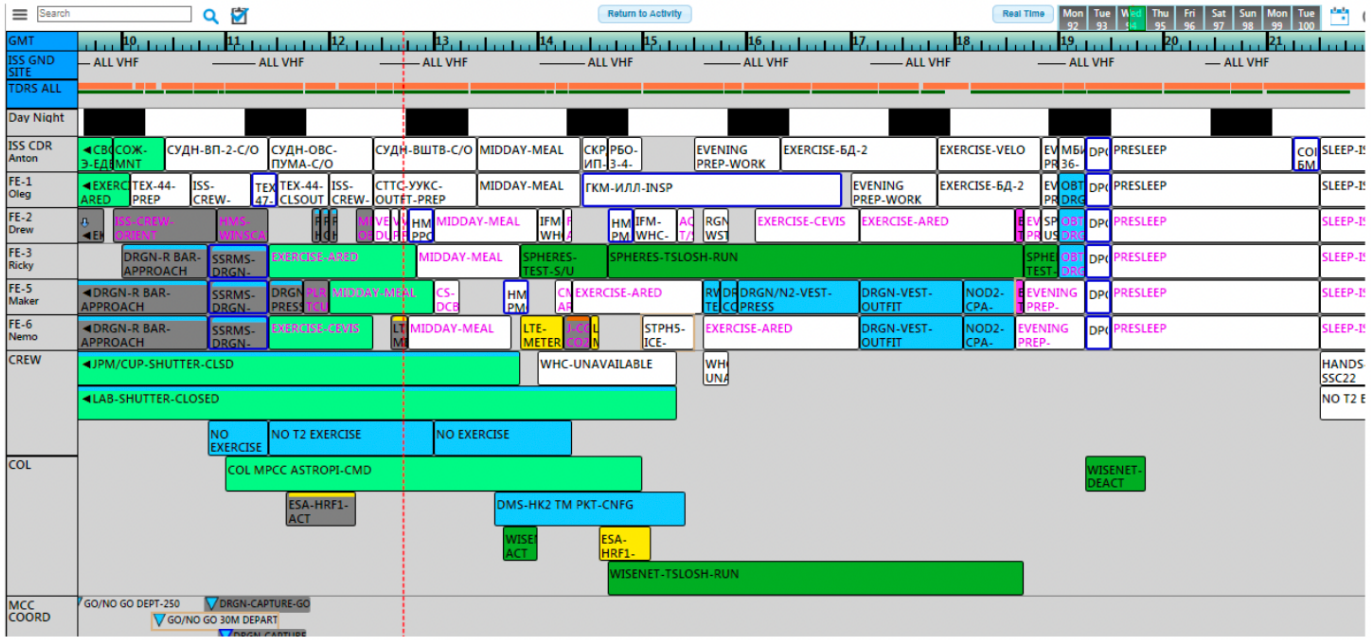
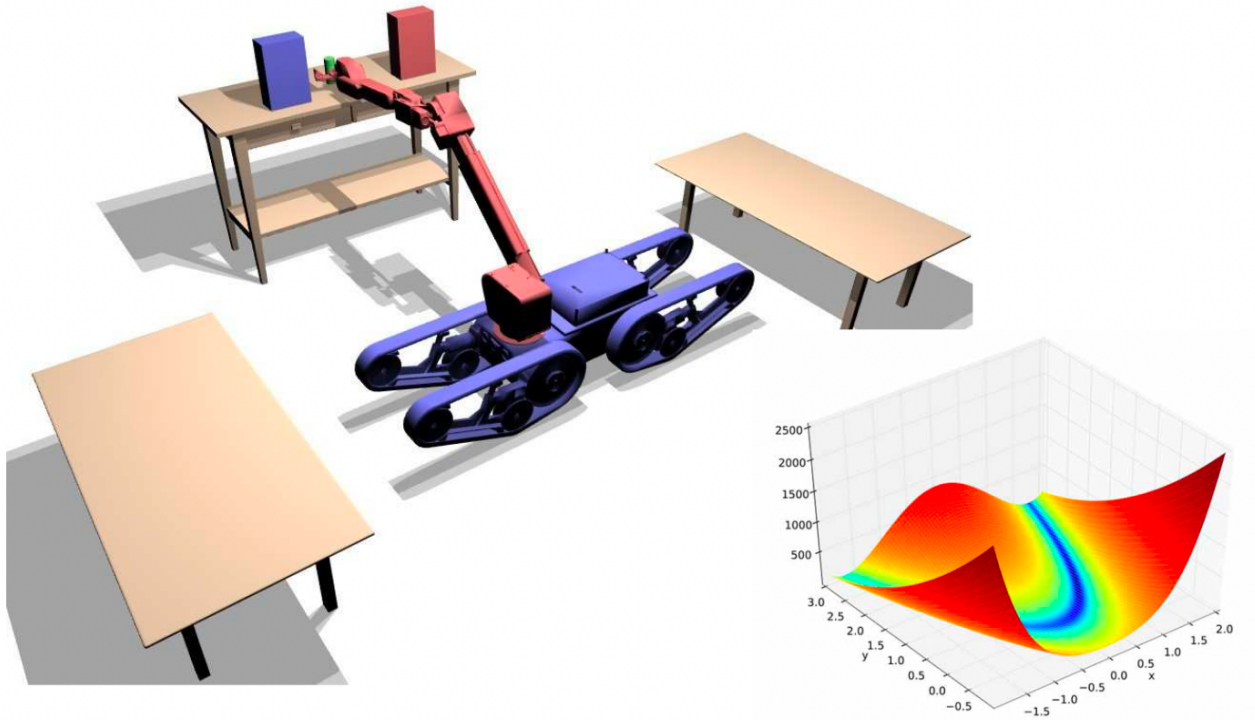
Beyond Classical Planning

Generalization of Classical Planning: Temporal Planning



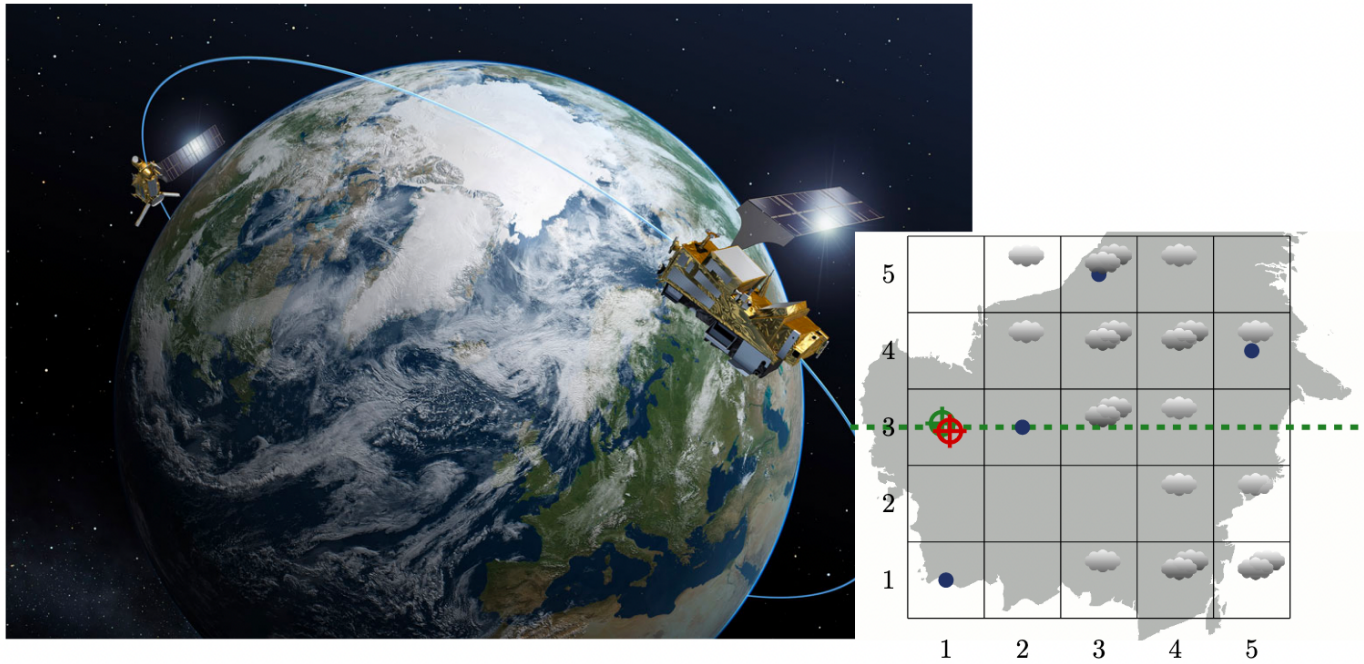
- timetable for astronauts on ISS
- concurrency required for some experiments
- optimize makespan

Generalization of Classical Planning: Numeric Planning



- kinematics of robotic arm
- state space is continuous
- preconditions and effects described by complex functions

Generalization of Classical Planning: MDPs

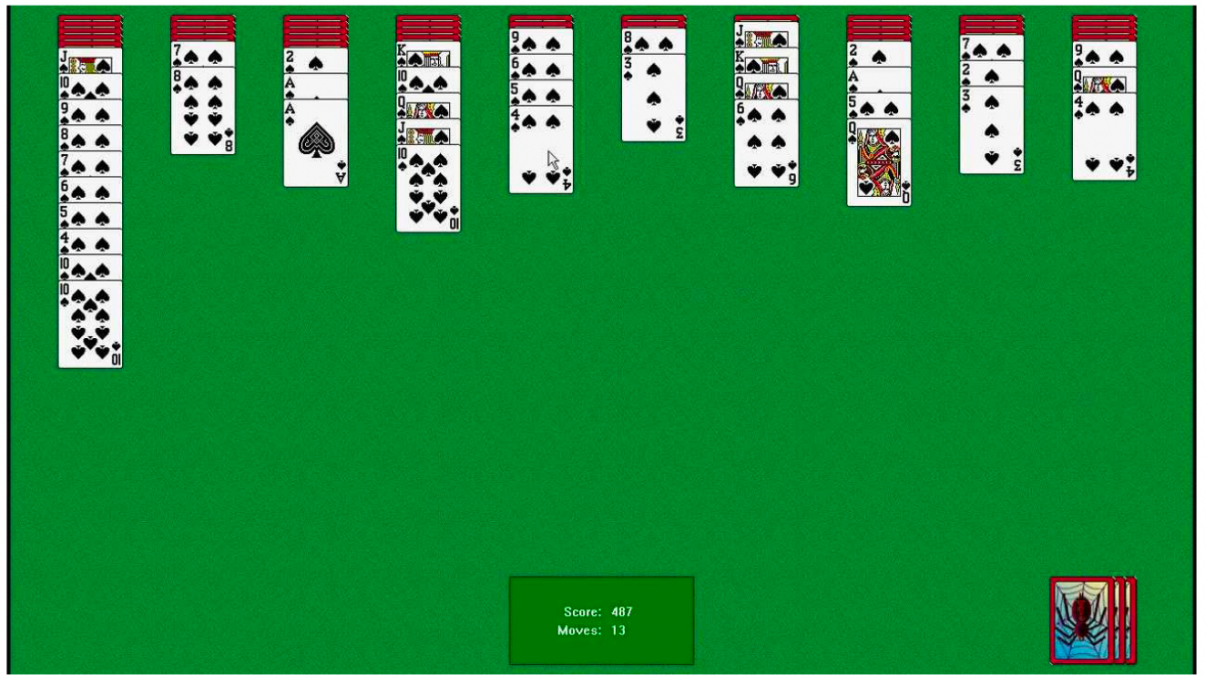


- satellite takes images of patches on earth
- weather forecast is uncertain
- find solution with lowest expected cost

Generalization of Classical Planning: Multiplayer Games

- Chess
- there is an opponent with a contradictory objective

Generalization of Classical Planning: POMDPs



- Solitaire
- some state information cannot be observed
- must reason over belief for good behaviour

Overview

- Numeric and Temporal
- Plan constraints and preferences
- Non-deterministic
- Multi-agent

- Hybrid continuous and discrete will be covered next week (PDDL+)

Numeric resources

- You can describe numeric resources in PDDL, (classical one), but it is a bit cumbersome.

```
(define (domain vehicle)
 (:requirements :strips :typing)
 (:types vehicle location fuel-level)
 (:predicates (at ?v - vehicle ?p - location)
              (fuel ?v - vehicle ?f - fuel-level)
              (accessible ?v - vehicle ?p1 ?p2 - location)
              (next ?f1 ?f2 - fuel-level))

 (:action drive
   :parameters (?v - vehicle ?from ?to - location
                  ?fbefore ?fafter - fuel-level)
   :precondition (and (at ?v ?from)
                      (accessible ?v ?from ?to)
                      (fuel ?v ?fbefore)
                      (next ?fbefore ?fafter))
   :effect (and (not (at ?v ?from))
                (at ?v ?to)
                (not (fuel ?v ?fbefore))
                (fuel ?v ?fafter))
 )
)
```

PDDL2.1: resources

```
(define (domain metricVehicle)
 (:requirements :strips :typing :fluents)
 (:types vehicle location)
 (:predicates (at ?v - vehicle ?p - location)
              (accessible ?v - vehicle ?p1 ?p2 - location))
 (:functions  (fuel-level ?v - vehicle)
              (fuel-used ?v - vehicle)
              (fuel-required ?p1 ?p2 - location)
              (total-fuel-used))

 (:action drive
   :parameters (?v - vehicle ?from ?to - location)
   :precondition (and (at ?v ?from)
                      (accessible ?v ?from ?to)
                      (>= (fuel-level ?v) (fuel-required ?from ?to)))
   :effect (and (not (at ?v ?from))
                (at ?v ?to)
                (decrease (fuel-level ?v) (fuel-required ?from ?to))
                (increase (total-fuel-used) (fuel-required ?from ?to))
                (increase (fuel-used ?v) (fuel-required ?from ?to)))
 )
)
```

```
(define (problem metricVehicle-example)
 (:domain metricVehicle)
 (:objects
      truck car - vehicle
      Paris Berlin Rome Madrid - location)
 (:init
      (at truck Rome)
      (at car Paris)
      (= (fuel-level truck) 100)
      (= (fuel-level car) 100)
      (accessible car Paris Berlin)
      (accessible car Berlin Rome)
      (accessible car Rome Madrid)
      (accessible truck Rome Paris)
      (accessible truck Rome Berlin)
      (accessible truck Berlin Paris)
      (= (fuel-required Paris Berlin) 40)
      (= (fuel-required Berlin Rome) 30)
      (= (fuel-required Rome Madrid) 50)
      (= (fuel-required Rome Paris) 35)
      (= (fuel-required Rome Berlin) 40)
      (= (fuel-required Berlin Paris) 40)
      (= (total-fuel-used) 0)
      (= (fuel-used car) 0)
      (= (fuel-used truck) 0)
 )
 (:goal (and (at truck Paris)
             (at car Rome))
 )
 (:metric minimize (total-fuel-used))
)
```

Initialise numeric values with (= (type name) VALUE)

You can of course specify a numeric metric to minimise, to shape the quality of generated plans

PDDL2.1 Temporal aspects

Specify the duration of an action (can also be a formula)

It is possible (and mandatory) to specify when preconditions have to be satisfied, and when effects will apply

```
(:durative-action load-truck
   :parameters (?t - truck)
               (?l - location)
               (?o - cargo)
               (?c - crane)
   :duration (= ?duration 5)
   :condition (and (at start (at ?t ?l))
                   (at start (at ?o ?l))
                   (at start (empty ?c)
                   (over all (at ?t ?l))
                   (at end (holding ?c ?o))
   :effect (and (at end (in ?o ?t))
                (at start (holding ?c ?o))
                (at start (not (at ?o ?l)))
                (at end (not (holding ?c ?o)))

)
```

PDDL2.1

- Numeric and temporal aspects are entangled.
  - You can explicitly specify modifications over time via:

    (decrease (fuel-level ?p) (*#t (consumption-rate ?p)))

  - Bear always in mind that while the description is continuous, planning engines need to discretise in order to solve and handle problems.
  - Discretisation can lead to issues and invalid plans.

Temporal: Action concurrency

```
(:durative-action fly
     :parameters (?p - airplane ?a ?b - airport)
     :duration (= ?duration (flight-time ?a ?b))
     :condition (and (at start (at ?p ?a))
                     (over all (inflight ?p))
                     (over all (>= (fuel-level ?p) 0)))
     :effect (and (at start (not (at ?p ?a)))
                  (at start (inflight ?p))
                  (at end (not (inflight ?p)))
                  (at end (at ?p ?b))
                  (decrease (fuel-level ?p)
                            (* #t (fuel-consumption-rate ?p)))))

(:action midair-refuel
     :parameters (?p)
     :precondition (inflight ?p)
     :effect (assign (fuel-level ?p) (fuel-capacity ?p)))
```

Very easy example. You can also specify that actions must start together, finish together, completely overlap, etc… By playing with the pre and post conditions

PDDL3: soft goals and trajectory

Soft-goals: I don’t have absolutely to do it, but I’ll get a reward if I do.
Trajectory constraints: I must fulfil the request before reaching a goal state.

Changes can be applied to the goal description (as below) and to the metric to minimise

```
<GD> ::= (at end <GD>) | (always <GD>) | (sometime <GD>) |
         (within <num> <GD>) | (at-most-once <GD>) |
         (sometime-after <GD> <GD>) | (sometime-before <GD> <GD>)
         (always-within <num> <GD> <GD>) |
         (hold-during <num> <num> <GD> |
         (hold-after <num> <GD> | ...
```

Non-determistic Planning

Non-deterministic

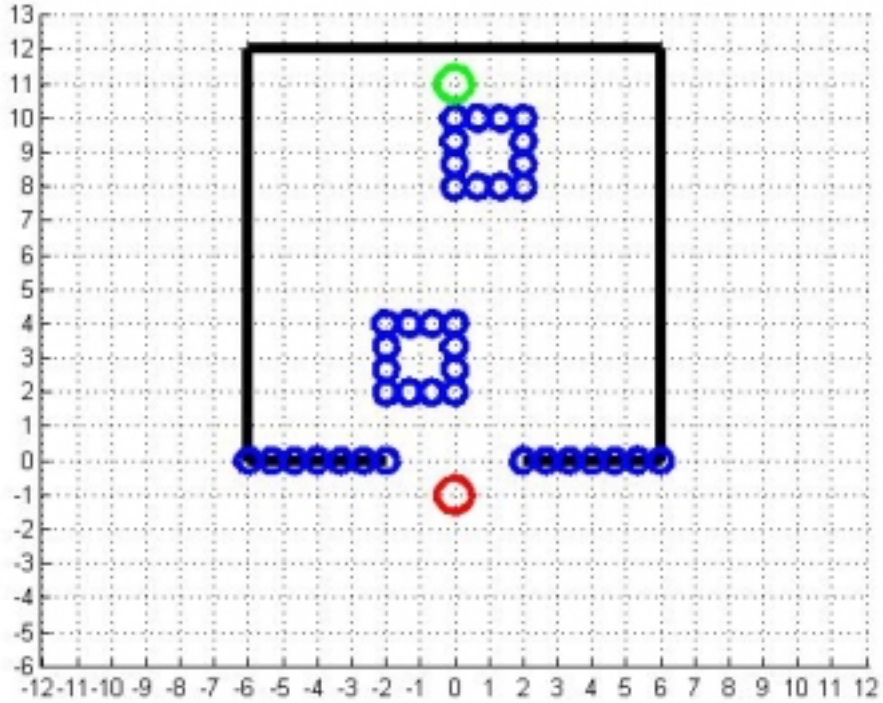
- Conformant planning
- Contingent planning
- Probabilistic planning
- plan-monitor-replan

Conformant Planning

- Very extreme:
  - I have no idea of my initial state, have no sensors to observe the situation, and my actions can possibly fail.
  - The generated plan must be extremely robust, and should **always** work.
    - Such a plan may not even exist!

Conformant Planning

- Typical example
  - A robot with no sensors (green on top) that has to leave a room
  - Any idea of a conformant plan?



- Limitations: Many real world domains are too complicated for this approach, and you can't come up with plans that work regardless of what the state of the world is.

Contingent Planning

- Very similar to conformant, but actions may fail:
  - actions may fail, but you can observe the results.
- Approach: plan ahead for different possible results of each action
- A solution plan looks like a tree of actions, where the path to follow is selected at execution time according to what is observed.

Example

- Suppose now that we have a non-deterministic blocks-world.
- Moving a block may be successful or not.
  - If successful, the block ends up where we wanted.
  - If unsuccessful, the block falls on the table.
- To model this, we need to allow the effects of an action to include disjunctions:

Action(Move(block, from, to),
PRECOND: On(block, from) AND Clear(block) AND Clear(to)
EFFECT: NOT(On(block, from)) AND Clear(from) AND ((On(block, to) AND NOT(Clear(to))) OR (On(block, table))))

Action(MoveToTable(block, from),
PRECOND: On(block, from) AND Clear(block)
EFFECT: On(block, Table) AND NOT(On(block, from)) AND Clear(from)

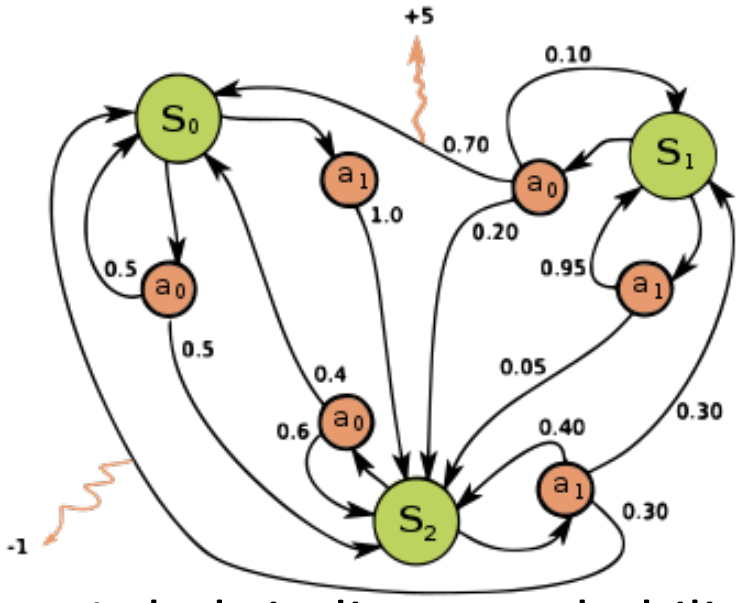
- We also need to allow plans to have if statements.
- Example:
  - move(A, B, C)
    - If on(A, C) then done
    - else: move(A, table, C)
      - If on(A, C) then done
      - else:move(A, table, C)
        - If on(A, C) then done ...
- A plan is

Solutions

- Theoretically, the execution of such a plan may never terminate.
- In practice, one of the attempts should be successful, and the plan should eventually terminate.
- This behaviour matches real-world plan execution:
  - Usually there is no 100% guarantee that a plan will work.
  - In practice, well-made plans tend to work.
  - Contingent plans look like decision trees. At each node, we choose a subtree based on a condition.

Probabilistic Planning

- This type of planning is based on the idea that you have full observability, but each action has a probability distribution over its effect.
  - Typically, this is dealt with via a Markov Decision Process framework.



Example from Wikipedia. Green -> states, red actions. Labels indicate probability

Probabilistic planning

- It has its own track in the IPC
- The reference language is RDDL, Relational Dynamic Influence Diagram Language
  - (there is also (P)PDDL, but it's less used)
- In RDDL, states, actions, and observations are parameterized variables and the evolution of a fully or partially observed (stochastic)process is specified via (stochastic) functions over next state variables conditioned on current state and action variables

PPDDL

- Introduced in 2003, basically an extension of PDDL2.1 with probabilities
- Issues: no concurrency, no time, only allows 1 independent event to affect fluent

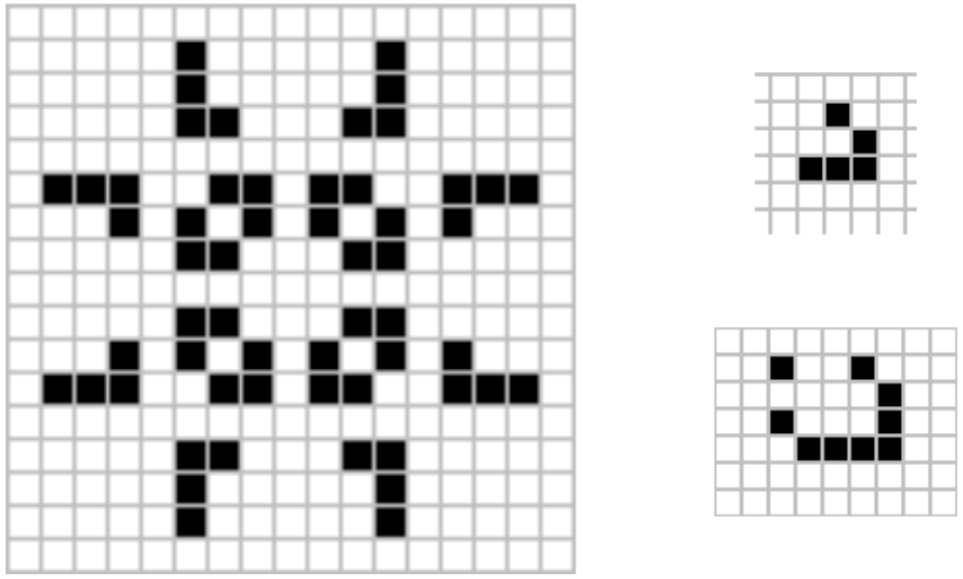
```
(define (domain bomb-and-toilet)
   (:requirements :conditional-effects :probabilistic-effects)
   (:predicates (bomb-in-package ?pkg) (toilet-clogged)
                (bomb-defused))
   (:action dunk-package
            :parameters (?pkg)
            :effect (and (when (bomb-in-package ?pkg)
                               (bomb-defused))
                         (probabilistic 0.05 (toilet-clogged)))))

(define (problem bomb-and-toilet)
   (:domain bomb-and-toilet)
   (:requirements :negative-preconditions)
   (:objects package1 package2)
   (:init (probabilistic 0.5 (bomb-in-package package1)
                         0.5 (bomb-in-package package2)))
   (:goal (and (bomb-defused) (not (toilet-clogged)))))
```

Figure 1: PPDDL encoding of "Bomb and Toilet" example.

RDDL example of use

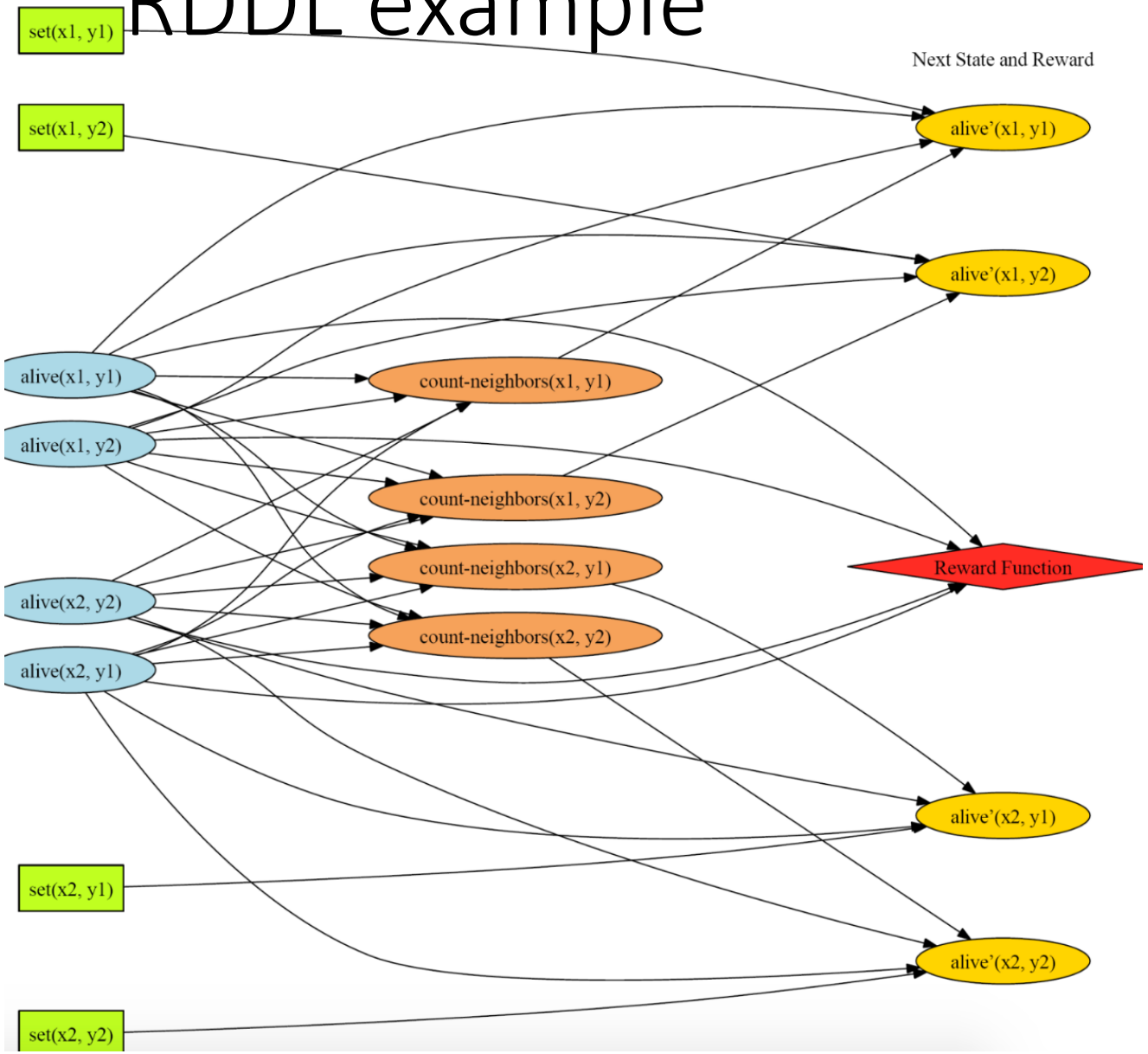
Lifting: Conway's Game of Life

- 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

RDDL example



Example of how actions and variables affect each other (lifted overview

RDDL example

```
// 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

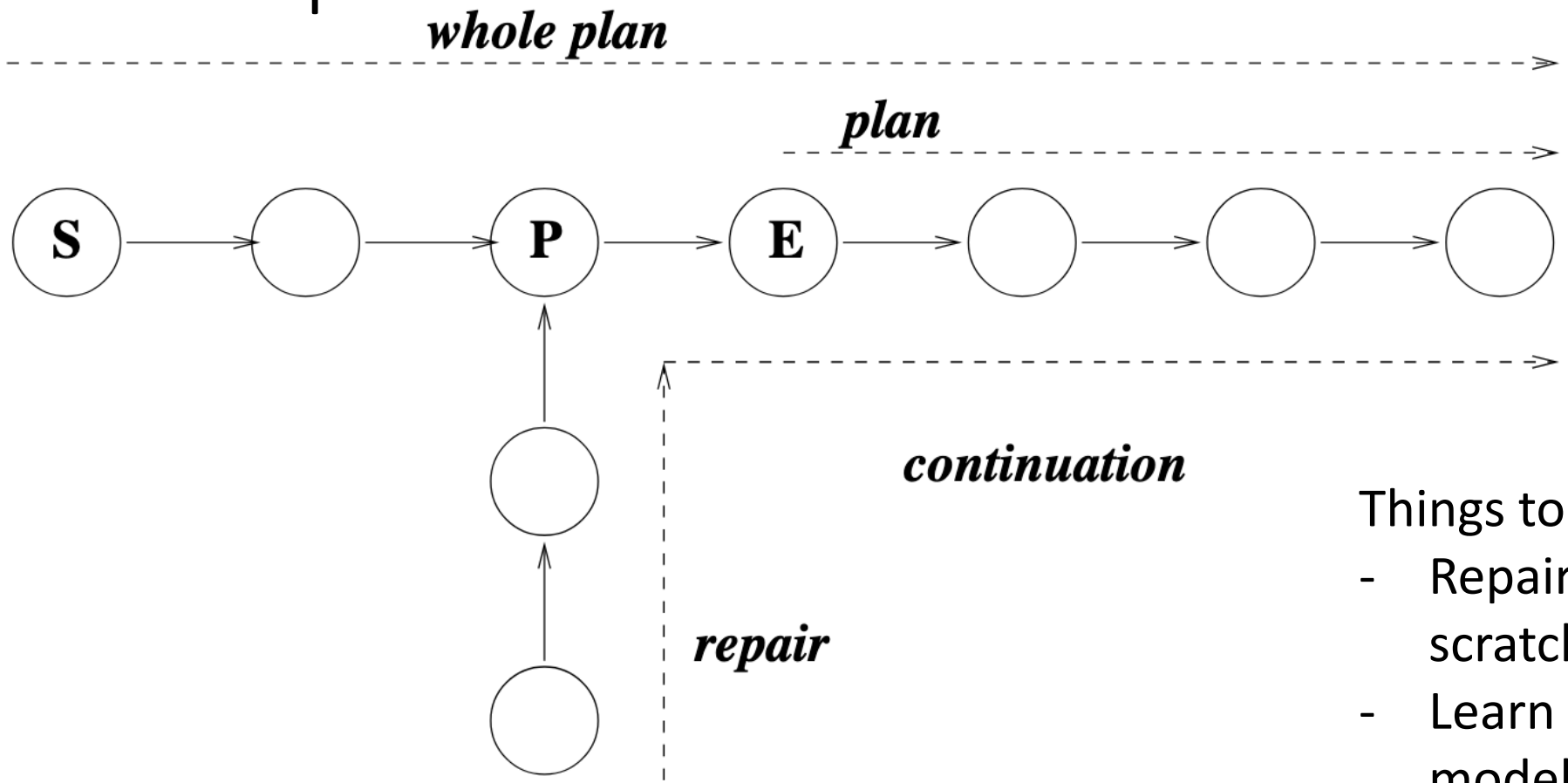
Plan – monitor - replan

- Contingency planning:
  - Prepare in advance. Useful when some conditions needed for the contingency plan can be gathered before execution.
- Execution monitoring:
  - ignore contingencies during planning, then handle them as they arise. Useful when planning time is a concern: not everything can be planned for.
  - Basic idea: handle failures if they arise during execution

Replanning

- Before executing the next action, the agent monitors the enviroment.
- There are different choices as to exactly what to monitor.
  - **Action monitoring**: verify that the preconditions of the next action still hold.
  - **Plan monitoring**: verify that the remaining plan will still succeed, given the current state of the world.
  - **Goal monitoring**: check to see if, given the current state of the world, there is a better plan to follow.
- Action monitoring is the most simple and efficient.
- Goal monitoring can take advantage of unexpected changes that may make it easier to achieve the goal.

Example



Things to consider:

- Repair or replan from scratches?
- Learn / modify domain model?
- “don’t touch” conditions?

The robot video shown in week1, is based on this framework!

Multi-agent

Multi-agent

- There can be more than one agent to control..
  - Centralised: complete plan *for* all agents in the same common search episode.
    - A problem with this approach is that the complexity grows exponentially with the number of agents in general.
    - Share Agents' knowledge (no privacy).
  - Distributed planning consists of each agent solving its own planning task; planning is performed *by* the agents.
    - Issues with coordination (social law, communication, etc.)
    - Issues with plans that can invalidate others'…
      - Notion of loosely and tightly coupled

MA-PDDL

- Introduced in 2012 by extending PDDL3
  - Not extremely used, but at least an attempt to standardise the field
  - Mostly for centralised planning
    - Can specify many agents, but the plan is shared

- Example: https://github.com/aig-upf/universal-pddl-parser-multiagent/blob/master/domains/codmap15/blocksworld/domain/domain.pddl

- There is a lot of work in the area, so it is hard to indicate how it will evolve in the future.
  - A competition was organised a few years ago (2015)

Of course…

- There are extensions that we did not have time to explore during the lecture.
  - Conditional effects.
    - (when CONDITION_FORMULA EFFECT_FORMULA)
      The interpretation is that the specified effect takes place only if the specified condition formula is true in the state where the action is executed. Conditional effects are usually placed within quantifiers.
    - A universally quantified formula:
      (forall (?V1 ?V2 ...) EFFECT_FORMULA)

Temporal and numeric planning

- lpg-td and metric ff can handle numeric and temporal planning.

- https://bitbucket.org/enricode/the-enhsp-planner/src/master/
  - ENHSP, another very good planner for **numeric**, developed in Java. Does not support durative-actions

  - Looking at the Zenotravel numeric domain, can you think about a fly-extrafast action that requires the full tank, and arrives with empty tank? Modify problem1 so that the metric to maximise is fuel consumption, and give it a try
    - https://bitbucket.org/enricode/the-enhsp-planner/src/master/ijcai16_benchmarks/ZenoTravel/Numeric/zenonumeric.pddl

References

- PDDL 2.1
  - https://arxiv.org/pdf/1106.4561.pdf
- PDDL 3
  - http://www.cs.yale.edu/homes/dvm/papers/pddl-ipc5.pdf
  - Alfonso Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition:PDDL3 and experimental evaluation of the planners. Artificial Intelligence,173(5-6):619–668, 2009.
- Conformant planning
  - D. E. Smith and D. Weld. Conformant Graphplan. Proceedings of AAAI'98, pages 889–896.
  - Blai Bonet and Robert Givan. 5th international planning competition: Non-deterministic track call for participation. In Proceedings of the 5th International Planning Competition IPC 2006
- Contingent Planning
  - Joerg Hoffmann and Ronen Brafman. Contingent planning via heuristic forward search with implicit belief states. In Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)
- Replanning with robots
  - https://core.ac.uk/download/pdf/196599596.pdf
- RDDL
  - http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf
  - http://users.cecs.anu.edu.au/~ssanner/Papers/RDDL_Tutorial.pdf
- Multiagent Planning
  - https://link.springer.com/article/10.1007/s10115-018-1202-1