

AAAI 2024 Lab

Introduction to Domain Modeling in RDDL

Part 1: Language Overview

Scott Sanner and Ayal Taitler



Multiple Target Audiences

- ICAPS folks familiar with (P)PDDL wondering what RDDL is and when they might use it
- Planning language agnostics who are simply interested in planning for MDPs and POMDPs
- RL researchers interested in how to specify and exploit complex model structure

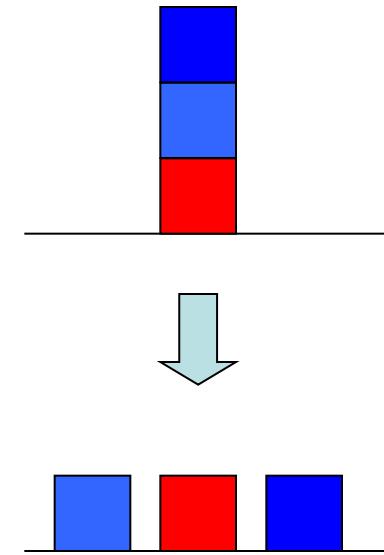
RDDL Tutorial Outline

- Part 1: Language Overview
 - What is probabilistic planning in PPDDL?
 - Why do we need RDDL?
 - RDDL by example
 - Overview of RDDL solution methodologies
- Part 2: PyRDDLGym

Stochastic Domain Languages as of 2009

- **Probabilistic PDDL (PPDDL)**
 - more expressive than PSTRIPS
 - for example, *probabilistic universal* and *conditional effects*:

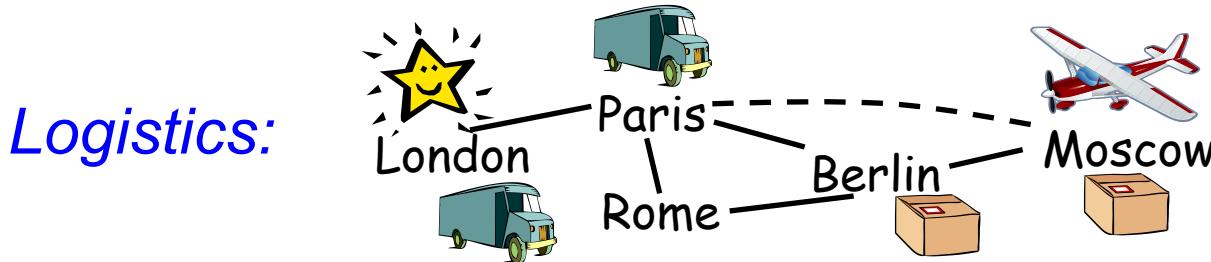
```
(:action put-all-blue-blocks-on-table
  :parameters ()
  :precondition ()
  :effect (prob 0.9
    (forall (?b)
      (when (Blue ?b)
        (not (OnTable ?b))))))
```



- **Idea:** make some effects stochastic
- **Question:** is this sufficient to model realistic problems?

More Realistic: Logistics?

- PPDDL Description:



```
(:action load-box-on-truck-in-city  
:parameters (?b - box ?t - truck ?c – city)  
:precondition (and (BIn ?b ?c) (TIn ?t ?c))  
:effect (prob 0.7 (and (On ?b ?t) (not (BIn ?b ?c))))
```

- Can instantiate problems for any domain objects
 - 3 trucks: 2 planes: 3 boxes:
- But wait... only one truck can move at a time???
 - No concurrency, no time: **will FedEx care?**

Expressivity Limitations of PPDDL

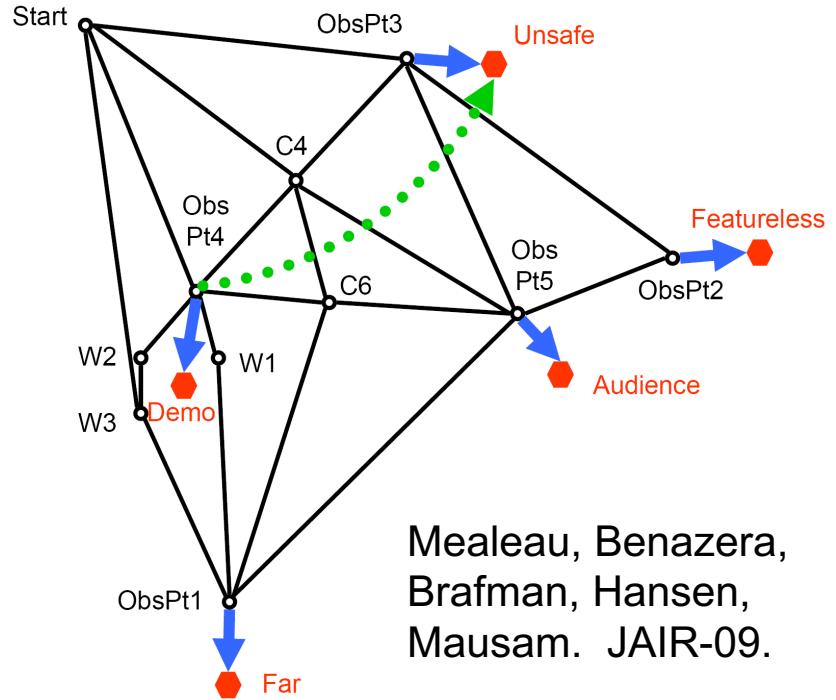
- Many PPDDL domains were tweaks of PDDL domains
 - Recipe: add success probability on some effects
 - e.g., $\text{load-plane}(p,x)$ succeeds with prob 0.9
 - IPPC 2004/6, could win by determinizing / replanning
 - led to work on “probabilistically interesting” PPDDL problems (Little & Thiebaux, 2007)
- But what stochastic expressiveness is needed for modeling real-world domains?
 - Then we can ask what language is appropriate

Observation

- Planning languages direct 5+ years of research
 - PDDL and variants
 - Probabilistic PDDL (PPDDL)
- Why?
 - Domain design is time-consuming
 - So everyone (students) use existing benchmarks
 - Need for comparison
 - Planner code not always released
 - Only means of comparison is on competition benchmarks
- Implication:
 - We should choose our languages & problems well
 - Let's ask what problems we want to model / solve

What probabilistic problems
might we want to model?

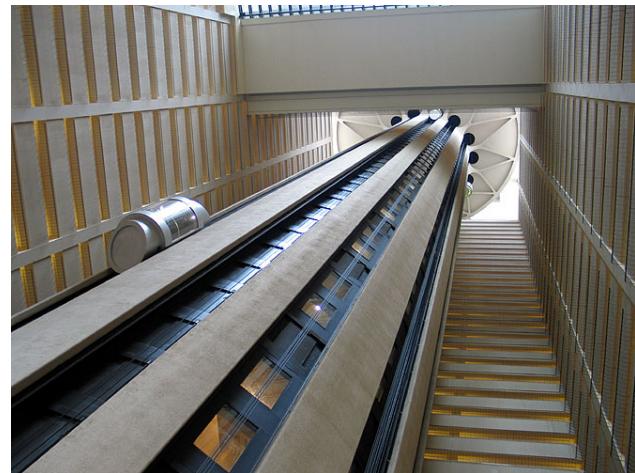
Mars Rovers



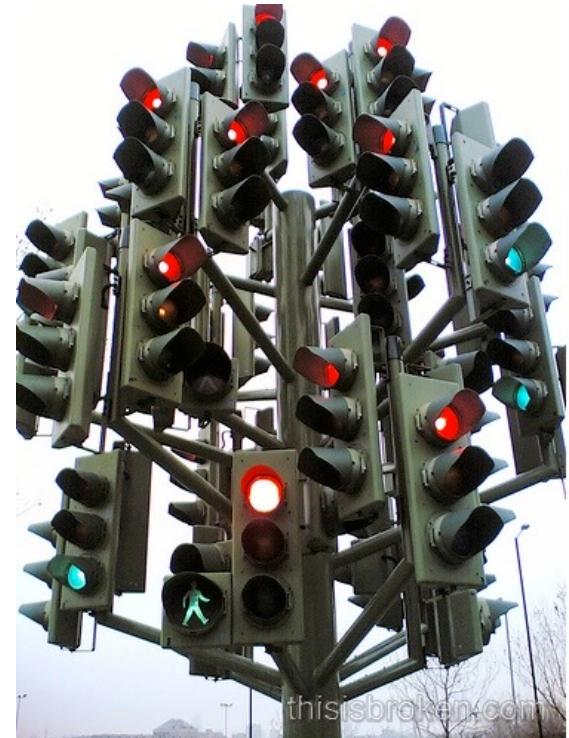
- **Continuous**
 - Time, robot position / pose, sun angle, battery reserves...
- **Partially observable**
 - Even worse: high-dimensional partially observable

Elevator Control

- **Concurrent Actions**
 - Elevator: up/down/stay
 - 6 elevators: 3^6 actions
- **Exogenous / Non-boolean**
 - Random integer arrivals (e.g., Poisson) at every floor
- **Complex Objective**
 - Minimize sum of wait times
 - Could even be nonlinear function (squared wait times)
- **Complex Action Constraints**
 - People might get annoyed if elevator reverses direction



Traffic Control



- Concurrent
 - Multiple lights
- Indep. Exogenous Events
 - Multiple vehicles
- Continuous Variables
 - Nonlinear dynamics
- Partially observable
 - Only observe stoplines

RDDL Tutorial Outline

- Part 1: Language Overview
 - What is probabilistic planning in PPDDL?
 - Why do we need RDDL?
 - RDDL by example
 - Overview of RDDL solution methodologies
- Part 2: PyRDDLGym

What are we missing in PPDDL?

- Independent concurrent stochastic actions & events
 - Exogenous events that scale with domain size
 - Random person arrivals at elevator floors, traffic movement
 - Resolution of stochastic or concurrent event conflicts
 - Two elevators admit passengers from same floor
 - Preconditions over joint actions (not per action)
 - Joint traffic light configurations must adhere to safety constraints
- Remedy: action-centric (P)PDDL → fluent-centric RDDL

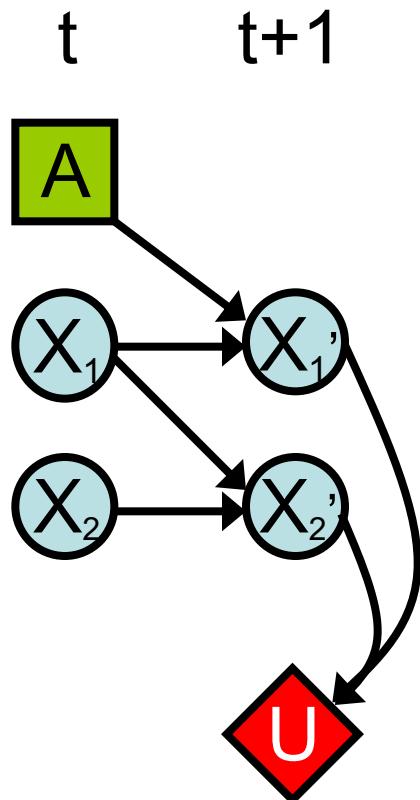
Need expressive decision-making formalism that supports complex stochastic fluent updates

Relational Dynamic Bayes Net
+ Influence Diagram (RDDL)

a.k.a. Relational Factored MDP

Dynamical Models & Influence Diagrams

- Dynamic Bayes Nets (DBNs) ...

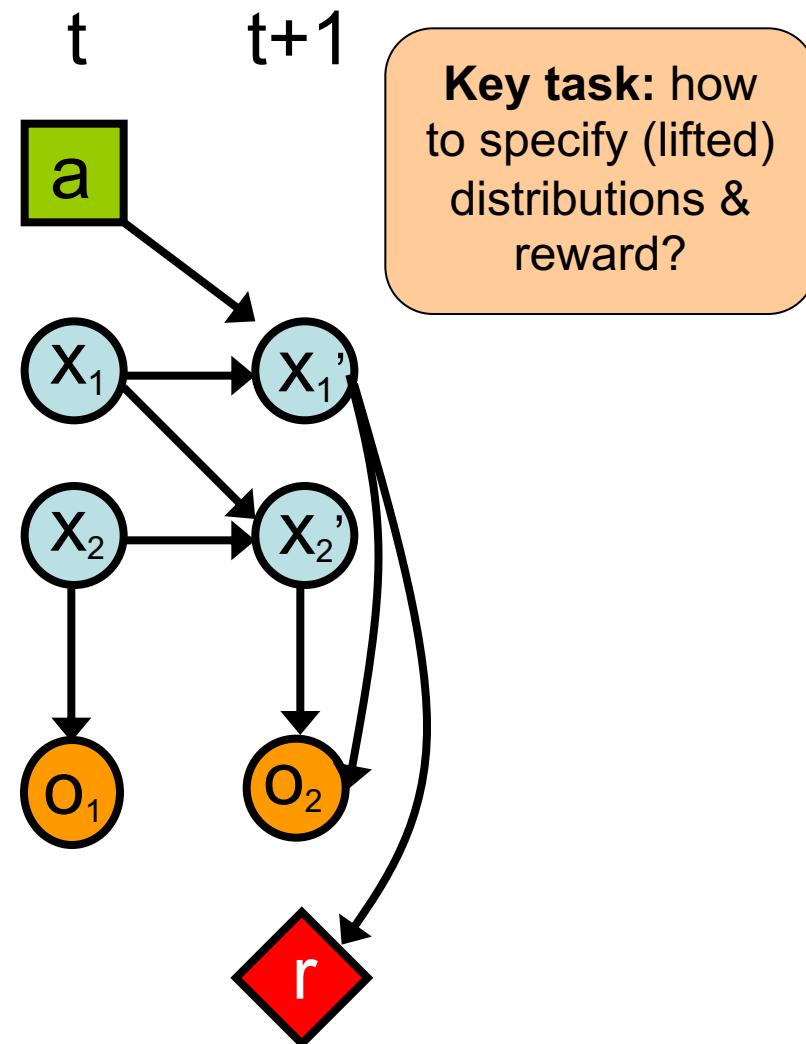


- Influence Diagrams (IDs)...

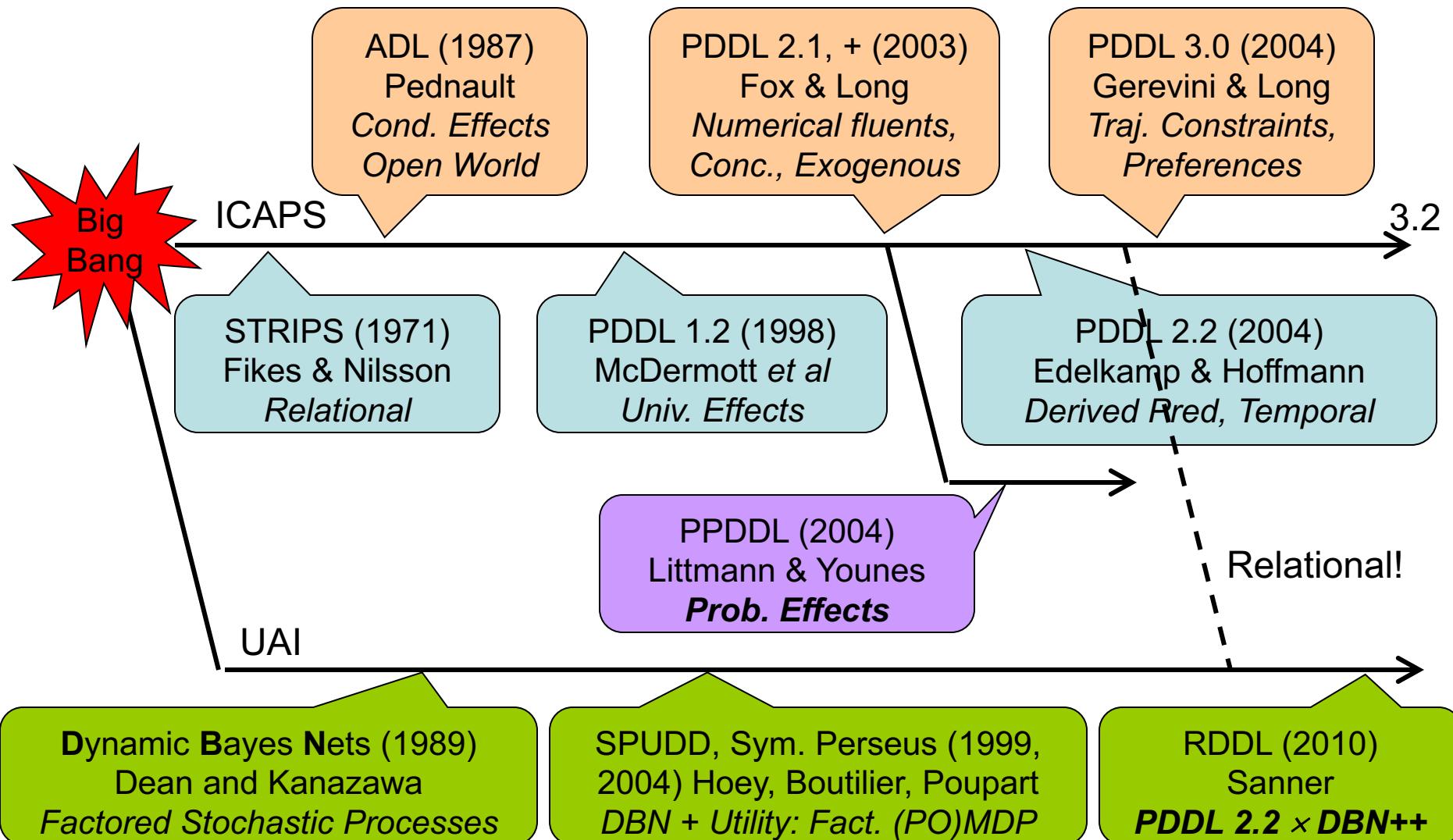
- Action nodes [squares]
 - Not random variables
 - Rather “controlled” variables
- Utility nodes <diamonds>
 - A utility conditioned on state, e.g.
 $U(X_1', X_2') = \text{if } (X_1' = X_2') \text{ then } 10 \text{ else } 0$

What is RDDL?

- Relational Dynamic Influence Diagram Language
 - Relational [DBN + Influence Diagram]
- Think of it as a Relational Factored (PO)MDP
 - Fluent updates are probabilistic programs



A Brief History of (ICAPS) Time



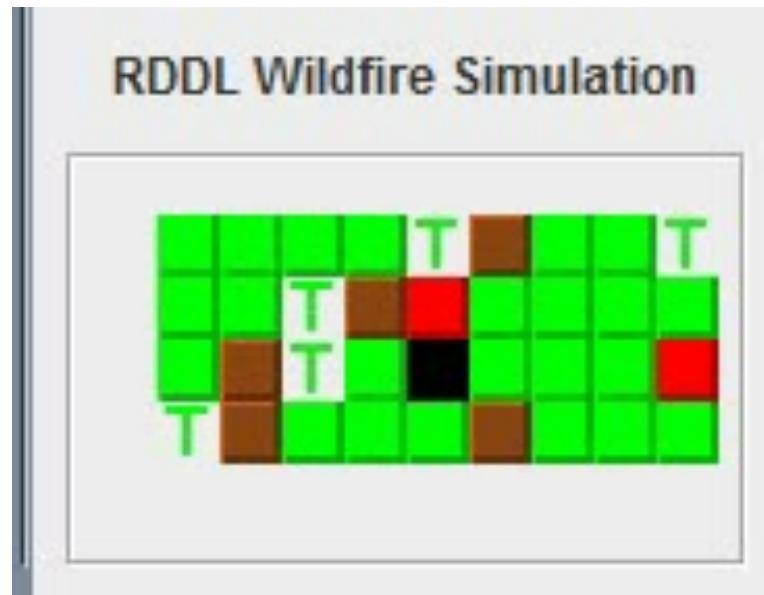
RDDL Tutorial Outline

- Part 1: Language Overview
 - What is probabilistic planning in PPDDL?
 - Why do we need RDDL?
 - **RDDL by example**
 - Overview of RDDL solution methodologies
- Part 2: PyRDDLGym

Example:

How to specify a problem in
RDDL (that cannot be
expressed in PPDDL)

Wildfire Domain



- Contributed by Zhenyu Yu (School of Economics and Management, Tongji University)
 - Karafyllidis, I., & Thanailakis, A. (1997). *A model for predicting forest fire spreading using gridular automata*. Ecological Modelling, 99(1), 87-97.

Wildfire in RDDL

Each cell may independently
stochastically ignite

```
cpfs {  
  
burning'(?x, ?y) =  
    if ( put-out(?x, ?y) )  
        then false  
    else if (~out-of-fuel(?x, ?y) ^ ~burning(?x, ?y))  
        then Bernoulli( 1.0 / (1.0 + exp[4.5 - (sum{?x2: x_pos, ?y2: y_pos}  
            (NEIGHBOR(?x, ?y, ?x2, ?y2) ^ burning(?x2, ?y2)))]) )  
    else  
        burning(?x, ?y); // State persists  
  
out-of-fuel'(?x, ?y) = out-of-fuel(?x, ?y) | burning(?x, ?y);  
  
};  
  
reward =  
    [sum{?x: x_pos, ?y: y_pos} [ COST_CUTOFF*cut-out(?x, ?y) ]]  
+ [sum{?x: x_pos, ?y: y_pos} [ COST_PUTOFF*put-out(?x, ?y) ]]  
+ [sum{?x: x_pos, ?y: y_pos} [ COST_NONTARGET_BURN*[ burning(?x, ?y) ^ ~TARGET(?x, ?y) ]]  
+ [sum{?x: x_pos, ?y: y_pos}  
    [ COST_TARGET_BURN*[ (burning(?x, ?y) | out-of-fuel(?x, ?y)) ^ TARGET(?x, ?y) ]]];
```

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);

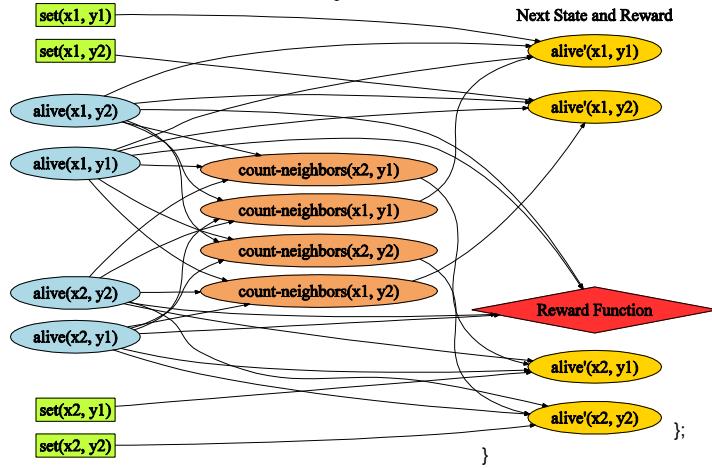
    NEIGHBOR(x1,y2,x1,y1);
    NEIGHBOR(x1,y2,x2,y1);
    NEIGHBOR(x1,y2,x2,y2);

    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);
};

}
```

Current State and Actions



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,y1,x3,y1);
    NEIGHBOR(x1,y1,x3,y2);
    NEIGHBOR(x1,y1,x3,y3);

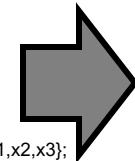
    NEIGHBOR(x1,y2,x1,y1);
    NEIGHBOR(x1,y2,x2,y1);
    NEIGHBOR(x1,y2,x2,y2);
    NEIGHBOR(x1,y2,x3,y1);
    NEIGHBOR(x1,y2,x3,y2);
    NEIGHBOR(x1,y2,x3,y3);

    NEIGHBOR(x1,y3,x1,y2);
    NEIGHBOR(x1,y3,x2,y1);
    NEIGHBOR(x1,y3,x2,y2);
    NEIGHBOR(x1,y3,x3,y1);
    NEIGHBOR(x1,y3,x3,y2);
    NEIGHBOR(x1,y3,x3,y3);

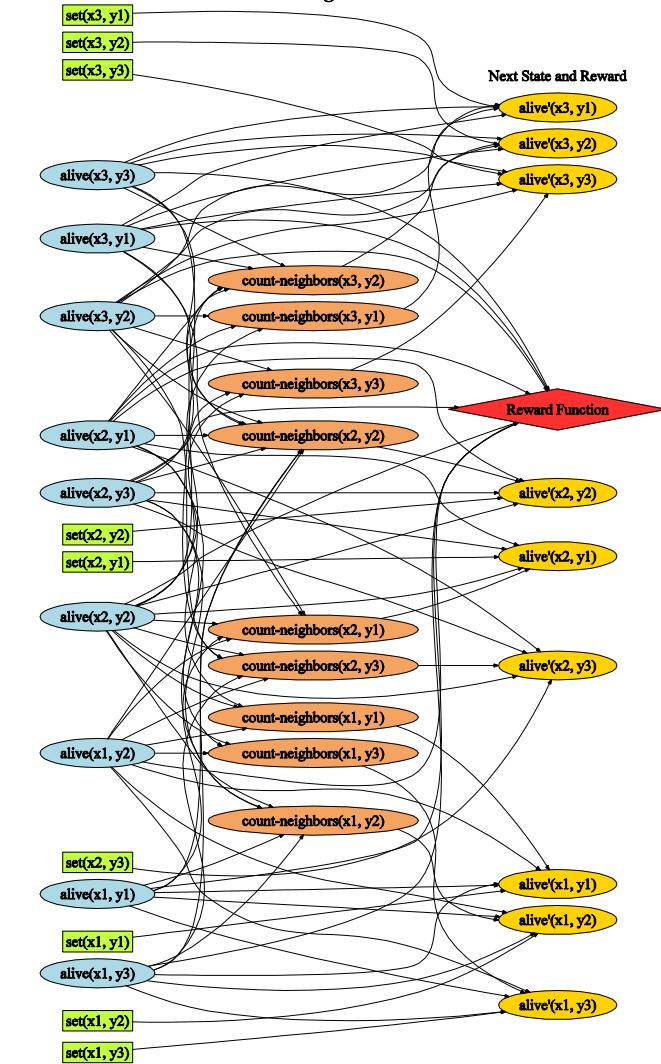
    NEIGHBOR(x2,y1,x1,y1);
    NEIGHBOR(x2,y1,x2,y1);
    NEIGHBOR(x2,y1,x3,y1);
    NEIGHBOR(x2,y2,x1,y1);
    NEIGHBOR(x2,y2,x2,y1);
    NEIGHBOR(x2,y2,x3,y1);
    NEIGHBOR(x2,y3,x1,y1);
    NEIGHBOR(x2,y3,x2,y1);
    NEIGHBOR(x2,y3,x3,y1);

    NEIGHBOR(x3,y1,x1,y2);
    NEIGHBOR(x3,y1,x2,y2);
    NEIGHBOR(x3,y1,x3,y2);
    NEIGHBOR(x3,y2,x1,y2);
    NEIGHBOR(x3,y2,x2,y2);
    NEIGHBOR(x3,y2,x3,y2);
    NEIGHBOR(x3,y3,x1,y2);
    NEIGHBOR(x3,y3,x2,y2);
    NEIGHBOR(x3,y3,x3,y2);
};

}
```



Current State and Actions

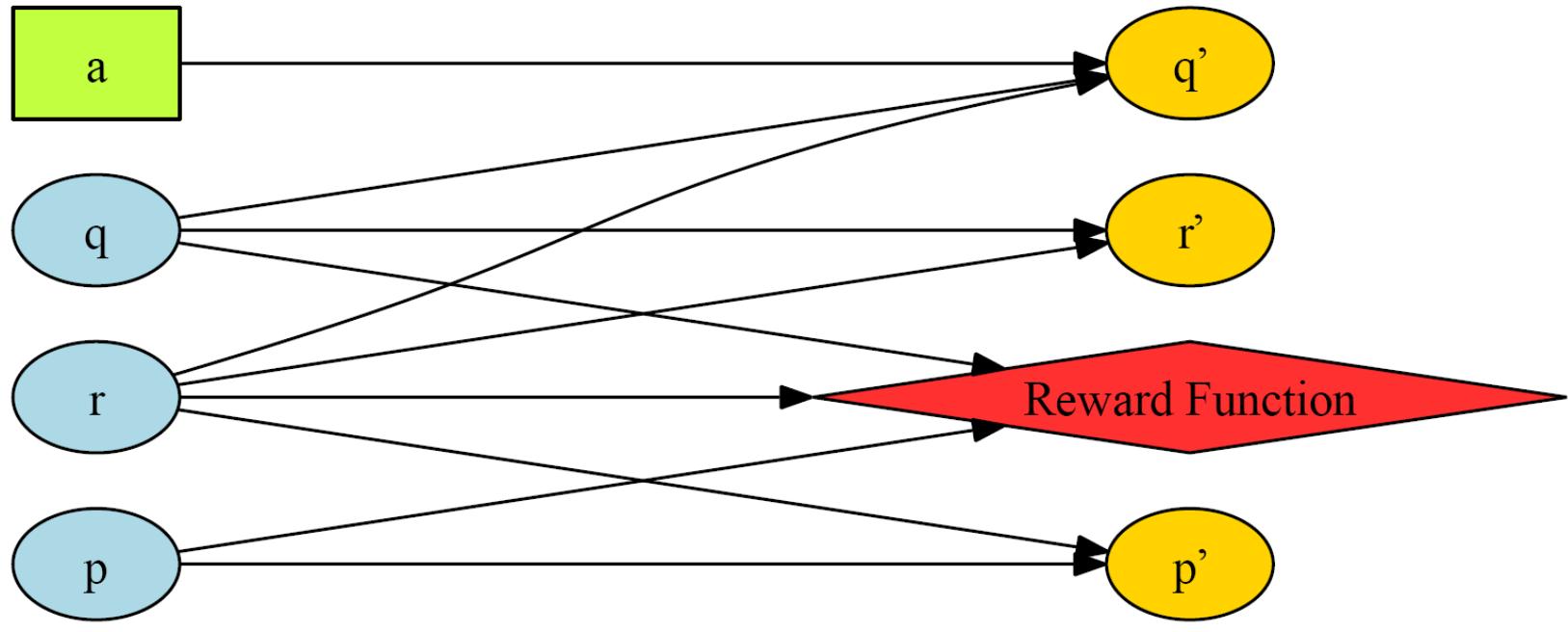


We're getting ahead of
ourselves

Let's see how RDDL can specify
a binary discrete DBN+ID

How to Represent Factored MDP?

Current State and Actions



Next State and Reward

p	r	p'	$P(p' p,r)$
<i>true</i>	<i>true</i>	<i>true</i>	0.9
<i>true</i>	<i>true</i>	<i>false</i>	0.1
<i>true</i>	<i>false</i>	<i>true</i>	0.3
<i>true</i>	<i>false</i>	<i>false</i>	0.7
<i>false</i>	<i>true</i>	<i>true</i>	0.3
<i>false</i>	<i>true</i>	<i>false</i>	0.7
<i>false</i>	<i>false</i>	<i>true</i>	0.3
<i>false</i>	<i>false</i>	<i>false</i>	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 action
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);
};

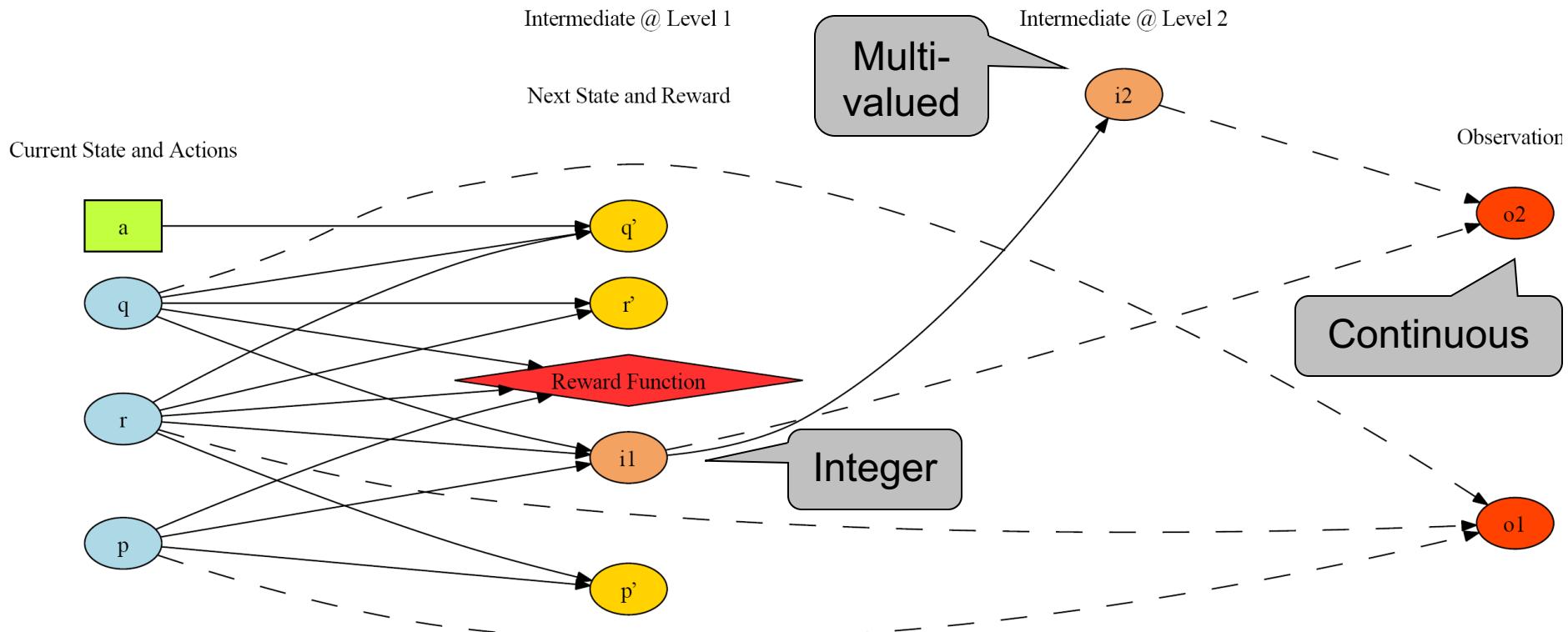
// Define the reward function; note that boolean functions are
// treated as 0/1 integers in arithmetic expressions
reward = p + q - r;
```

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

Let's look at a few more RDDL ingredients

- enum, integer, continuous fluents
- intermediate fluents
- observation fluents (POMDP)
- more control / stochastic constructs

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, };
    i2 : { interm-fluent, enum_level };

    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);
}
```

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  
            );
```

Multi-valued

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

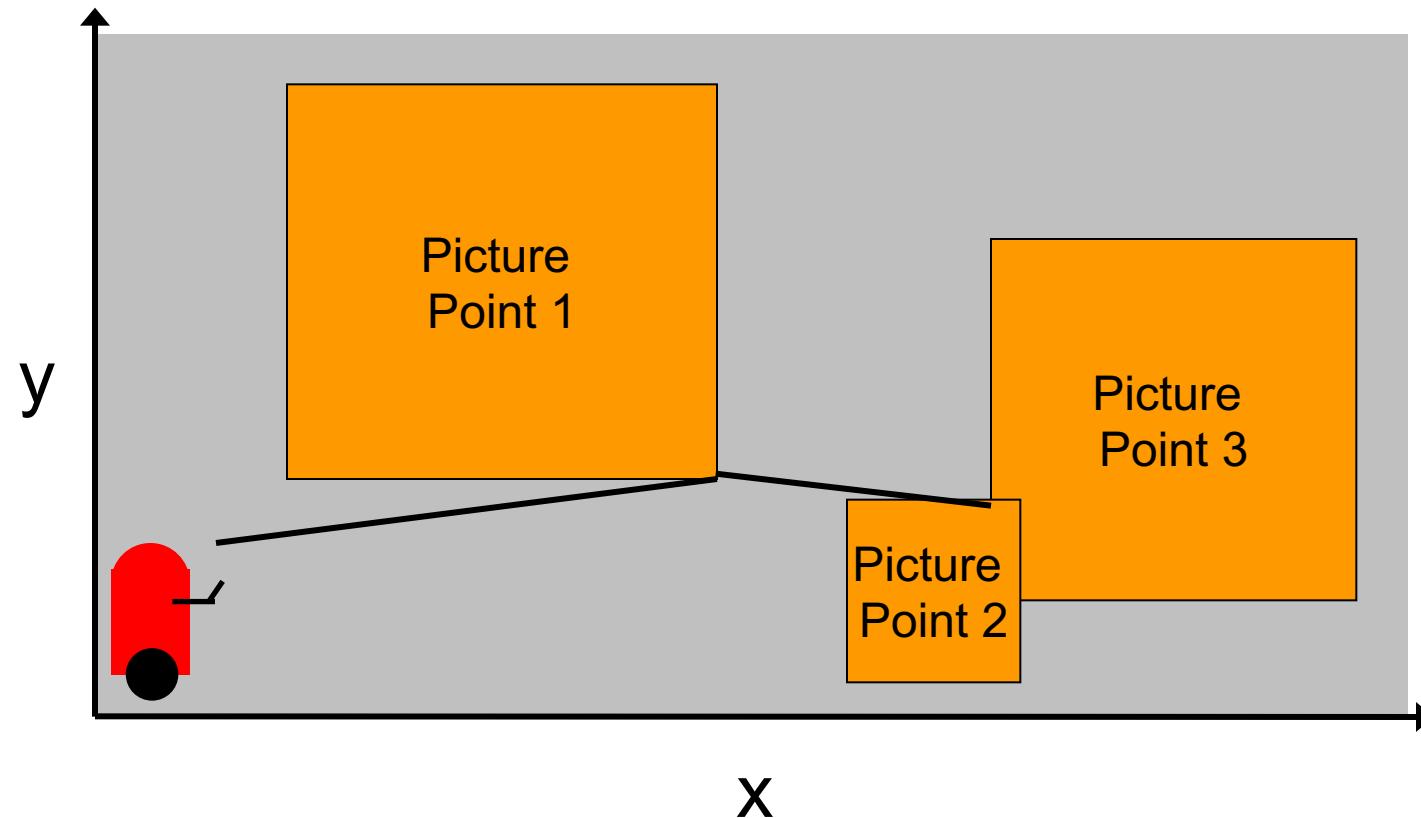
Mixture of
Normals

Variance comes from other
previously sampled variables

Finally: Mars Rover example

- lifting
- non-fluents
- aggregation expressions
- joint action preconditions

Lifted Continuous MDP in RDDL: Simple Mars Rover



Simple Mars Rover: Part I

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

```
pvariables {
```

Constant
picture
points,
bounding box

```
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 };
```

Rover position
(only one
rover)
and time

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

Rover
actions

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

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);
```

White noise, variance
proportional to distance moved

```
// Time update
```

```
time' = if (snapPicture)
```

Fixed time for picture

```
then (time + 0.25)
```

```
else (time +
```

```
[if (xMove > 0) then xMove else -xMove] +  
[if (yMove > 0) then yMove else -yMove]);
```

Time proportional to
distance moved

```
};
```

Simple Mars Rover: Part III

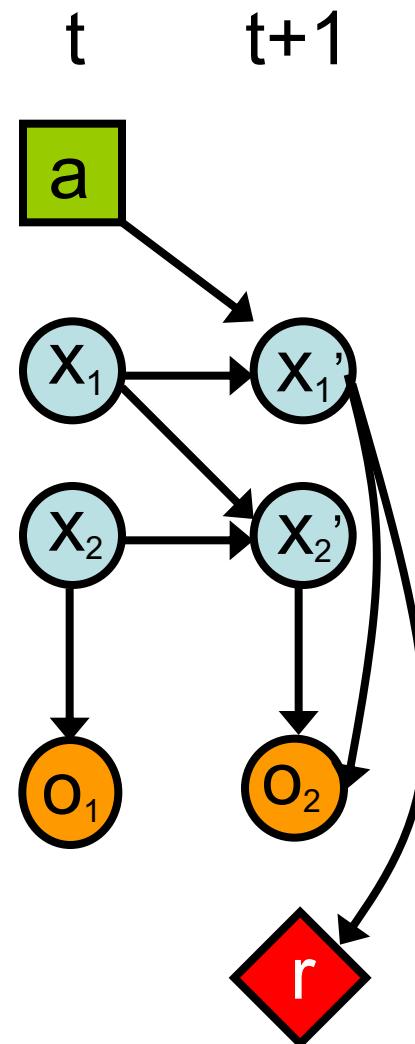
```
// We get a reward for any picture taken within picture box error bounds  
// and the time limit.  
reward = if (snapPicture ^ (time <= MAX_TIME))  
    then sum_{?p : picture-point} [  
        if ((abs[ PICT_XPOS(?p) – xPos] <= PICT_ERROR_ALLOW(?p))  
            ^ (abs[ PICT_YPOS(?p) – yPos] 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

- Relational Dynamic Influence Diagram Language
 - Relational [DBN + Influence Diagram]
- Specify the probabilistic process over relations to generate next state
 - Generate “ground” DBN+ID given domain object instantiation



RDDL Recap 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, ...)
- Flexible fluent types
 - Binary (predicate) fluents
 - Multi-valued (enumerated) fluents
 - Integer and continuous fluents (from PDDL 2.1)

RDDL Recap II

- Semantics is ground DBN + Influence Diagram
 - Naturally supports independent exogenous events
- General expressions in transition / reward
 - Logical expressions ($\wedge, \vee, \Rightarrow, \Leftrightarrow, \forall, \exists$)
 - Arithmetic expressions ($+, -, *, /, \sum_x, \prod_x$)
 - In/dis/equality comparison expressions ($=, \neq, <, >, \leq, \geq$)
 - Conditional expressions (if-then-else, switch)
 - Standard Functions: $\text{pow}[\cdot]$, $\text{log}[\cdot]$, $\text{abs}[\cdot]$, $\text{max}[\cdot]$, $\text{sin}[\cdot]$
 - Basic probability distributions
 - Bernoulli, Discrete, Normal, Poisson

Logical expr. {0,1}
so can use in
arithmetic expr.

RDDL Recap III

- Goal + General (PO)MDP objectives
 - Arbitrary reward
 - goals, numerical preferences (c.f., PDDL 3.0)
 - Finite horizon
 - Discounted or undiscounted
- State/action constraints
 - Encode legal action-preconditions
 - (concurrent) action preconditions
 - Assert state-invariants
 - serve as integrity constraint checks on state
 - e.g., an elevator cannot be in two locations

What RDDL does not do...

- RDDL just provides a language for specifying complex (PO)MDPs
 - For an MDP: $\langle S, A, T, R \rangle$
 - For a POMDP: $\langle S, A, T, R, O, Z \rangle$
- RDDL does not define a policy
- RDDL does not specify a planning methodology
 - It's up to **external planners** to perform planning, learning, or inference on the RDDL domain model

RDDL Tutorial Outline

- **Part 1: Language Overview**
 - What is probabilistic planning in PPDDL?
 - Why do we need RDDL?
 - RDDL by example
 - **Overview of RDDL solution methodologies**
- **Part 2: PyRDDLGym**

Common question from RL crowd: Why RDDL vs. a Simulator in C++?

Answer: Want a language that can be **compiled** into other formalisms for planning and domain analysis such as abstraction (e.g., Tensorflow / PyTorch / Jax, XADDs, Gurobi – constrained optimization).

RDDL is a disciplined subset of modern languages designed to facilitate compilation.

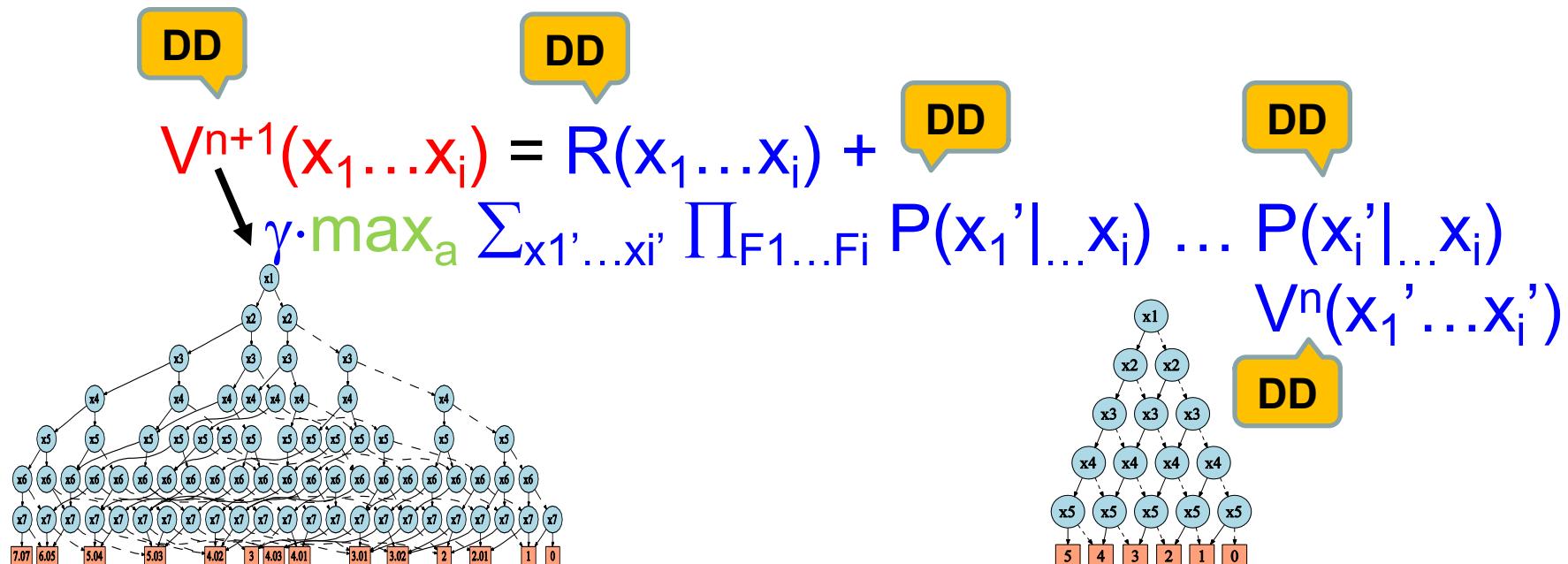
RDDL Planning Overview

- SOTA: **compile instance to planning formalism**
 - MCTS (Discrete Search) (PROST, Keller et al, ICAPS-12) – discrete only
 - Symbolic Methods (Decision Diagrams – XADDs)
 - Planning by Backprop (Tensorplan, JaxPlan, SOGBOFA)
 - Planning by Optimization in Gurobi (Raghavan et al, AAAI-17)
- Generalized Planning: “solve” at **lifted domain level**
 - Relational / First-order MDPs (Kharden et al, Sanner et al)
 - Graph neural network policies (Symnet 1/2/3: Mausam et al)
 - Plan / policy should work for all instances

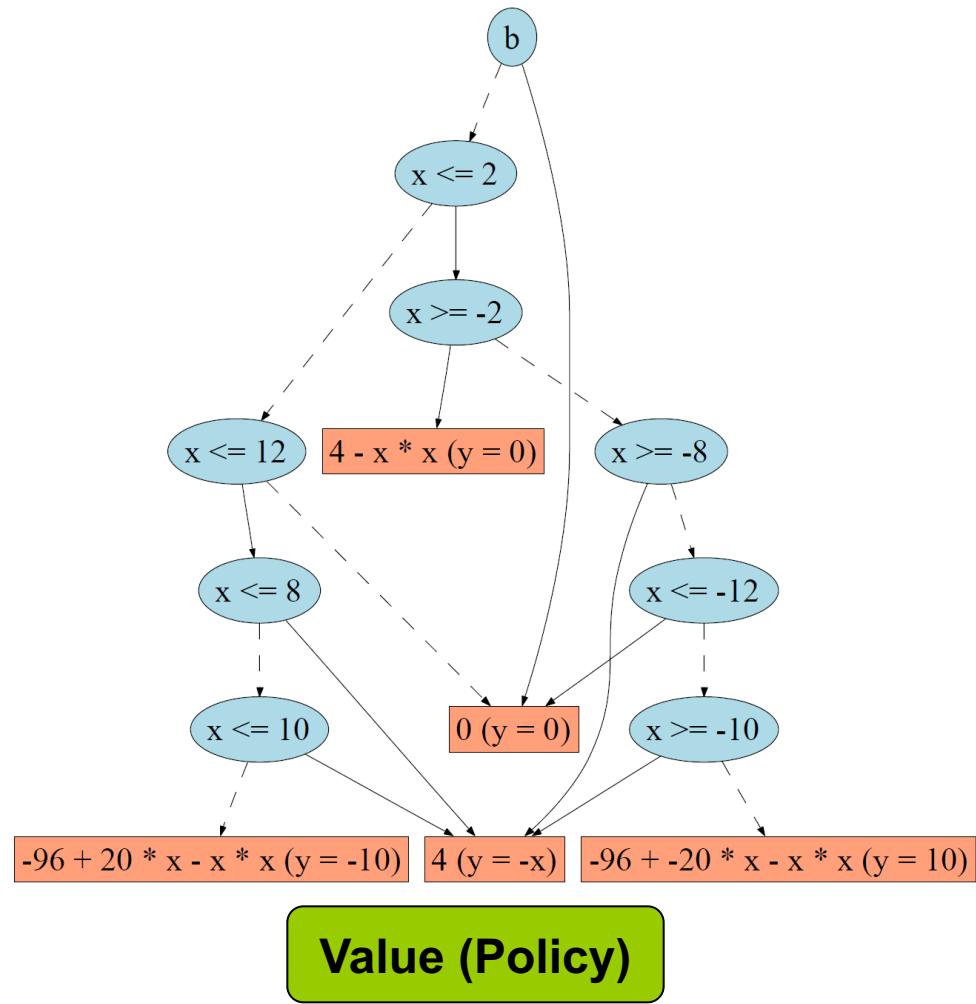
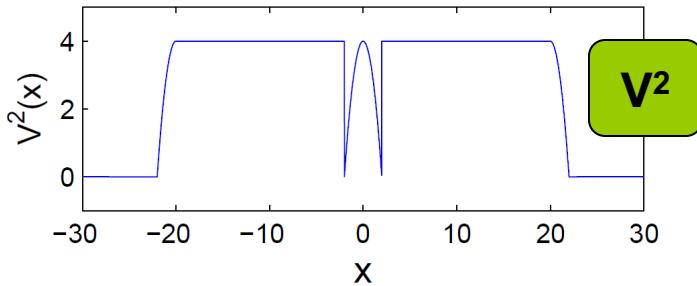
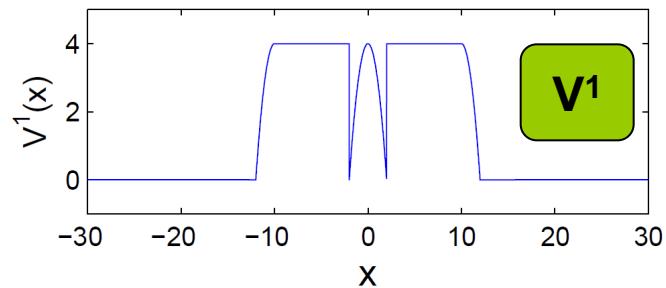
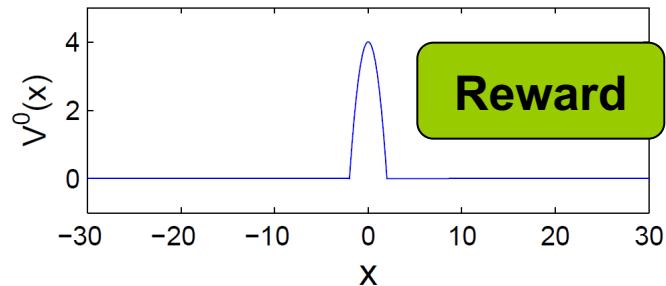
Symbolic Decision Diagram Methods

SPUDD for Factored MDPs

- Value Iteration using ADDs (SPUDD)
 - Can use ADDs or any DD that supports +,* ,max
 - Bounded approximations (APRICODD)

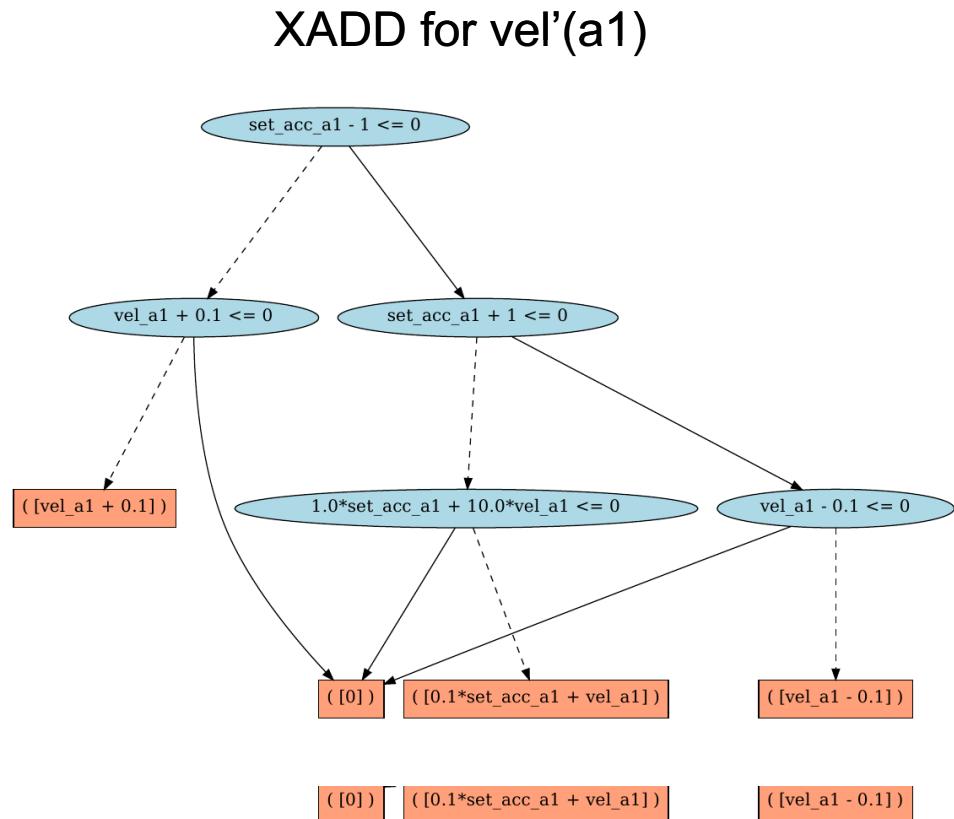
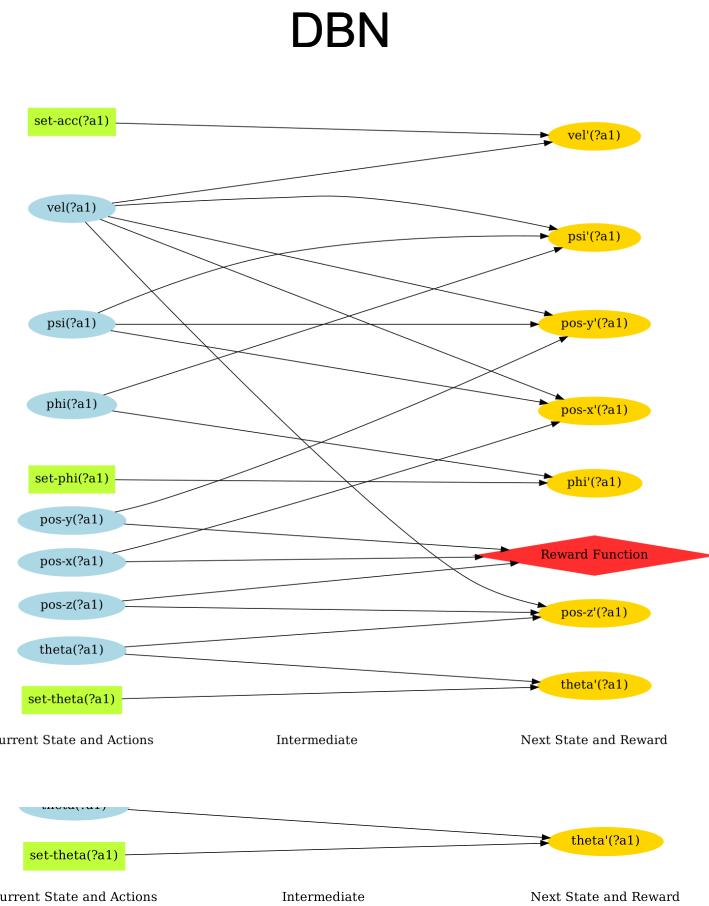


XADDs for Discrete+Continuous MDPs



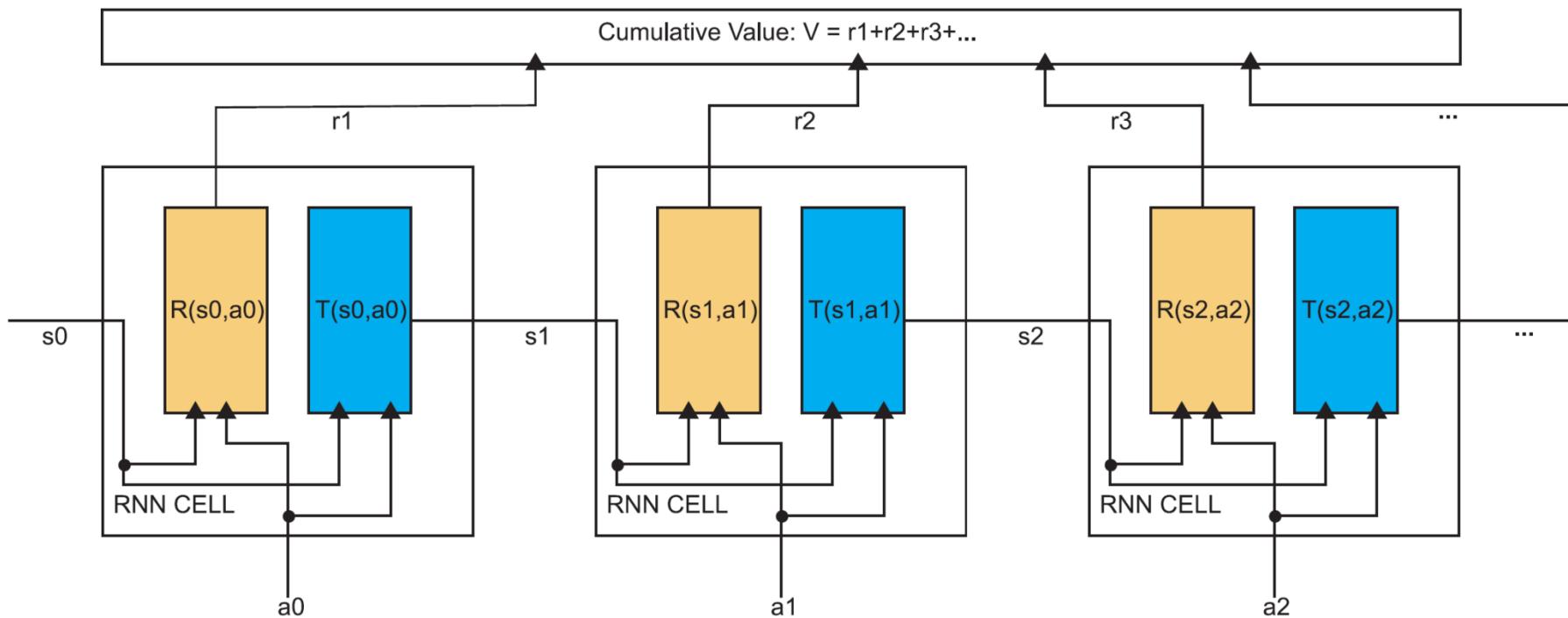
RDDL Compiles to (X)ADDs!

- UAV Problem



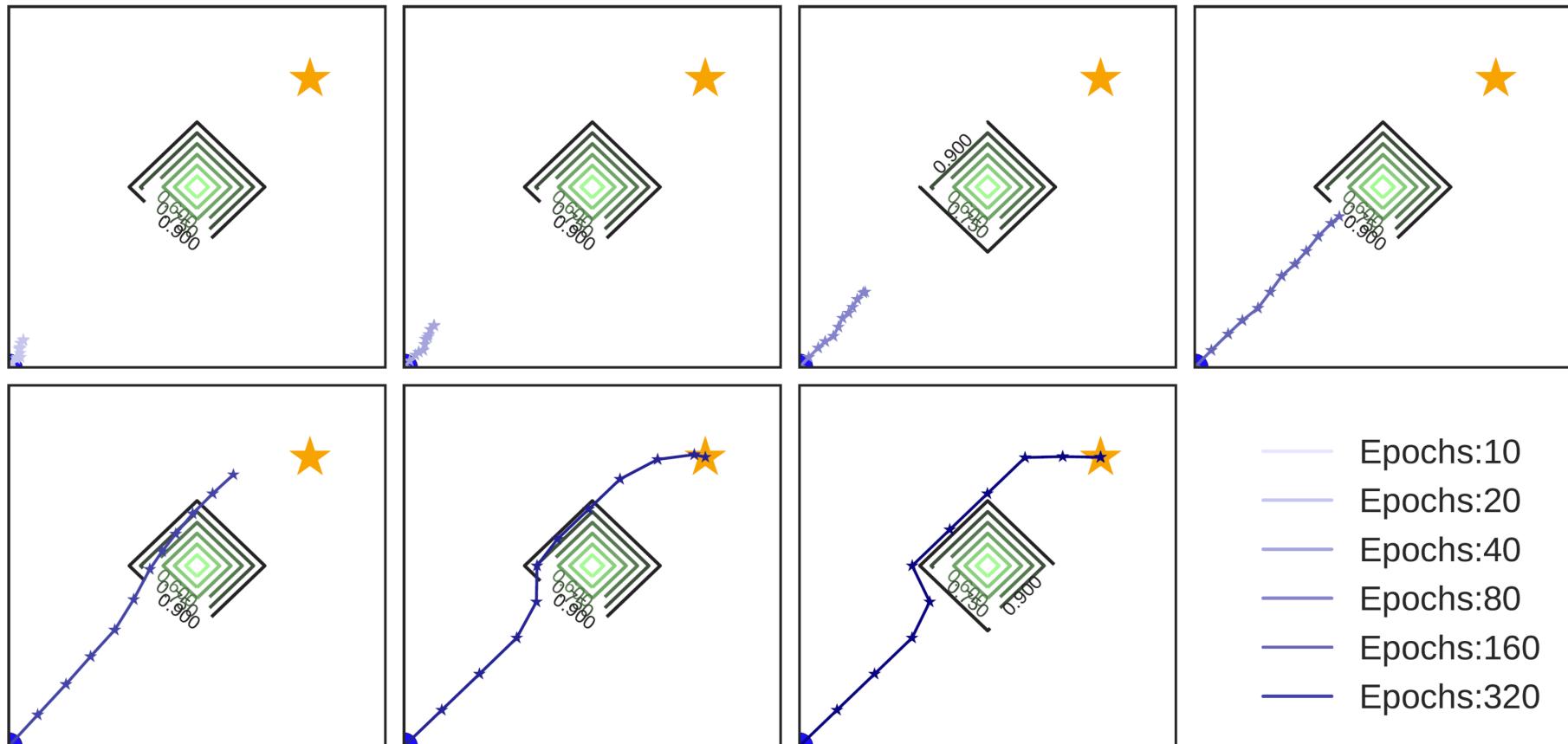
Planning by Backprop

Tensorplan: Embed Reward and Transition in an RNN and Optimize *End-to-end*!

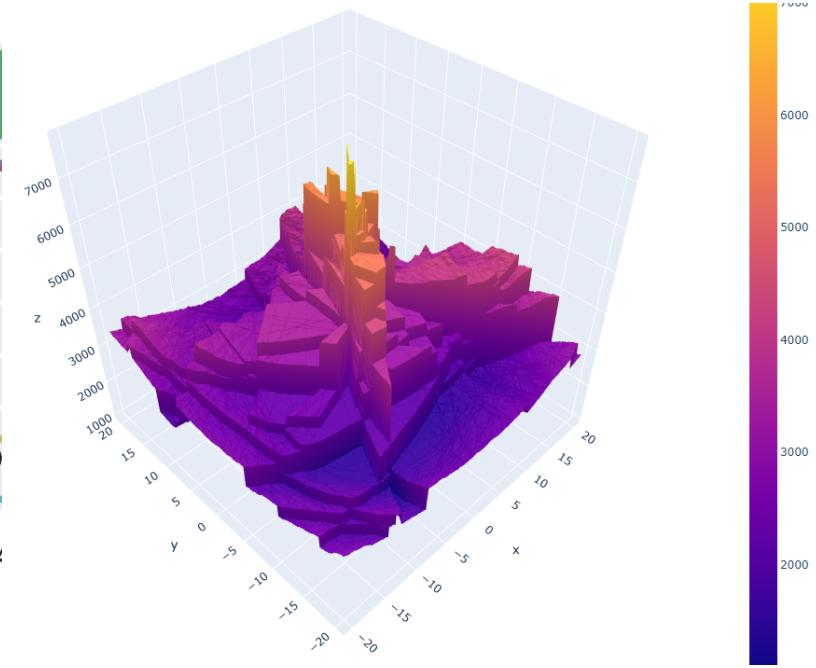
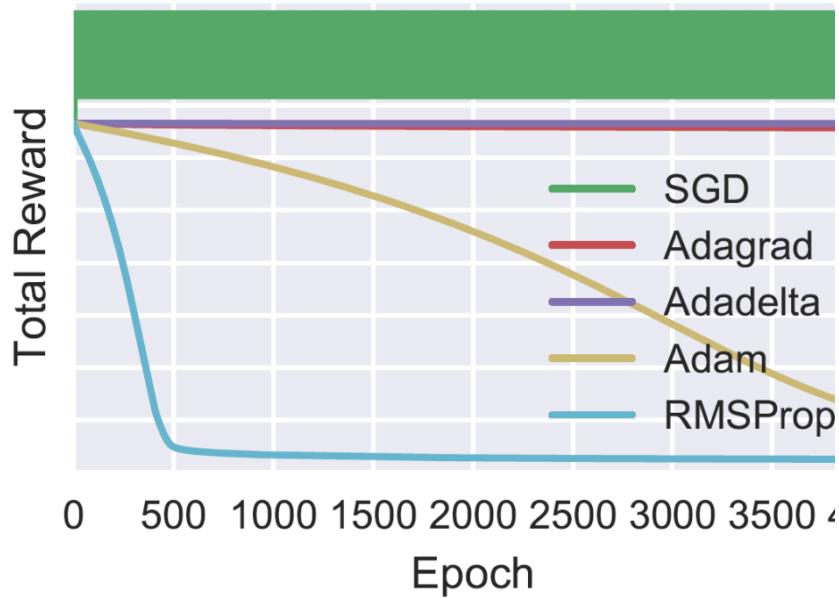


GPU-based Path Planning via Tensorflow

- RMSProp makes for a great non-convex optimizer!



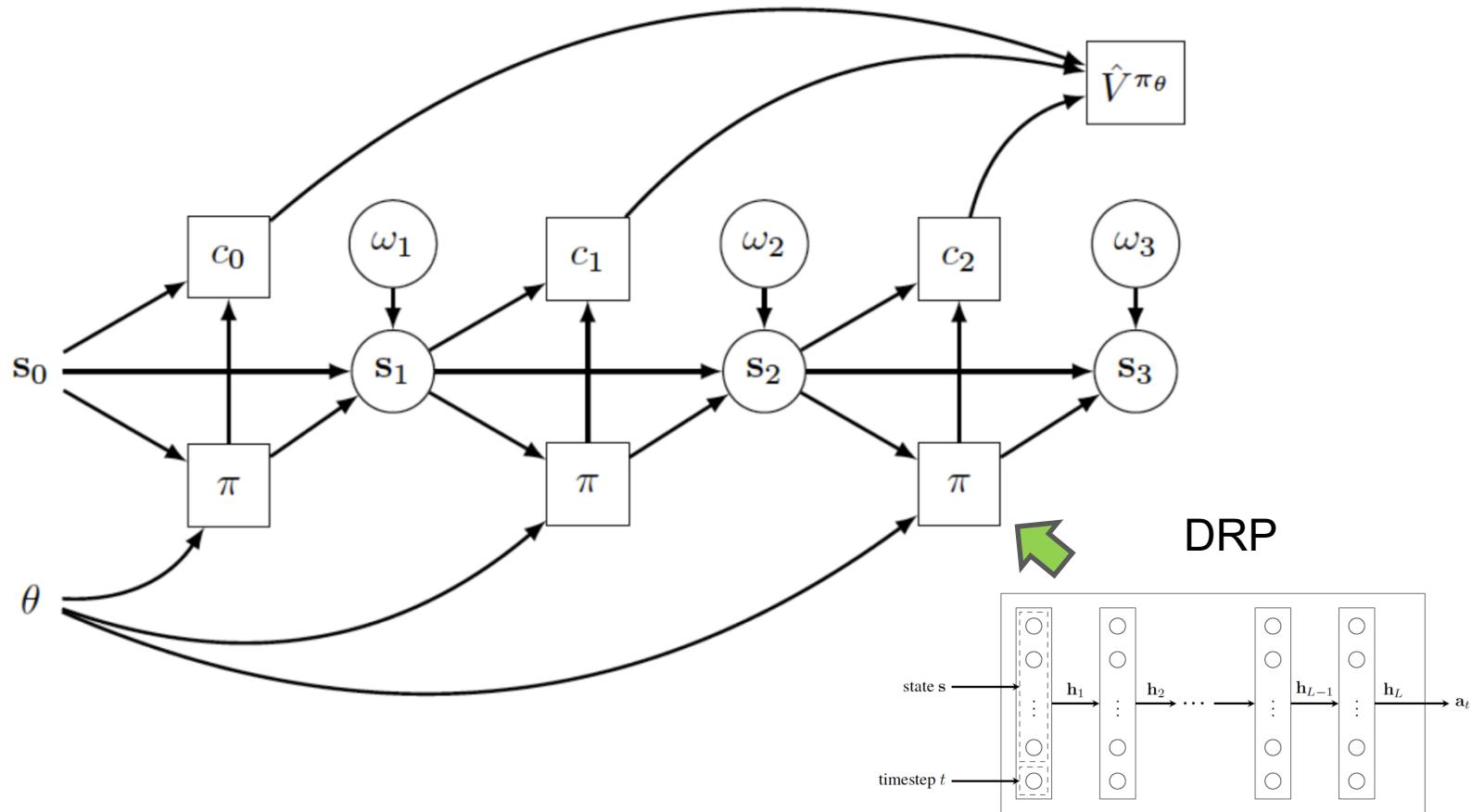
Need Modern Non-convex Gradient Methods



RMSProp is the best-performing optimizer for planning, likely b/c it can handle piecewise structure.

Learning Deep Reactive Policies (DRPs)

Stochastic RNNs

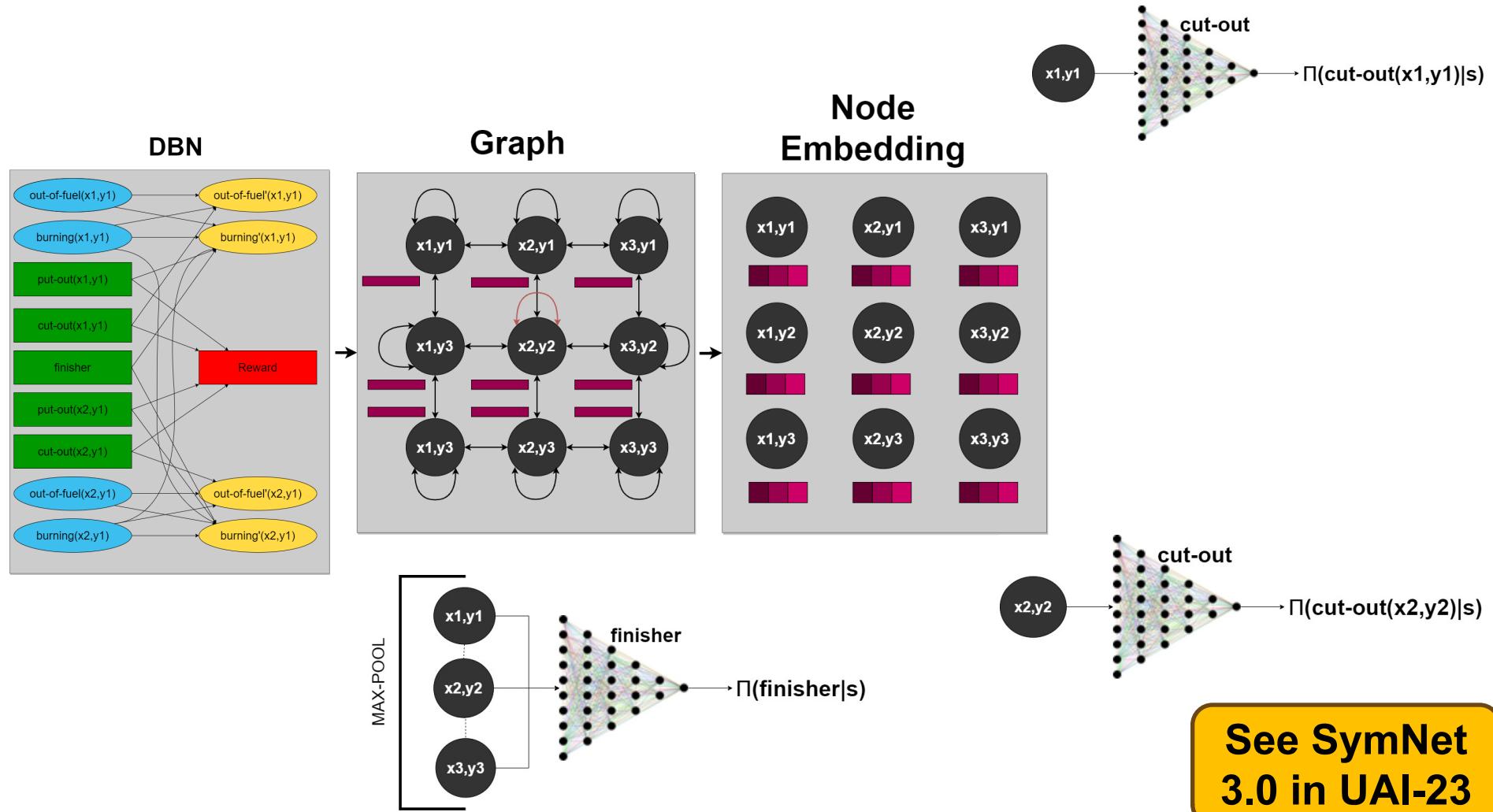


Lifted Approaches: Generalized Planning for RDDL

SymNet (Mausam and students)

SymNet 2.0 (Mausam et al, ICML-22)

Compile RDDL DBN into GNN, Embed, Decode to Actions
(GNN learning is domain instance independent)



RDDL Tutorial Outline

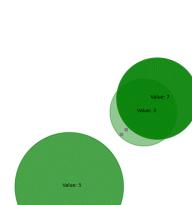
- **Part 1: Language Overview**
 - What is probabilistic planning in PPDDL?
 - Why do we need RDDL?
 - RDDL by example
 - Overview of RDDL solution methodologies
- **Part 2: PyRDDLGym**

pyRDDLGym

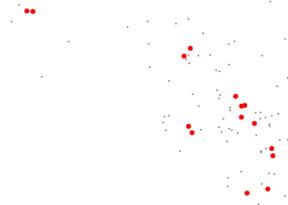
Includes OpenAI Gym interface, JaxPlanner, XADDs, etc.
<https://github.com/ataitler/pyRDDLGym>



Wildfires

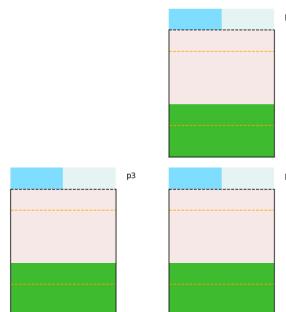


Mars Rover

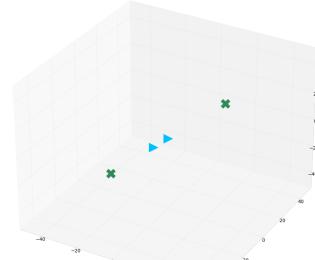


Recommender System

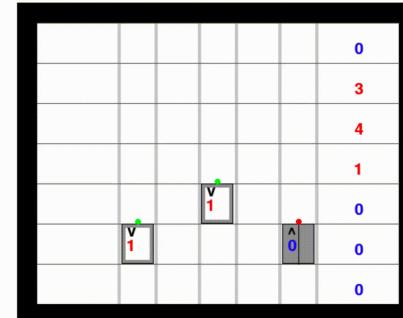
More on
PyRDDLGym
with worked
examples in
Part 2!



Power Generation



UAVs



Elevators