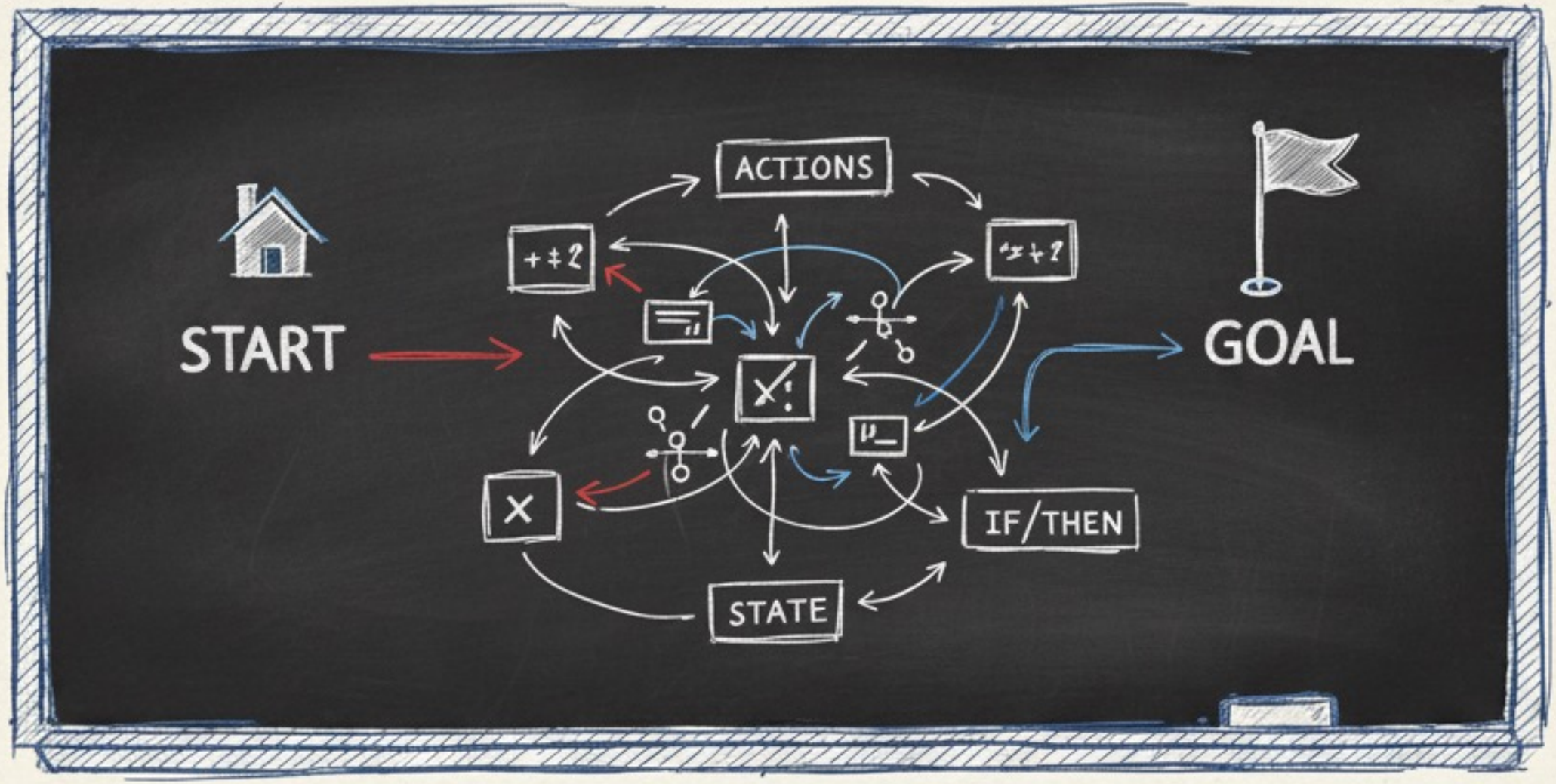
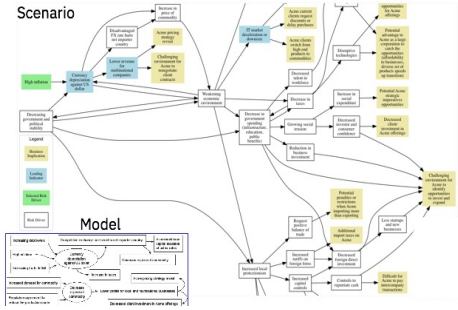
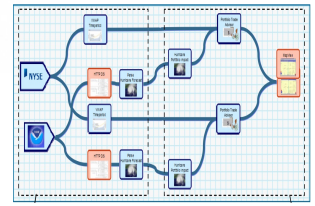
# Session 1: What is AI Planning

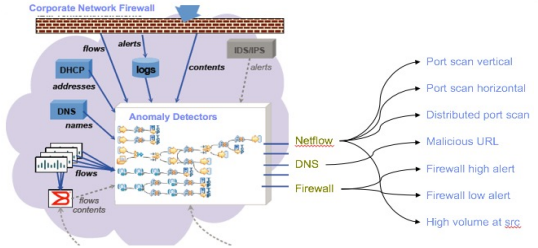


# What is AI Planning

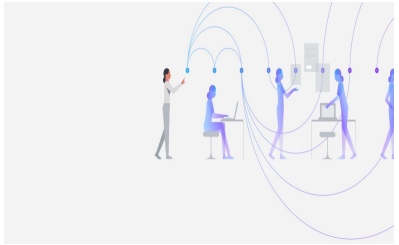
A subfield of Artificial Intelligence focused on **automating the decision-making process**.

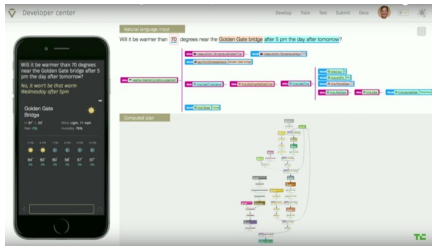


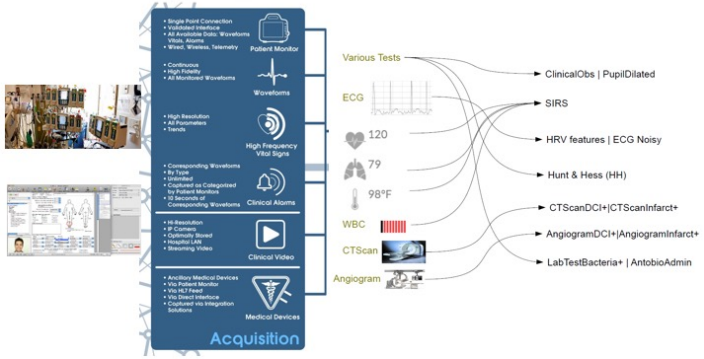



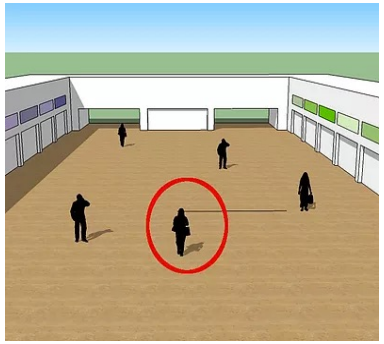


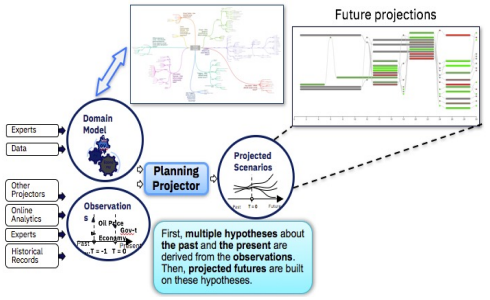


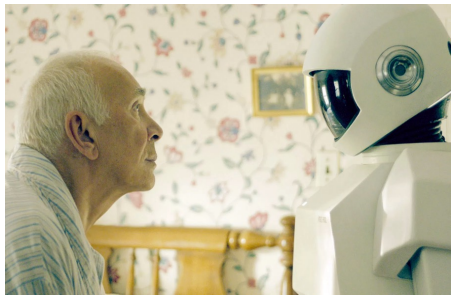


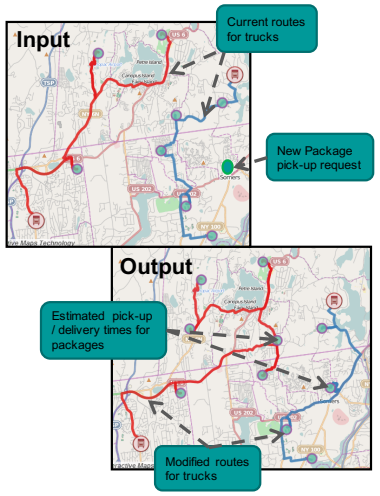









# Planning Problem

The 24 Game is a mathematical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. The game is played with a list of four numbers, and the player must use all four numbers exactly once, using any combination of addition, subtraction, multiplication, or division, to arrive at the number 24.

Mathematical Model: $\mathcal{S} = \langle S, s_0, S_G, A, f, c \rangle$

- finite and discrete state space $S$
- a known initial state $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a deterministic transition function
  $s' = f(a, s)$ for $a \in A(s)$
- non-negative action costs $c(a, s)$

A solution is a sequence of applicable actions that maps $s_0$ into $S_G$, and it is optimal if it minimizes sum of action costs

Planning Language

types
objects
predicates
initial state
actions
goal
preconditions
effects

Black Box

goal: S -> {T,F}

succ: S ->$2^{SxA}$

# Planning Formalisms

- Classical planning: initial state is **known**, action dynamics is **deterministic** and **instantaneous**, **discrete and finite** state space, **flat** hierarchies, **hard** goals that must be achieved
- Numeric Planning: numeric state variables, **infinite** state spaces
- Net-benefit/oversubscription planning: **utility of states, possibly no hard goals**
- Conformant planning: initial state is **uncertain**
- Probabilistic planning: action dynamics is **probabilistic**, and state spaces **do not have to be discrete or finite**
- Non-deterministic planning (ND): action dynamics is **non-deterministic**
- Temporal Planning: actions have **durations** and **temporal constraints**
- Hierarchical Task Network (HTN): initial state/goal are task networks (a set of tasks and constraints), with **recursive decomposition of high-level tasks into lower-level sub-tasks**

# Computational Problems

Classical/Numeric Planning

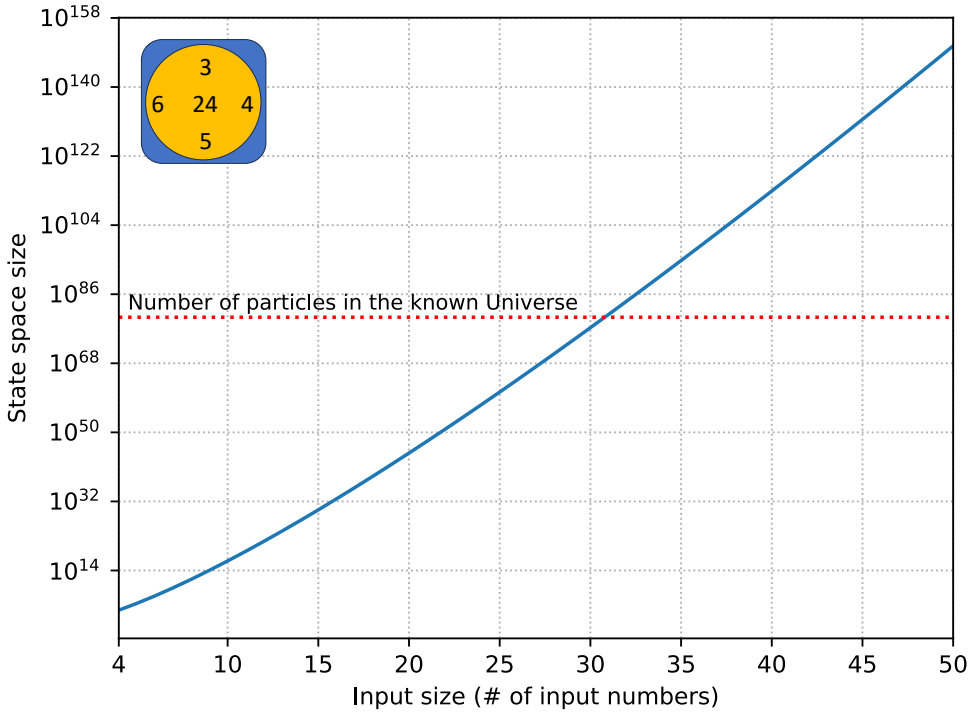
- **Agile planning**: find a plan, quicker is better
- **Satisficing planning**: find a plan, cheaper plans a better
- **Cost-optimal planning**: find a plan that minimizes summed operator cost
- **Top-k planning**: find k plans such that no cheaper plans exist
- **Top-quality planning**: find all plans up to a certain cost
- **Diverse planning**: variety of problems, aiming at obtaining diverse set of plans, considering plan quality as well

Beyond Classical Planning (examples)

- **Net-benefit planning**: find a plan that minimizes the difference between utility of end state and summed operator cost
- **Oversubscription planning**: find a plan that maximizes the utility of end state, bounding summed operator cost
- **FOND planning**: find a policy that guarantees eventual goal achievement under fairness of outcomes assumption
- **PO Probabilistic planning**: find a policy that maximizes expected utility

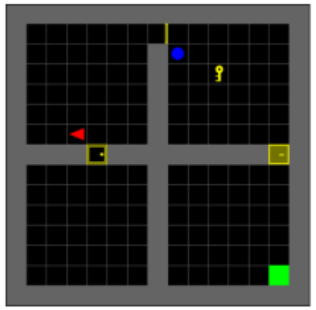
# Planning Complexity

- Essentially the problem of finding a path in a graph
- Can be done in poly-time  in the size of the graph
- The planning language representation is more compact
- The simplest setting of classical agile planning (solving planning tasks given in planning language) is PSPACE-hard
- Numeric planning: undecidable in general, under strict restrictions becomes PSPACE-hard
- Temporal planning: undecidable in general, under strict restrictions becomes EXPSPACE-hard, even stricter restrictions make it PSPACE-hard



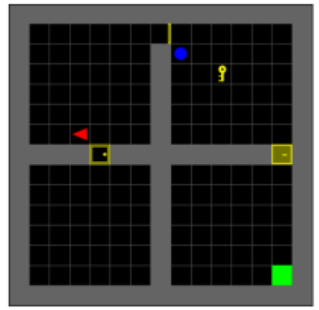
# Planning Languages – PDDL

```
(define (domain grid)
(:requirements :strips :typing)
  (:types place key shape - object)
(:predicates (conn ?x - place ?y - place) (locked ?x - place) (holding ?k - key)
    (key-shape ?k - key ?s - shape) (lock-shape ?x - place ?s - shape)
    (at ?r - key ?x - place) (at-robot ?x - place) (open ?x - place) (arm-empty ))

(:action unlock-door
  :parameters (?curpos - place ?lockpos - place ?key - key ?shape - shape)
  :precondition (and (conn ?curpos ?lockpos) (key-shape ?key ?shape)
    (lock-shape ?lockpos ?shape) (at-robot ?curpos) (locked ?lockpos) (holding ?key))
  :effect (and (not (locked ?lockpos))))
(:action open-door
  :parameters (?curpos - place ?lockpos - place)
  :precondition (and (conn ?curpos ?lockpos) (at-robot ?curpos) (not (locked ?lockpos)))
  :effect (and (open ?lockpos)))
(:action move
  :parameters (?curpos - place ?nextpos - place)
  :precondition (and (at-robot ?curpos) (conn ?curpos ?nextpos) (open ?nextpos))
  :effect (and (at-robot ?nextpos) (not (at-robot ?curpos))))
(:action pickup
  :parameters (?curpos - place ?key - key)
  :precondition (and (at-robot ?curpos) (at ?key ?curpos) (arm-empty ))
  :effect (and (holding ?key) (not (at ?key ?curpos)) (not (arm-empty ))))
(:action pickup-and-loose
  :parameters (?curpos - place ?newkey - key ?oldkey - key)
  :precondition (and (at-robot ?curpos) (holding ?oldkey) (at ?newkey ?curpos))
  :effect (and (holding ?newkey) (at ?oldkey ?curpos)
        (not (holding ?oldkey)) (not (at ?newkey ?curpos))))
(:action putdown
  :parameters (?curpos - place ?key - key)
  :precondition (and (at-robot ?curpos) (holding ?key))
  :effect (and (arm-empty ) (at ?key ?curpos) (not (holding ?key)))))
```



```
(define (problem grid-x2-y2-t1-k1-l1)
(:domain grid)
(:objects
  f0-0f f1-0f  f0-1f f1-1f  - place
  shape0 - shape
  key0 - key)

(:init
  (key-shape key0 shape0)
  (lock-shape f1-0f shape0)
  (conn f0-0f f0-1f) (conn f0-1f f1-1f) (conn f1-0f f1-1f)
  (arm-empty) (at-robot f0-1f) (at key0 f1-1f)
  (open f0-1f) (open f1-1f) (locked f1-0f))

(:goal (and (at key0-0 f2-0f) (at key0-1 f1-3f))))
```

# Planning Languages – PDDL



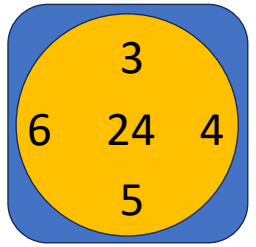
```
(:action move
  :parameters (?curpos - place ?nextpos - place)
  :precondition (and (at-robot ?curpos) (conn ?curpos ?nextpos) (open ?nextpos))
  :effect (and (at-robot ?nextpos) (not (at-robot ?curpos))))
```

# Planning Languages – PDDL

```
(define (domain sokoban-sequential)
  (:requirements :typing :action-costs)
  (:types thing location direction - object
          player stone - thing)
  (:predicates (clear ?l - location) (at ?t - thing ?l - location)
    (IS-GOAL ?l - location) (IS-NONGOAL ?l - location)
    (at-goal ?s - stone) (MOVE-DIR ?from ?to - location ?dir - direction))
  (:functions (total-cost) - number)

  (:action move
   :parameters (?p - player ?from ?to - location ?dir - direction)
   :precondition (and (at ?p ?from) (clear ?to) (MOVE-DIR ?from ?to ?dir))
   :effect       (and (not (at ?p ?from)) (not (clear ?to)) (at ?p ?to) (clear ?from)))

  (:action push-to-nongoal
   :parameters (?p - player ?s - stone   ?ppos ?from ?to - location ?dir - direction)
   :precondition (and (at ?p ?ppos) (at ?s ?from) (clear ?to)
     (MOVE-DIR ?ppos ?from ?dir) (MOVE-DIR ?from ?to ?dir) (IS-NONGOAL ?to))
   :effect       (and (not (at ?p ?ppos)) (not (at ?s ?from)) (not (clear ?to))
     (at ?p ?from) (at ?s ?to) (clear ?ppos) (not (at-goal ?s)) (increase (total-cost) 1)))

  (:action push-to-goal
   :parameters (?p - player ?s - stone   ?ppos ?from ?to - location ?dir - direction)
   :precondition (and (at ?p ?ppos) (at ?s ?from) (clear ?to)
     (MOVE-DIR ?ppos ?from ?dir) (MOVE-DIR ?from ?to ?dir) (IS-GOAL ?to))
   :effect       (and (not (at ?p ?ppos)) (not (at ?s ?from)) (not (clear ?to))
     (at ?p ?from) (at ?s ?to) (clear ?ppos) (at-goal ?s) (increase (total-cost) 1)))
)
```

```
(define (problem p012-microban-sequential)
  (:domain sokoban-sequential)
  (:objects
    up down left right - direction
    player-01 - player
    pos-1-1 pos-1-2 ... pos-9-8 - location
    stone-01 stone-02 - stone
  )
  (:init
    (IS-NONGOAL pos-1-1) (IS-NONGOAL pos-1-2) ... (IS-NONGOAL pos-9-8)
    (IS-GOAL pos-6-5) (IS-GOAL pos-6-6)
    (MOVE-DIR pos-1-5 pos-1-6 down) ... (MOVE-DIR pos-9-3 pos-8-3 left)
    (at player-01 pos-5-5) (at stone-01 pos-3-3) (at stone-02 pos-4-4)
    (clear pos-1-5) (clear pos-1-6) ... (clear pos-9-3)
    (= (total-cost) 0))

  (:goal (and     (at-goal stone-01)   (at-goal stone-02)))
  (:metric minimize (total-cost))
)
```

# Planning Languages – PDDL




```
(define (problem c01)
  (:domain countdown)
   (:objects n1 n2 n3 n4 - num)
  (:init
   (= (value n1) 3) (= (value n2) 4) (= (value n3) 5) (= (value n4) 6)
   (= (targetvalue) 24)
   (= (numactive) 4)
   (active n1) (active n2) (active n3) (active n4)
  )
  (:goal (and (goalreached)))
)
```

```
(define (domain countdown)
  (:types num - object)
  (:predicates (active ?o - num) (goalreached))
  (:functions (value ?o - num) (targetvalue) (numactive))

  (:action add
   :parameters (?a ?b - num)
   :precondition (and (not (= ?a ?b)) (active ?a) (active ?b))
   :effect       (and (decrease (numactive) 1) (increase (value ?a) (value ?b)) (not (active ?b))))
  (:action subtract
   :parameters (?a ?b - num)
   :precondition (and (not (= ?a ?b)) (active ?a) (active ?b) (>= (value ?a) (value ?b)))
   :effect       (and (not (active ?b)) (decrease (numactive) 1) (decrease (value ?a) (value ?b))))
  (:action multiply
   :parameters (?a ?b - num)
   :precondition (and (not (= ?a ?b)) (active ?a) (active ?b))
   :effect       (and (not (active ?b)) (decrease (numactive) 1) (assign (value ?a) (* (value ?a) (value ?b)))))
  (:action divide
   :parameters (?a ?b - num)
   :precondition (and (> (value ?b) 0) (not (= ?a ?b)) (active ?a) (active ?b))
   :effect       (and (not (active ?b)) (decrease (numactive) 1) (assign (value ?a) (/ (value ?a) (value ?b)))))
  (:action checkgoal
   :parameters (?a - num)
   :precondition (and (active ?a) (= (numactive) 1) (= (value ?a) (targetvalue)))
   :effect       (and (goalreached)))
)
```

# Planning Languages – PDDL-> STRIPS->FDR

**PDDL**

Grounding

A Strips Planning task is 5-tuple $\Pi = \langle F, O, c, I, G \rangle$:

- $F$: finite set of atoms (boolean variables)
- $O$: finite set of operators (actions) of form $\langle Add, Del, Pre \rangle$ (Add/Delete/Preconditions; subsets of atoms)
- $c : O \mapsto \mathbb{R}^{0+}$ captures operator cost
- $I$: initial state (subset of atoms)
- $G$: goal description (subset of atoms)

Plan: sequence of applicable actions that maps $I$ into a state consistent with $G$

A STRIPS Planning task $\Pi = \langle F, O, c, I, G \rangle$ determines state model $\mathcal{S}(\Pi)$ where

- the states $s \in S$ are collections of atoms from $F$
- the initial state $s_0$ is $I$
- the goal states $s$ are such that $G \subseteq s$
- the actions $a$ in $A(s)$ are ops in $O$ s.t. $Pre(a) \subseteq s$
- the next state is $s' = s - Del(a) + Add(a)$
- action costs $c(a, s) = c(a)$

♠ Solutions of $\mathcal{S}(\Pi)$ are plans of $\Pi$

**STRIPS**

Translation

An FDR Planning task is 5-tuple $\Pi = \langle V, O, s_0, s_*, cost \rangle$:

- $V$: finite set of finite-domain multi-valued variables
- $O$: finite set of operators (actions) of form $\langle pre, eff \rangle$ (Preconditions/Effects; partial variable assignments)
- $cost : O \mapsto \mathbb{R}^{0+}$ captures operator cost
- $s_0$: initial state (variable assignment)
- $s_*$: goal description (partial variable assignment)

Plan: sequence of applicable actions that maps $s_0$ into a state consistent with $s_*$

**SAS+/FDR**

# Planning Languages – Beyond PDDL

- PPDDL (Probabilistic PDDL) – extension of PDDL to support probabilistic planning
- RDDL (Relational Dynamic influence Diagram Language) – captures POMDPs and more

- PDDL+/OPT – extensions of PDDL with support for processes and events, as well as ontologies

- MA-PDDL (Multi Agent PDDL)

- NDDL/APPL – NASA’s planning languages