance moved

Simple Mars Rover: Part III

```
// We get a reward for any picture taken within picture box error bounds
// and the time limit.
reward = if (snapPicture ^ (time <= MAX_TIME))
             then sum_{?p : picture-point} [
                 if ((xPos >= PICT_XPOS(?p) - PICT_ERROR_ALLOW(?p))
                        ^ (xPos <= PICT_XPOS(?p) +  PICT_ERROR_ALLOW(?p))
                        ^ (yPos >= PICT_YPOS(?p) - PICT_ERROR_ALLOW(?p))
                        ^ (yPos <= PICT_YPOS(?p) + PICT_ERROR_ALLOW(?p)))
                 then PICT_VALUE(?p)
                 else 0.0 ]
             else 0.0;

state-action-constraints {

          // Cannot snap a picture and move at the same time
          snapPicture => ((xMove == 0.0) ^ (yMove == 0.0));
};
```

Reward for all pictures taken within bounding box!

Cannot move and take picture at same time.

RDDL Software

Open source & online at
http://code.google.com/p/rddlsim/

RDDL Java Software Overview

- BNF grammar and parser
- Simulator
- Automatic translations
  - LISP-like format (easier to parse)
  - SPUDD & Symbolic Perseus (boolean subset)
  - Ground PPDDL (boolean subset)
- Client / Server
  - Evaluation scripts for log files
- Visualization
  - DBN Visualization
  - Domain Visualization – *see* how your planner is doing

RDDL vs. PPDDL (In)equivalance

- For a fixed domain instance and discrete noise
  - RDDL and PPDDL are expressively equivalent
  - Both convertible to Influence Diagram + DBN
- For lifted domain specification (no instance)
  - There exist lifted models in RDDL that **cannot** be expressed in lifted PPDDL
    - SysAdmin
      - transition probability function of state
      - reward sum over all objects
    - Traffic
      - indefinite concurrent actions, constraints
    - Simple Mars Rover
      - Gaussian noise

Summary of Part 1: Modeling

- Many real-world problems naturally modeled with continuous variables
- MDPs and POMDPs can formalize almost any continuous problem
- RDDL (and to some extent PPDDL) allow very compact lifted models of these domains

Tutorial Outline

1. Modeling Continuous Problems
   a) Why continuous?
   b) MDPs and POMDPs
   c) (P)PDDL and RDDL

2. Solving Continuous Problems
   a) Exact dynamic programming
      - Data structures
   b) Open problems
   c) Survey of other solution methods
   d) Connections to control and scheduling

Part 2a: Solutions

Exact dynamic programming

Discrete and Continuous (DC-)MDPs

- Mixed discrete / continuous state

$$(\vec{b}, \vec{x}) = (b_1, \ldots, b_n, x_1, \ldots, x_m) \in \{0,1\}^n \times \mathbb{R}^m$$

- Discrete action set $a \in \mathcal{A}$
- DBN factored transition model

$$P(\vec{b}', \vec{x}' | \vec{b}, \vec{x}, a) =$$

$$\underbrace{\left(\prod_{i=1}^{n} P(b_i' | \vec{b}, \vec{x}, a) \right)}_{\text{discrete}} \underbrace{\left(\prod_{j=1}^{m} P(x_j' | \vec{b}, \vec{b}', \vec{x}, a) \right)}_{continuous}$$

- Action-dependent reward

$$R_a(\vec{b}, \vec{x}) = x_1^2 + x_1 x_2$$

Exact Dynamic Programming for DC-MDPs

- Value of policy in state is expected sum of rewards
- Want optimal value $V^{h,*}$ over horizons $h \in 0..H$
  - Implicitly provides optimal horizon-dependent policy
- Compute inductively via **Value Iteration** for $h \in 0..H$
  - **Regression step:**

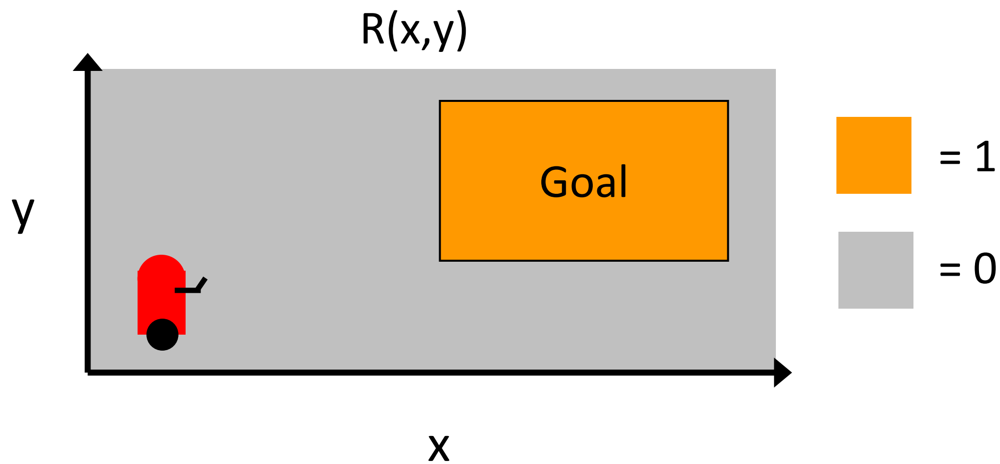
$$Q_a^{h+1}(\vec{b},\vec{x}) = R_a(\vec{b},\vec{x}) + \gamma \cdot \sum_{\vec{b}'} \int_{\vec{x}'} \left(\prod_{i=1}^{n} P(b_i'|\vec{b},\vec{x},a) \prod_{j=1}^{m} P(x_j'|\vec{b},\vec{b}',\vec{x},a) \right) V^h(\vec{b}',\vec{x}')d\vec{x}'$$

  - **Maximization step:**

$$V_{h+1} = \max_{a \in A} Q_a^{h+1}(\vec{b},\vec{x})$$

Exact Solutions to n-D DC-MDPs: Domain

- 2-D Navigation
- State: $(x,y) \in \mathbb{R}^2$
- Actions:
  - move-x-2
    - x' = x + 2
    - y' = y
  - move-y-2
    - x' = x
    - y' = y + 2
- Reward:
  - R(x,y) = I[(x > 5) ^ (x < 10) ^ (y > 2) ^ (y < 5)]

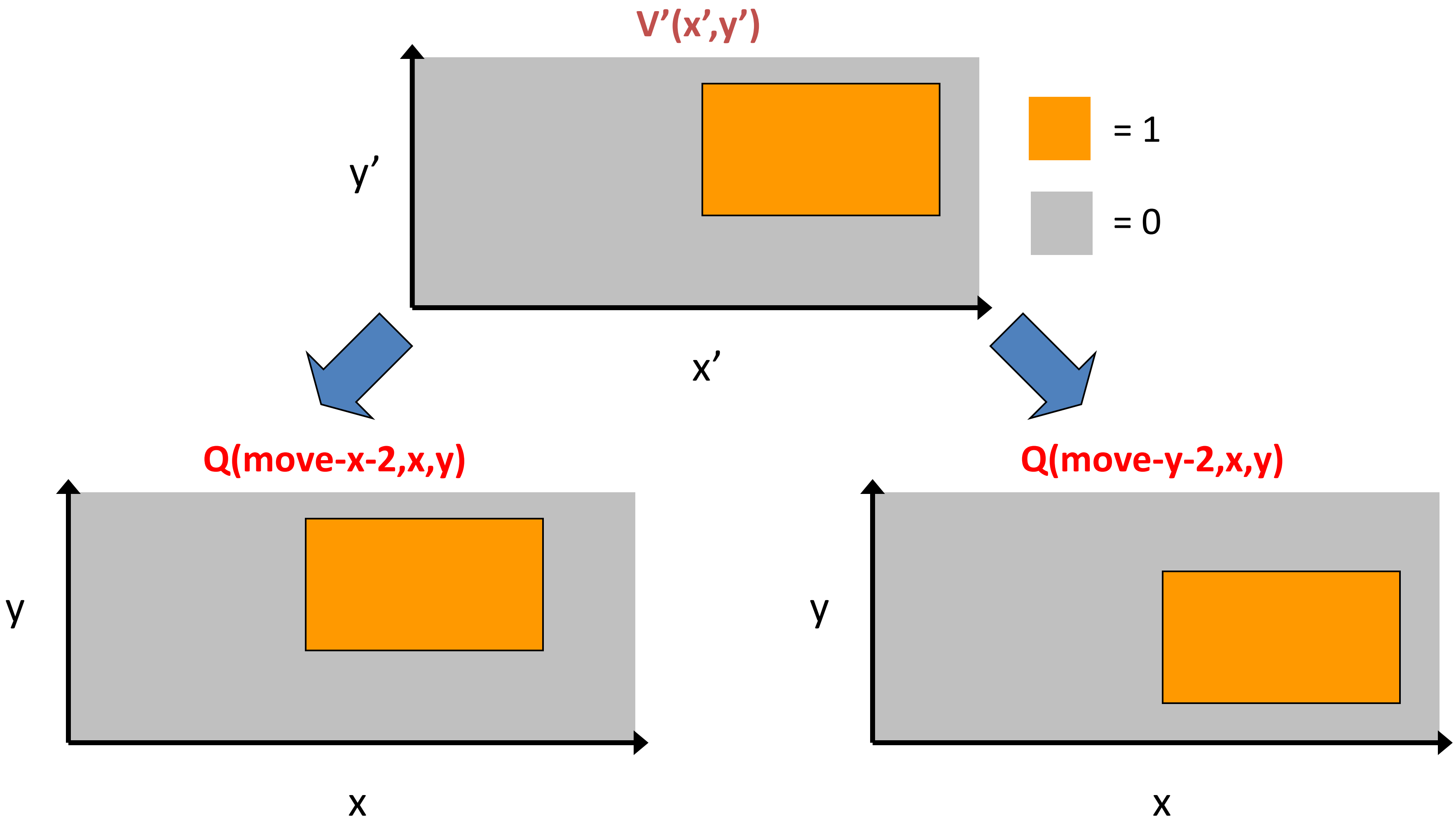


Assumptions:

1. Continuous transitions are deterministic and linear
2. Discrete transitions can be stochastic
3. Reward is piecewise rectilinear

Exact Solutions to n-D DC-MDPs: Regression

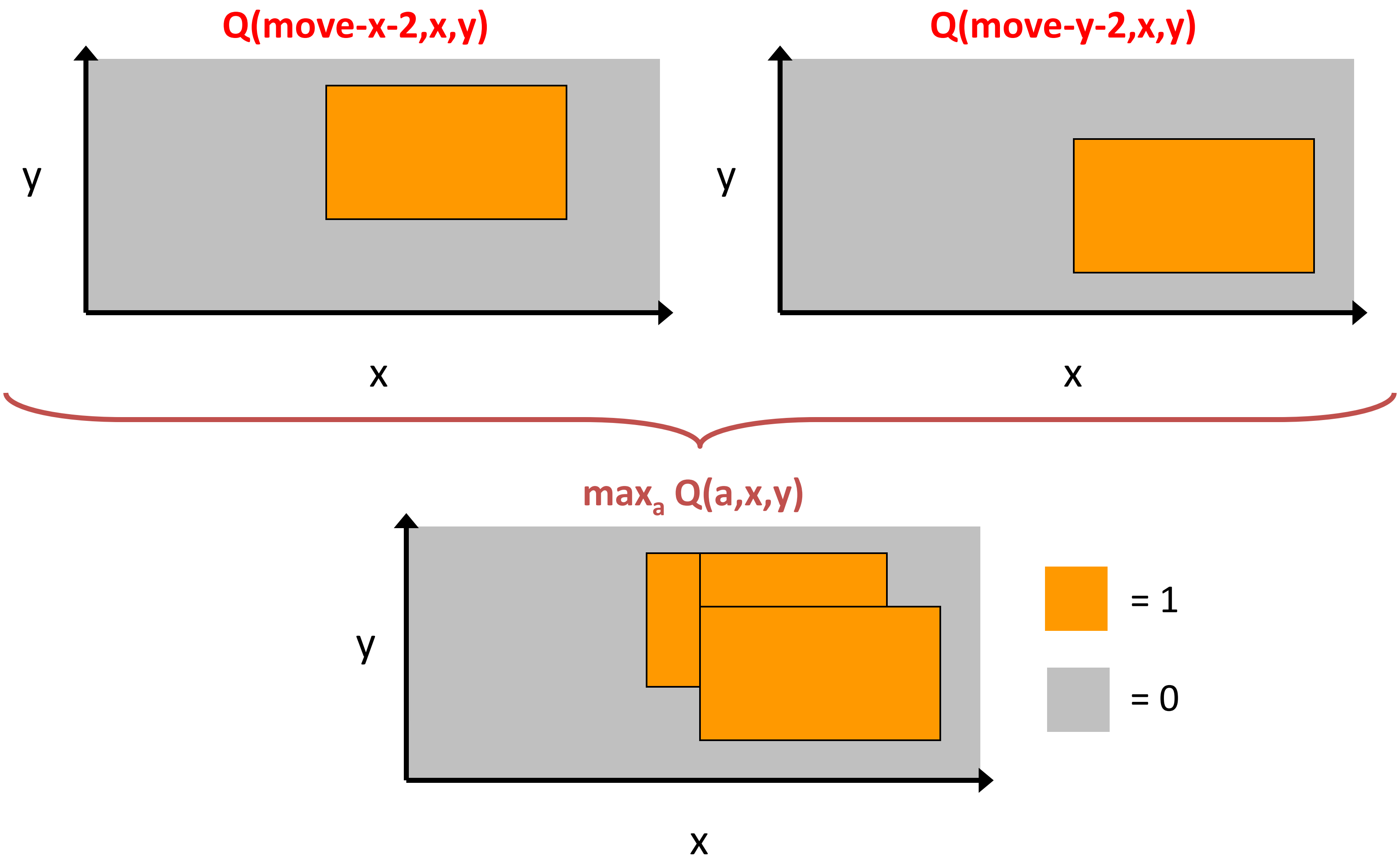
- Continuous regression is just translation of "pieces"



Feng *et al*, UAI-04

Exact Solutions to n-D DC-MDPs: Maximization

- Q-value maximization yields piecewise rectilinear solution



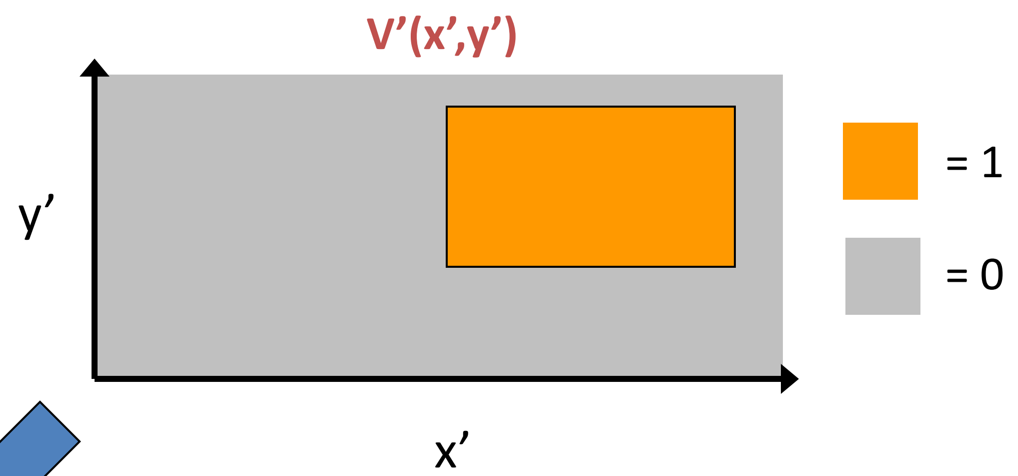
Previous Work Limitations I

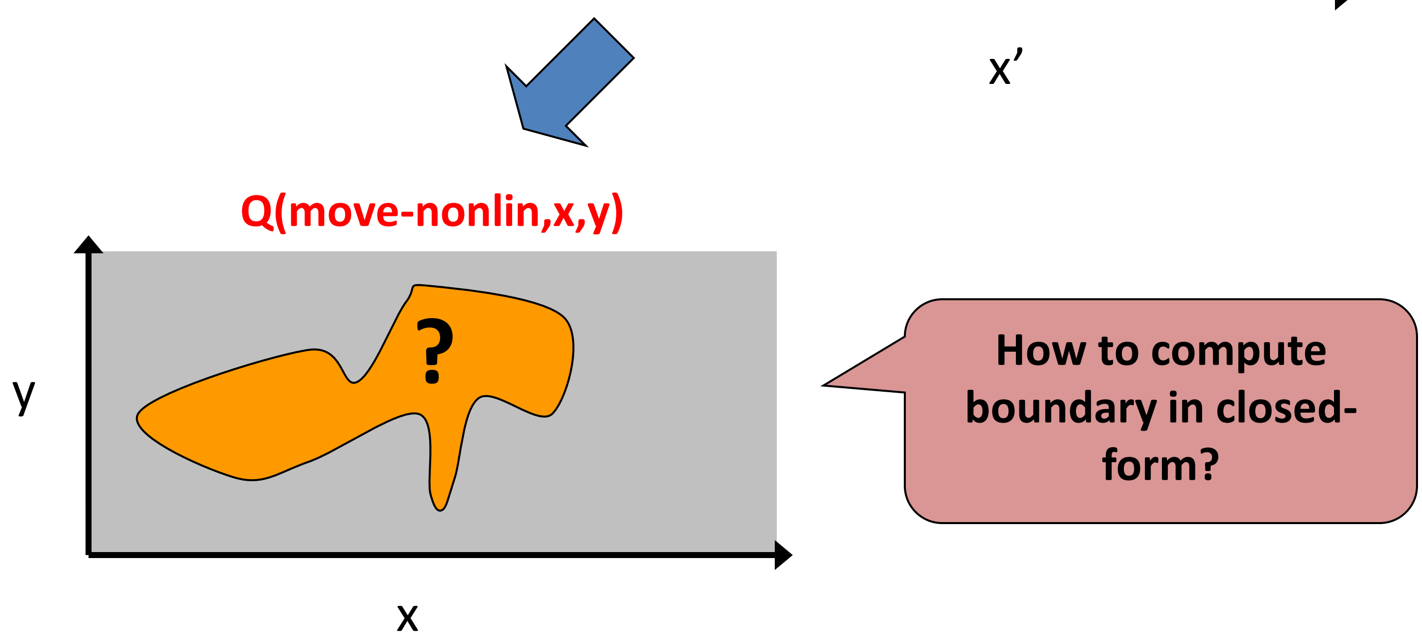
- Exact regression when transitions nonlinear?

Action move-nonlin:

– $x' = x^3 y + y^2$

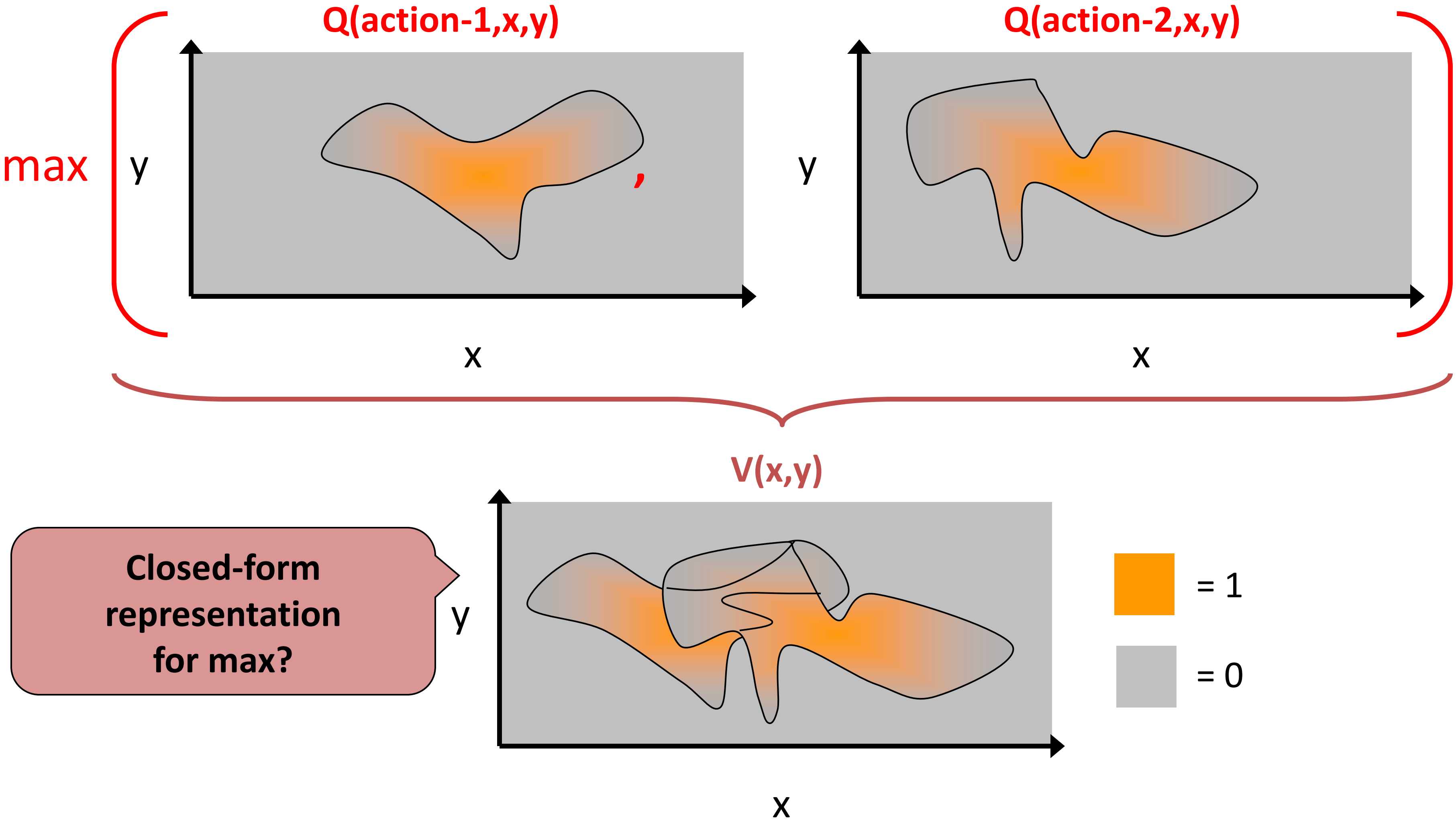
– $y' = y * \log(x^2 y)$





Previous Work Limitations II

- max(.,.) when reward/value arbitrary piecewise?



Brief History of Exact DP for Continuous MDPs

- Time-dependent MDPs (1-D)
  - Fascinating solution by Boyan and Littman (NIPS-00)
  - Recent extensions by Rachelson

- General n-D Solutions
  - Bresina, Dearden, Meuleau, Ramkrishnan, Smith, Washington, R. (UAI 2002) stress importance
  - Feng, Dearden, Meuleau, Washington, (UAI 2004) introduce first restricted exact solutions (hyperrectangular)
  - Li and Littman (AAAI 2005), more expressive dynamics, approximate solutions
  - Sanner, Delgado, Barros (UAI 2011) extend to expressive domains
  - Zamani, Sanner, Fang (AAAI 2012) extend to continuous actions under some restrictions

A solution to previous limitations:

Symbolic Dynamic Programming (SDP)

Joint work with:

Karina Valdivia Delgado
Leliane Nunes de Barros

Ehsan Abbasnejad
Zahra Zamani

Cheng
Fang

Universidade de São Paulo

THE AUSTRALIAN NATIONAL UNIVERSITY

Massachusetts Institute of Technology

Symbolic Dynamic Programming requires a Symbolic Representation

Piecewise Case Statement!

Piecewise Functions (Cases)

$$z = f(x,y) = \begin{cases} \boxed{(x > 3) \wedge (y \cdot \ \ x) : \quad x + y} \\ (x \cdot \ \ 3) \vee (y > x) : \quad x^2 + xy^3 \end{cases}$$

Partition

Constraint

Value

Linear constraints and value

Linear constraints, constant value

Quadratic constraints and value

Case Operations: ⊕, ⊗

$$\begin{cases} \phi_1 : & f_1 \\ \phi_2 : & f_2 \end{cases} \oplus \begin{cases} \psi_1 : & g_1 \\ \psi_2 : & g_2 \end{cases} = \qquad ?$$

Case Operations: ⊕, ⊗

$$\begin{cases} \phi_1 : & f_1 \\ \phi_2 : & f_2 \end{cases} \oplus \begin{cases} \psi_1 : & g_1 \\ \psi_2 : & g_2 \end{cases} = \begin{cases} \phi_1 \wedge \psi_1 : & f_1 + g_1 \\ \phi_1 \wedge \psi_2 : & f_1 + g_2 \\ \phi_2 \wedge \psi_1 : & f_2 + g_1 \\ \phi_2 \wedge \psi_2 : & f_2 + g_2 \end{cases}$$

- Similarly for ⊗
  - Expressions trivially closed under +, *
- What about max?
  - $\max(f_1, g_1)$ not pure arithmetic expression ☹

Case Operations: max

$$\max\left(\begin{cases} \phi_1 : & f_1 \\ \phi_2 : & f_2 \end{cases} , \begin{cases} \psi_1 : & g_1 \\ \psi_2 : & g_2 \end{cases} \right) = \qquad ?$$

Case Operations: max

$$
\max\left(\begin{cases} \phi_1 : & f_1 \\ \phi_2 : & f_2 \end{cases}, \begin{cases} \psi_1 : & g_1 \\ \psi_2 : & g_2 \end{cases} \right) = \begin{cases} \phi_1 \wedge \psi_1 \wedge f_1 > g_1 : & f_1 \\ \phi_1 \wedge \psi_1 \wedge f_1 \cdot \; g_1 : & g_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 > g_2 : & f_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 \cdot \; g_2 : & g_2 \\ \phi_2 \wedge \psi_1 \wedge f_2 > g_1 : & f_2 \\ \phi_2 \wedge \psi_1 \wedge f_2 \cdot \; g_1 : & g_1 \\ \phi_2 \wedge \psi_2 \wedge f_2 > g_2 : & f_2 \\ \phi_2 \wedge \psi_2 \wedge f_2 \cdot \; g_2 : & g_2 \end{cases}
$$

Key point: still in case form!

Size blowup? We'll get to that…

All Case Ops for Dynamic Programming?

- Value Iteration for $h \in 0..H$
  - Regression step:

$$Q_a^{h+1}(\vec{b}, \vec{x}) = R_a(\vec{b}, \vec{x}) + \gamma \cdot$$
$$\sum_{\vec{b}'} \int_{\vec{x}'} \left(\prod_{i=1}^{n} P(b_i' | \vec{b}, \vec{x}, a) \prod_{j=1}^{m} P(x_j' | \vec{b}, \vec{b}', \vec{x}, a) \right) V^h(\vec{b}', \vec{x}') d\vec{x}'$$

  - Maximization step:

$$V_{h+1} = \max_{a \in A} Q_a^{h+1}(\vec{b}, \vec{x})$$

  - Almost there: we need to define $\sum_{b'}$ and $\int_{x'}$

SDP Regression Step

- **Binary variable $\sum$**
  - As done in SPUDD: Hoey *et al*, UAI-99

$$\sum_{b_i \in \{0,1\}} f(\vec{b}, \vec{x}) = f(\vec{b}, \vec{x})|_{b_i=1} \oplus f(\vec{b}, \vec{x})|_{b_i=0}$$

$$\sum_{b_1 \in \{0,1\}} \begin{cases} \phi_1 \wedge b_1 : & f_1 \\ \phi_1 \wedge \neg b_1 : & f_2 \\ \neg \phi_1 : & f_3 \end{cases} = \begin{cases} \phi_1 : & f_1 \\ \neg \phi_1 : & f_3 \end{cases} \oplus \begin{cases} \phi_1 : & f_2 \\ \neg \phi_1 : & f_3 \end{cases}$$

SDP Regression Step

- **Continuous variables x_j**
  - $\int_x \delta[x - y] f(x) dx = f(y)$ triggers symbolic *substitution*
  - e.g., $\int_{x'_j} \delta[x'_j - g(\vec{x})] V' dx'_j = V'\{x'_j / g(\vec{x})\}$

$$\int_{x'_1} \delta[x'_1 - (x_1^2 + 1)] \left(\begin{cases} \underline{x'_1} < 2 : & \underline{x'_1} \\ \underline{x'_1} \geq 2 : & \underline{x'^2_1} \end{cases} \right) dx'_1 = \begin{cases} \underline{x_1^2 + 1} < 2 : & \underline{x_1^2 + 1} \\ \underline{x_1^2 + 1} \geq 2 : & \underline{(x_1^2 + 1)}^2 \end{cases}$$

  - If ***g*** is *case*: need *conditional substitution*
    - see Sanner, Delgado, Barros (UAI 2011)

That's SDP!

In theory

Exact for any reward, discrete noise transition dynamics!

- Value Iteration for h∈0..H
  - Regression step:

$$Q_a^{h+1}(\vec{b},\vec{x}) = R_a(\vec{b},\vec{x}) + \gamma \cdot$$

$$\sum_{\vec{b}'} \int_{\vec{x}'} \left(\prod_{i=1}^{n} P(b_i'|\vec{b},\vec{x},a) \prod_{j=1}^{m} P(x_j'|\vec{b},\vec{b}',\vec{x},a) \right) V^h(\vec{b}',\vec{x}')d\vec{x}'$$

  - Maximization step:

$$V_{h+1} = \max_{a \in A} Q_a^{h+1}(\vec{b},\vec{x})$$

Data Structures for Continuous Planning

Case $\rightarrow$ XADD

SDP needs an efficient data structure for

- compact, minimal case representation
- efficient case operations

BDD / ADDs

Quick Introduction

Function Representation (Tables)

- How to represent functions: $B^n \rightarrow R$?
- How about a fully enumerated table…
- …OK, but can we be more compact?

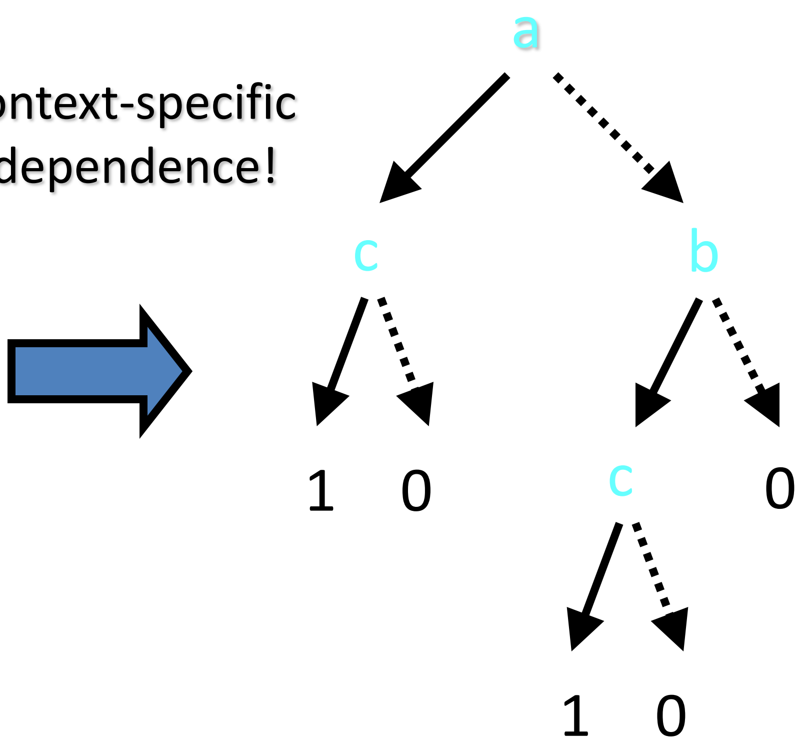
a	b	c	F(a,b,c)
0	0	0	0.00
0	0	1	0.00
0	1	0	0.00
0	1	1	1.00
1	0	0	0.00
1	0	1	1.00
1	1	0	0.00
1	1	1	1.00

Function Representation (Trees)

- How about a tree? Sure, can simplify.

a	b	c	F(a,b,c)
0	0	0	0.00
0	0	1	0.00
0	1	0	0.00
0	1	1	1.00
1	0	0	0.00
1	0	1	1.00
1	1	0	0.00
1	1	1	1.00

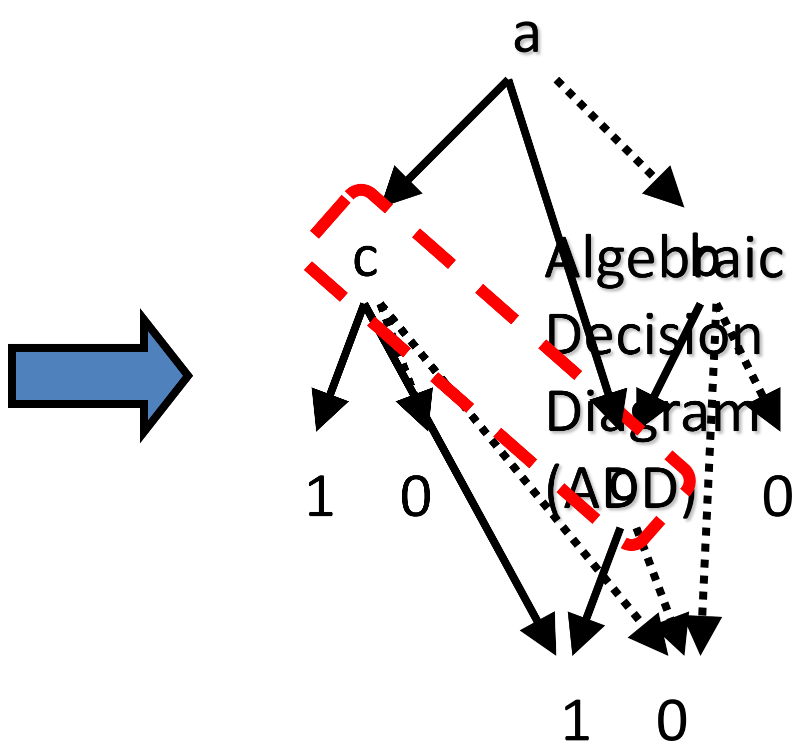
Context-specific independence!



Function Representation (ADDs)

- Why not a directed acyclic graph (DAG)?

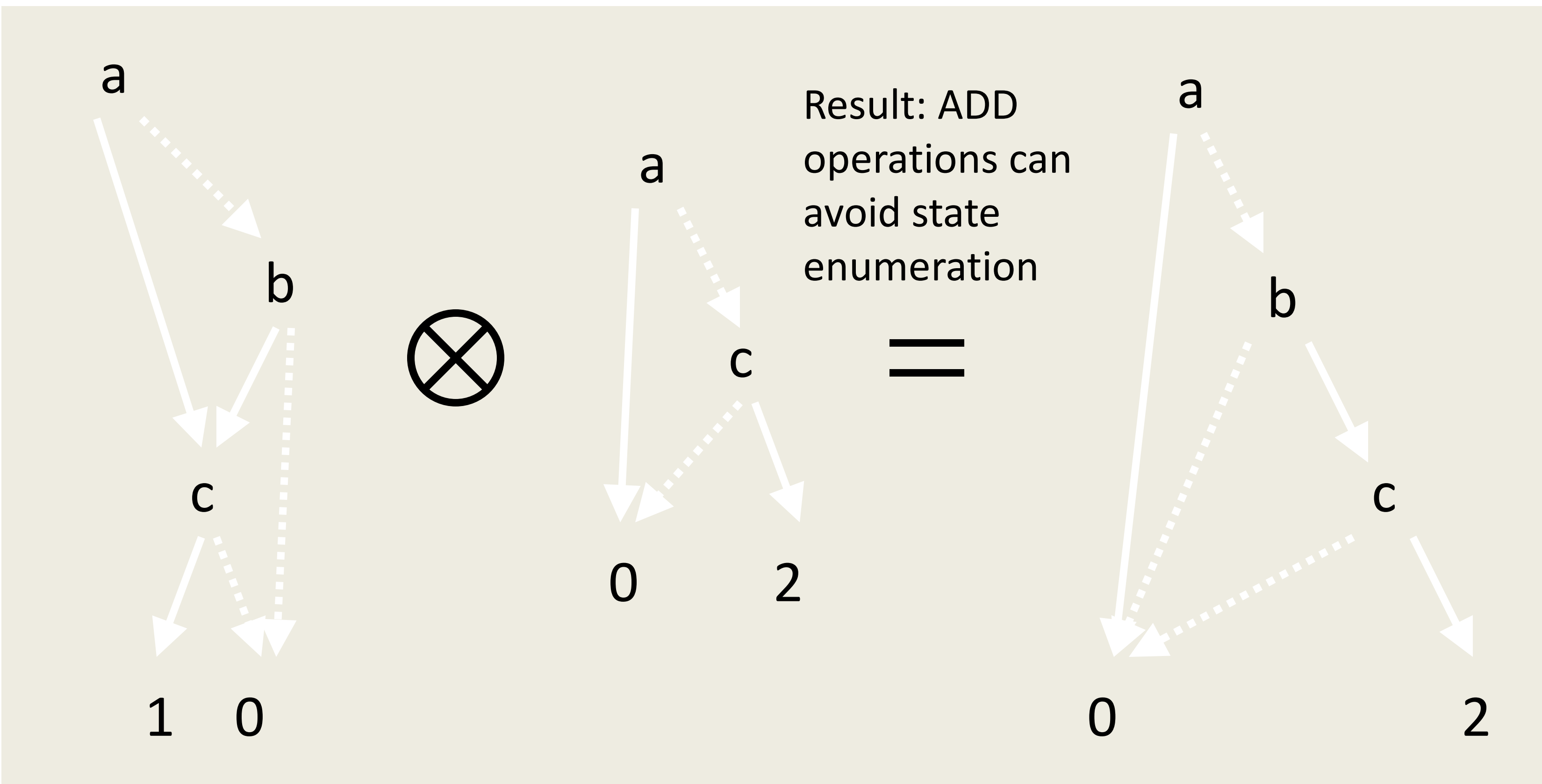
a	b	c	F(a,b,c)
0	0	0	0.00
0	0	1	0.00
0	1	0	0.00
0	1	1	1.00
1	0	0	0.00
1	0	1	1.00
1	1	0	0.00
1	1	1	1.00



Think of BDDs as {0,1} subset of ADD range

Binary Operations (ADDs)

- Why do we order variable tests?
- Enables us to do efficient binary operations…



Case $\rightarrow$ XADD

XADD = continuous variable e**x**tension
of **a**lgebraic **d**ecision **d**iagram

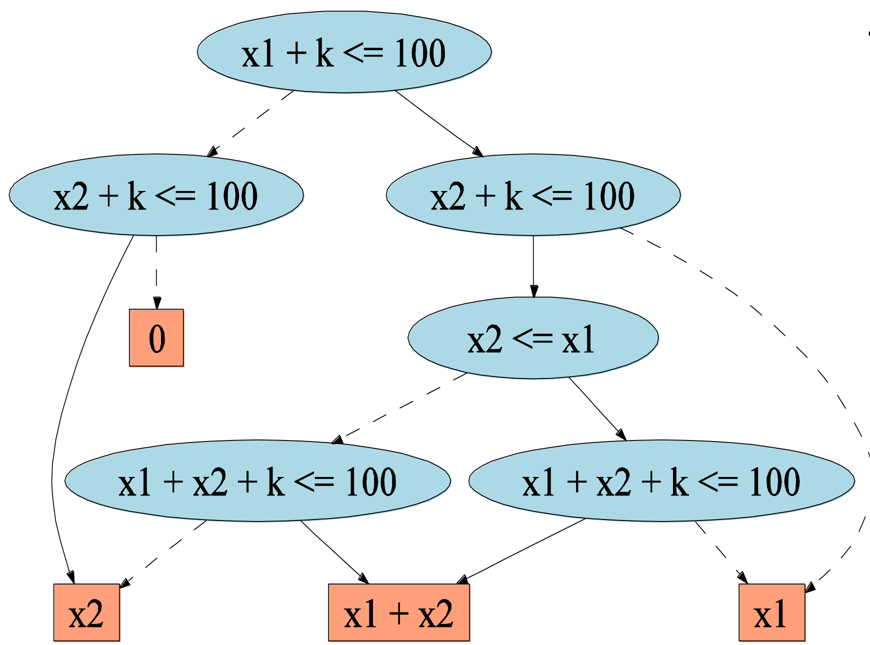
Efficient XADD data structure for cases
• strict ordering of atomic inequality tests

➔ compact, minimal case representation
➔ efficient case operations

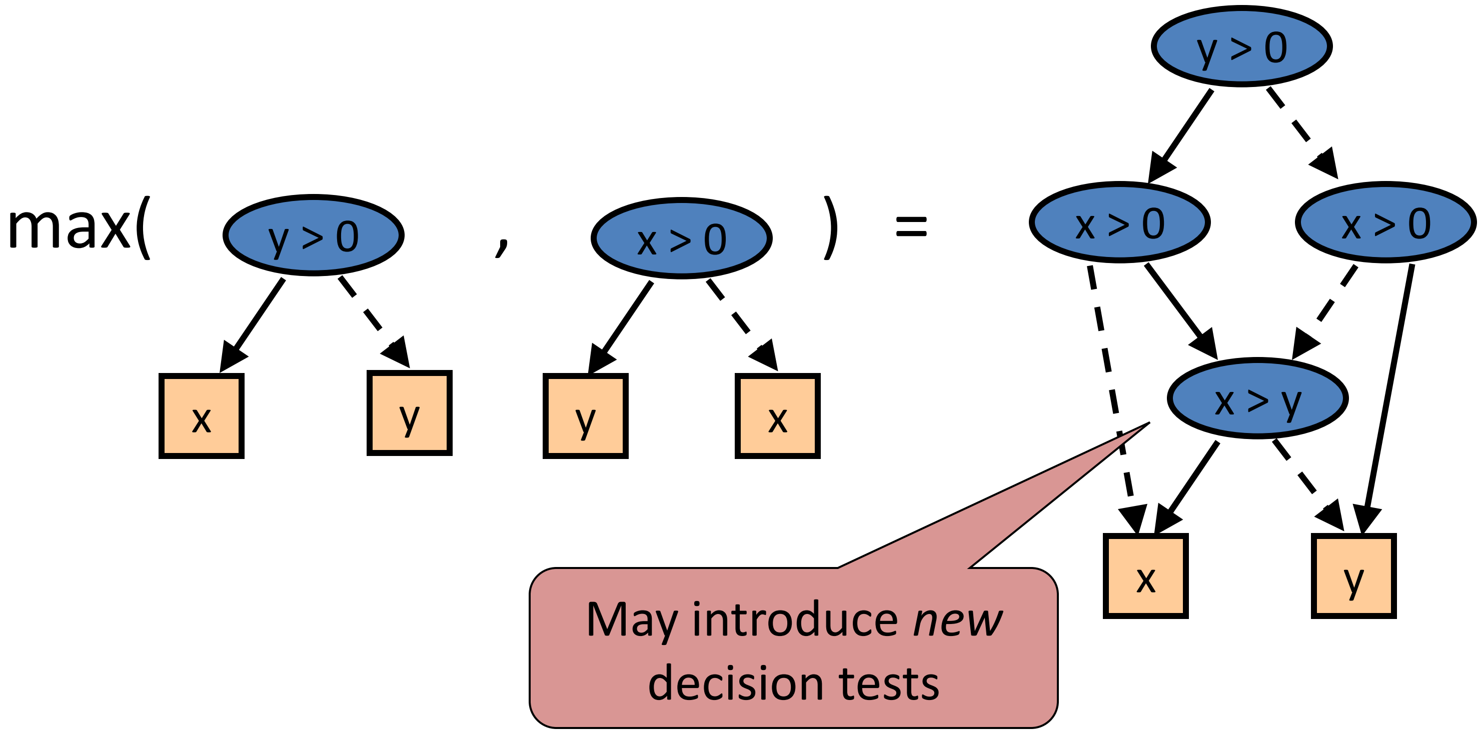
XADDs

- Extended ADD representation of case statements

$$V = \begin{cases} x_1 + k > 100 \wedge x_2 + k > 100 : & 0 \\ x_1 + k > 100 \wedge x_2 + k \leq 100 : & x_2 \\ x_1 + k \leq 100 \wedge x_2 + k > 100 : & x_1 \\ x_1 + x_2 + k > 100 \wedge x_1 + k \leq 100 \wedge x_2 + k \leq 100 \wedge x_2 > x_1 : & x_2 \\ x_1 + x_2 + k > 100 \wedge x_1 + k \leq 100 \wedge x_2 + k \leq 100 \wedge x_2 \leq x_1 : & x_1 \\ x_1 + x_2 + k \leq 100 : & x_1 + x_2 \end{cases}$$



XADD Maximization

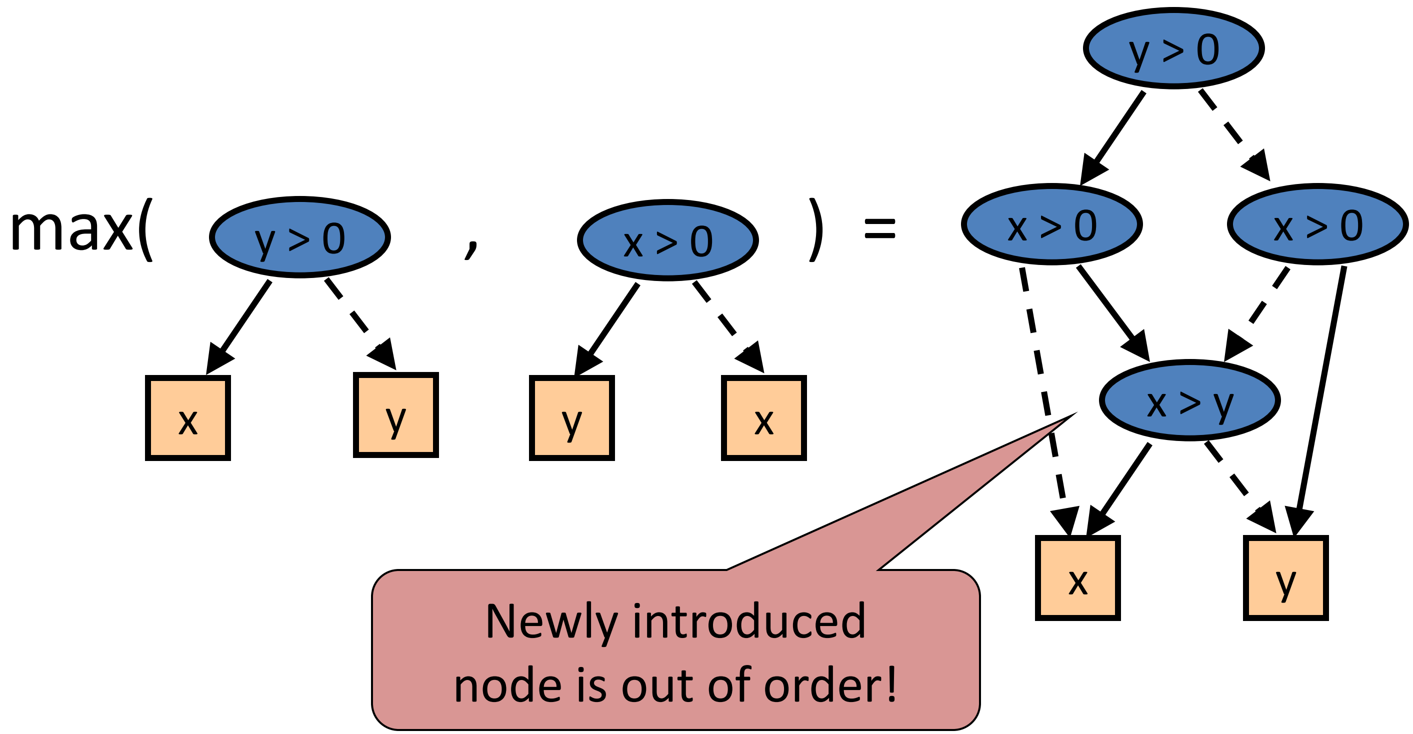


Maintaining XADD Orderings I

- Max may get variables out of order

Decision ordering (root→leaf)

- $x > y$
- $y > 0$
- $x > 0$

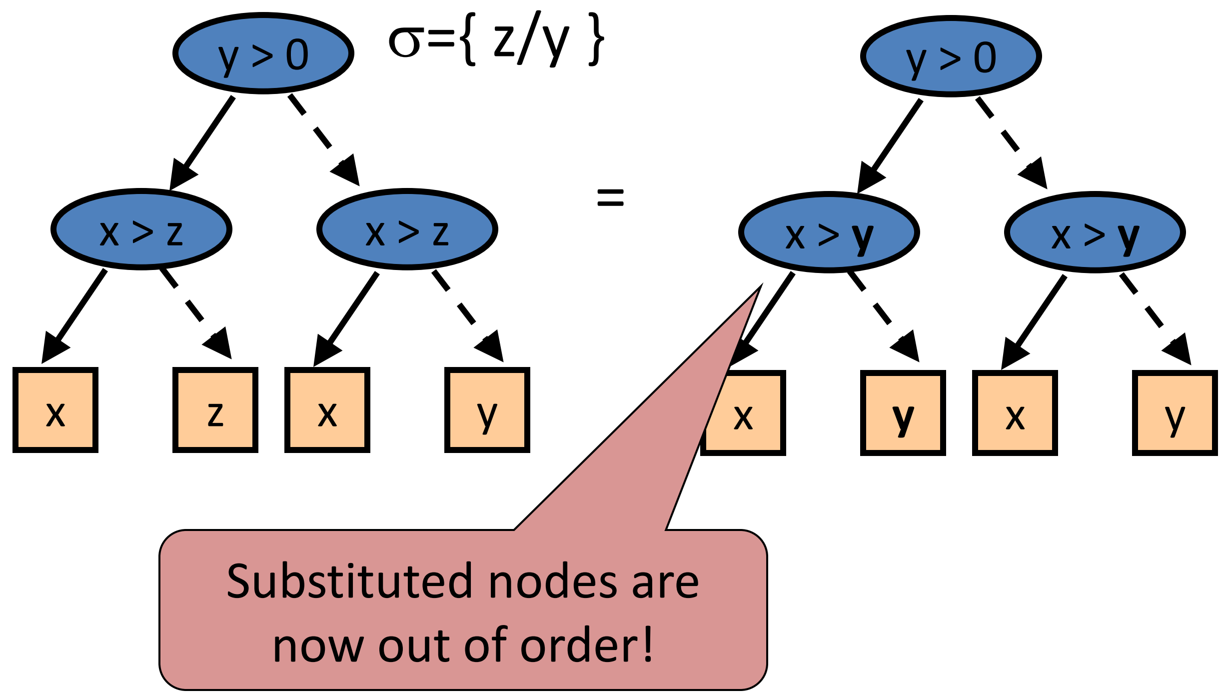


Maintaining XADD Orderings II

- Substitution may get vars out of order

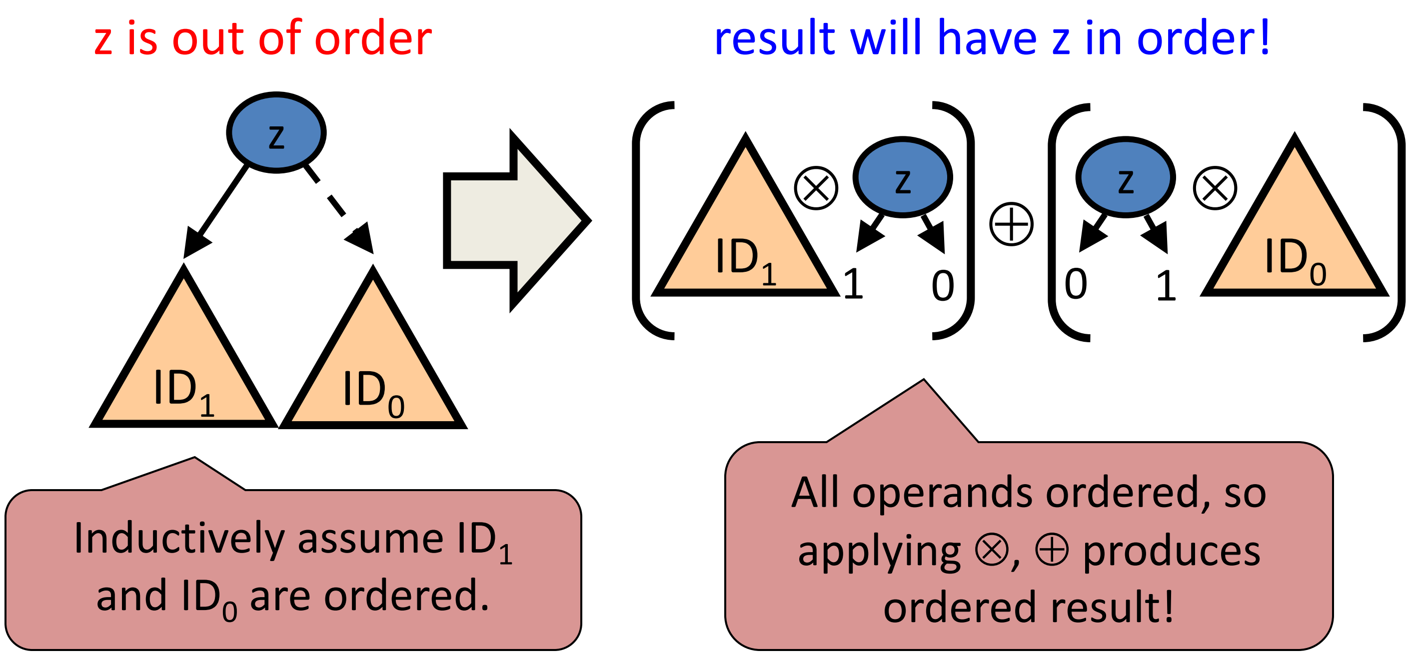
Decision ordering (root→leaf):

- x > y
- y > 0
- x > z

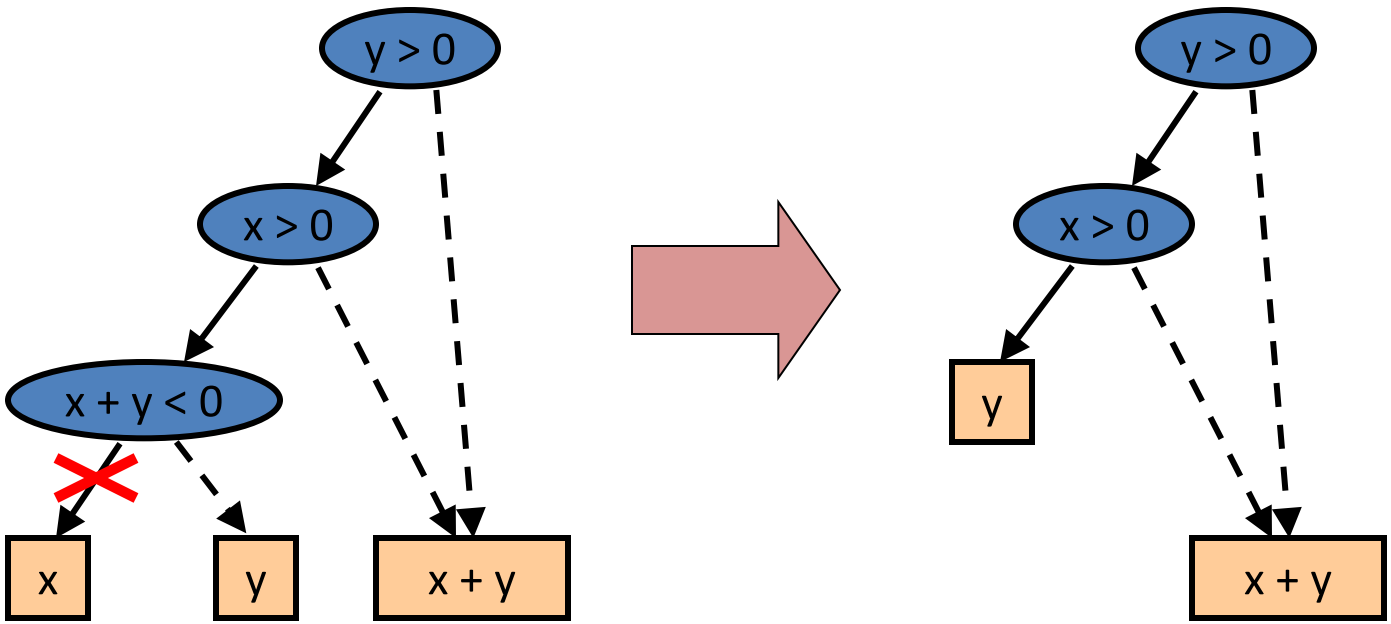


Correcting XADD Ordering

- Obtain *ordered* XADD from *unordered* XADD
  - key idea: binary operations maintain orderings



XADD Pruning



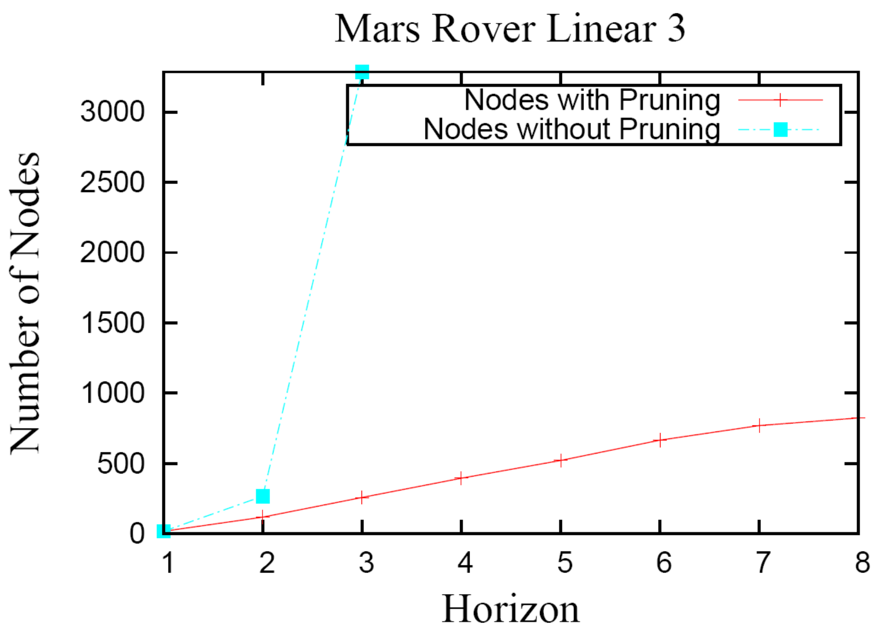
Node unreachable –
$x + y < 0$ always false if
$x > 0$ & $y > 0$

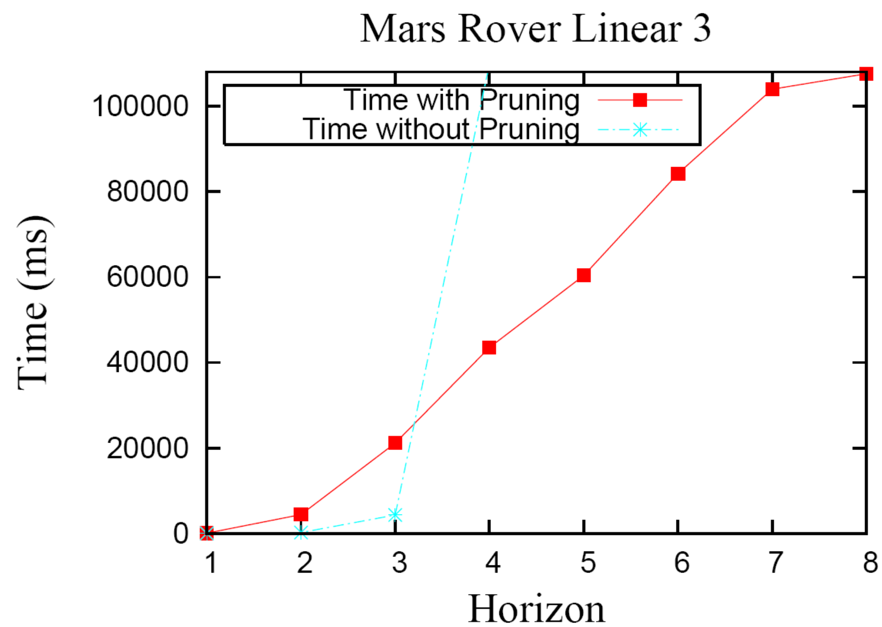
If **linear**, can detect with feasibility checker of LP solver & prune

Take-home point:
SDP impossible without XADD

How well does it work?

Results: XADD Pruning vs. No Pruning





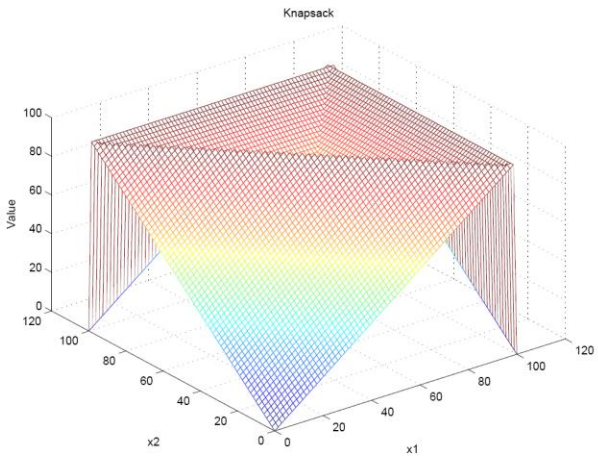
Summary:

- **without pruning: superlinear vs. horizon**
- **with pruning: linear vs. horizon**

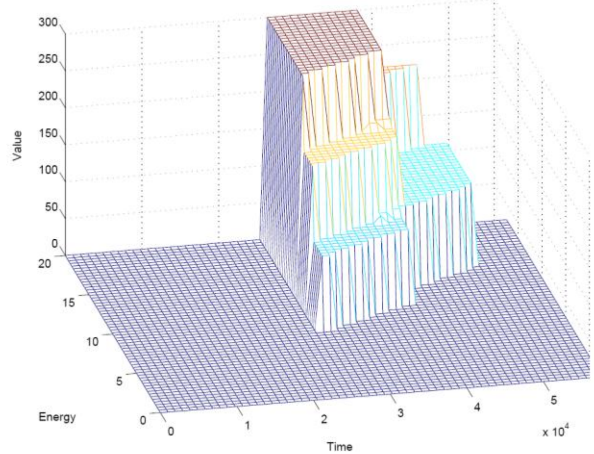
Worth the effort to prune!

Exact 3D Value Functions

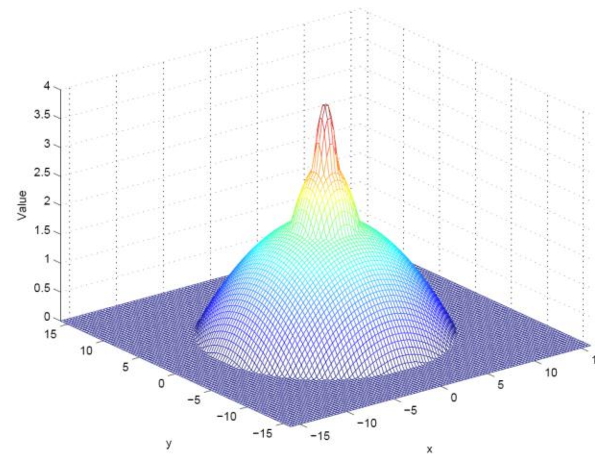
Knapsack



Mars Rover Linear



Mars Rover Nonlinear



Exact value functions in case form:

- **linear & nonlinear piecewise boundaries!**
- **nonlinear function surfaces!**

Continuous Actions

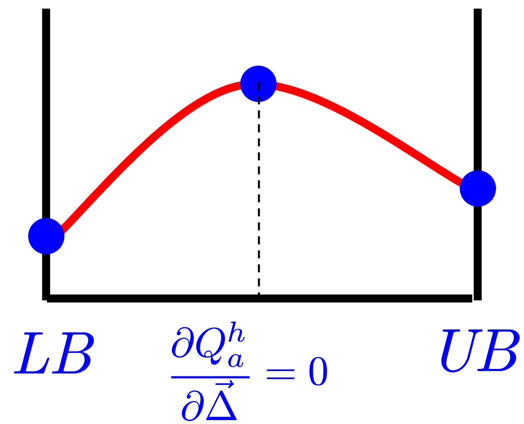
- Inventory control
  - Reorder based on stock, future demand
  - Action: $a(\vec{\Delta}); \vec{\Delta} \in \mathbb{R}^{|a|}$
- Need max $_\Delta$ in Bellman backup

$$V_{h+1} = \max_{a \in A} \max_{\vec{\Delta}} Q_a^{h+1}(\vec{b}, \vec{x}, \vec{\Delta})$$

- Track maximizing Δ substitutions to recover π

Max-out Case Operation

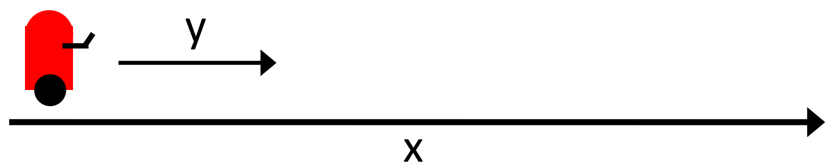
- $\max_x$ case(x) can be done partition-wise
  - In a *single* case partition …*max* w.r.t. critical points
    - Derive LB, UB in case form
    - Derivative Der0 in case form
    - max(case(x/LB), case(x/UB), case(x/Der0))
  - Can even track substitutions to recover optimal policy

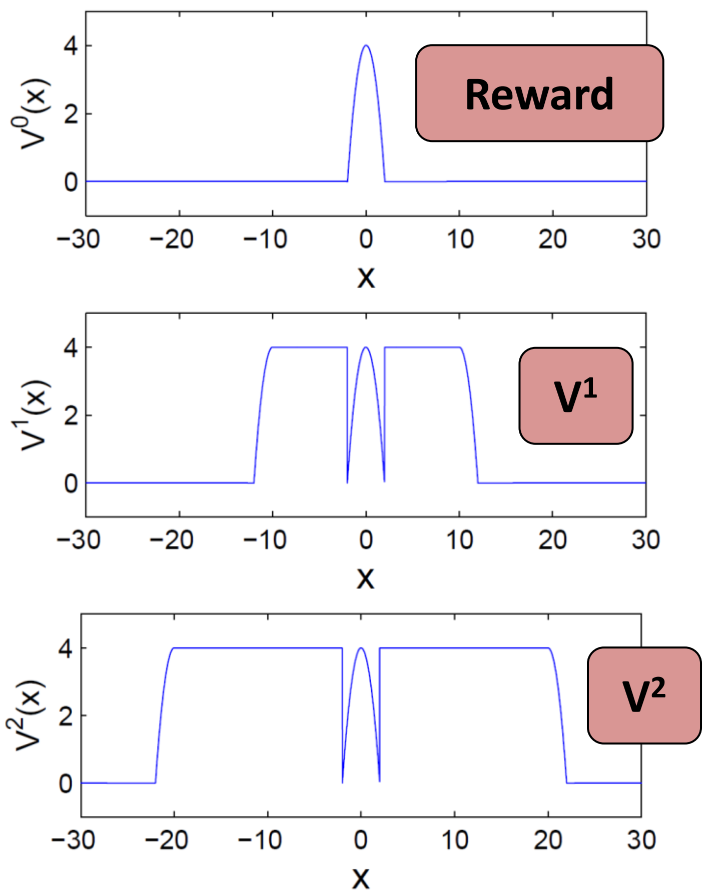


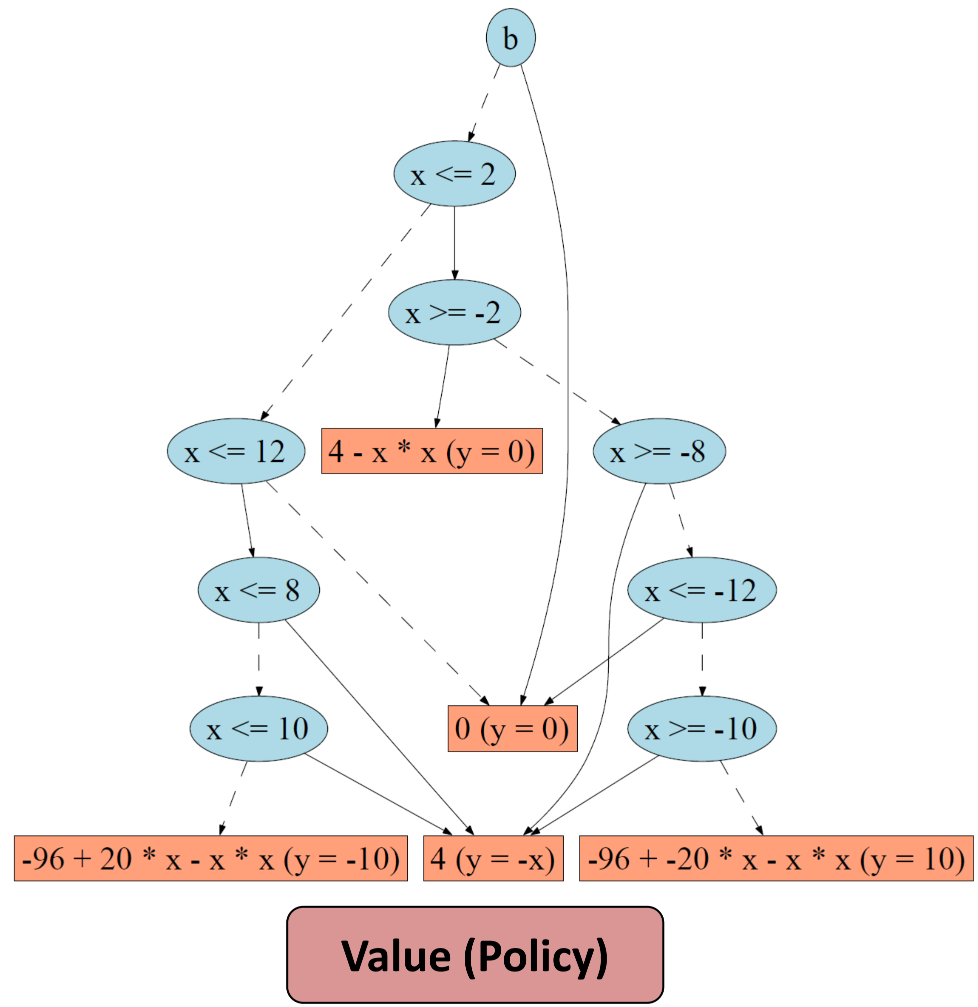
See AAAI 2012 (Zamni, Sanner, Fang) for details

First exact solutions to multivariate inventory in 50 years!

Illustrative Value and Policy







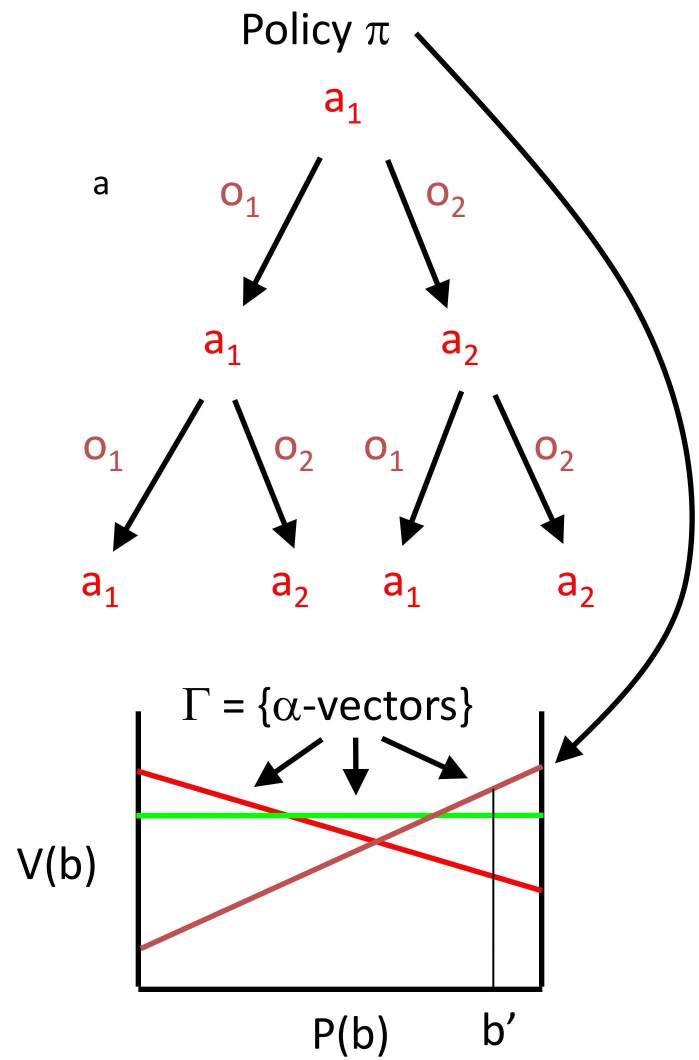
Fully Stochastic DC-MDP

- Add continuous noise ε to transitions
  - $x' = x + 2 + \varepsilon$
    - or $x' = x * \varepsilon + 2$
  - $\varepsilon \sim \mathcal{N}(\varepsilon; 0, \sigma^2)$
    - or $\varepsilon \sim \mathcal{N}(\varepsilon; f_1(x), f_2(x))$

  - Introduce intermediate vars ε for noise
    - Must be integrated out
    - Requires non-δ continuous integral $\int$
      - See AAAI-12 (Abbasnejad and Sanner) for $\int$ operation
      - Unfortunately **not closed-form** for SDP in MDPs ☹

Partially Observable – Continuous

- **POMDPs**
  - Standard discrete observation solution enumerates conditional policy trees
  - Continuous observations…
    - ∞ policy trees!
- **But in many cases…**
  - Policy only dependent upon **finite partitioning** of observation space
  - SDP methods allow one to **derive** this partitioning and apply discrete solutions!
    - If (temperature > 10)
      then …
      else …



Summary: Exact Solutions

- Solutions to continuous state (PO)MDPs
  - Discrete action MDPs **Sanner et al, UAI-11**
  - Continuous action MDPs (incl. exact policy) **Zamani et al, AAAI-12**
  - Extensions to full continuous noise
    - Initial work on required integration **Sanner et al, AAAI-12**
  - Discrete action, continuous observation POMDPs **In progress**

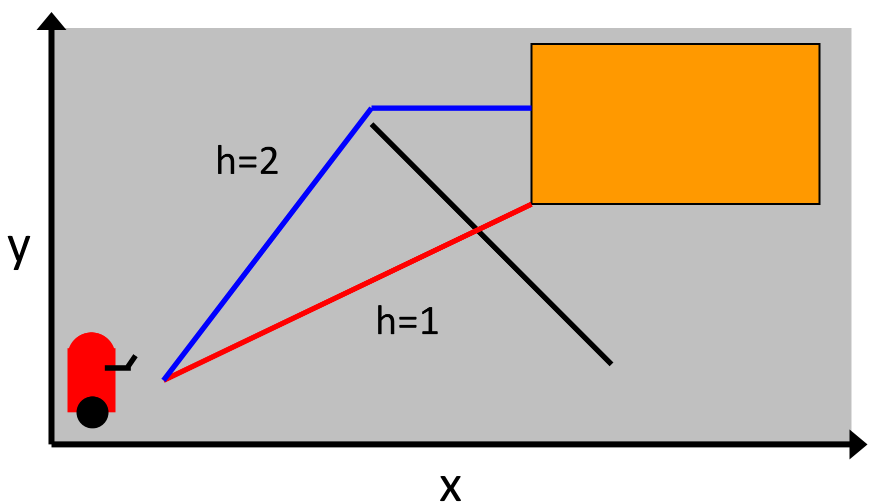
Part 2b: Solutions

Open problems
(some work in progress)

Nonlinearity and Continuous Actions

- Robotics
  - Need **nonlinear** cos, sin
  - Can use cubic spline

- General path planning
  - Not obvious, but requires **bilinear** constraints for obstacle specification



Real-time Dynamic Programming (RTDP)

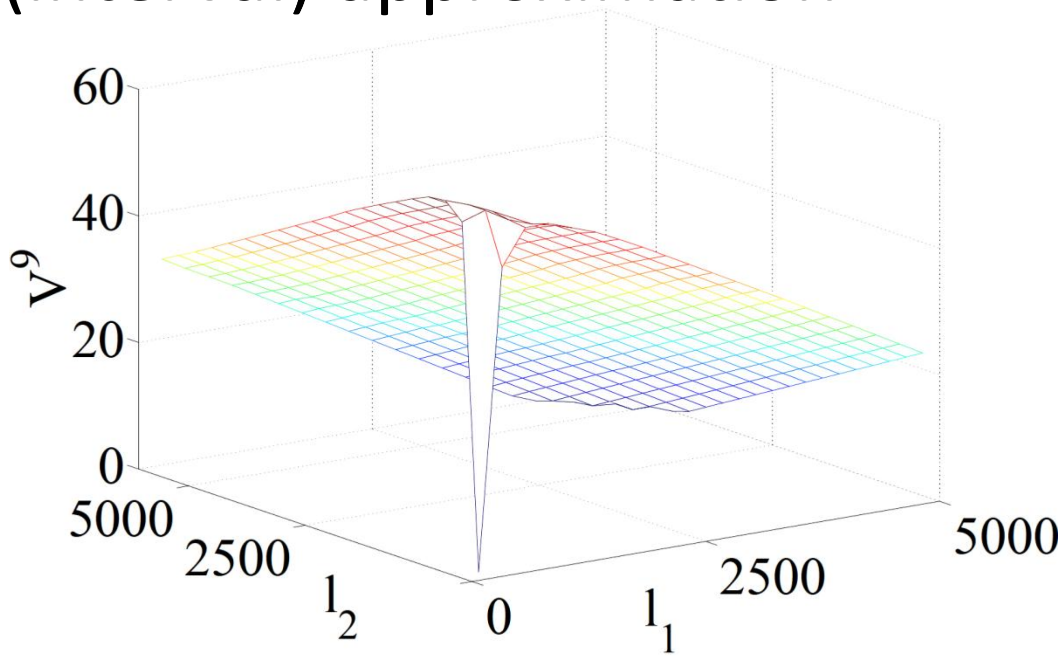
- ***Reachability*** and drawbacks of synch. DP (VI)

  - Better to think of ***relevance*** to optimal policy

- How to do RT-SDP for **continuous** problems?
  - HAO* (Meuleau et al, JAIR-09) provides some hints
  - Or instead do HAO* using SDP for DP operation

Approximation

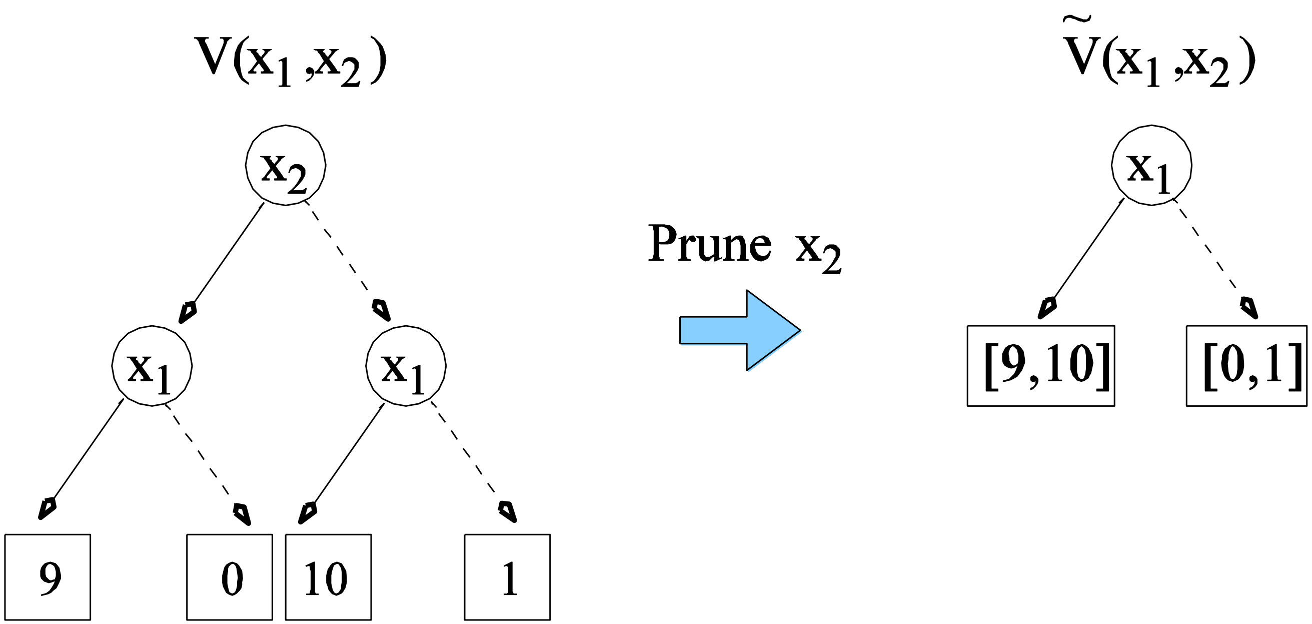
- Bounded (interval) approximation



  - This XADD has > 1000 nodes!
  - Should only require < 10 nodes!

Can use ADD to Maintain Bounds!

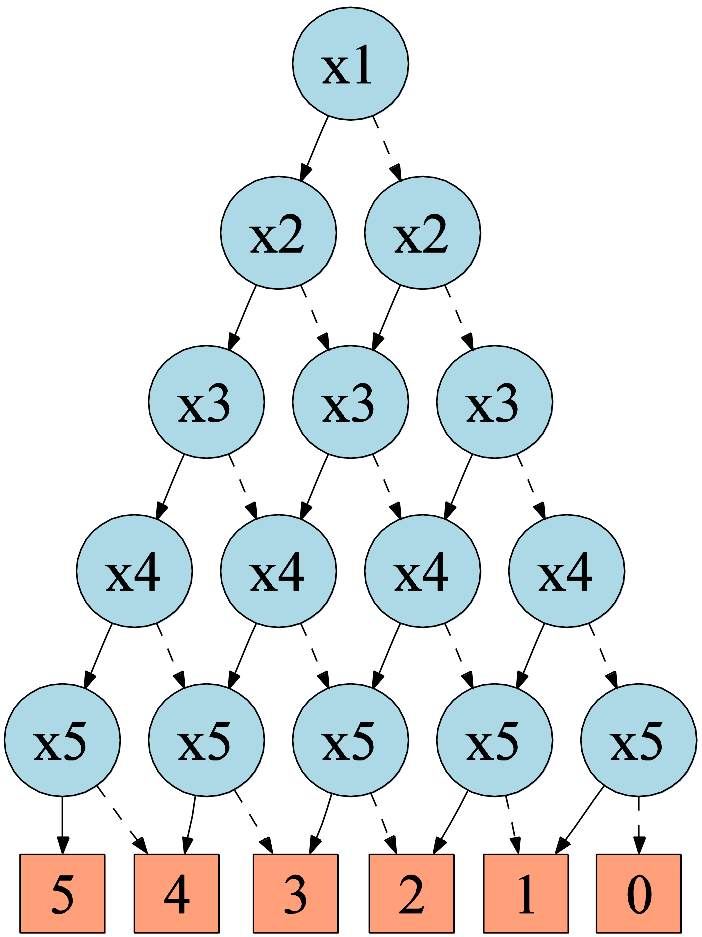
- Change leaf to represent range [L,U]
  - Normal leaf is like [V,V]
  - When merging leaves…
    - keep track of min and max values contributing

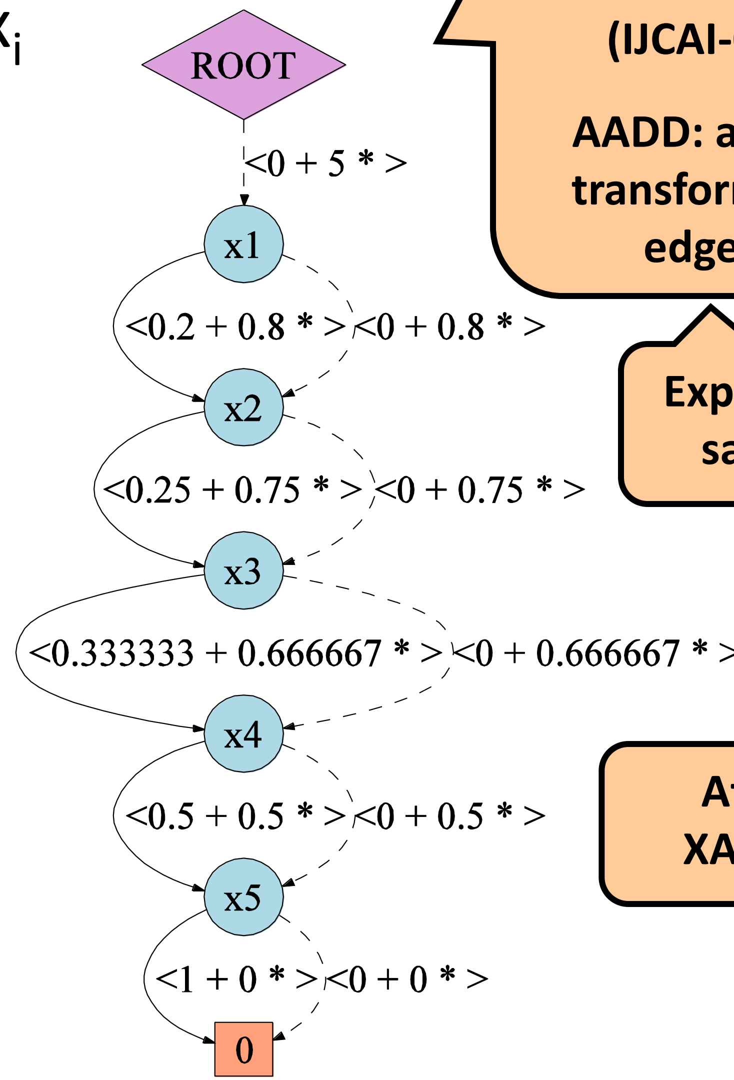


How to approximate for XADDs – expressions in decisions and leaves?

(X)ADDs vs. (X)AADDs

- Additive functions: $\sum_{i=1..n} x_i$





Sanner & McAllester (IJCAI-05)

AADD: affine transform on edges

Exponential savings!

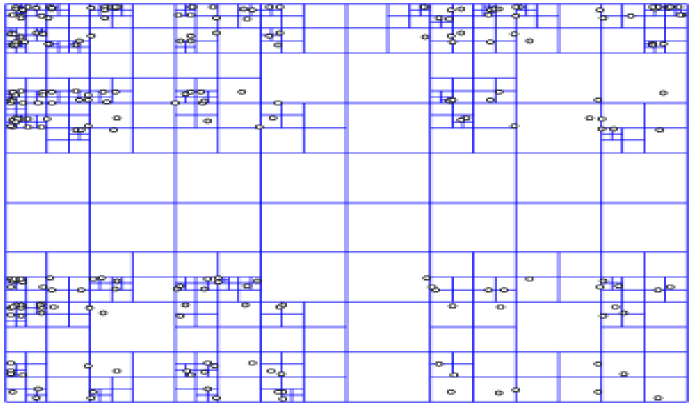
Affine XADDs?

Part 2c: Solutions

Survey of other methods

(Adaptive) Discretization

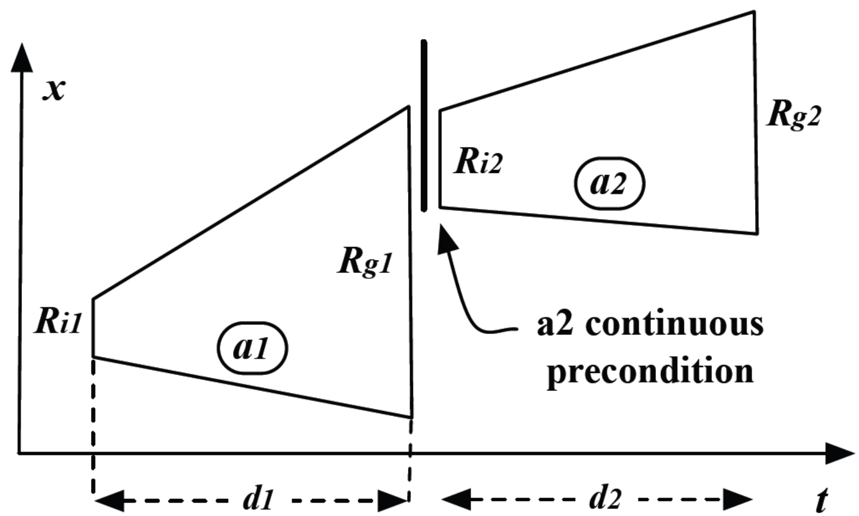
- Approximate by discretizing continuous variables
  - Then apply discrete solution!
  - Can often bound error, but O(N^D)
  - (Adaptively) discretize model:
    - Still O(N^D)
    - Adaptivity is an artform



  - Munos and Moore, MLJ 2002.
  - Nouri, Weinstein, Littman, NIPS 2008.

Search – Bounded

- Deterministic
  - Geometric reasoning
  - KongMing
    (Li, Williams,
    ICAPS 2008)
  - COLIN
    (Coles, Coles,
    Fox, Long,
    IJCAI 2009, JAIR 2012)



- Uncertainty
  - HAO* - AO* search using dynamic programming
    (extends previous DP methods to search!)

Search – Sampling

- UCT extremely effective for many MDPs
  - Maintain a partial tree for visited states
  - Treat each node in the tree as a bandit problem
    - Hence UCB for trees – UCT (Kocsis, Szepesvari, ECML 2006)

- Extensions of UCT for continuous actions and state
  - (Mansley, Weinstein, Littman, ICAPS 2012)

Direct Optimization

- Deterministic Planning
  - Extend SAT compilation to continuous variables
  - Use LP-SAT (SAT + linear constraints)
  - TM-LPSAT (Shin, Davis, AIJ 2005)
- Uncertain (MDP)
  - Approximate Bellman fixed point directly
  - (Kveton, Hauskrecht, and Guestrin, JAIR 2006)
  - Requires *a priori* knowledge of basis functions

Part 2d: Solutions

Connections to Control and Scheduling (very brief)

Control

- Overlap with (PO)MDPs for discrete time control
  - Almost always have **continuous actions** in control
    - E.g., servos
    - But rarely discrete time

- Different problem for continuous time control
  - Modeled as partial differential equations (PDEs)
    - E.g., airplane stabilizer control
  - Policy must be **continuous time** as well
    - Not act and wait until next time step
    - But apply **continuous control signal as a function of observation inputs**
    - Rely on specialized PDE solutions

Scheduling

- Cornerstone of scheduling is **concurrency**
  - Deliveries
  - Factory processes

- But importantly: **asynchronous concurrency**
  - Processes start and end at different times
  - Not well-modeled as synchronous, discrete time (PO)MDP

    - If **not stochastic**, can view from constraints perspective (Bartak's tutorial)

    - If **stochastic**, might consider Generalized Semi-(PO)MDPs (Younes and Simmons, AAAI 2004)

Summary of Part 2: Solutions

- Express model in language of your choice (RDDL!)
  - Compile to a factored (PO)MDP
- Exact dynamic programming for factored (PO)MDPs
  - Important to say what optimal solution looks like!
  - Many open problems for bounded / exact solutions
- Not all problems can be solved exactly
  - Useful to take hints from heuristic / approximation literature
  - Much work to be done in generalizing discrete MDP techniques
- In some cases (control and continuous time scheduling), factored MDP and POMDP insufficient
  - Need to extend or seek alternate models

Tutorial Summary

- Many **real-world planning problems require continuous models**
  - Need compact, expressive languages (e.g., RDDL)
- Need to **understand exact solutions** & limits
- Need to develop **effective practical solutions**
  - Wide open area for research