

ICAPS 2014 Tutorial

# Introduction to Planning Domain Modeling in RDDL

Scott Sanner



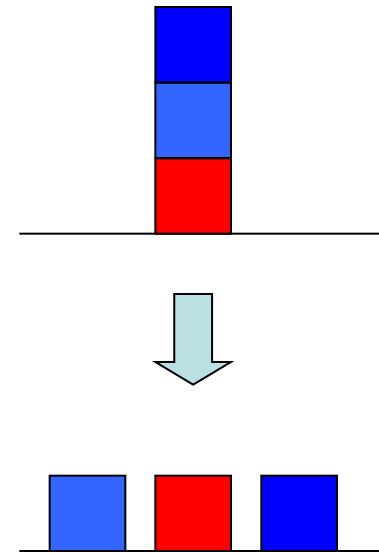
# Observation

- Planning languages direct 5+ years of research
  - PDDL and variants
  - PPDDL
- Why?
  - Domain design is time-consuming
    - So everyone uses the 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...

# Current Stochastic Domain Language

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

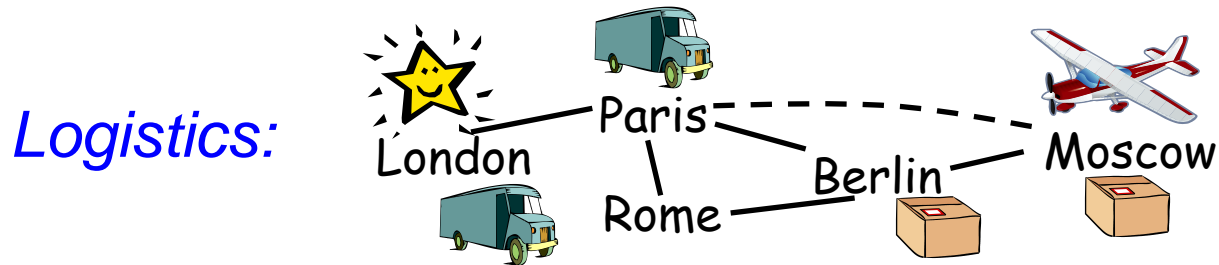
```
(:action put-all-blue-blocks-on-table
:parameters ( )
:precondition ( )
:effect (probabilistic 0.9
        (forall (?b)
          (when (Blue ?b)
            (not (OnTable ?b))))))
```



- **But wait, not just BlocksWorld...**
  - Colored BlocksWorld
  - Exploding BlocksWorld
  - Moving-stacks BlocksWorld
- Difficult problems *but where to apply solutions???*

# More Realistic: Logistics

- Compact relational 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* (and (On ?b ?t) (not (BIn ?b ?c))))

- Can instantiate problems for any domain objects

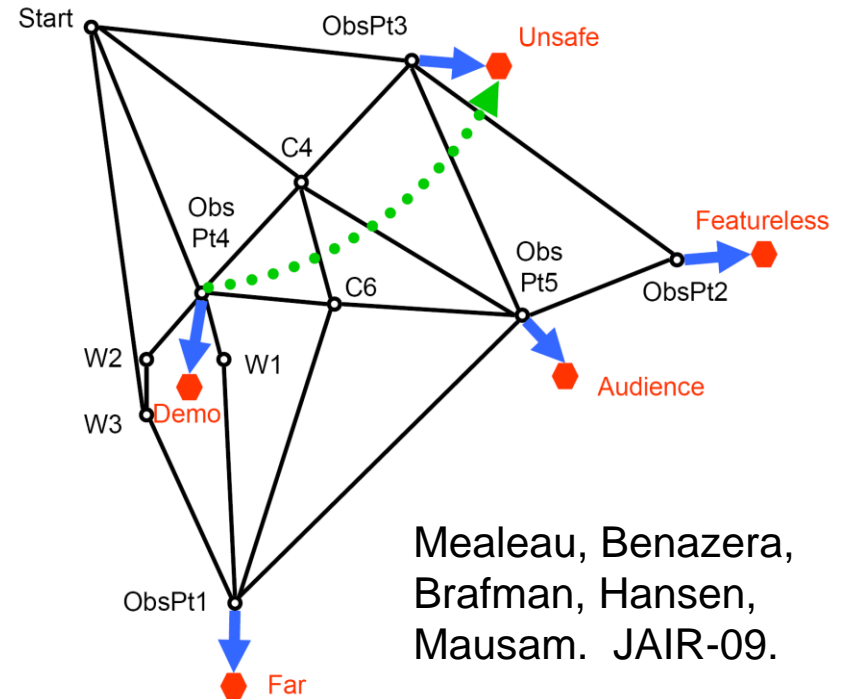
- 3 trucks:  2 planes:  3 boxes: 

- But wait... only one truck can move at a time???

- No concurrency, no time: **will FedEx care?**

What stochastic problems  
should we care about?

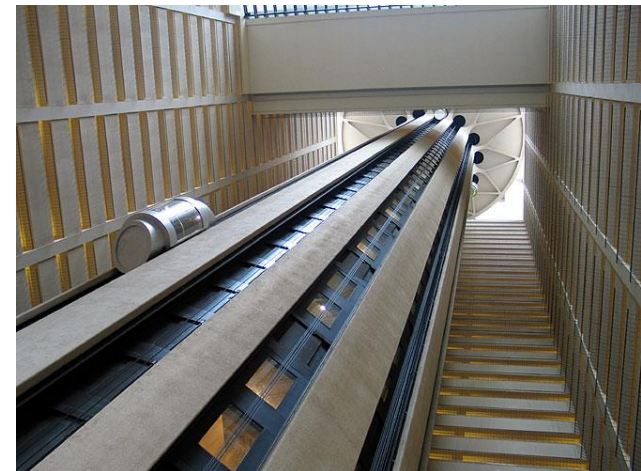
# Mars Rovers



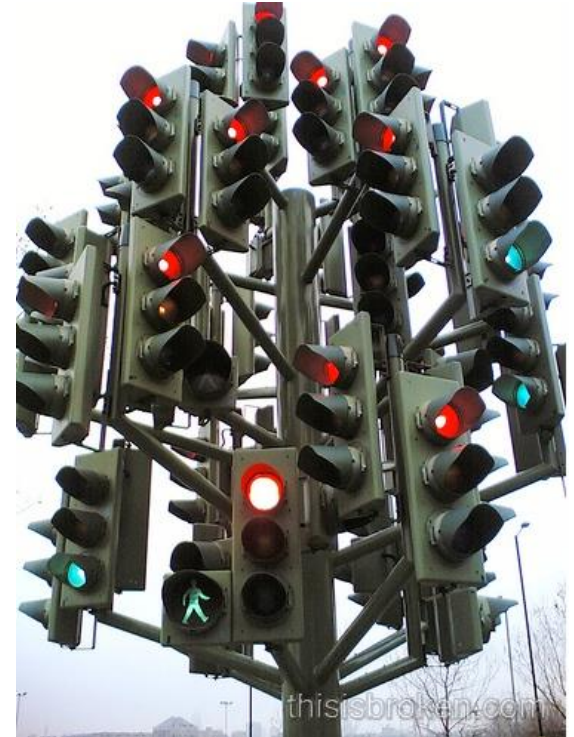
- **Continuous**
  - Time, robot position / pose, sun angle, ...
- **Partially observable**
  - Even worse: high-dimensional partially observable

# Elevator Control

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



# Traffic Control



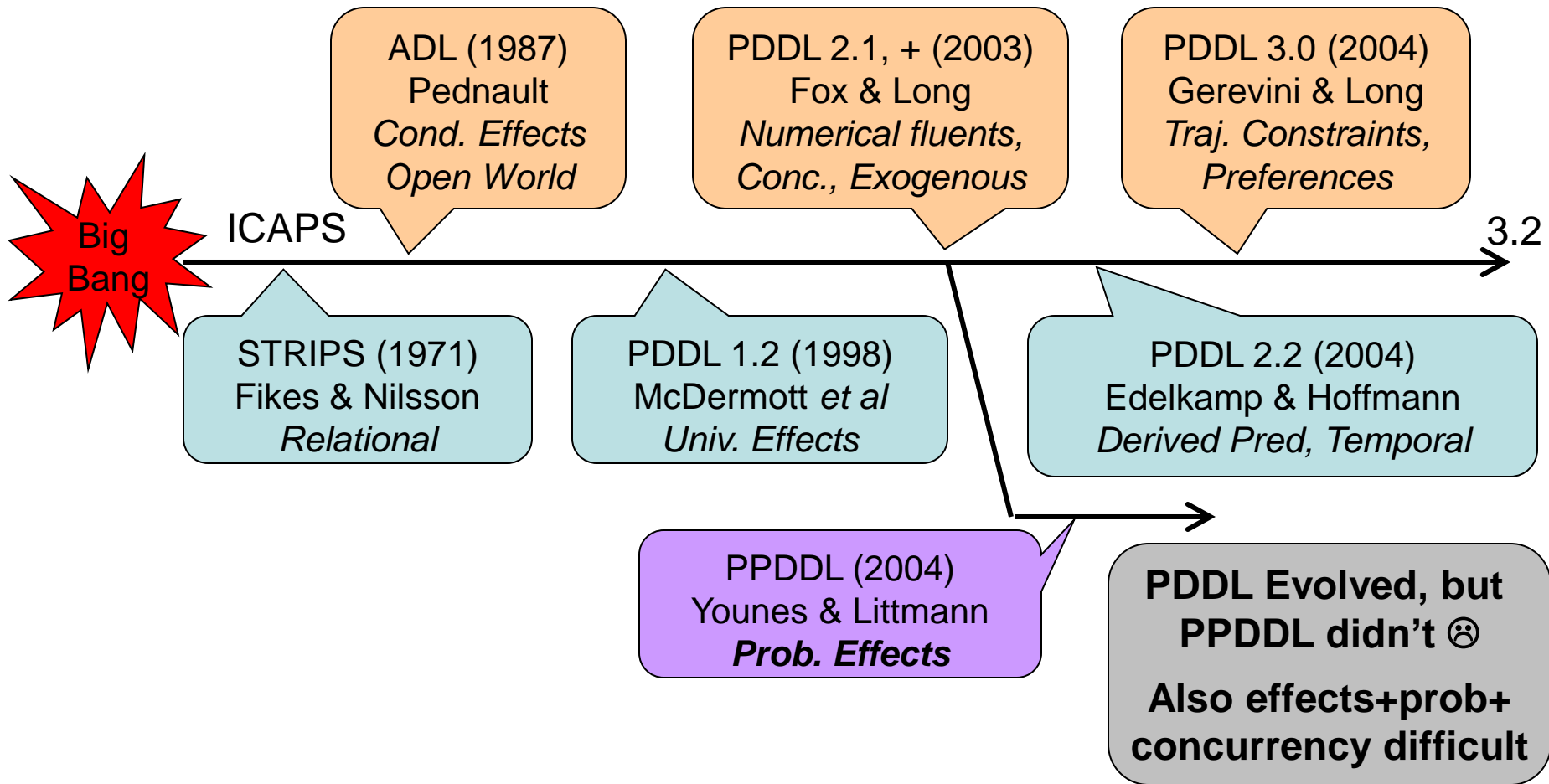
- **Concurrent**
  - Multiple lights
- **Continuous Variables**
  - Nonlinear dynamics
- **Indep. Exogenous Events**
  - Multiple vehicles
- **Partially observable**
  - Only observe stoplines



Can PPDDL model  
these problems?

No? What happened?

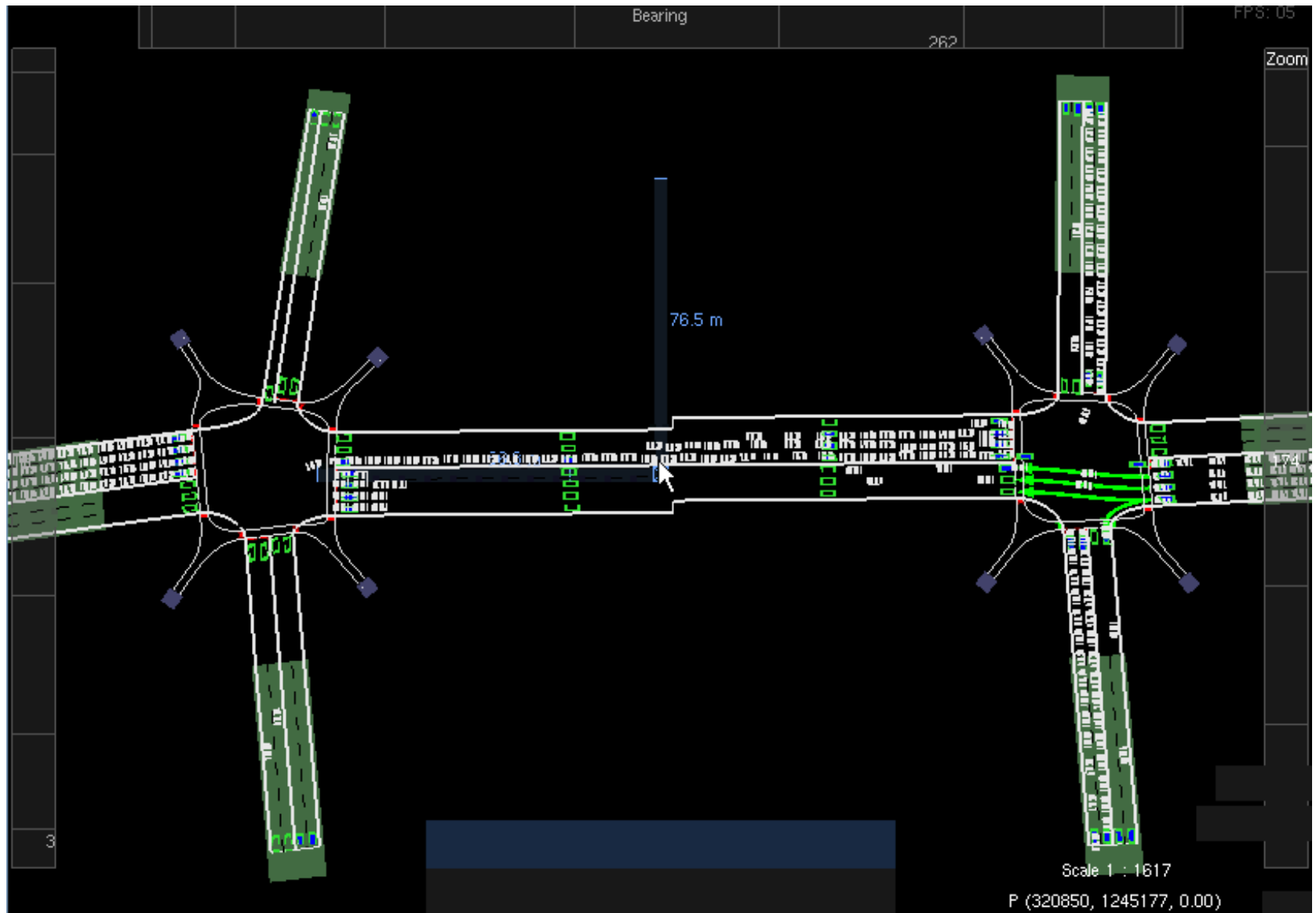
# A Brief History of (ICAPS) Time



What would it take to model  
more realistic problems?

Let's take a deeper look at  
traffic control...

# Birth of RDDDL: Solving Traffic Control



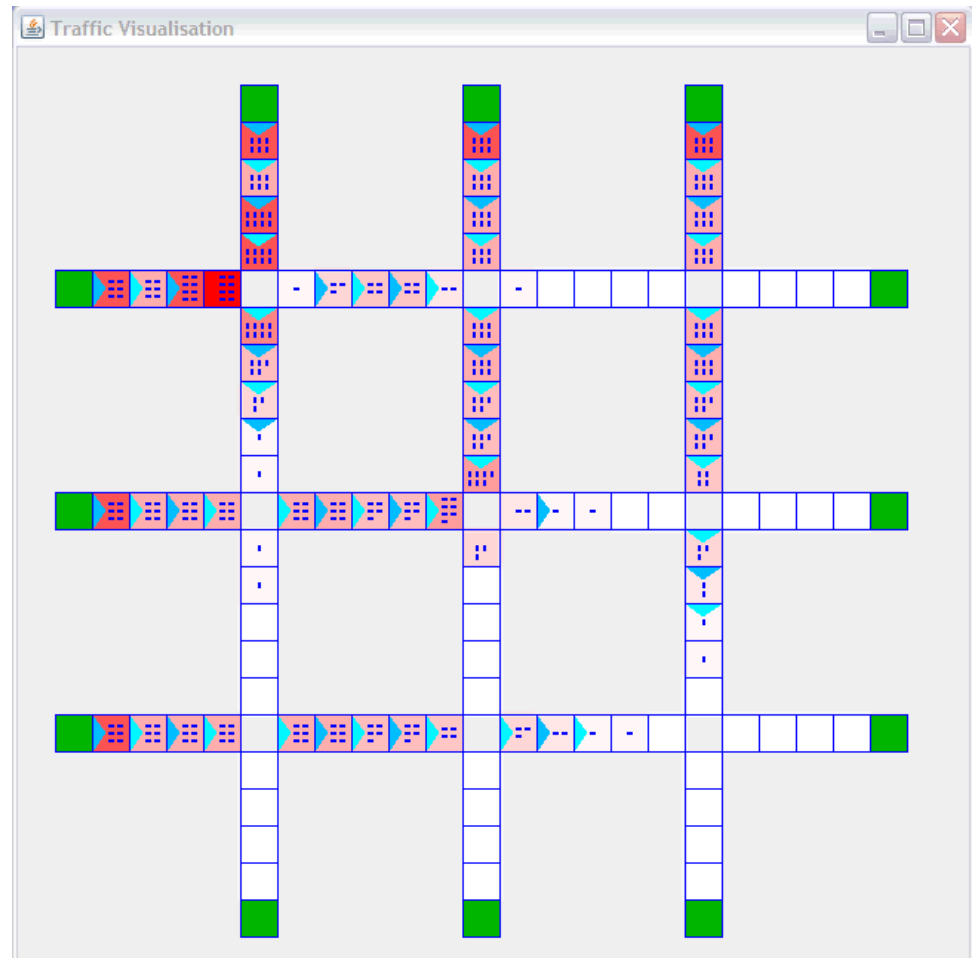
# What's missing in PPDDL, Part I

- **Need Unrestricted Concurrency:**
  - In PPDDL, would have to enumerate joint actions
  - In PDDL 2.1: *restricted concurrency*
    - conflicting actions not executable
    - when effects probabilistic, some chance most effects conflict
      - really need *unrestricted concurrency* in probabilistic setting
- **Multiple Independent Exogenous Events:**
  - PPDDL only allows 1 independent event to affect fluent
    - E.g, what if cars in a queue change lanes, brake randomly?

**Looking ahead... will need something more like Relational DBN**

# What's missing in PPDDL, Part II

- **Expressive transition distributions:**
  - (Nonlinear) stochastic difference equations
  - E.g., cell velocity as a function of traffic density
- **Partial observability:**
  - In practice, only observe stopline



# What's missing in PPDDL, Part III

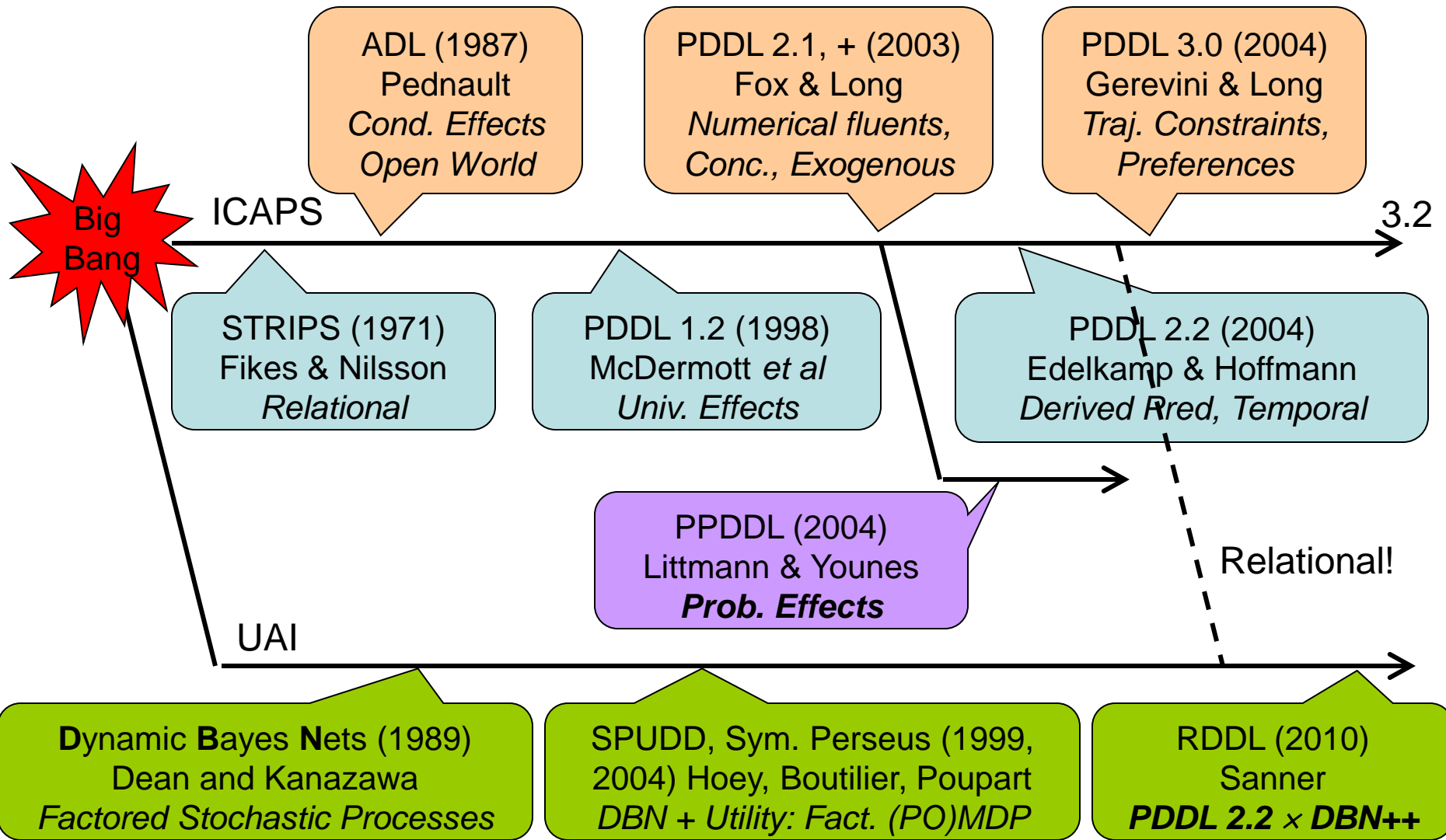
- Distinguish fluents from nonfluents:
  - E.g., topology of traffic network
  - Lifted planners must know this to be efficient!
- **Expressive rewards & probabilities:**
  - E.g., sums, products, nonlinear functions, ratios, conditionals
- **Global state-action constraints:**
  - Concurrent domains 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...

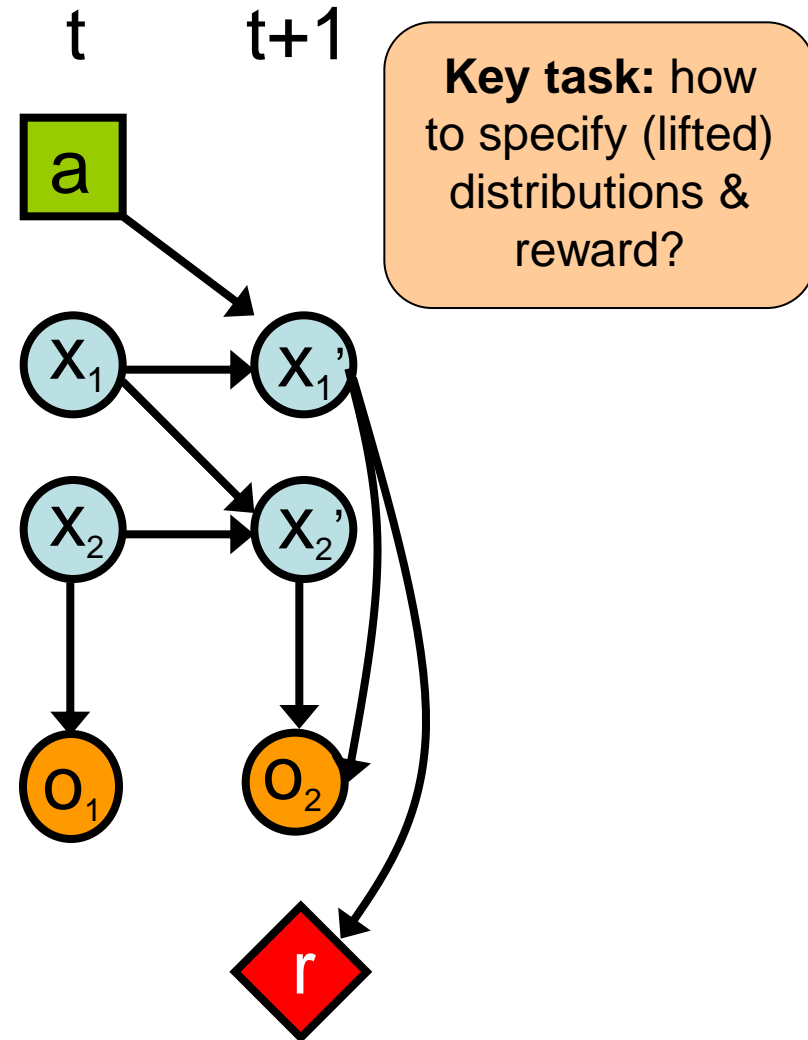


# A Brief History of (ICAPS) Time



# What is RDDDL?

- **Relational Dynamic Influence Diagram Language**
  - Relational [DBN + Influence Diagram]
- Think of it as Relational SPUDD / Symbolic Perseus
  - on speed



# 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, ...)
- **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**
  - Unambiguous specification of transition semantics
    - Supports unrestricted concurrency
  - 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)
  - Basic probability distributions
    - Bernoulli, Discrete, Normal, Poisson

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

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

# RDDL Principles 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 actions
    - (concurrent) action preconditions
  - Assert state invariants
    - e.g., a package cannot be in two locations

# RDDL Grammar

Let's examine BNF  
grammar in infinite tedium!

OK, maybe not.  
(Grammar [online](#) if you want it.)

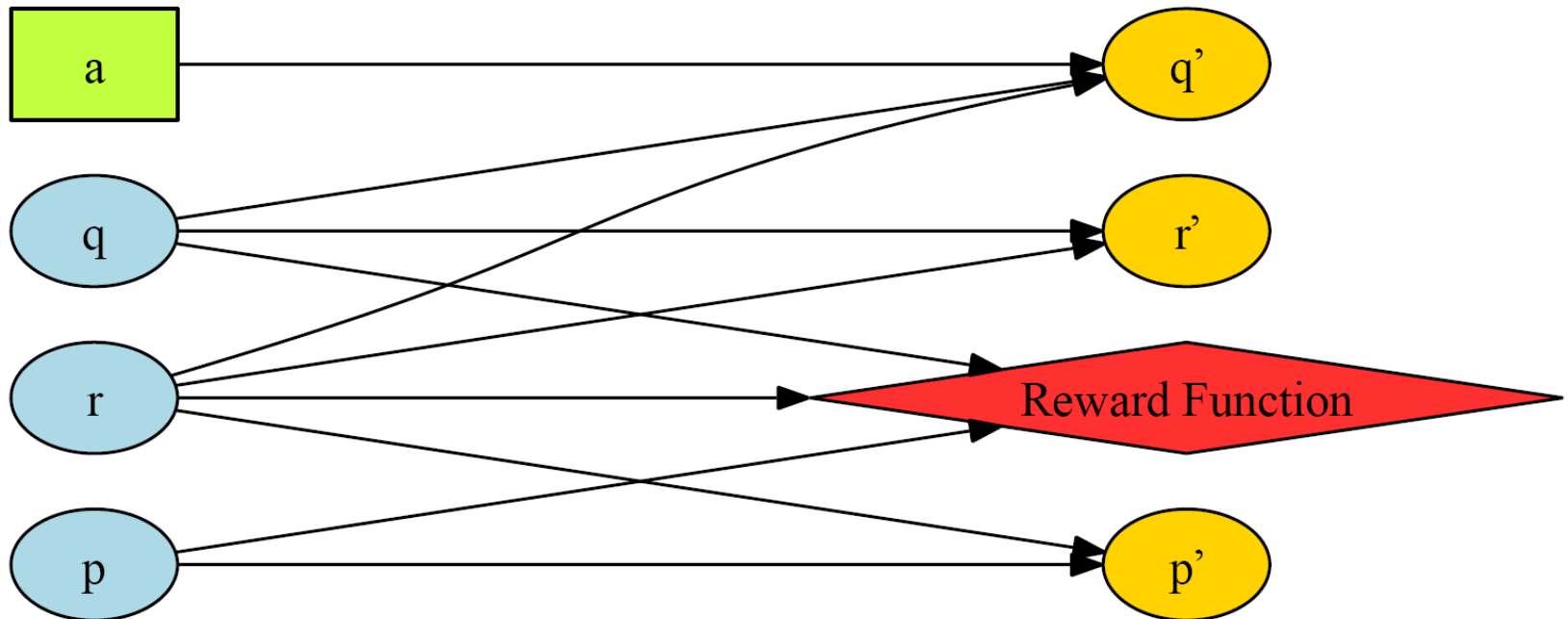
# RDDL Examples

Easiest to understand  
RDDL in use...

# 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);

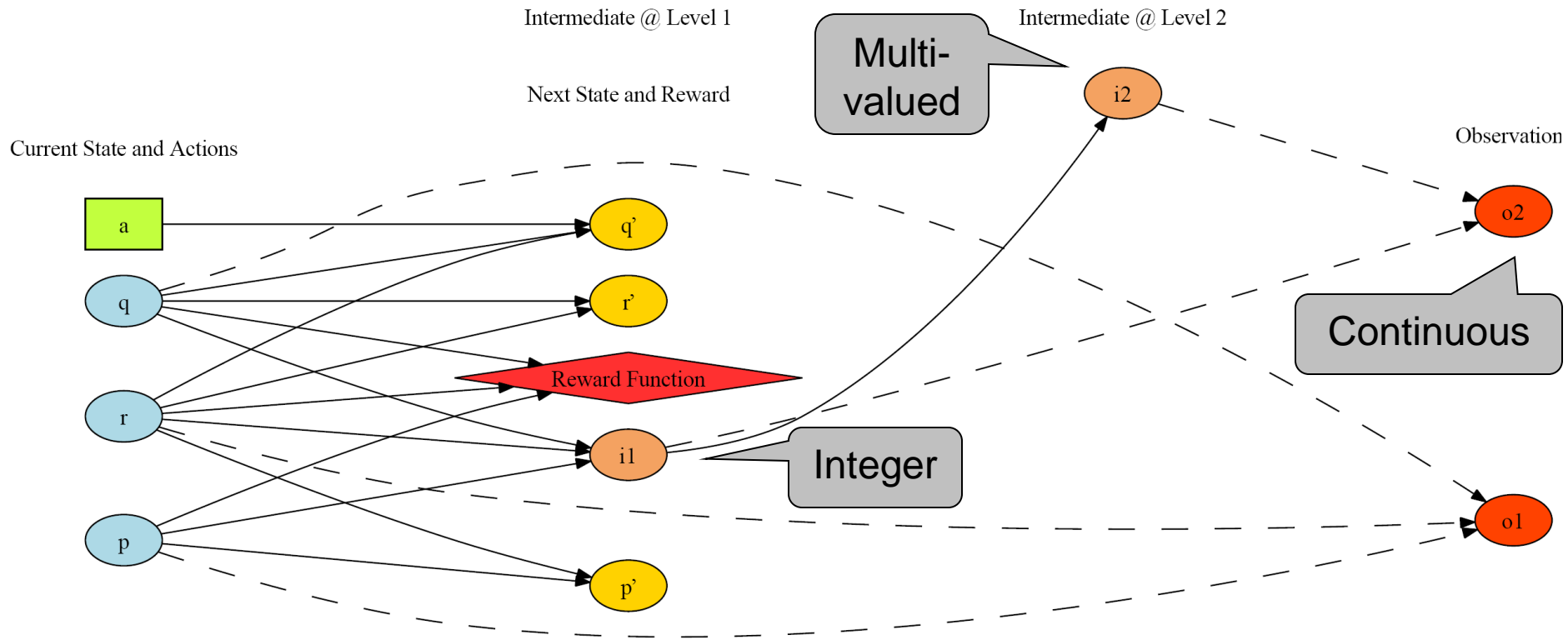
    q' = if (q ^ r) then Bernoulli(.9)
        else if (a) then Bernoulli(.3) else Bernoulli(.8);

    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*”

# 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);
};
```

# 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

# RDDL so far...

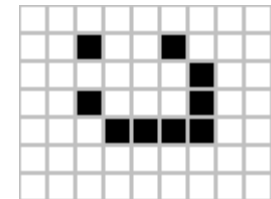
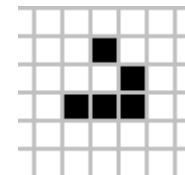
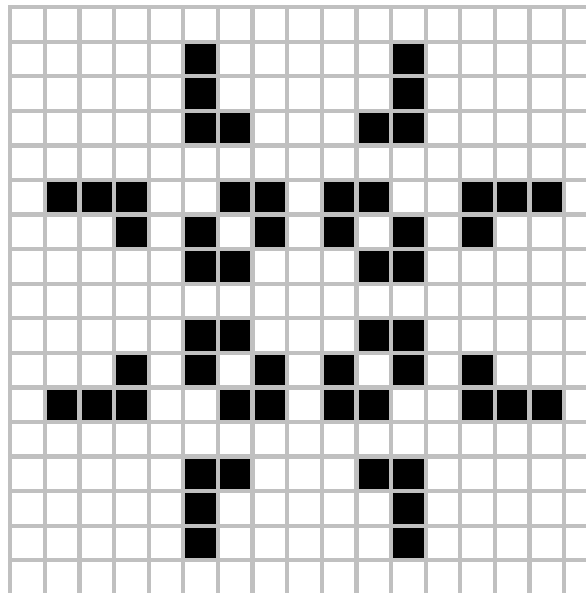
- Mainly SPUDD / Symbolic Perseus with a different syntax 😊
  - A few enhancements
    - concurrency
    - constraints
    - integer / continuous variables
- **Real problems (e.g., traffic) need lifting**
  - An intersection model
  - A vehicle model
    - Specify each intersection / vehicle model once!

# 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](http://en.wikipedia.org/wiki/Conway's_Game_of_Life)

- Compact RDDDL specification for **any grid size?** Lifting.

Current State and Actions

Intermediate @ Level 1

Next State and Reward

set(x1, y1)

set(x1, y2)

alive(x1, y1)

alive(x1, y2)

alive(x2, y2)

alive(x2, y1)

set(x2, y1)

set(x2, y2)

count-neighbors(x1, y1)

count-neighbors(x1, y2)

count-neighbors(x2, y1)

count-neighbors(x2, y2)

alive'(x1, y1)

alive'(x1, y2)

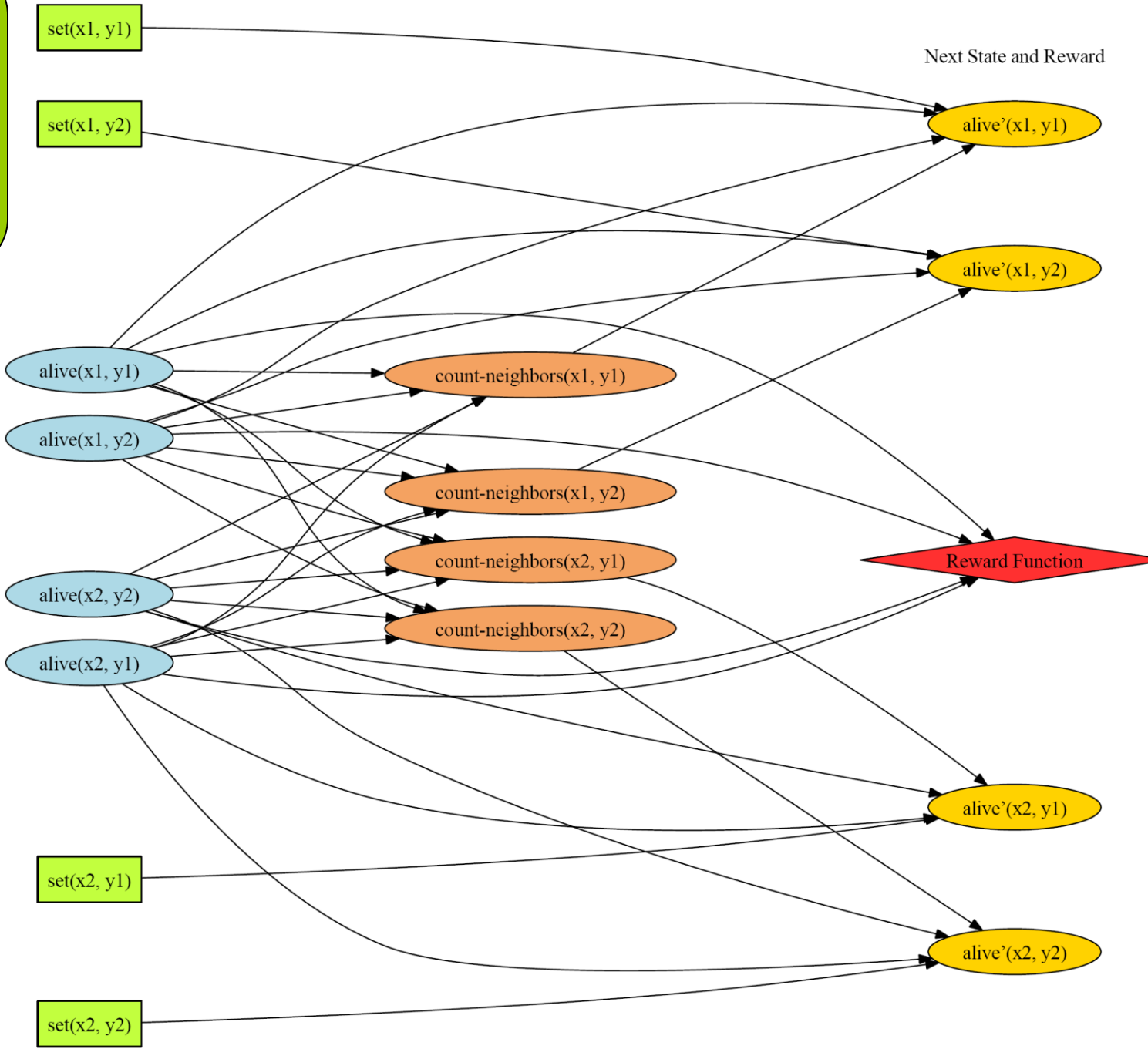
alive'(x2, y1)

alive'(x2, y2)

Reward Function

Concurrency  
as factored  
action variables  
  
How many  
possible joint  
actions here?

# Lifted MDP: Game of Life



# A Lifted MDP

```
// Store alive-neighbor count
count-neighbors(?x,?y) =
```

Intermediate variable: like derived predicate

```
  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);
```

Using counts to  
decide next state

```
};
```

```
// Reward is number of alive cells
```

```
reward = sum_{?x : x_pos, ?y : y_pos} alive(?x,?y);
```

Additive reward!

```
state-action-constraints {
```

```
  // Assertion: ensure PROB_REGENERATE is a valid probability
  (PROB_REGENERATE >= 0.0) ^ (PROB_REGENERATE <= 1.0);
```

State constraints,  
preconditions

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

```
};
```



# Nonfluent and Instance Definition

```
// 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};
};

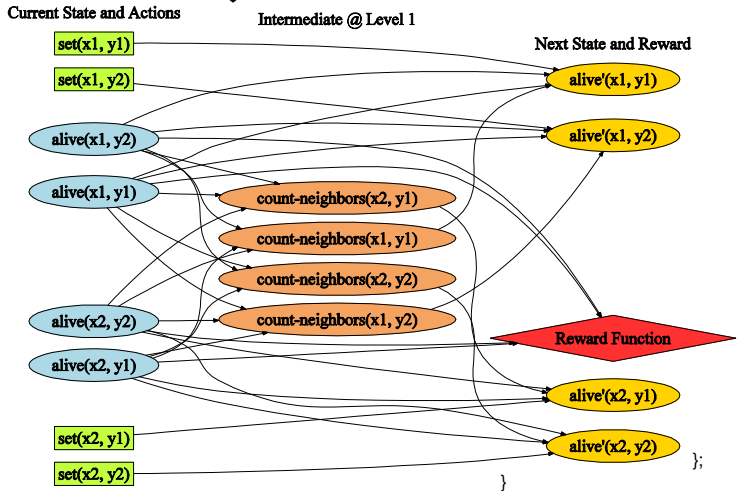
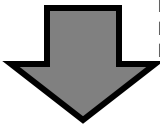
non-fluents {
    PROB_REGENERATE = 0.9;

    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);
};
    
```



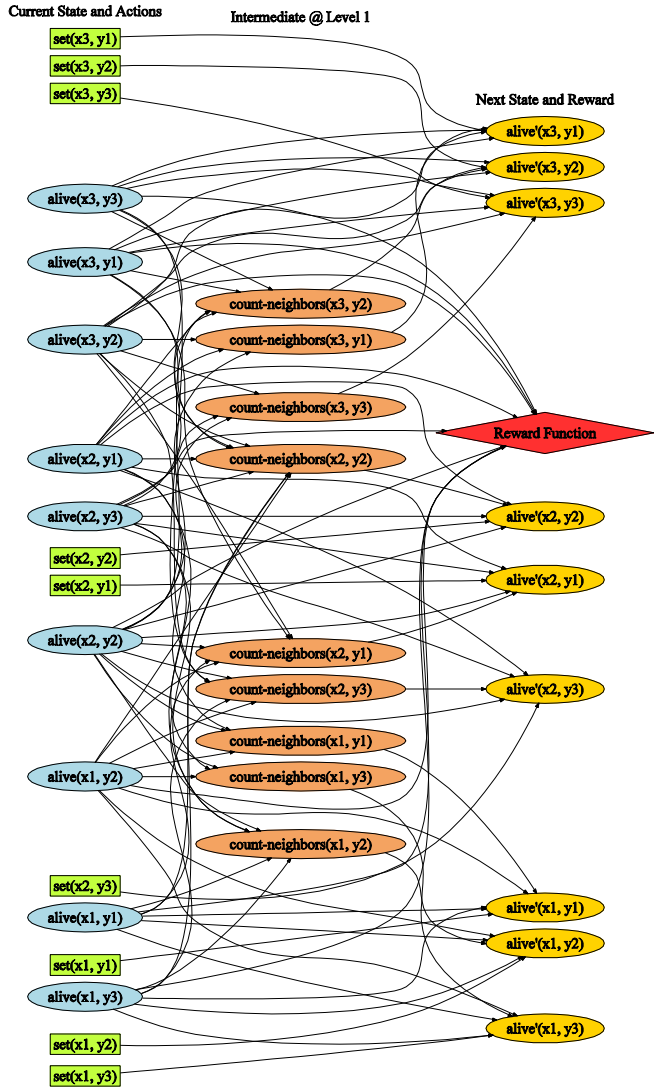
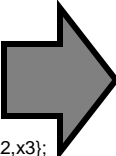
non-fluents game3x3 {

```

domain = game_of_life;

objects {
    x_pos : {x1,x2,x3};
    y_pos : {y1,y2,y3};
};

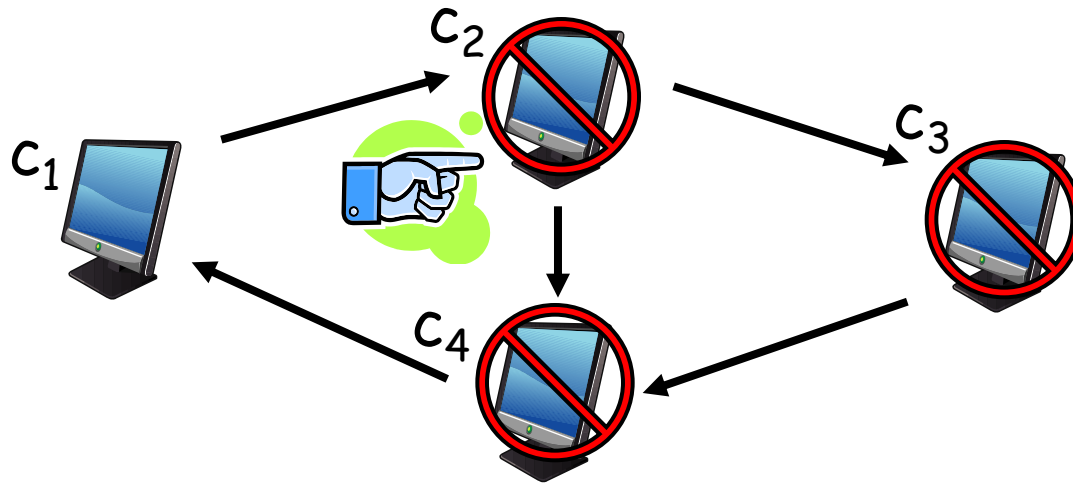
non-fluents {
    NEIGHBOR(x1,y1,x1,y2);
    NEIGHBOR(x1,y1,x2,y1);
    NEIGHBOR(x1,y1,x2,y2);
    NEIGHBOR(x1,y2,x1,y1);
    NEIGHBOR(x1,y2,x2,y1);
    NEIGHBOR(x1,y2,x2,y2);
    NEIGHBOR(x1,y2,x2,y3);
    NEIGHBOR(x1,y2,x1,y3);
    NEIGHBOR(x1,y3,x1,y2);
    NEIGHBOR(x1,y3,x2,y2);
    NEIGHBOR(x1,y3,x2,y3);
    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,y2);
    NEIGHBOR(x3,y1,x3,y2);
    NEIGHBOR(x3,y2,x3,y1);
    NEIGHBOR(x3,y2,x2,y2);
    NEIGHBOR(x3,y2,x3,y3);
    NEIGHBOR(x3,y3,x2,y3);
    NEIGHBOR(x3,y3,x2,y2);
    NEIGHBOR(x3,y3,x3,y2);
    NEIGHBOR(x3,y3,x2,y2);
    NEIGHBOR(x3,y3,x3,y2);
};
    
```



# Complex Lifted Transitions: SysAdmin

SysAdmin (Guestrin et al, 2001)

- Have  $n$  computers  $C = \{c_1, \dots, c_n\}$  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

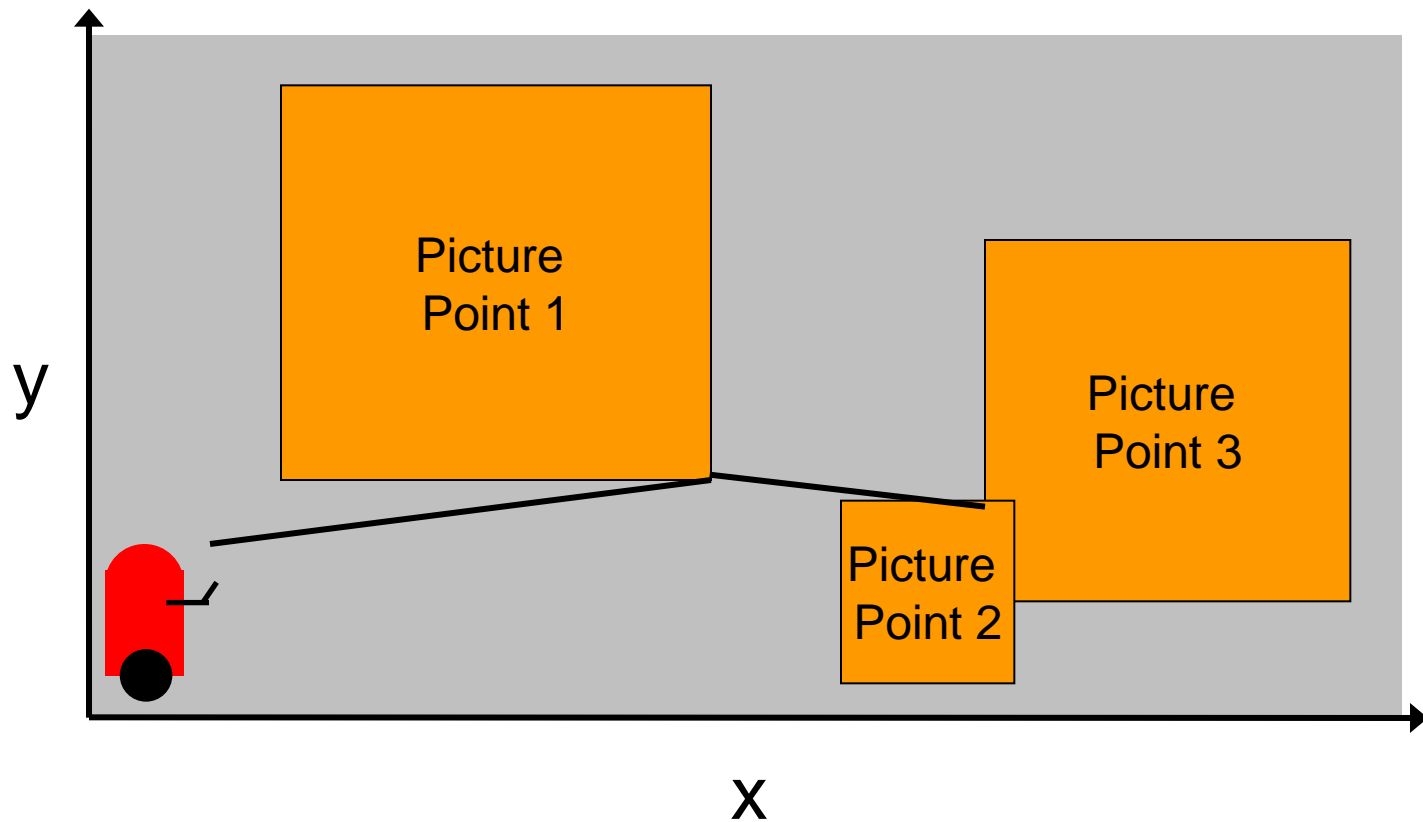
# Complex Lifted Transitions

SysAdmin (Guestrin et al, 2001)

```
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!

# 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 };
```

Constant  
picture  
points,  
bounding box

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

Rover position  
(only one  
rover)  
and time

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

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]);
```

Fixed time for picture

Time proportional to distance moved

White noise, variance proportional to distance moved

```
};
```

nb., This is RDDDL1, in RDDDL2, now have vectors and functions like abs()

# 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;
```

Reward for all pictures taken  
within bounding box!

Cannot move and take  
picture at same time.

```
state-action-constraints {
```

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

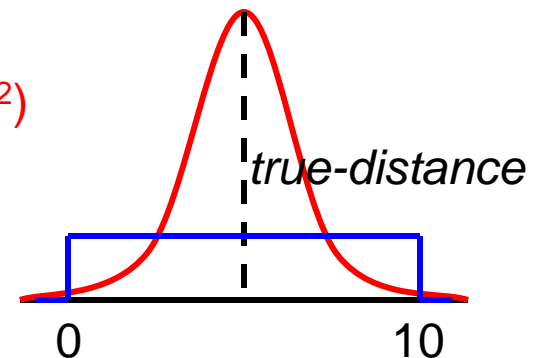
```
};
```



# How to Think About Distributions

- **Transition distribution is stochastic program**
  - Similar to BLOG (Milch, Russell, et al), IBAL (Pfeffer)
  - Leaves of programs are distributions
    - Think of SPUDD / Sym. Perseus decision diagrams as having Bernoulli leaves
- *Procedural specification of sampling process*
  - Use intermediate DBN variables for storage
  - E.g., drawing a distance measurement in robotics
    - **boolean Noise** := sample from **Bernoulli (.1)**
    - **real Measurement** := If (*Noise* == true)
      - Then sample from **Uniform(0, 10)**
      - Else sample from **Normal(true-distance,  $\sigma^2$ )**

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



# 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**
  - Unambiguous specification of transition semantics
    - Supports unrestricted concurrency
  - Naturally supports independent exogenous events
- **General expressions in transition / reward**
  - Logical expressions ( $\wedge, \vee, \Rightarrow, \Leftrightarrow, \forall, \exists$ )
  - Arithmetic expressions ( $+, -, *, /, \Sigma_x, \Pi_x$ )
  - In/dis/equality comparison expressions ( $=, \neq, <, >, \leq, \geq$ )
  - Conditional expressions (if-then-else, switch)
  - Basic probability distributions
    - Bernoulli, Discrete, Normal, Poisson

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

$\Sigma_x, \Pi_x$  aggregators over  
domain objects extremely  
powerful

# 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 actions
    - (concurrent) action preconditions
  - Assert state invariants
    - e.g., a package cannot be in two locations

# RDDL Software

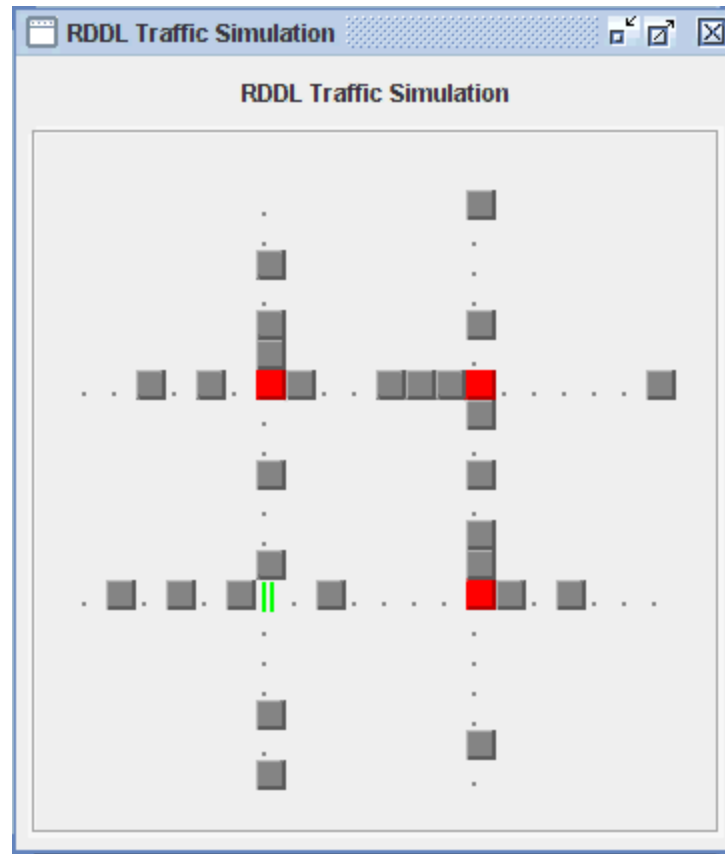
Open source & online at

<http://code.google.com/p/rddlsim/>

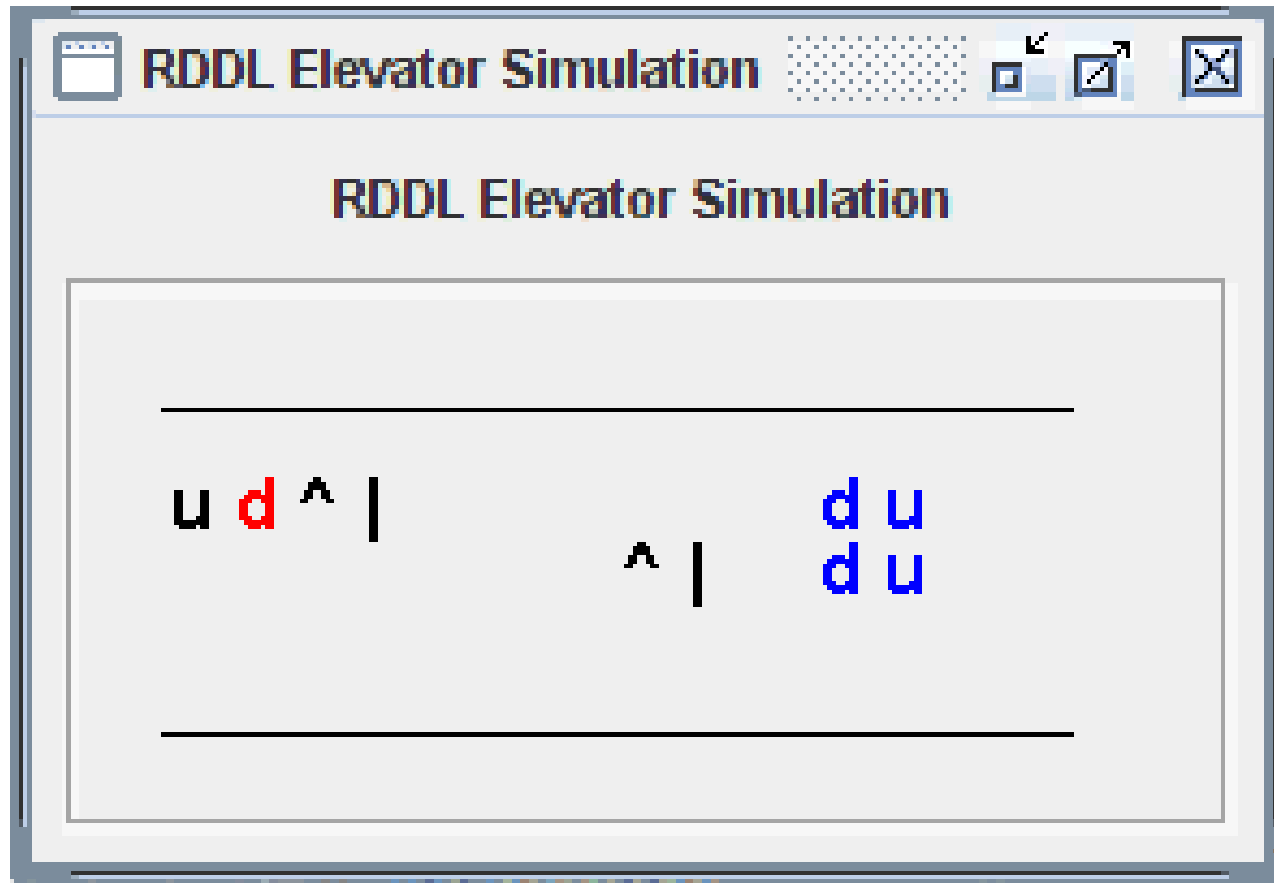
# Java Software Overview

- 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

# Visualization of Boolean Traffic



# Visualization of Boolean Elevators





# Submit your own Domains in RDDDL!

Field only makes true progress  
working on realistic problems

# RDDL2 (with Thomas Keller)

- Elementary functions
  - abs, sin, cos, log, exp, pow, sqrt, etc.
- Vectors
  - Need for some distributions (multinomial, multivariate normal)
- Object fluents and bounded integers
  - \$ to differentiate object names from parameter-free fluents
  - @ to differentiate bounded-range integers from integers
  - Auto-casting where possible
- Derived fluents
  - Like intermediate but can use in preconditions
- Indefinite horizon (goal-oriented problems)
- Recursion!
  - Fluents can self-reference as long as define a DAG

# RDDL Domain Examples

- See IPPC 2011 (Discrete)
  - [http://users.cecs.anu.edu.au/~ssanner/IPPC\\_2011/index.html](http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/index.html)
- See IPPC 2014 (Discrete)
  - [https://cs.uwaterloo.ca/~mgrzes/IPPC\\_2014/](https://cs.uwaterloo.ca/~mgrzes/IPPC_2014/)
- See IPPC 2014/5 (Continuous)
  - [http://users.cecs.anu.edu.au/~ssanner/IPPC\\_2014/index.html](http://users.cecs.anu.edu.au/~ssanner/IPPC_2014/index.html)

# Ideas for other RDDDL Domains

- UAVs with partial observability
- (Hybrid) Control
  - Linear-quadratic control (Kalman filtering with control)
  - Discrete and continuous actions – avoided by planning
  - Nonlinear control
- Dynamical Systems from other fields
  - Population dynamics
  - Chemical / biological systems
  - Physical systems
    - Pinball!
  - Environmental / climate systems
- Bayesian Modeling
  - Continuous Fluents can represent parameters
    - Beta / Bernoulli / Dirichlet / Multinomial / Gaussian
  - Then progression is a Bayesian update!
    - Bayesian reinforcement learning

# RDDL3?

- **Effects-based specification?**
  - Easier to write than current fluent-centered approach
  - But how to resolve conflicting effects in unrestricted concurrency
- **Timed processes?**
  - Concurrency + time quite difficult
  - Should we simply use languages like RMPL (Williams *et al*)
    - Or could there be RDDL + RMPL hybrids?

Enjoy RDDDL!

(no lack of difficult  
problems to solve!)

Questions?