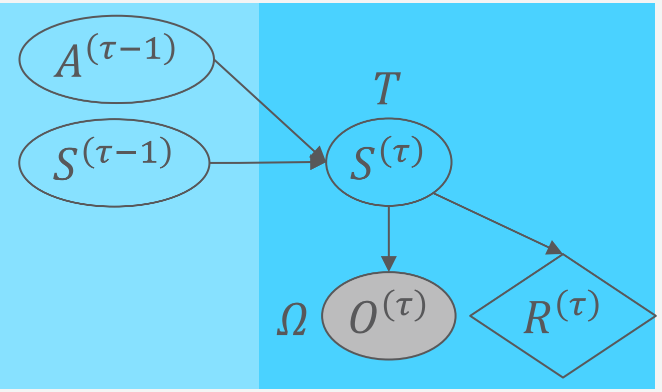
Universität
Münster



Automated Planning and Acting

Decision Making: Structure

living.knowledge

Tanya Braun
Research Group Data Science, Computer Science Department

Content: Planning and Acting

1. With **Deterministic** Models
2. With **Refinement** Methods
3. With **Temporal** Models
4. With **Nondeterministic** Models
5. With **Probabilistic** Models
6. By **Decision Making**
   A. *Foundations*
   B. *Extensions*
   C. *Structure*
      - Lifted DecPOMDPs
      - Factored MDPs
      - First-order MDPs
7. **Human-aware** Planning

Outline: Decision Making – Structure

Structure by Groups in the Agent Set

- Agent types
- Partitioned decPOMDPs

Structure by Features in the State Space

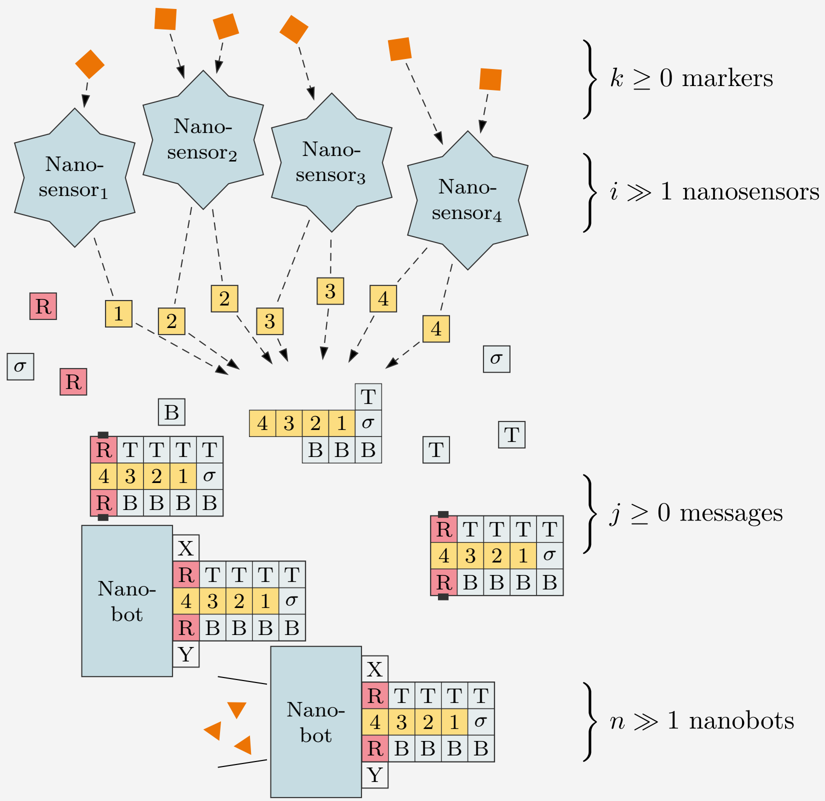
- Dynamic Bayesian networks
- Factored MDPs

Structure by Relations in the State Space

- Situation calculus
- First-order MDPs

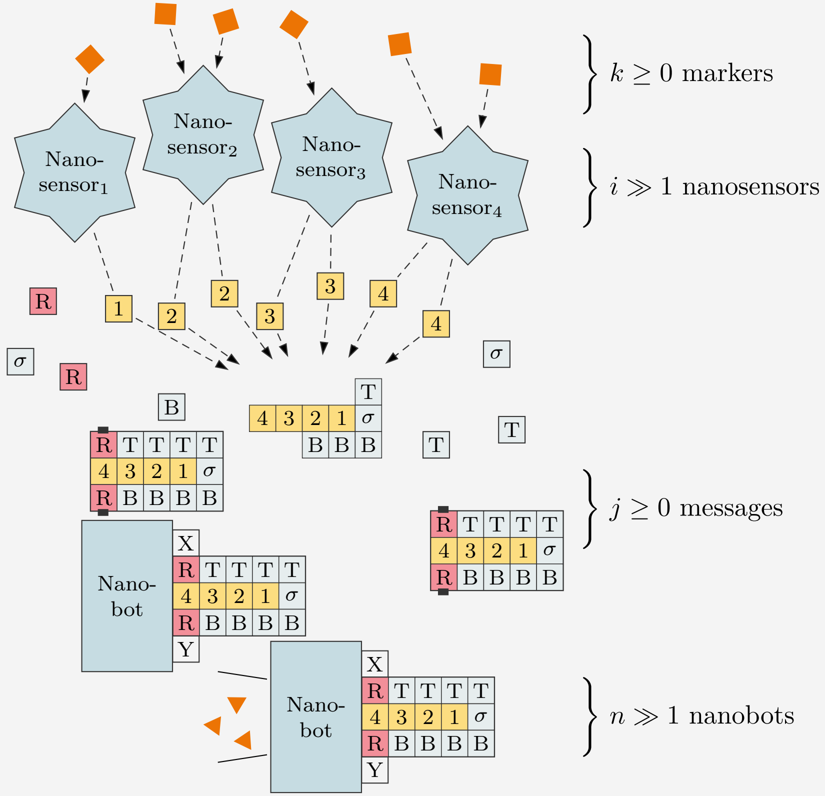
Example: Medical Nanoscale Systems

- Nanoscale systems regularly consist of $>$ 10,000 nanoagents
  - Different types of agents: nanosensors, nanobots
- Application: DNA-based medical system
  - E.g., for diagnosis (modelled as an AND gate)
    - Nanosensors receptive to individual markers for a specific disease
      - Release individual tiles in presence of their individual markers
    - Tiles assemble themselves to form messages
    - Nanobots receptive to completely formed messages
      - Release markers of their own that signify presense of the disease
- Formal model necessary to argue about
  - Success rates
  - Sizes of agent sets



Example: Medical Nanoscale Systems as a DecPOMDP

- Set of agents I consisting of nanosensors, nanobots
- Observations O_i: markers / messages present (or not)
  - Noisy process → probabilistic behaviour
- Actions A_i: release of tiles / markers (or not)
  - Noisy process → probabilistic behaviour
- Environment → probabilistic behaviour
  - Presence in general of agents, markers, tiles, messages, or position more specifically → movement over time
- Reward: Qualitative measure
  - Positive diagnosis only in presence of disease



Reprise: Worst-case Complexity of DecPOMDP

- Space complexity
  - Transition model: $\mathcal{O}(s \cdot s \cdot a^N)$
  - Sensor model: $\mathcal{O}(s \cdot o^N)$ or $\mathcal{O}(s \cdot o^N \cdot a^N)$
  - Reward function: $\mathcal{O}(s)$ or $\mathcal{O}(s \cdot a^N)$
- Runtime complexity of brute-force search
  - Evaluation cost of a joint policy: $\mathcal{O}\left(s \cdot o^{Nh}\right)$
  - Policy space: $\mathcal{O}\left(a^{\frac{N(o^h-1)}{o-1}}\right)$
- Notations
  - $s = |S|$
    - State space size
  - $a = \max_{i \in I} |A_i|$
    - Largest individual action space size
  - $o = \max_{i \in I} |O_i|$
    - Largest individual action space size
  - h
    - Horizon

Agent Types & Partitioned DecPOMDPs

- Types: Agents with the same sets of actions and observations
  - E.g., two nanosensors 1,2 receptive to the same marker and releasing the same tile
    - $A_1 = A_2 = \{0,1\}$; 0: do nothing, 1: release tile
    - $O_1 = O_2 = \{0,1\}$; 0: marker not present, 1: marker present

→ Partitions the set of agents regarding actions, observations
  - Agent set $I = \{I_1, \dots, I_K\}$ with $I_1, \dots, I_K$ a partitioning of I ($I = \bigcup_k I_k, I_k \cap I_{k'} = \emptyset, I_k \neq \emptyset$)
  - For each partition I_k: one set of actions A_k, one set of observations O_k for all agents in I_k
  - Expectation that $K \ll N$
- Additional constraints / assumptions on same behaviour in T, R, Ω

→ Partitions the set of agents completely, enabling more compact encodings
- How?

Counting DecPOMDPs

- Counting constraint / assumption in T, R, Ω
  - Formal: All permutations $\sigma(\vec{a}_k)$ of a partition action $\vec{a}_k$ map to the same probability
  - Enables counting how many agents do something and not which in particular did
    - Encode in a histogram $[\#(a_1), \dots, \#(a_l)]$ how many agents did actions $A_k = \{a_1, \dots, a_l\}$
    - Number of histograms $\binom{|I_k|+l-1}{l-1} \leq |I_k|^l$

S	S'	$A_1^\#$	$\bar{T}(s,s',a_1') = P(s'\|s,a_1')$
0	0	[0,2]	0.01
0	0	[1,1]	0.02
0	0	[2,0]	0.03
0	1	[0,2]	0.015
0	1	[1,1]	0.012
0	1	[2,0]	0.01
1	0	[0,2]	0.01
		⋮	

S	S'	A_1	A_2	$T(s,s',a_1,a_2) = P(s'\|s,a_1,a_2)$
0	0	0	0	0.01
0	0	0	1	0.02
0	0	1	0	0.02
0	0	1	1	0.03
0	1	0	0	0.015
0	1	0	1	0.012
0	1	1	0	0.012
0	1	1	1	0.01
1	0	0	0	0.01
		⋮		

Counting DecPOMDPs

- Complexity-wise, with $n = \max_k |I_k|$
  - Transition model: $\mathcal{O}(s \cdot s \cdot n^{Ka})$
  - Sensor model: $\mathcal{O}(s \cdot n^{Ko})$
  - Reward function: $\mathcal{O}(s)$
  - Evaluation cost: $\mathcal{O}\left(s \cdot n^{Koh}\right)$
  - Reduction if $K \ll N$
- Unfortunately,
  - Policy space: $\mathcal{O}\left(n^{\frac{aK(n^{ho}-1)}{n^{o}-1}}\right)$
- Ongoing research how to use counting efficiently

S	S'	$A_1^{\#}$	$\bar{T}(s,s',a_1') = P(s' \mid s,a_1')$
0	0	[0,2]	0.01
0	0	[1,1]	0.02
0	0	[2,0]	0.03
0	1	[0,2]	0.015
0	1	[1,1]	0.012
0	1	[2,0]	0.01
1	0	[0,2]	0.01
		⋮	

S	S'	A_1	A_2	$T(s,s',a_1,a_2) = P(s' \mid s,a_1,a_2)$
0	0	0	0	0.01
0	0	0	1	0.02
0	0	1	0	0.02
0	0	1	1	0.03
0	1	0	0	0.015
0	1	0	1	0.012
0	1	1	0	0.012
0	1	1	1	0.01
1	0	0	0	0.01
		⋮		

Ismorphic DecPOMDPs

- Isomorphic constraint / assumption in T, R, Ω:
  Conditional independence between agents of a partition given joint state
  → Enables factorisation of T, R, Ω
  - E.g., $T(s, s', a_1, a_2) = \underbrace{T_1(s, s', a_1)} \cdot \underbrace{T_2(s, s', a_2)} = \prod_{i \in I_k} T'(s, s', a_i)$

    $T_1 = T_2 = T'$
- Space complexities
  - Transition model: $\mathcal{O}(s \cdot s \cdot a^K)$
  - Sensor model: $\mathcal{O}(s \cdot o^K)$
  - Reward function: $\mathcal{O}(s)$
- Ongoing research how to solve isomorphic DecPOMDPs efficiently

S	S'	A_i	$T'(s, s', a_i) = P(s' \mid s, a_i)$
0	0	0	0.01
0	0	1	0.03
0	1	0	0.015
0	1	1	0.01
1	0	0	0.01
	⋮		

Interim Summary: Structure by Groups in the Agent Set

- Types of agents with identical action and observation space
- Partitioned DecPOMDP if agent types + constraints of transition / sensor / reward function
- Counting DecPOMDP
  - Permutations of actions of agents of the same partition map to the same probability / reward
  - Count occurrences → encode in histograms
- Isomorphic DecPOMDP
  - Further independences between agents of a partition
- Space complexity polynomial at worst but using compact encoding for efficient decision making not yet solved

Outline: Decision Making – Structure

Structure by Groups in the Agent Set

- Agent types
- Partitioned decPOMDPs

Structure by Features in the State Space

- Dynamic Bayesian networks
- Factored MDPs

Structure by Relations in the State Space

- Situation calculus
- First-order MDPs

State Space

- So far: State space treated as a black box with a set of different states as domain of a random variable S
- However, state space often has structure
  - n different features that describe a state space
  - Encode in n individual random variables S_i with respective domains $\mathrm{dom}(S_i) = \{v_1, \dots, v_{d_i}\}$
    - State space size then describable as $|S| = \prod_i d_i \leq d^n, d = \max_i d_i$
      - I.e., exponential in the number of random variables
- Given (conditional) independences between different S_i, factorisation of probability distributions in model possible
  - Applicable to MDPs, POMDPs, DecPOMDPs, partitioned DecPOMDPs
  - Most work exists for factored MDPs (also the simplest case to consider)

Factorisation in General

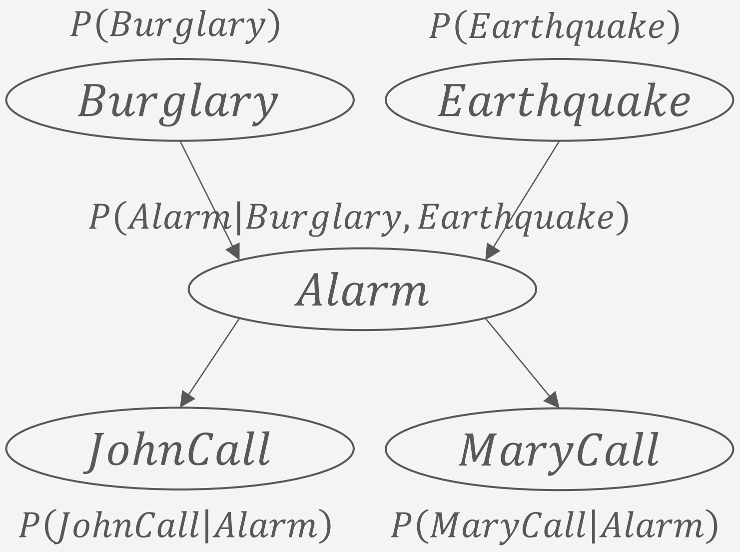
- (Conditional) independences:
  - $A \perp B$ (A, B independent) $\Leftrightarrow P(A,B) = P(A) \cdot P(B)$
  - $A \perp B \mid C$ (A, B conditionally independent given C) $\Leftrightarrow P(A,B|C) = P(A|C) \cdot P(B|C)$
    - Alternate version: $A \perp B \mid C \Leftrightarrow P(A|B,C) = P(A|C)$
- (Conditional) independences allow for factorising a distribution into smaller factors
  - In general: Factorisation of a full joint probability distribution $P(S_1, \dots, S_n)$ into m factors over subsets $\boldsymbol{C}$ of random variables that form $P(S_1, \dots, S_n)$ after multiplication (and normalisation):

$$P(S_1, \dots, S_n) = \frac{1}{Z} \prod_{j=1}^{m} \phi(\boldsymbol{C}_j)$$

    - Where $\boldsymbol{C}_j$ refers to sets of random variables that are mutually dependent on each other
    - Memory complexity: $\mathcal{O}(d^n)$ vs. $\mathcal{O}\left(m \cdot d^{|\boldsymbol{C}_{max}|}\right)$

Probabilistic Graphical Models (PGMs)

- PGMs use a graph structure to represent dependences
  - Common formalism: Bayesian network (BN) B
    - Directed acyclic graph
      - Nodes: random variables S_i
      - Edges: if S_i depends on S_j, edge $S_j \longrightarrow S_i$
    - Factors: conditional probability distributions (CPDs) $\forall i\ P(S_i|\text{pa}(S_i))$
      - Roots: $\text{pa}(S_i) = \emptyset \rightarrow$ Prior distributions $P(S_i)$
      - Usually not depicted in graph; have to be denoted somewhere
      - Semantics: $P(S_1, \ldots, S_n) = \prod_{i=1}^{n} P(S_i|\text{pa}(S_i))$
  - Not further considered here:
    Undirected version with potential functions ϕ as factors:
    - Factor graphs, Markov networks
    - Same semantics, different graphical representation



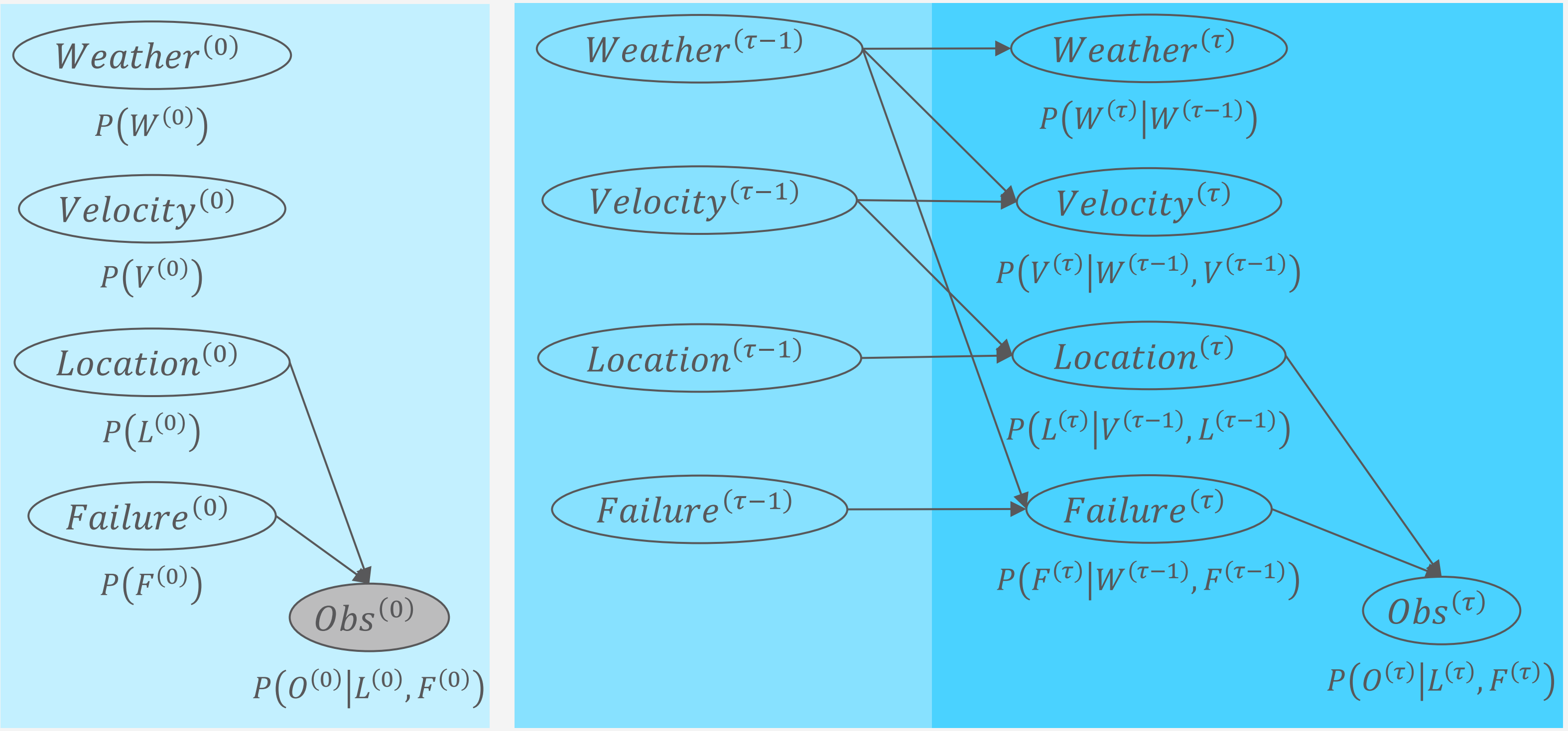
Full joint probability distribution size: d^5
Sizes of CPDs: $d + d + d^3 + d^2 + d^2$
Given $d = 2$: $2^5 = 32$ vs. 20
(As probabilities add to 1:
size -1 for each probability distribution in each CPD,
i.e., $1 + 1 + 4 + 2 + 2 = 10$)

Dynamic Bayesian Networks

- MDP models a sequential, i.e., temporal, stationary, Markovian probabilistic setting
  - Factorisation also needs to encode a sequential, stationary, Markovian probabilistic setting
- Popular modeling formalism used:
  Dynamic BN (DBN) is a two-tuple $\left(B^{(0)}, B^{(\rightarrow)}\right)$
  - Template variables S_i indexed by time step τ in BNs
    → Can be instantiated for particular time steps t
  - BN $B^{(0)}$ for time step 0 to encode
    - If set to uniform distributions or using DBN for fix point calculations, can be safely ignored
  - BN $B^{(\rightarrow)}$ for time step τ with connections from time step $\tau - 1$ (copy pattern)
    - Markov-1 → Only connections from $\tau - 1$ to τ
    - Stationary → $B^{(\rightarrow)}$ identical for all $t \in \{1, \ldots\}$
  - Semantics: unroll for T time steps and multiply

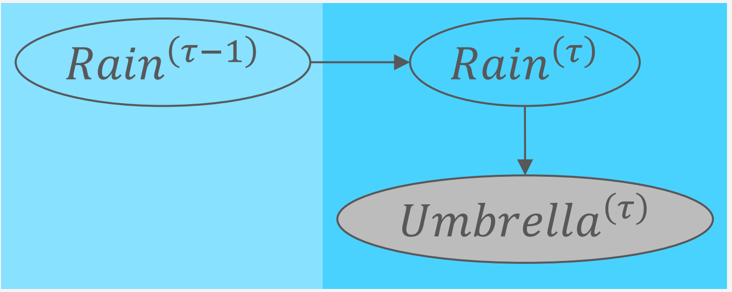
Dynamic Bayesian Networks: Example

- Left: vehicle localization task, where a moving car tries to track its current location using the data obtained from a, possibly faulty, sensor



- Right: Toy example of a special case of a DBN with one latent and one observable variable (*hidden Markov model, HMM*)

$P(r^{(0)})$
0.5



$R^{(t-1)}$	$P(r^{(t)}|R^{(t-1)})$
true	0.7
false	0.3

$R^{(t)}$	$P(u^{(t)}|R^{(t)})$
true	0.9
false	0.2

Factored MDPs

- MDP with its state space S structured according to $S_1, \dots, S_n$, which in general means that
  - Transition probability distribution $T(S', S, A) = P(S'|S, A)$ is given by $T(S'_1, \dots, S'_n, S_1, \dots, S_n, A) = P(S'_1, \dots, S'_n | S_1, \dots, S_n, A)$
    - Or using the template notation: $T\left(S^{(\tau)}, S^{(\tau-1)}, A^{(\tau-1)}\right) = P\left(S^{(\tau)} \middle| S^{(\tau-1)}, A^{(\tau-1)}\right)$ is given by $T\left(S_1^{(\tau)}, \dots, S_n^{(\tau)}, S_1^{(\tau-1)}, \dots, S_n^{(\tau-1)}, A^{(\tau-1)}\right) = P\left(S_1^{(\tau)}, \dots, S_n^{(\tau)} \middle| S_1^{(\tau-1)}, \dots, S_n^{(\tau-1)}, A^{(\tau-1)}\right)$
    - Note that the overall size of T does not increase as the state space size is identical
  - Given that $S_1, \dots, S_n$ represent features of (hopefully weakly) connected parts of a system, T can be factored according to (conditional) independences → often represented using a DBN
    - Factorisation of T:

$$T(S', S, A) = P(S'_1, \dots, S'_n | S_1, \dots, S_n, A) = \prod_{i=1}^{n} P(S'_i | \mathrm{pa}(S'_i)) =: T_B$$

Factored MDPs: Actions and Rewards

- To be correct, the DBN just described is a standard DBN extended with random variable nodes for actions, whose assignment we want to determine, and reward nodes to denote that a reward function outputs a reward depending on the state (and action)
  - BN extended with so-called decision and utility nodes called influence or decision diagram

Side note: Since the state in MDPs is fully observable, every random variable in a DBN is observable, which is not the general case for DBNs, where usually there is a set of latent variables, which are never observed and as such often queried, and a set of evidence variables, which are usually observed (save for sensor failures).

Factored MDPs: Actions and Rewards

- What about rewards?
  If the reward remains a function over the complete state space without any factorisation, we have not gained much
- But remember: Multi-attribute utility theory
  - Reward function with preference independence between subsets of random variables
    → additive reward function
    - Factorisation of R:

$$R(S) = R(S_1, \dots, S_n) = \sum_{j=1}^{m} R_j(\boldsymbol{C}_j)$$

      - Best case $R(S_1, \dots, S_n) = \sum_{i=1}^{n} R_i(S_i)$
    - Compare factorisation of T: $T(S', S, A) = P(S_1', \dots, S_n' | S_1, \dots, S_n, A) = \prod_{i=1}^{n} P(S_i' | \mathrm{pa}(S_i'))$

Factored MDPs: Space Complexity

- With a structured state space, representation size down
  - Given
    - State space with n features and a maximum domain size of d
    - DBN over n features and a maximum domain size of d, with $c = \max_{i \in \{1,\ldots,n\}} |\mathrm{pa}(S_i)| + 1$
    - Given action space of size a
  - Space complexity
    - Transition function $T(S', S, A)$: $\mathcal{O}(d^n \cdot a)$ vs. $\mathcal{O}(n \cdot d^c \cdot a)$
    - Reward function $R(S)$: $\mathcal{O}(d^n)$ vs. $\mathcal{O}(n \cdot d^c)$

Solving Factored MDPs

- Bellman equation:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s' \in \mathrm{dom}(S)} P(s'|a,s)U(s')$$

  - Becomes

$$U(s_1, \dots, s_n) = \sum_{j=1}^{m} R_j(\boldsymbol{C}_j) + \gamma \max_{a \in A(s_1,\dots,s_n)} \sum_{s_1' \in \mathrm{dom}(S_1)} \dots \sum_{s_n' \in \mathrm{dom}(S_n)} \prod_{i=1}^{N} P\left(s_i^{(\tau)} \middle| \mathrm{pa}\left(s_i^{(\tau)}\right)\right) U(s_1', \dots, s_n')$$

- Unfortunately, a factored MDP does not induce a factored value function U
  - One way to go: concentrate on value functions that have a factored representation
    - Approximate the unfactored value function with a factored one

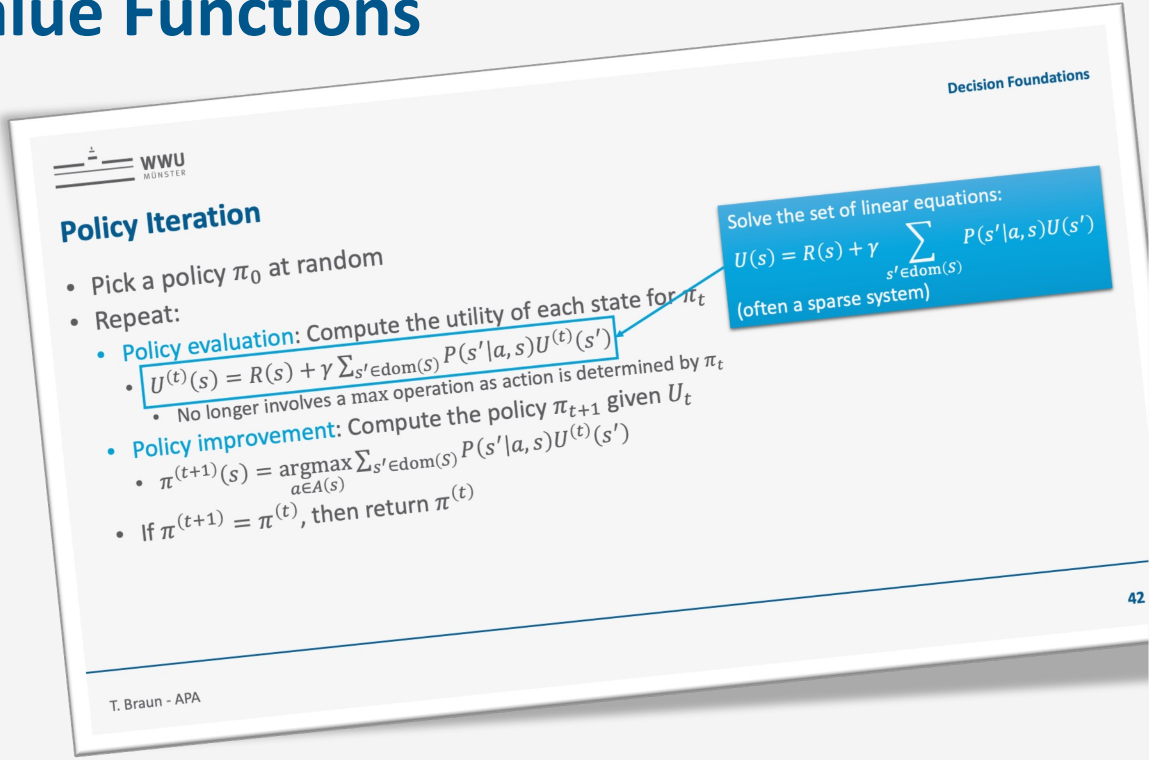
Linear Value Functions

- Linear value function $\mathcal{V}$ over a set of basis functions $H = \{h_1, \ldots, h_k\}$
  - Function $\mathcal{V}$ that can be written as $\mathcal{V}(s_1, \ldots, s_n) = \sum_{j=1}^{k} w_j \cdot h_j(s_1, \ldots, s_n)$ for some coefficients $\mathrm{w} = (w_1, \ldots, w_k)'$
    - Let $\mathcal{H}$ be the linear subspace of $\mathbb{R}^n$ spanned by H
    - Let H be an $n \times k$ matrix whose columns are the k basis functions viewed as vectors
    - Then, $\mathcal{V}$ can be written as Hw
  - Equivalent expressive power to, e.g., single layer neural network
    - Features corresponding to the basis functions
    - Optimise the coefficients w to obtain a good approximation for true value function
  - Separates the problem of defining a reasonable space of features and the induced space $\mathcal{H}$, from the problem of searching within the space
    - Former problem is typically purview of domain experts, latter is focus of analysis + algorithmic design

Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman: Efficient Solution Algorithms for Factored MDPs. In *Journal of Artificial Intelligence Research*, 2003.

Approximate Policy Iteration with Linear Value Functions

- Restrict policy iteration algorithm to only use value functions $\mathcal{V}$ within the provided $\mathcal{H}$
  - Policy improvement as before
  - Policy evaluation changes
    - Whenever policy iteration takes a step that results in a $\mathcal{V}$ outside of $\mathcal{H}$, project result back into $\mathcal{H}$ by finding a value function within $\mathcal{H}$ closest to $\mathcal{V}$
- Projection operator Π
  - Mapping $\Pi : \mathbb{R}^n \rightarrow \mathcal{H}$
  - Π is said to be a projection w.r.t. a norm $||\cdot||$ if $\Pi\mathcal{V} = \mathrm{Hw}^*$ such that $\mathrm{w}^* \in \arg\min_{\mathrm{w}} ||\mathrm{Hw} - \mathcal{V}||$
    - Π is the linear combination of the basis functions that is closest to $\mathcal{V}$ w.r.t. chosen norm



Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman: Efficient Solution Algorithms for Factored MDPs. In *Journal of Artificial Intelligence Research*, 2003.

Approximate Policy Iteration with Linear Value Functions

- Policy evaluation for a policy $\pi^{(t)}$
  - Value function — the value of acting according to the current policy $\pi^{(t)}$ — is approximated through a linear combination of basis functions
- Given $\pi^{(t)}$, i.e., actions are fixed,
  - $T(S', S, A) = T\left(S', S, \pi^{(t)}\right) = T(S', S)$
- Policy evaluation can be written in terms of matrices and vectors
  - $\mathcal{V}$ and R as n-dimensional vectors and T as an $n \times n$-dimensional matrix, denoted $\mathrm{V}, \mathrm{R}, \mathrm{T}$
  - Then, $\mathcal{V} = \mathrm{R} + \gamma \mathrm{T} \mathcal{V}$
    - System of linear equations with one equation for each state → approximate solution within $\mathcal{H}$:
      $$\mathrm{w}^{(t)} = \arg\min_{\mathrm{w}} \|\mathrm{Hw} - (\mathrm{R} + \gamma \mathrm{THw})\| = \arg\min_{\mathrm{w}} \left\|(\mathrm{H} - \gamma \mathrm{TH})\mathrm{w}^{(t)} - \mathrm{R}\right\|$$
      - Problem: How to choose $\|\cdot\|$ wisely, i.e., providing error bounds?

Approximate Policy Iteration with Linear Value Functions

- Convergence and error analysis for MDPs use max-norm ($\mathcal{L}_\infty$)
  → Tie projection operator to $\mathcal{L}_\infty$ norm
- Minimising the $\mathcal{L}_\infty$ norm studied in optimisation literature as the problem of finding the Chebyshev solution to an overdetermined linear system of equations
  - I.e., finding w^* such that $\mathrm{w}^* \in \arg\min_{\mathrm{w}} \|C\mathrm{w} - b\|_\infty$
    - $C = (\mathrm{H} - \gamma \mathrm{TH}), b = R$
  - Algorithm due to Stiefel (1960) solves problem by linear programming:
    - Variables: $w_1, \dots, w_k, \phi;$
    - Minimise: $\phi;$
    - Subject to: $\phi \geq \sum_{j=1}^{k} c_{ij} \cdot w_j - b_i$ and $\phi \geq b_i - \sum_{j=1}^{k} c_{ij} \cdot w_j,$ $i = 1, \dots, n.$
      - At solution (w^*, ϕ^*), w^* is the Chebyshev solution and ϕ^* is the $\mathcal{L}_\infty$ projection error

Only $k+1$ variables but $2n$ constraints: Impractical in general but in factored MDPs with linear value functions, constraints can be represented efficiently → tractable

Factored Value Functions

- Factored (linear) value function
  - Linear function over the basis set $h_1, \dots, h_k$ where scope of each basis function h_i restricted to some subset of variables $\boldsymbol{C}_i \subset S$
  - Goal: the scopes of $h_1, \dots, h_k$ correspond to cliques in graph of DBN representing transition model T

- Not considered so far: How can we use this factored function to our advantage in policy evaluation where we need to
  - Solve the value function as a combination of $h_1, \dots, h_k$ and
    - Problem: Sum over exponential state space
  - Optimise the weights to have a good approximation
    - Problem: LP with exponentially many constraints

Factored Value Functions: Use in Q Value Function

- Efficient computation of value function using $h_1, \dots, h_k$ ($\boldsymbol{s} = s_1, \dots, s_n$) using Q value function

$$Q(\boldsymbol{s}, a) = R(\boldsymbol{s}, a) + \gamma \sum_{\boldsymbol{s}' \in \boldsymbol{S}} P(\boldsymbol{s}' | \boldsymbol{s}, a) \mathcal{V}(\boldsymbol{s}) = R(\boldsymbol{s}, a) + \gamma \sum_{\boldsymbol{s}' \in \boldsymbol{S}} P(\boldsymbol{s}' | \boldsymbol{s}, a) \sum_i w_i h_i(\boldsymbol{s}')$$

  - Define $G(\boldsymbol{s}, a)$ with $g_i(\boldsymbol{s}, a) \coloneqq \sum_{\boldsymbol{s}' \in \boldsymbol{S}} P(\boldsymbol{s}' | \boldsymbol{s}, a) h_i(\boldsymbol{s}')$

$$G(\boldsymbol{s}, a) \coloneqq \sum_{\boldsymbol{s}' \in \boldsymbol{S}} P(\boldsymbol{s}' | \boldsymbol{s}, a) \sum_i w_i h_i(\boldsymbol{s}') = \sum_i w_i \sum_{\boldsymbol{s}' \in \boldsymbol{S}} P(\boldsymbol{s}' | \boldsymbol{s}, a) h_i(\boldsymbol{s}') = \sum_i w_i g_i(\boldsymbol{s}, a)$$

    - Can compute each basis function separately

Factored Value Functions: Use in Q Value Function

- Consider $g(\boldsymbol{s}, a) \coloneqq \sum_{\boldsymbol{s}' \in \boldsymbol{S}} P(\boldsymbol{s}' | \boldsymbol{s}, a) h(\boldsymbol{s}') = T_B h$
  - $P(\boldsymbol{s}' | \boldsymbol{s}, a)$ factored as a DBN T_B
  - h has restricted scope over $\boldsymbol{C}$
- Sum over $\boldsymbol{C}'$ conditioned on ancestors $\boldsymbol{R} = \mathrm{anc}(\boldsymbol{C}')$ of $\boldsymbol{C}'$ in T_B

$$g_i(\boldsymbol{s}, a) = \sum_{\boldsymbol{s}' \in \boldsymbol{S}'} P(\boldsymbol{s}' | \boldsymbol{s}, a) h_i(\boldsymbol{s}') = \sum_{\boldsymbol{s}' \in \boldsymbol{S}'} P(\boldsymbol{s}' | \boldsymbol{s}, a) h_i(\boldsymbol{c}')$$

$$= \sum_{\boldsymbol{c}' \in \boldsymbol{C}'} P(\boldsymbol{c}' | \boldsymbol{s}, a) h_i(\boldsymbol{c}') \underbrace{\sum_{\boldsymbol{r}' \in \boldsymbol{S}' \setminus \boldsymbol{C}'} P(\boldsymbol{r}' | \boldsymbol{s}, a)}_{=1} = \sum_{\boldsymbol{c}' \in \boldsymbol{C}'} P(\boldsymbol{c}' | \boldsymbol{r}, a) h_i(\boldsymbol{c}')$$

  - Depends on the number of values $\boldsymbol{R}$ can take, which depends on $\boldsymbol{C}'$ and complexity of dynamics represented in T_B, i.e., connectivity of graph B

Factored Value Functions: Use in LP with Exponentially Many Constraints

- Constraints of form $\phi \geq \sum_i w_i c_i(\boldsymbol{s}) - b(\boldsymbol{s}), \forall \boldsymbol{s} \in \boldsymbol{S}$
  - $\phi, w_1, \dots, w_k$ free variables
  - $\boldsymbol{s}$ ranges over all states
- Can be replaced by one equivalent non-linear constraint $\phi \geq \max_{\boldsymbol{s}} \sum_i w_i c_i(\boldsymbol{s}) - b(\boldsymbol{s})$
  - Tackle problem of representing non-linear constraint by
    - Computing maximum assignment for a fixed set of weights
      - Simpler problem: Given fixed weights w_i, compute $\phi^* = \max_{\boldsymbol{s}} \sum_i w_i c_i(\boldsymbol{s}) - b(\boldsymbol{s})$
    - Representing non-linear constraint by small set of linear constraints using a construction called factored LP

Factored Value Functions: Use in LP with Exponentially Many Constraints

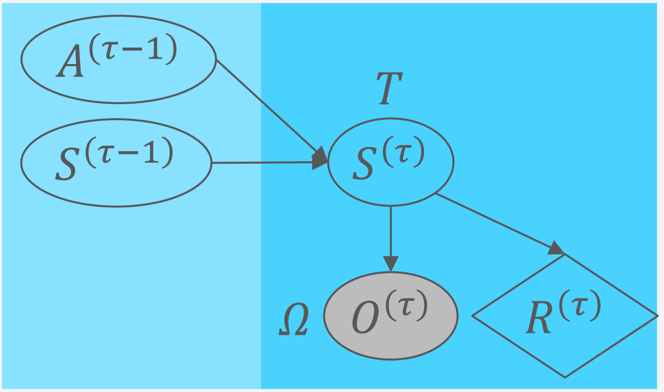
- Computing maximum assignment for a fixed set of weights
  - Given fixed weights w_i, compute $\phi^* = \max_{\boldsymbol{s}} \sum_i w_i c_i(\boldsymbol{s}) - b(\boldsymbol{s})$
  - Remember: Each $c(\boldsymbol{s})$ involves only a subset $\boldsymbol{C}$ of $\boldsymbol{S}$
- Follow idea of variable elimination in Bayesian networks
  - Eliminate one variable $S \in \boldsymbol{S}$ at a time by
    - Combining all functions involving S and
    - Replacing the result with a new function in which we keep only the mappings for each $\boldsymbol{s} \setminus \{S\}$ where S leads to a maximum value
  - Cost exponential in the width of network (largest number of variables combined in a function during overall computation)

Factored Value Functions: Use in LP with Exponentially Many Constraints

- Factored LP to construct a (polynomial) set of constraints for the exponential set of constraints $\phi \geq \sum_i w_i c_i(\boldsymbol{s}) + \sum_j b_j(\boldsymbol{s})$ to use to compute max-norm projections
  - Set of constraints $\Omega = \emptyset$, set of intermediate functions $\mathcal{F} = \emptyset$
  - For each c_i with scope $\boldsymbol{Z}$:
    - For each assignment $\boldsymbol{z}$ to $\boldsymbol{Z}$, create new LP variable $u_{\boldsymbol{z}}^{f_i}$, add $u_{\boldsymbol{z}}^{f_i} = w_i c_i(\boldsymbol{z})$ to Ω and $f_i = w_i c_i(\boldsymbol{z})$ to $\mathcal{F}$
  - For each b_j with scope $\boldsymbol{z}$:
    - For each assignment $\boldsymbol{z}$ to $\boldsymbol{Z}$, create new LP variable $u_{\boldsymbol{z}}^{f_j}$, add $u_{\boldsymbol{z}}^{f_j} = b_j(\boldsymbol{z})$ to Ω and $f_j = b_j(\boldsymbol{z})$ to $\mathcal{F}$
  - Eliminate all variables $S \in \{S_1, \dots, S_n\}$
    - Select functions $\boldsymbol{F}$ from $\mathcal{F}$ containing S
    - Define a new function e over all variables $\boldsymbol{Z}$ in $\boldsymbol{F}$ minus S to represent $\max_S \sum_{f \in \boldsymbol{F}} f$ to replace $\boldsymbol{F}$ in $\mathcal{F}$
    - For each assignment $\boldsymbol{z}$ to $\boldsymbol{Z}$, add constraint $u_{\boldsymbol{z}}^e \geq \sum_{f \in \boldsymbol{F}} u_{\boldsymbol{z}_f}^f$

Factored POMDP

- Difference between MDP and POMDP: partial observability of state
  - State S no longer directly observable → latent
  - Additional sensor model $\Omega(O, S) = P(O|S)$ for observation O
- Given a factorisation of state space
  - Sensor model becomes $\Omega(O, S_1, \dots, S_n) = P(O|S_1, \dots, S_n)$
    - Alternate version using template notation: $\Omega(O^\tau, S_1^\tau, \dots, S_n^\tau) = P(O^\tau|S_1^\tau, \dots, S_n^\tau)$
  - O could also be possibly factored if more than one observation signal incoming
    - $\Omega(O_1^\tau, \dots, O_k^\tau, S_1^\tau, \dots, S_n^\tau) = P(O_1^\tau, \dots, O_k^\tau|S_1^\tau, \dots, S_n^\tau)$
  - Given (conditional) independences, Ω can also be factored like T and represented by a BN B^τ or incorporated into the DBN $(B_0, B_\rightarrow)$ representing T



Graph representation of a POMDP

Interim Summary: *Structure by Features in the State Space*

- State space characterised by set of attributes
  - (Conditional) independences allow for factorisation of functions in MDP
  - Probabilistic graphical models represent such factorisations
- Factored MDP: MDP with a DBN as a representation of the transition model
  - Reduction in space complexity
  - Factored transition function does not lead to factored value function
- Factored (linear) value functions over a set of basis functions
  - Enable computing policy evaluation efficiently
- Approximate policy iteration
  - Project results outside of subspace spanned by basis functions back into subspace

Outline: Decision Making – Structure

Structure by Groups in the Agent Set

- Agent types
- Partitioned decPOMDPs

Structure by Features in the State Space

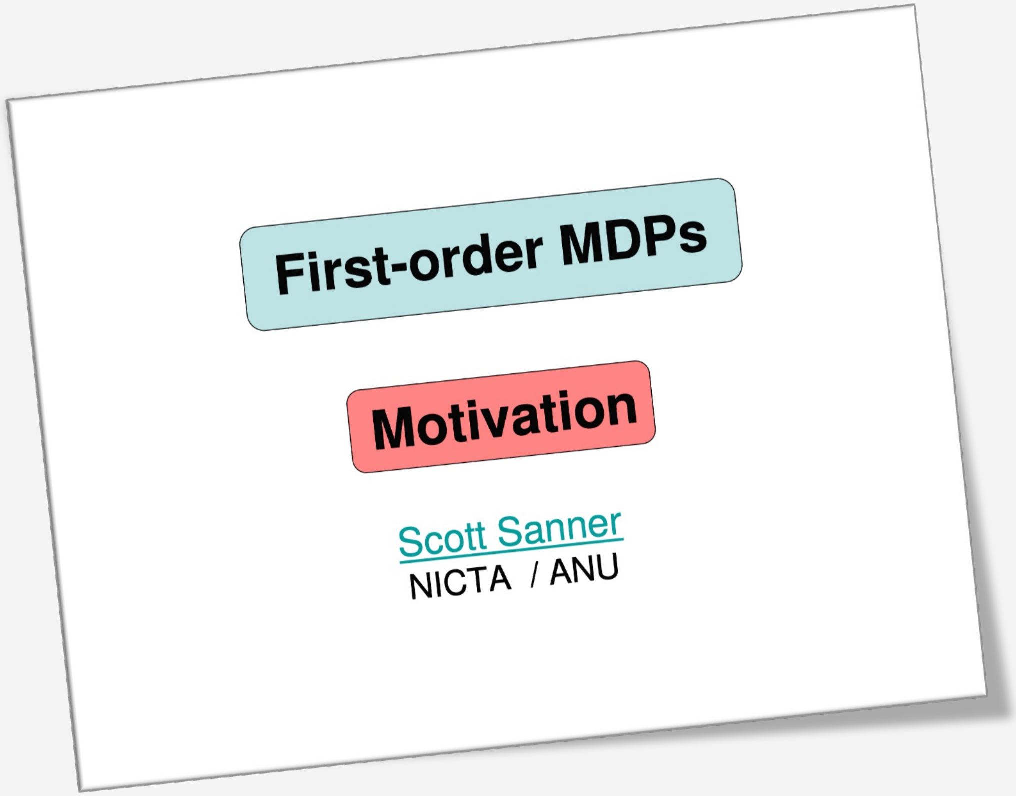
- Dynamic Bayesian networks
- Factored MDPs

Structure by Relations in the State Space

- Situation calculus
- First-order MDPs

Acknowledgement

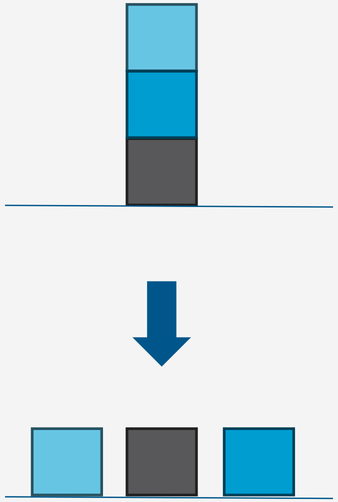
- Thanks to Scott Sanner!



Motivation: Planning Languages

- Common languages:
  - STRIPS
  - PDDL
    - More expressive than STRIPS
    - For example, universal and conditional effects:

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



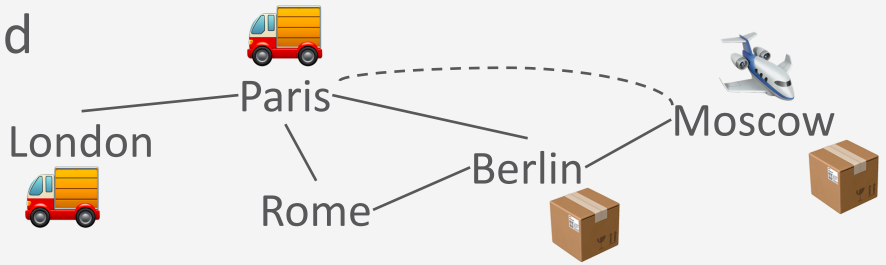
  - General Game Playing (GGP)
    - One or more agents

Motivation: Benefits of Relational Languages

- STRIPS, PDDL, GGP are relational languages…
  - Refer to relational fluents:
    - E.g., $BoxIn(?\,b, ?\,c)$, $OnTable(?\,b)$
  - Specify relations between objects
  - Change over time
- Use first-order logic to specify…
  - Action preconditions
  - Action effects
  - Goals / rewards
    - E.g., `(forall (?b ?c) ((Destination ?b ?c) ⇒ (BoxIn ?b ?c)))`
- Are domain-independent and often compact!

Motivation: How to Solve?

- Relational planning problem
  - E.g., box world




```
(:action load-box-on-truck-in-city
        :parameters (?b - box ?t - truck ?c - city)
        :precondition (and (BoxIn ?b ?c) (TruckIn ?t ?c))
        :effect (and (On ?b ?t) (not (BoxIn ?b ?c))))
```

- Solve *ground* problem for each domain instance?
  - E.g., instance with 3 trucks 🚚🚚🚚, 2 planes ✈️✈️, 3 boxes 📦📦📦
- Or solve lifted specification for *all* domains at once?

Box World: Full (Relational) Specification

- Relational fluents: $BoxIn(Box, City), TruckIn(Truck, City), BoxOn(Box, Truck)$
- Goal: $[\exists Box : b.\, BoxIn(b, paris)]$
- Actions:
  - $load(Box : b, Truck : t)$:
    - Effects:
      - when $[\exists City : c.\, BoxIn(b, c) \wedge TruckIn(t, c)]$ then $[BoxOn(b, t)]$
      - $\forall City : c.$ when $[BoxIn(b, c) \wedge TruckIn(t, c)]$ then $[\neg BoxIn(b, c)]$
  - $unload(Box : b, Truck : t)$:
    - Effects:
      - $\forall City : c.$ when $[BoxOn(b, t) \wedge TruckIn(t, c)]$ then $[BoxIn(b, c)]$
      - when $[\exists City : c.\, BoxOn(b, t) \wedge TruckIn(t, c)]$ then $[\neg BoxOn(b, t)]$
  - $drive(Truck : t, City : c)$:
    - Effects:
      - when $[\exists City : c_1.\, TruckIn(t, c_1)]$ then $[TruckIn(t, c)]$
      - $\forall City : c_1.\, when$ $[TruckIn(t, c_1)]$ then $[\neg TruckIn(t, c_1)]$

Solving Ground Box World

- Apply planner to Box World grounded with respect to domain, e.g.,
  - Domain object instantiations:
    - $Box = \{box_1, box_2, box_3\}, Truck = \{truck_1, truck_2\}, City = \{paris, berlin, rome\}$
  - Ground fluents:
    - $BoxIn$: $\{BoxIn(box_1, paris), BoxIn(box_2, paris), BoxIn(box_3, paris), BoxIn(box_1, berlin), BoxIn(box_2, berlin), BoxIn(box_3, berlin), BoxIn(box_1, rome), BoxIn(box_2, rome), BoxIn(box_3, rome)\}$
    - $TruckIn$: $\{TruckIn(truck_1, paris), TruckIn(truck_2, paris), TruckIn(truck_1, berlin), TruckIn(truck_2, berlin), TruckIn(truck_1, rome), TruckIn(truck_2, rome)\}$
    - $BoxOn$: $\{BoxOn(box_1, truck_1), BoxOn(box_2, truck_1), BoxOn(box_3, truck_1), BoxOn(box_1, truck_2), BoxOn(box_2, truck_2), BoxOn(box_3, truck_2)\}$
  - Ground actions:
    - $load$: $\{load(box_1, truck_1), load(box_2, truck_1), load(box_3, truck_1), load(box_1, truck_2), load(box_2, truck_2), load(box_3, truck_2)\}$
    - $unload$: $\{unload(box_1, truck_1), unload(box_2, truck_1), unload(box_3, truck_1), unload(box_1, truck_2), unload(box_2, truck_2), unload(box_3, truck_2)\}$
    - $drive$: $\{drive(truck_1, paris), drive(truck_2, paris), drive(truck_1, berlin), drive(truck_2, berlin), drive(truck_1, rome), drive(truck_2, rome)\}$
  - Goal: $[BoxIn(box_1, paris) \vee BoxIn(box_2, paris) \vee BoxIn(box_3, paris)]$

Number of fluents exponential in arity

Number of actions exponential in arity

Goal description exponential in number of nested quantifiers

A First-order Solution to Box World

- Derive solution deductively at lifted PDDL level → Optimal for any domain instantiation!

  if $(\exists b. BoxIn(b, paris))$ **then**

  do $noop$

  else if $\left(\exists b^*, t^*. TruckIn(t^*, paris) \wedge BoxOn(b^*, t^*)\right)$ **then**

  do $unload(b^*, t^*)$

  else if $(\exists b, c, t^*. BoxOn(b, t^*) \wedge TruckIn(t, c))$ **then**

  do $drive(t^*, paris)$

  else if $(\exists b^*, c, t^*. BoxIn(b^*, c) \wedge TruckIn(t^*, c))$ **then**

  do $load(b^*, t^*)$

  else if $(\exists b, c_1^*, t^*, c_2. BoxIn(b, c_1^*) \wedge TruckIn(t^*, c_2))$ **then**

  do $drive(t^*, c_1^*)$

  else do $noop$

- Great, but how do I obtain this solution?

Situation Calculus

- Logic formalism designed for representing and reasoning about dynamic domains
  - First introduced by John McCarthy in 1963
- Basic elements
  - Actions that can be performed in the world
  - Situations
  - Fluents that describe the state of the world
- Domain
  - Action precondition axioms, one for each action
  - Successor state axioms, one for each fluent
  - Axioms describing the world in various situations
  - Foundational axioms of the situation calculus: situations are histories, induction on situations

Situation Calculus: Ingredients

- Actions
  - First-order terms with action parameters
  - E.g., $load(b, t)$, $unload(b, t)$, $drive(t, c)$
- Situations
  - Term that encoes action history
  - E.g., s, s_0, $do(load(b, t), s)$, $do(load(b, t), drive(t, c), s)$
- Fluents
  - Relation whose truth value varies between situations
  - E.g., $BoxOn(b, t, s)$, $TruckIn(t, c, s)$, $Box(t, c, s)$
- Effects?

Situation Calculus: PDDL to Effects

- Translate action effects into positive and negative effect axioms
  - $load(Box : b, Truck : t)$:
    - when $[\exists City : c. BoxIn(b,c) \wedge TruckIn(t,c)]$ then $[BoxOn(b,t)]$
    - $\forall City : c.$ when $[BoxIn(b,c) \wedge TruckIn(t,c)]$ then $[\neg BoxIn(b,c)]$
  - $unload(Box : b, Truck : t)$:
    - $\forall City : c.$ when $[BoxOn(b,t) \wedge TruckIn(t,c)]$ then $[BoxIn(b,c)]$
    - when $[\exists City : c. BoxOn(b,t) \wedge TruckIn(t,c)]$ then $[\neg BoxOn(b,t)]$
  - $drive(Truck : t, City : c)$:
    - when $[\exists City : c_1. TruckIn(t,c_1)]$ then $[TruckIn(t,c)]$
    - $\forall City : c_1. when\ [TruckIn(t,c_1)]$ then $[\neg TruckIn(t,c_1)]$

- $[\exists c. a = load(b,t) \wedge BoxIn(b,c,s) \wedge TruckIn(t,c,s)] \Rightarrow BoxOn(b,t,do(a,s))$
- $[\exists t. a = load(b,t) \wedge BoxIn(b,c,s) \wedge TruckIn(t,c,s)] \Rightarrow \neg BoxIn(b,c,do(a,s))$
- $[\exists t. a = unload(b,t) \wedge BoxOn(b,t,s) \wedge TruckIn(t,c,s)] \Rightarrow BoxIn(b,c,do(a,s))$
- $[\exists c. a = unload(b,t) \wedge BoxOn(b,t,s) \wedge TruckIn(t,c,s)] \Rightarrow \neg BoxOn(b,t,do(a,s))$
- $[\exists c_1. a = drive(t,c) \wedge TruckIn(t,c_1,s)] \Rightarrow TruckIn(t,c,do(a,s))$
- $[\exists c. a = drive(t,c) \wedge TruckIn(t,c_1,s)] \Rightarrow \neg TruckIn(t,c_1,do(a,s))$

Situation Calculus: PDDL to Effects

- Use rule to combine multiple effects $C_1 \Rightarrow F, C_2 \Rightarrow F$ over the same fluent F into effect axioms: $\gamma_F^+(\vec{x}, a, s) \Rightarrow F(\vec{x}, do(a, s)), \gamma_F^-(\vec{x}, a, s) \Rightarrow F(\vec{x}, do(a, s))$
  - Rule: $[(C_1 \Rightarrow F) \wedge (C_2 \Rightarrow F)] \equiv [(C_1 \vee C_2) \Rightarrow F]$
  - As a sort of shorthand notation
    - E.g.,
      - $[\exists c. a = load(b,t) \wedge BIn(b,c,s) \wedge TIn(t,c,s)] \Rightarrow BOn(b,t,do(a,s)) \rightarrow \gamma_{BOn}^+(\vec{x}, a, s) \Rightarrow BOn(\vec{x}, do(a,s))$
      - $[\exists c. a = unload(b,t) \wedge BOn(b,t,s) \wedge TIn(t,c,s)] \Rightarrow \neg BOn(b,t,do(a,s)) \rightarrow \gamma_{BOn}^-(\vec{x}, a, s) \Rightarrow \neg BOn(\vec{x}, do(a,s))$
      - $[\exists t. a = unload(b,t) \wedge BOn(b,t,s) \wedge TIn(t,c,s)] \Rightarrow BIn(b,c,do(a,s)) \rightarrow \gamma_{BIn}^+(\vec{x}, a, s) \Rightarrow BIn(\vec{x}, do(a,s))$
      - $[\exists \mathrm{t}. a = load(b,t) \wedge BIn(b,c,s) \wedge TIn(t,c,s)] \Rightarrow \neg BIn(b,c,do(a,s)) \rightarrow \gamma_{BIn}^-(\vec{x}, a, s) \Rightarrow \neg BIn(\vec{x}, do(a,s))$
      - $[\exists c_1. a = drive(t,c) \wedge TIn(t,c_1,s)] \Rightarrow TIn(t,c,do(a,s)) \rightarrow \gamma_{TIn}^+(\vec{x}, a, s) \Rightarrow TIn(\vec{x}, do(a,s))$
      - $[\exists c. a = drive(t,c) \wedge TIn(t,c_1,s)] \Rightarrow \neg TIn(t,c_1,do(a,s)) \rightarrow \gamma_{TIn}^-(\vec{x}, a, s) \Rightarrow \neg TIn(\vec{x}, do(a,s))$

Frame Problem

- Positive and negative effect axioms specify what changes
  - $\gamma^+_{BOn}(\vec{x},a,s) \Rightarrow BOn(\vec{x},do(a,s))$ $\gamma^-_{BOn}(\vec{x},a,s) \Rightarrow \neg BOn(\vec{x},do(a,s))$
  - $\gamma^+_{BIn}(\vec{x},a,s) \Rightarrow BIn(\vec{x},do(a,s))$ $\gamma^-_{BIn}(\vec{x},a,s) \Rightarrow \neg BIn(\vec{x},do(a,s))$
  - $\gamma^+_{TIn}(\vec{x},a,s) \Rightarrow TIn(\vec{x},do(a,s))$ $\gamma^-_{TIn}(\vec{x},a,s) \Rightarrow \neg TIn(\vec{x},do(a,s))$
- Assume completeness regarding these effect axioms:
  - That is, assume that $\gamma^+_F(\vec{x},a,s) \Rightarrow F(\vec{x},do(a,s)), \gamma^-_F(\vec{x},a,s) \Rightarrow \neg F(\vec{x},do(a,s))$ characterise all the conditions under which an action a changes the value of fluent F
  - Formalise as explanation closure axioms
    - $\neg F(\vec{x},s) \wedge F(\vec{x},do(a,s)) \Rightarrow \gamma^+_F(\vec{x},a,s)$ $\equiv \neg F(\vec{x},s) \wedge \neg\gamma^+_F(\vec{x},a,s) \Rightarrow \neg F(\vec{x},do(a,s))$
      - If F was false and was made true by doing action a, then condition γ^+_F must have been true
    - $F(\vec{x},s) \wedge \neg F(\vec{x},do(a,s)) \Rightarrow \gamma^-_F(\vec{x},a,s)$ $\equiv F(\vec{x},s) \wedge \neg\gamma^-_F(\vec{x},a,s) \Rightarrow F(\vec{x},do(a,s))$
      - If F was true and was made false by doing action a then condition γ^-_F must have been true

Frame Problem

- Frame problem: How to (*compactly*) specify what does not change?
  - Intuition: “What does not change, remains the same.”
    - Reiter’s so-called Default Solution
  - Not so easy to specify
    - Moving one thing might move another thing, even though the other thing is never directly touched
    - How to distinguish between relevant and irrelevant side effects? And use that efficiently?

- Default solution to frame problem given as successor state axioms (SSAs), which we construct next

Successor State Axioms (SSAs)

- Inputs / Requirements
  - Unique names for actions / arguments
  - Positive and negative effect axioms
    - $\gamma_F^+(\vec{x}, a, s) \Rightarrow F\big(\vec{x}, do(a, s)\big), \gamma_F^-(\vec{x}, a, s) \Rightarrow F\big(\vec{x}, do(a, s)\big)$
  - Explanation closure axioms
    - $\neg F(\vec{x}, s) \wedge F\big(\vec{x}, do(a, s)\big) \Rightarrow \gamma_F^+(\vec{x}, a\ , s), F(\vec{x}, s) \wedge \neg F\big(\vec{x}, do(a, s)\big) \Rightarrow \gamma_F^-(\vec{x}, a\ , s)$
  - Integrity: $\neg \exists \vec{x}, a, s. \gamma_F^+(\vec{x}, a, s) \wedge \gamma_F^-(\vec{x}, a, s)$
- SSA for each F:
  - $F\big(\vec{x}, do(a, s)\big) \equiv \gamma_F^+(\vec{x}, a, s) \vee \big(F(\vec{x}, s) \wedge \neg \gamma_F^-(\vec{x}, a, s)\big)$
  - Shorthand:
    - $F\big(\vec{x}, do(a, s)\big) \equiv \Phi_F(\vec{x}, a, s)$

Successor State Axioms (SSAs): Example

- SSA for each F: $F\big(\vec{x}, do(a,s)\big) \equiv \gamma_F^+(\vec{x},a,s) \vee \big(F(\vec{x},s) \wedge \neg\gamma_F^-(\vec{x},a,s)\big)$
  - Shorthand: $F\big(\vec{x}, do(a,s)\big) \equiv \Phi_F(\vec{x},a,s)$
- $BoxOn\big(b,t,do(a,s)\big) \equiv \Phi_{BoxOn}(b,t,a,s)$
  $\equiv [\exists c.\, a = load(b,t) \wedge BoxIn(b,t,s) \wedge TruckIn(t,c,s)]$
  $\vee (BoxOn(b,t,s) \wedge \neg[\exists c.\, a = unload(b,t) \wedge BoxOn(b,t,s) \wedge TruckIn(t,c,s)])$
- $BoxIn\big(b,c,do(a,s)\big) \equiv \Phi_{BoxIn}(b,c,a,s)$
  $\equiv [\exists t.\, a = unload(b,t) \wedge BoxOn(b,t,s) \wedge TruckIn(t,c,s)]$
  $\vee (BoxIn(b,c,s) \wedge \neg[\exists t.\, a = load(b,t) \wedge BoxIn(b,c,s) \wedge TruckIn(t,c,s)])$
- $TruckIn\big(t,c,do(a,s)\big) \equiv \Phi_{TruckIn}(t,c,a,s)$
  $\equiv [\exists c_1.\, a = drive(t,c) \wedge TruckIn(t,c_1,s)]$
  $\vee (TruckIn(t,c,s) \wedge \neg[\exists c_1.\, a = drive(t,c) \wedge TruckIn(t,c_1,s)])$

Regression

- Idea: Use SSAs to regress from goal towards a (possibly only partially defined) intial state
  - A bit like lifted backward search

- Regression
  - If ϕ held after action a, then *regression* is the ϕ' that held before action a
  - Exploit following properties
    - $Regr(\neg\psi) = \neg Regr(\psi)$
    - $Regr(\psi_1 \land \psi_2) = Regr(\psi_1) \land Regr(\psi_2)$
    - $Regr\big((\exists x)\psi\big) = (\exists x)Regr(\psi)$
    - $Regr\left(F\big(\vec{x}, do(a,s)\big)\right) = \Phi_F(\vec{x}, a, s)$

Regression: Example

- Given: $\exists b. BoxIn(b, paris, do(unload(b^*, t^*), s))$
- Regress through $unload(b^*, t^*)$
  - $Regr\left(\exists b. BoxIn(b, paris, do(unload(b^*, t^*), s))\right)$
    $= \exists b. Regr\left(BoxIn(b, paris, do(unload(b^*, t^*), s))\right)$
    $= \exists b. \Phi_{BoxIn}(b, paris, unload(b^*, t^*), s)$
    $= \exists b. [\exists t. unload(b^*, t^*) = unload(b, t) \wedge BoxOn(b, t, s) \wedge TruckIn(t, paris, s)]$
    $\vee (BoxIn(b, paris, s)$
    $\wedge \neg[\exists t. unload(b^*, t^*) = load(b, t) \wedge BoxIn(b, paris, s) \wedge TruckIn(t, paris, s)])$
    $= [\exists b, t. b = b^* \wedge t = t^* \wedge BoxOn(b, t, s) \wedge TruckIn(t, paris, s)] \vee \exists b. BoxIn(b, paris, s)$
    $= [(\exists b. b = b^*) \wedge (\exists t. t = t^*) \wedge BoxOn(b^*, t^*, s) \wedge TruckIn(t^*, paris, s)]$
    $\vee \exists b. BoxIn(b, paris, s)$
    $= [BoxOn(b^*, t^*, s) \wedge TruckIn(t^*, paris, s)] \vee \exists b. BoxIn(b, paris, s)$

- If ϕ held after action a, then *regression* is the ϕ' that held before action a
- Exploit following properties
  - $Regr(\neg\psi) = \neg Regr(\psi)$
  - $Regr(\psi_1 \wedge \psi_2) = Regr(\psi_1) \wedge Regr(\psi_2)$
  - $Regr((\exists x)\psi) = (\exists x) Regr(\psi)$
  - $Regr\left(F(\vec{x}, do(a, s))\right) = \Phi_F(\vec{x}, a, s)$

Cannot be made true
$\rightarrow \phi \wedge \neg[\bot] \equiv \phi \wedge \top \equiv \phi$

Make non-empty domain assumption for b, t

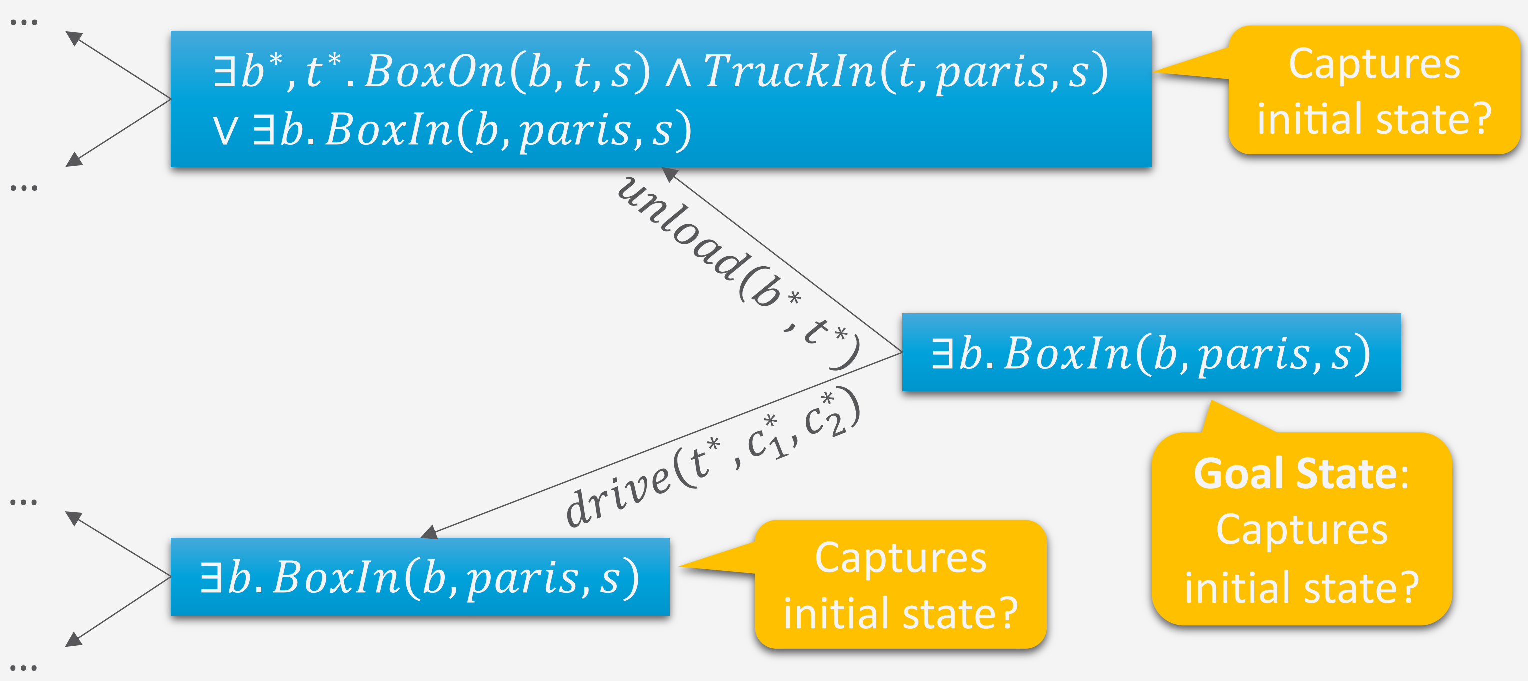
Regression: Example

- Given: $\exists b. BoxIn\big(b, paris, do(unload(b^*, t^*), s)\big)$
- Regress through $unload(b^*, t^*)$
  - $Regr\left(\exists b. BoxIn\big(b, paris, do(unload(b^*, t^*), s)\big)\right)$
    $= [BoxOn(b^*, t^*, s) \land TruckIn(t^*, paris, s)] \lor \exists b. BoxIn(b, paris, s)$
- To get action instantiations of $unload(b^*, t^*)$, query knowledge base (KB, i.e., planning domain)
  - Existentially quantify b^*, t^* and obtain instances via query extraction w.r.t. KB
    - KB consists of first-order state and action abstraction → do not have to enumerate all states, b^*, t^*
    - $\exists b^*, t^*. Regr\left(\exists b. BoxIn\big(b, paris, do(unload(b^*, t^*), s)\big)\right)$
      $= \exists b^*, t^*. [BoxOn(b^*, t^*, s) \land TruckIn(t^*, paris, s)] \lor \exists b. BoxIn(b, paris, s)$
      $= [\exists b^*, t^*. BoxOn(b^*, t^*, s) \land TruckIn(t^*, paris, s)] \lor \exists b. BoxIn(b, paris, s)$

Regression Planning

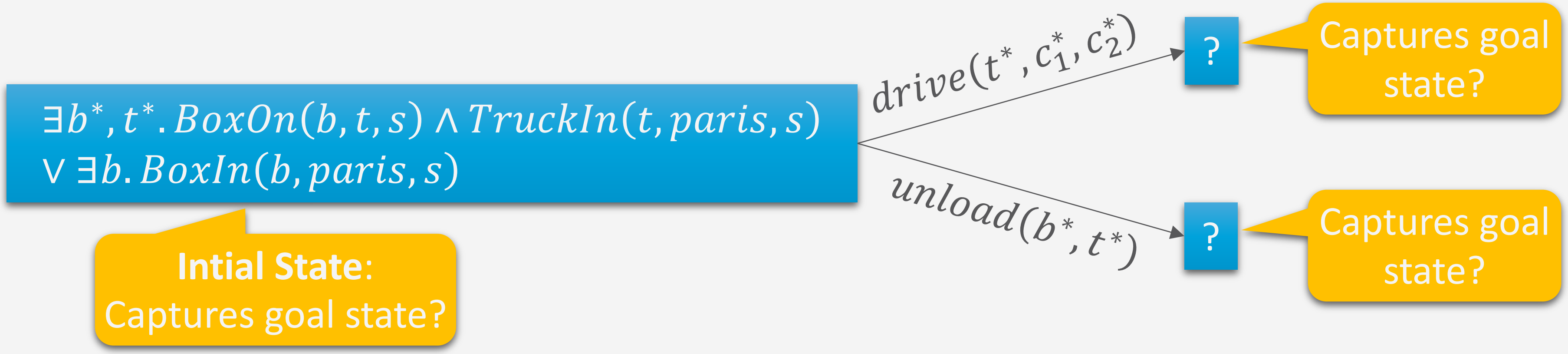
- Define abstract goal state
  - E.g., $\exists b.\, BoxIn(b, paris, s)$
  - Check if regression through action sequence holds in initial state

→ Goal regression planning
  - Provide initial state, actions
    - Initial state description can be partial
  - Use regression to tell whether goal will hold



Progression and Forward Search?

- Can we do lifted forward-search planning?



- Progression not first-order definable! (Reiter, 2001)
- Could progress ground state
  - But this does not exploit first-order structure

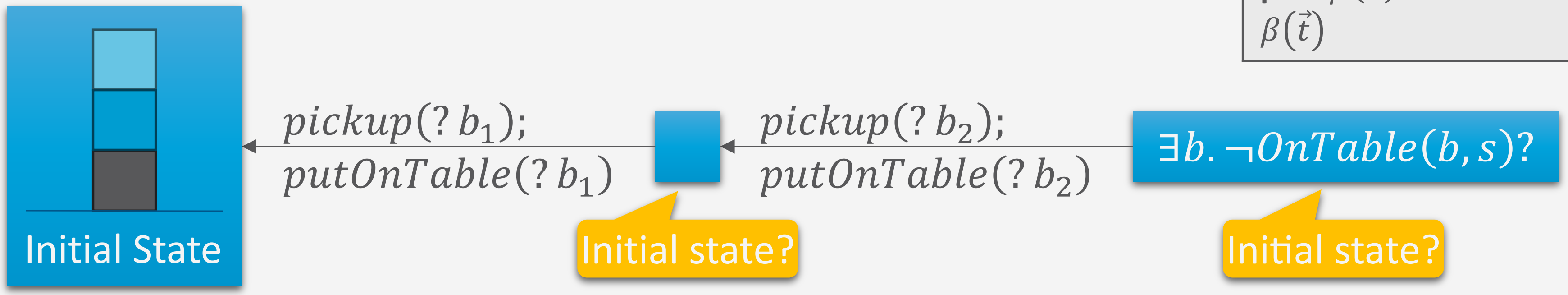
Golog: Restricted Plan Search

- Al**GO**l in **LOG**ic
  - Search the space of sequential action plans
  - Regress actions to initial state to test reachability
  - Restrict action space with program:

α	primitive action
$\phi?$	condition test
(δ_1, δ_2)	sequence
if ϕ **then** δ **endIf**	conditional
while ϕ **then** δ **endWhile**	loop
$(\delta_1 \mid \delta_2)$	nondeterministic choice of actions
$\pi\ \vec{x}\ [\delta]$	nondeterministic choice of arguments
δ^*	nondeterministic iteration
proc $\beta(\vec{x})\ \delta$ **endProc**	procedure call definition
$\beta(\vec{t})$	procedure call

Golog: Example

- Golog program
  - $(\pi b[\neg OnTable(b,s)?, pickup(b), putOnTable(b)])^*, \forall b. OnTable(b,s)?$
- Diagram of Golog planning



  - Heavily restricted action sequences
  - Program exploits first-order action abstraction
  - Initial state need not be fully known

α	primitive action
$\phi?$	condition test
(δ_1, δ_2)	sequence
if ϕ **then** δ **endIf**	conditional
while ϕ **then** δ **endWhile**	loop
$(\delta_1 \vert \delta_2)$	nondeterministic choice of actions
$\pi\ \vec{x}\ [\delta]$	nondeterministic choice of arguments
δ^*	nondeterministic iteration
proc $\beta(\vec{x})\ \delta$ **endProc**	procedure call definition
$\beta(\vec{t})$	procedure call

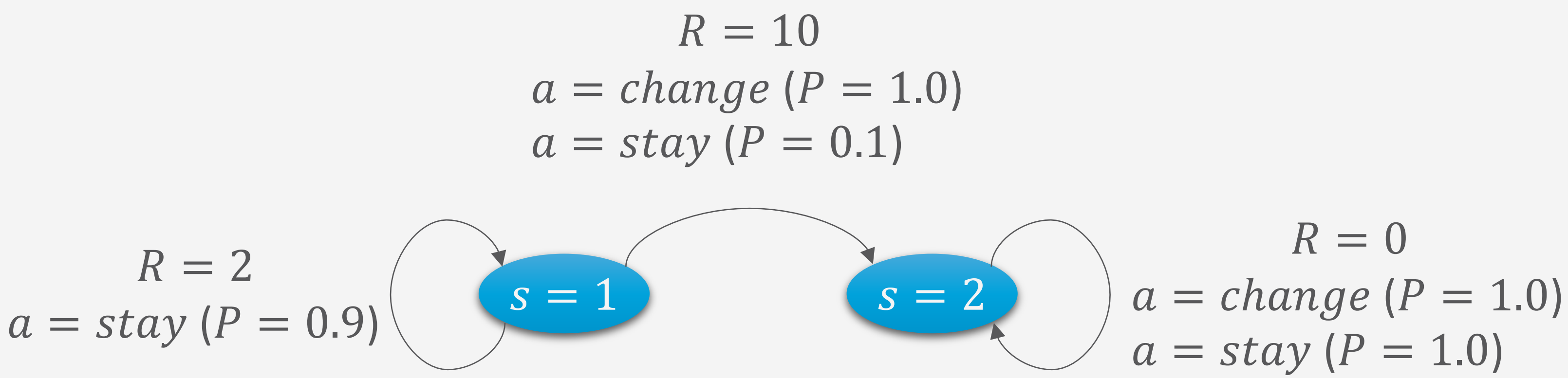
Interim (Interim) Summary

- Situation calculus to describe a relational world
  - Can convert PDDL (and state-variable domains) into effect axioms
  - Derive SSAs from effect axioms
    - Using default solution to frame problem
- Regression operator
  - Extract action instantiation to achieve goal
- Regression planning
  - Initial state need not be fully specified
  - Exploit state and action abstraction
    - Avoid enumerating all state and action instances

Next step: Extend this idea for decision-theoretic planning with uncertain action outcomes

First-order MDPs: MDPs

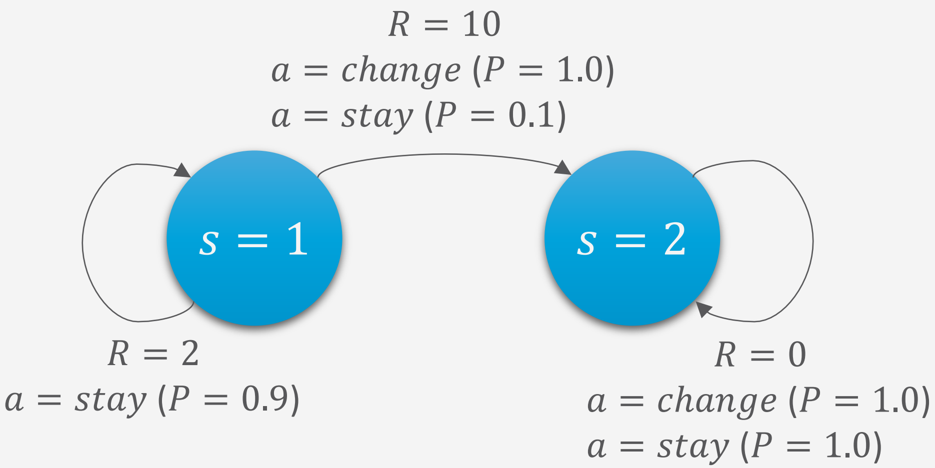
- MDP with discount factor
  - Tuple (S, A, T, R, γ)
    - State space S
      - E.g., $S = \{1,2\}$
    - Actions A
      - E.g., $A = \{stay, go\}$
    - Immediate reward function R
      - E.g., $R(s = 1, a = stay) = 2, \ldots$
    - Transition function T
      - E.g., $T(s = 1, a = stay, s' = 1) = P(s' = 1 | s = 1, a = stay) = 0.9$
    - Discount factor γ
- Acting → define policy $\pi : S \rightarrow A$

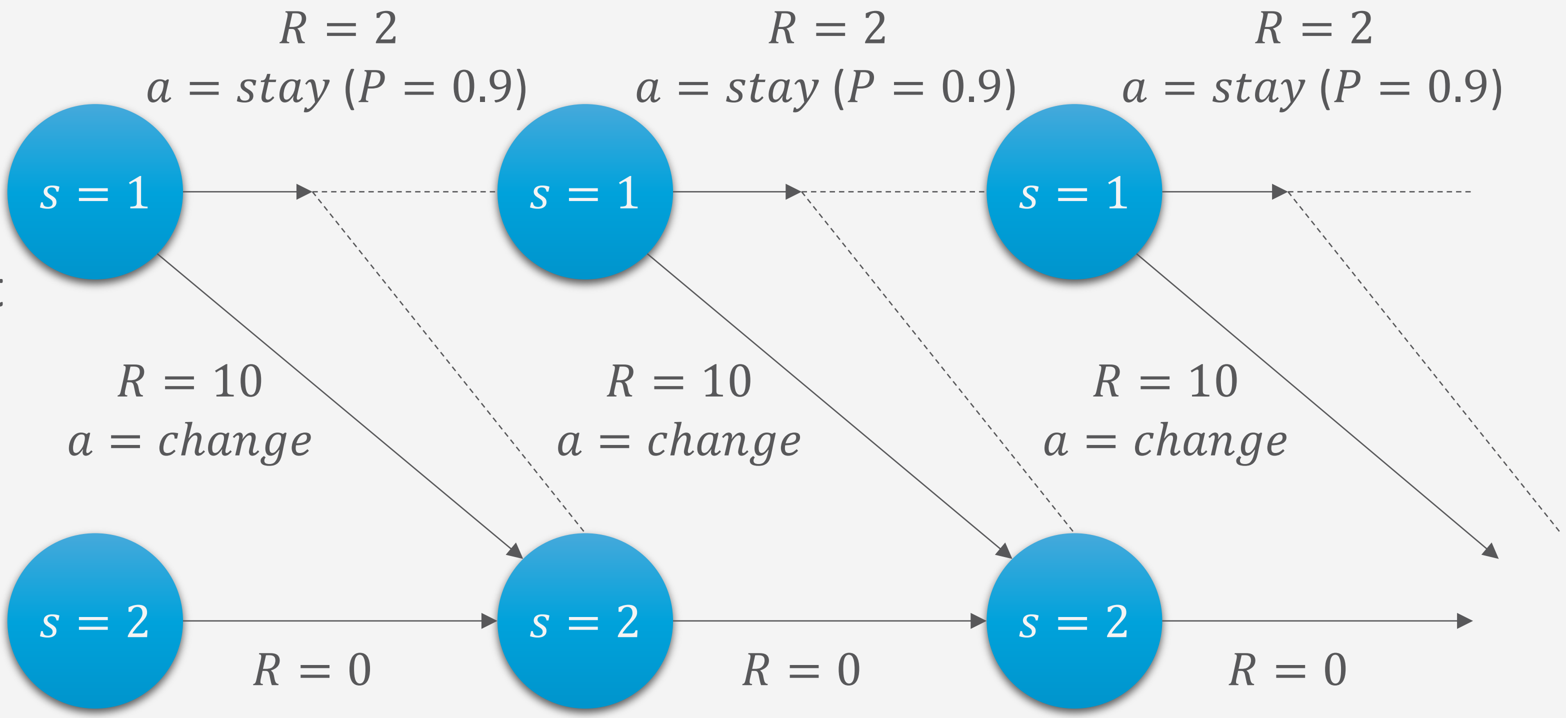


Policy, Value, Solution

- Immediate vs. long-term gain?
  - Reward criterion to optimise
    - Discount factor γ important ($\gamma = 0.9$ vs. $\gamma = 0.1$)
- Define value of policy π

  $V_\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^t \cdot r_t | s = s_0]$
  - Tells how much value to expect to get by following π starting from state s
- MDP optimal solution
  - Policy $\pi^*(s) = \text{argmax}_\pi V_\pi(s)$





Value Iteration & Value Function to Policy

- How to act optimally with t decisions?
  - Given optimal $t-1$-state-to-go value fct.
  - Take action a, then act so as to achieve V^{t-1} thereafter:

$$Q^t(s,a) := R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{t-1}(s')$$

  - Expected value of best action a at stage t?

$$V^t(s) \coloneqq \max_{a \in A}\{Q^t(s,a)\}$$

  - At ∞ horizon, get same value ($= V^*$)

$$\lim_{t \to \infty} \max_s |V^t(s) - V^{t-1}(s)| = 0$$

    - π^* says act the same at each decision stage for ∞ horizon

- Given arbitrary value V (optimal or not)
  - *Greedy* policy π_V takes action in each state that maximises expected value w.r.t. V

$$\pi_V(s) = \arg\max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s') \right\}$$

  - If can act so as to obtain V after doing action a in state s, π_V guarantees $V(s)$ in expectation

First-order MDP (FOMDP)

- Components of MDP in an FOMDP specified as a collection of *case statements*
  - E.g., express reward in Box World FOMDP as

$$rCase(s) = \begin{array}{|c|c|}\hline \forall b, c.\, Dest(b,c) \Rightarrow BoxIn(b,c,s) & 1 \\ \hline \neg\left(\forall b, c.\, Dest(b,c) \Rightarrow BoxIn(b,c,s)\right) & 0 \\ \hline \end{array}$$

- Operators: define unary and binary case operations
  - E.g., cross-sum ⊕ (or ⊖, ⊗) of cases

$$\begin{array}{|c|c|}\hline \phi & 10 \\ \hline \neg\phi & 20 \\ \hline \end{array} \oplus \begin{array}{|c|c|}\hline \varphi & 3 \\ \hline \neg\varphi & 4 \\ \hline \end{array} = \begin{array}{|c|c|}\hline \phi \land \varphi & 10 + 3 \\ \hline \phi \land \neg\varphi & 10 + 4 \\ \hline \neg\phi \land \varphi & 20 + 3 \\ \hline \neg\phi \land \neg\varphi & 20 + 4 \\ \hline \end{array}$$

Stochastic Actions and First-order Decision-theoretic Regression (FODTR)

- Stochastic actions using deterministic situation calculus
  - User's stochastic action, e.g., $A(x) = load(b,t)$
  - Nature's choice, e.g., $n(x) \in \{loadS(b,t), loadF(b,t)\}$
  - Probability distribution over nature's choice, e.g.,

Probability distribution → Adds up to 1 over success and failure choice
$0.1 + 0.9 = 1$
$0.6 + 0.4 = 1$

$P(loadS(b,t)|load(b,t)) =$

$snow(s)$	0.1
$\neg snow(s)$	0.6

$P(loadF(b,t)|load(b,t)) =$

$snow(s)$	0.9
$\neg snow(s)$	0.4

- First-order decision-theoretic regression (FODTR)
  - FODTR = *expectation* of regression:

$$FODTR[vCase(s), A(\vec{x})] = \boldsymbol{E}_{P(n(\vec{x})|A(\vec{x}))}\left[Regr\big(vCase(s), n(\vec{x})\big)\right]$$

FODTR & Q-Functions

- Result of FODTR is a case statement encoding a first-order Q-function

$$FODTR[vCase(s), A(\vec{x})] = R(s) \oplus \gamma \bigoplus_{j=1}^{k} P\left(n_j(\vec{x}), A(\vec{x}), s\right) \otimes Regr\left(V\left(do\left(n_j(\vec{x})\right), s\right)\right)$$

  - E.g.,

$$FODTR[vCase(s), unload(b^*, t^*)]$$
$$= rCase(s) \oplus \gamma \bigoplus_{j=1}^{k} pCase\left(n_j(\vec{x}), unload(b^*, t^*), s\right)$$

$\otimes$

$Regr\left(\exists b. BoxIn\left(b, paris, do\left(n_j(\vec{x}), s\right)\right)\right)$	10
$Regr\left(\neg\exists b. BoxIn\left(b, paris, do\left(n_j(\vec{x}), s\right)\right)\right)$	0

$rCase(s) =$

$\exists b. BoxIn(b, paris, s)$	10
$\neg(\exists b. BoxIn(b, paris, s))$	0

$pCase(loadS(b,t), load(b,t), s) =$

⊤	0.9

$pCase(unloadS(b,t), unload(b,t), s) =$

⊤	0.9

$pCase(driveS(b,t), drive(b,t), s) =$

⊤	1

FODTR & Q-Functions

$$FODTR[vCase(s), unload(b^*, t^*)]$$

$$= rCase(s) \oplus \gamma \bigoplus_{j=1}^{k} pCase\left(n_j(\vec{x}), unload(b^*, t^*), s\right) \otimes$$

$Regr\left(\exists b. BoxIn\left(b, paris, do\left(n_j(\vec{x}), s\right)\right)\right)$	10
$Regr\left(\neg\exists b. BoxIn\left(b, paris, do\left(n_j(\vec{x}), s\right)\right)\right)$	0.0

$$= rCase(s) \oplus \gamma \Bigg[$$

⊤	0.9

$\otimes$

$Regr\left(\exists b. BoxIn(b, paris, do(unloadS(b^*, t^*), s))\right)$	10
$Regr\left(\neg\exists b. BoxIn(b, paris, do(unloadS(b^*, t^*), s))\right)$	0.0

$\oplus$

⊤	0.1

$\otimes$

$Regr\left(\exists b. BoxIn\left(b, paris, do\left(unloadF(b^*, t^*)\right)\right)\right)$	10
$Regr\left(\neg\exists b. BoxIn\left(b, paris, do\left(unloadF(b^*, t^*)\right)\right)\right)$	0.0

$$\Bigg]$$

FODTR & Q-Functions

$$FODTR[vCase(s), unload(b^*, t^*)]$$

$$= rCase(s) \oplus \gamma \bigoplus_{j=1}^{k} pCase\left(n_j(\vec{x}), unload(b^*, t^*), s\right) \otimes$$

$Regr\left(\exists b.BoxIn\left(b, paris, do\left(n_j(\vec{x}), s\right)\right)\right)$	10
$Regr\left(\neg\exists b.BoxIn\left(b, paris, do\left(n_j(\vec{x}), s\right)\right)\right)$	0.0

$=$

$\exists b^*, t^*.BoxOn(b, t, s) \land TruckIn(t, paris, s) \lor \exists b.BoxIn(b, paris, s)$	8.1
¬“	0.0

$\oplus$

$\exists b.BoxIn(b, paris, s)$	0.9
¬“	0.0

$\oplus$

$\exists b.BoxIn(b, paris, s)$	10
¬“	0.0

$=$

$\exists b.BoxIn(b, paris, s)$	19.0	$\rightarrow noop$
¬“ $\land [\exists b, t.BoxOn(b, t, s) \land TruckIn(t, paris, s)]$	8.1	$\rightarrow unload(b^*, t^*)$
¬“	0.0	$\rightarrow noop$

Symbolic Dynamic Programming (SDP)

- What value if 0-stages-to-go?
  - Immediate reward: $V^0(s) = rCase(s)$
- What value if 1-state-to-go?
  - We know value for each action → Take maximum for each state

$V^1(s) = \max_s$ {

ϕ_1	9
ϕ_2	0

$= V^0(s, A_1)$

ϕ_3	3
ϕ_4	1

$= V^0(s, A_2)$

$V^1(s) =$

ϕ_1	9
else ϕ_3	3
else ϕ_4	1
else ϕ_2	0

ϕ_1	9
ϕ_2	0

$= V^0(s, A_1)$

ϕ_3	3
ϕ_4	1

$V^0(s, A_2)$

- Value iteration
  - Obtain V^{n+1} from V^n until $(V^{n-1} \ominus V^n) < \epsilon$

Value Iteration for $t = 1, 2$ of the Box World Example

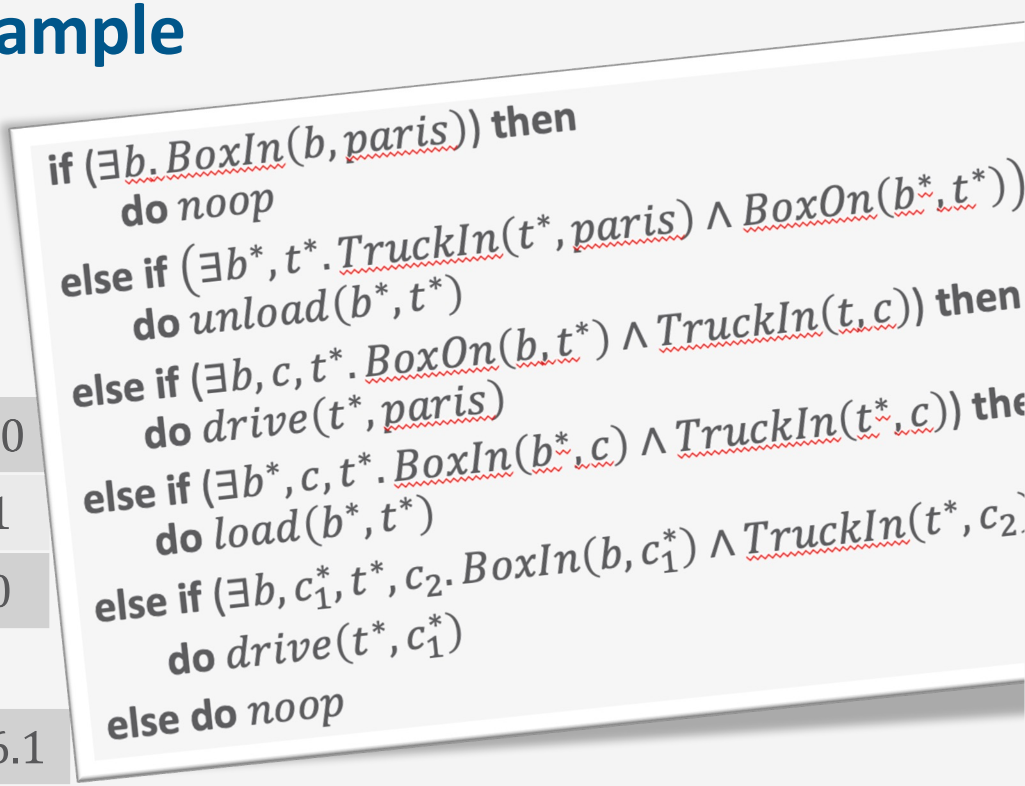
- With increasing iterations, the sequence of actions considered gets longer

$vCase^1(s) =$

$\exists b.BoxIn(b, paris, s)$	19.0
$\neg$“ $\wedge [\exists c.BoxOn(b, t, s) \wedge TruckIn(t^*, paris, s)]$	8.1
$\neg$“	0.0

$vCase^2(s) =$

$\exists b.BoxIn(b, paris, s)$	26.1
$\neg$“ $\wedge [\exists b, t.BoxOn(b, t, s) \wedge TruckIn(t, paris, s)]$	15.4
$\neg$“ $\wedge [\exists b, c, t.BoxOn(b, t, s) \wedge TruckIn(t, c, s)]$	7.3
$\neg$“	0.0



First-order Algebraic Decision Diagrams (FOADDs)

- We want to compactly represent arbitrary case statements
  - E.g.,

$case(s) =$

$\exists x. [A(x) \lor \forall y. A(x) \land B(x) \land \neg A(y)]$	1
$\neg(\exists x. [A(x) \lor \forall y. A(x) \land B(x) \land \neg A(y)])$	0

- Push down quantifiers, expose propositional structure → convert into FOADD

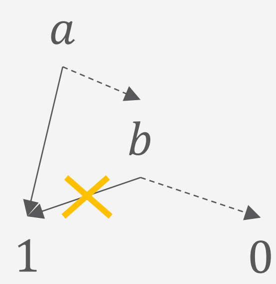
$$[\exists x. A(x)] \lor ([\exists x. A(x) \land B(x)] \land [\forall y. \neg A(y)])$$

Variable	Variable ⇔ FOL KB
a	$\equiv [\exists x. A(x)]$
b	$\equiv [\exists x. A(x) \land B(x)]$

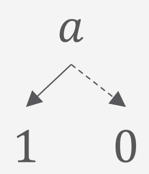
$case(s) =$

$a \lor (b \land \neg a)$	1
$\neg(a \lor (b \land \neg a))$	0

$case(s) =$



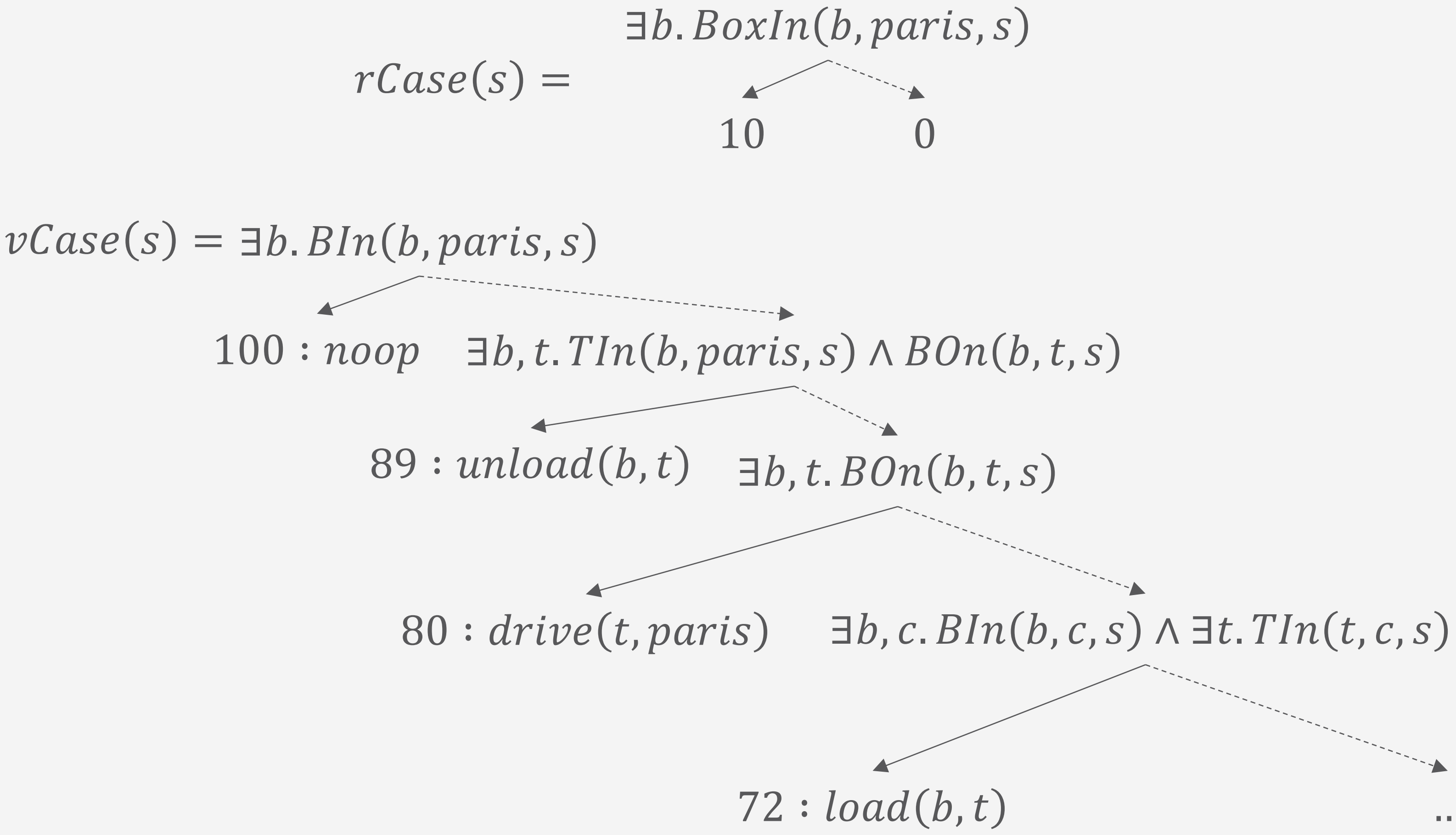
$=$



First-order context-specific independence

Results for SDP with FOADDs

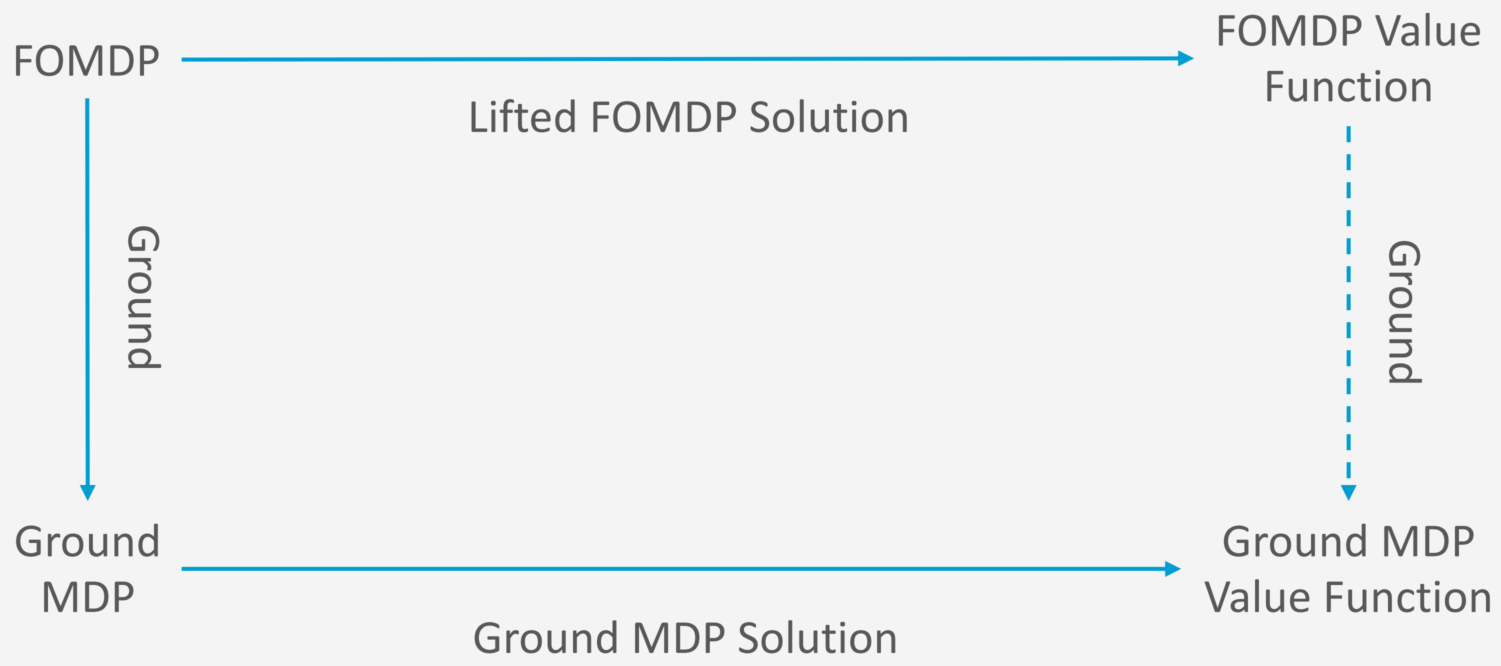
- Encode case statements with FOADDs
  - Solid line: true case
  - Dotted line: false case
- Use FOADD operations for structured SDP
  - E.g., Box World
    - Using $\gamma = 0.9$



Factored SDP for factored FOMDPs [Sanner and Boutilier, 2007]

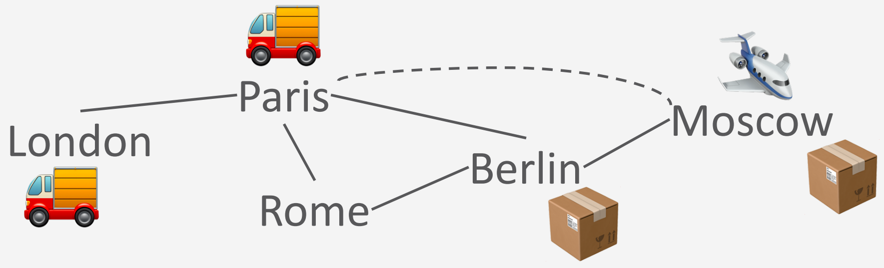
Correctness of SDP

- Show SDP for FOMDPs is correct w.r.t. ground MDP



Caveats of First-order Planning

- Many problems have topologies
  - E.g., reachability constraints in logistics



- If topology not fixed a priori
  - First-order solution must consider ∞ topologies
  - In general case, leads to ∞ values / policies
    - Universal rewards
    - Value function must distinguish ∞ cases
    - Policy will also likely be ∞

$rCase(s) =$

$\forall b, c. Dest(b,c) \Rightarrow BoxIn(b,c,s)$	1
$\neg(\forall b, c. Dest(b,c) \Rightarrow BoxIn(b,c,s))$	0

$V^t(s) =$

$\forall b, c. Dest(b,c) \Rightarrow BoxIn(b,c,s)$	1
One box not at destination	γ
Two boxes not at destination	γ^2
⋮	⋮
$t-1$ boxes not at destination	γ^{t-1}

Caveats of First-order Planning

- Unreachable states
  - PDDL domains often under-constrained
    - E.g., logistics: one box cannot be in two cities at once
  - Constraints implicitly obeyed in initial state
    - Action effects cannot violate constraints
      - Reachable legal states are small subset of all states
    - But (P)PDDL does not constrain legal states

- If no background theory to restrict legal states
  - First-order planning must solve for all states
    - When initial state unknown
  - Where majority of states are actually illegal
- First-order planning w/o initial state solves more difficult problem than search-based solutions
  - Initial state contains implicit constraint information
  - Reachable state space is small subset of all states

Suggests need for hybrid first-order / search-based approaches

A Note on First-order Modelling in Reinforcement Learning

- Novel propositional situations worth exploring may be instances of a well-known context in the relational setting → *exploitation* promising
  - E.g., household robot learning water-taps
    - Having opened one or two water-taps in a kitchen, one can expect other water-taps in kitchens to work similarly

  ⇒ Priority for exploring water-taps in kitchens in general reduced

  ⇒ Information gathered likely to carry over to water-taps in other places

  ❖ Hard to model in propositional setting: each water-tap is novel

Interim Summary

- FOMDPs are one model for lifted decision-theoretic planning
  - Exploit state and action abstraction for MDPs
- Use situation calculus specified action theory
- Use case statements to represent reward, probabilities
- Symbolic dynamic programming = lifted DP
  - Use FOADDs to compactly represent case statements
  - First-order context-specific independence to compactify FOADDs

Outline: Decision Making – Structure

Structure by Groups in the Agent Set

- Agent types
- Partitioned decPOMDPs

Structure by Features in the State Space

- Dynamic Bayesian networks
- Factored MDPs

Structure by Relations in the State Space

- Situation calculus
- First-order MDPs

$\Longrightarrow$ Next: Human-awareness