Exploiting First-Order Regression in Inductive Policy Selection

Charles Gretton, Sylvie Thiébaux

{charlesg,thiebaux}@csl.anu.edu.au

Computer Sciences Laboratory

Australian National University + NICTA

Overview

- Decision-theoretic planning
  - MDP, ***the*** model for decision-theoretic planning
  - What about the ***relational structure*** of domains (***situation-calculus***, PPDDL, Prob-STRIPS)?
- RMDPs, computing a generalised policy
  - Previous approaches:
    - Reasoning – decision theoretic regression
    - Learning – policy and/or value function
  - Situation-calculus specification
  - Algorithm
  - Results
- Future work

Overview

- Decision-theoretic planning
  - MDP, ***the*** model for decision-theoretic planning
  - What about the ***relational structure*** of domains (***situation-calculus***, PPDDL, Prob-STRIPS)?
- RMDPs, computing a generalised policy
  - Previous approaches:
    - Reasoning – decision theoretic regression
    - Learning – policy and/or value function
  - Situation-calculus specification
  - Algorithm
  - Results
- Future work

Markov Decision Process

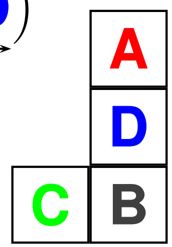
- An MDP is a 4-tuple $\langle \mathcal{E}, \mathcal{A}, \mathrm{Pr}, \mathcal{R} \rangle$
- Which includes fully observable states $\mathcal{E}$ and actions $\mathcal{A}$
- $\{\mathrm{Pr}(e, a, \bullet) \mid e \in \mathcal{E}, a \in \mathcal{A}(e)\}$ is a family of probability distributions over $\mathcal{E}$ such that $\mathrm{Pr}(e, a, e')$ is the probability of being in state e' after performing action a in state e
- $\mathcal{R} : \mathcal{E} \to \mathbb{R}$ is a reward function such that $\mathcal{R}(e)$ is the immediate reward for being in state e
- We want a stationary policy $\pi : \mathcal{E} \mapsto \mathcal{A}$. The value $V_\pi(e)$ of state e given π is:

$$V_\pi(e) = \lim_{n \to \infty} \mathsf{E}\left[\sum_{t=0}^{n} \beta^t \mathcal{R}(e_t) \mid \pi, e_0 = e\right]$$

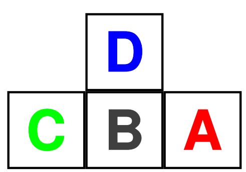
- π is optimal iff $V_\pi(e) \geq V_{\pi'}(e)$ for all $e \in \mathcal{E}$ and π'

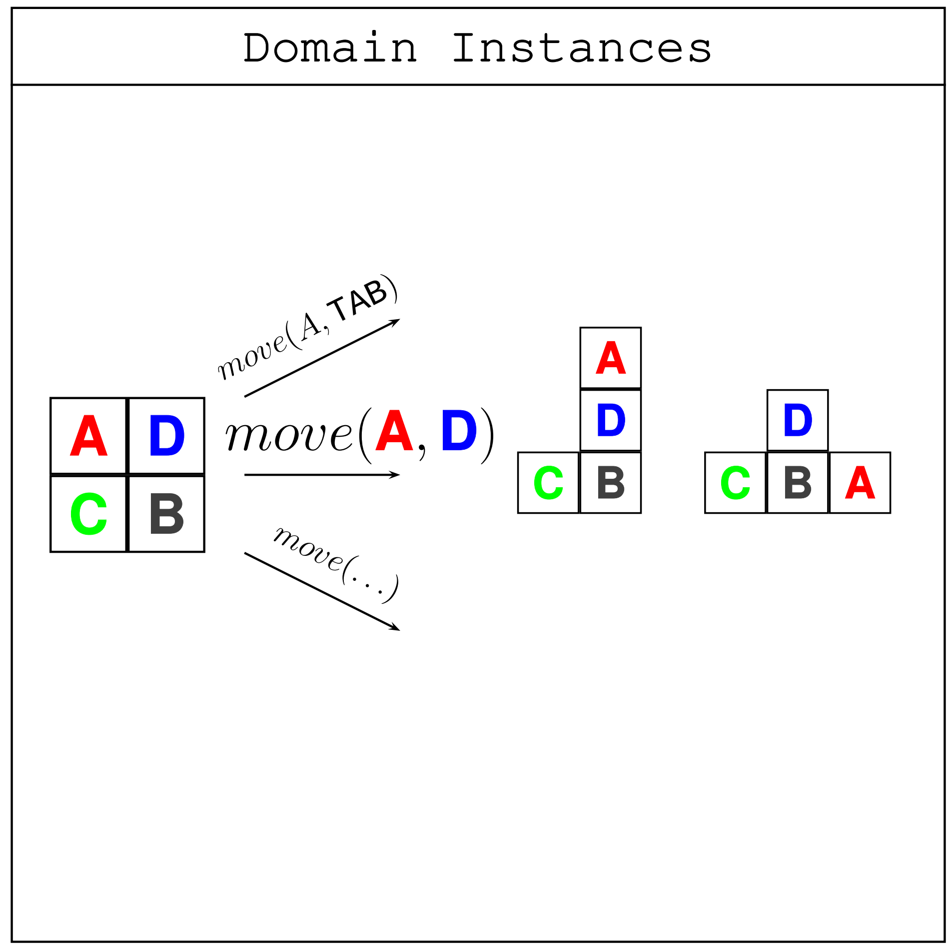
Planning (MDP)

$$\xrightarrow[\Pr = 0.9]{moveS(\mathsf{A},\mathsf{D})}$$

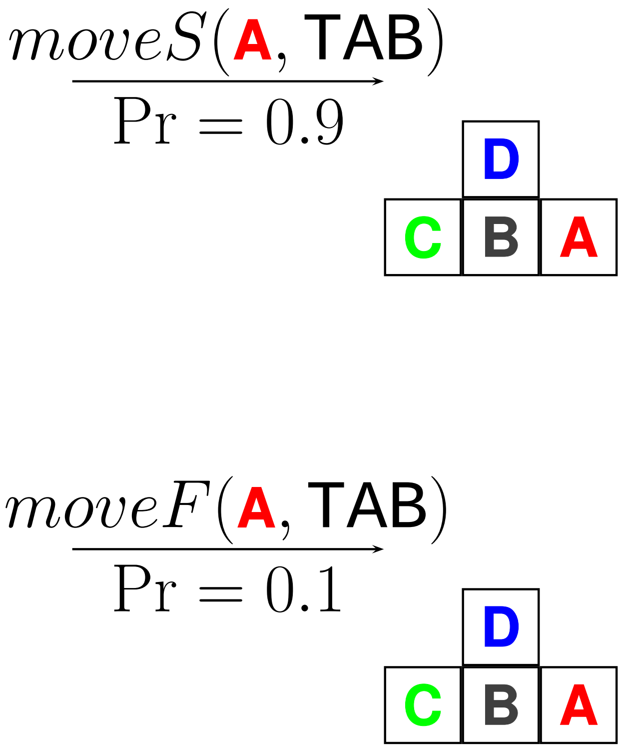


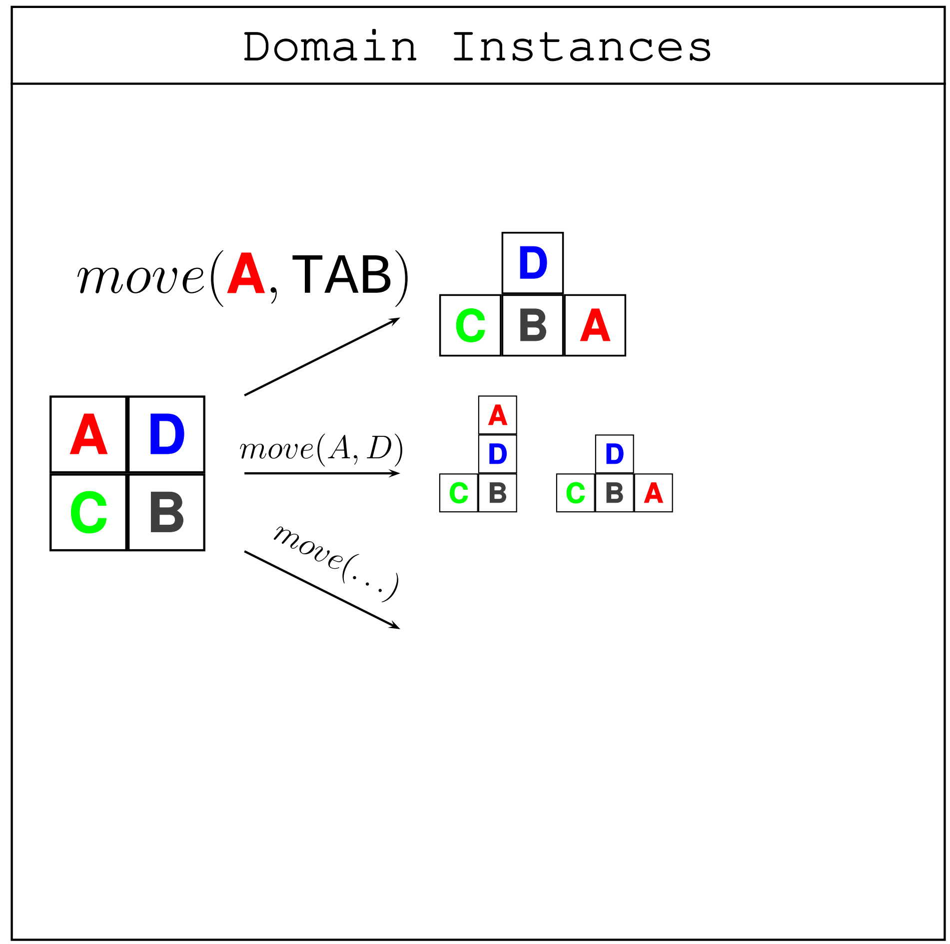
$$\xrightarrow[\Pr = 0.1]{moveF(\mathsf{A},\mathsf{D})}$$



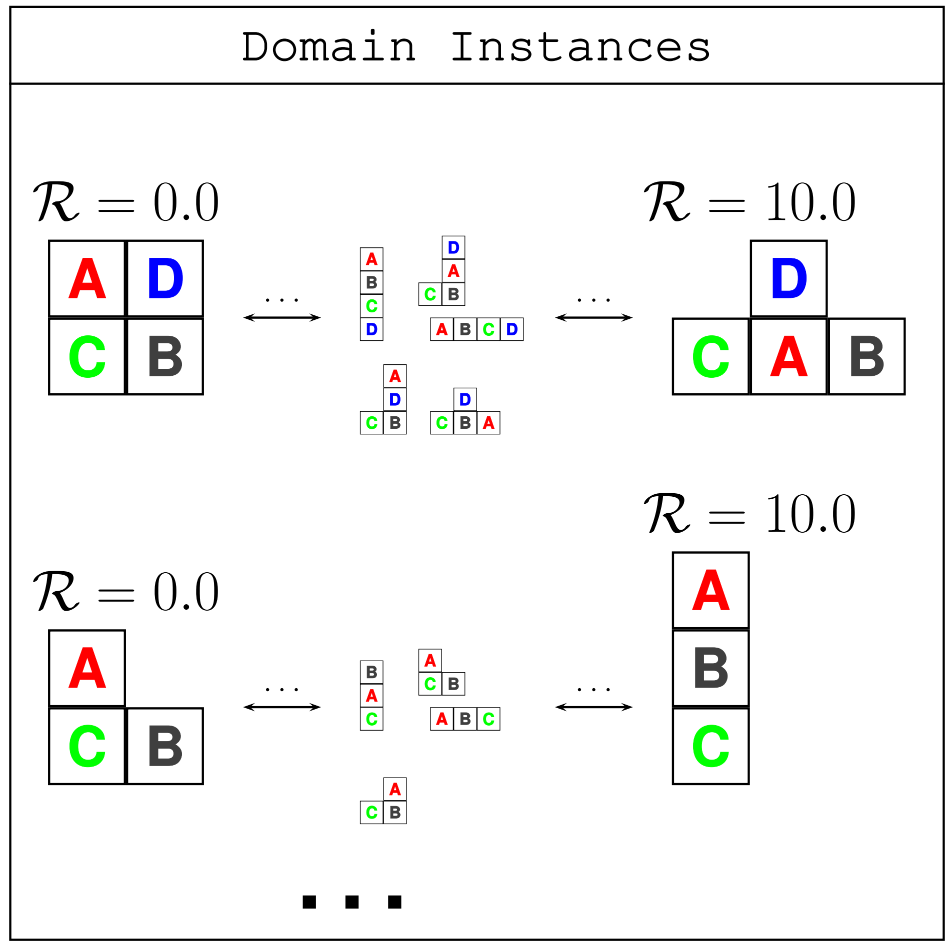


Planning (MDP)

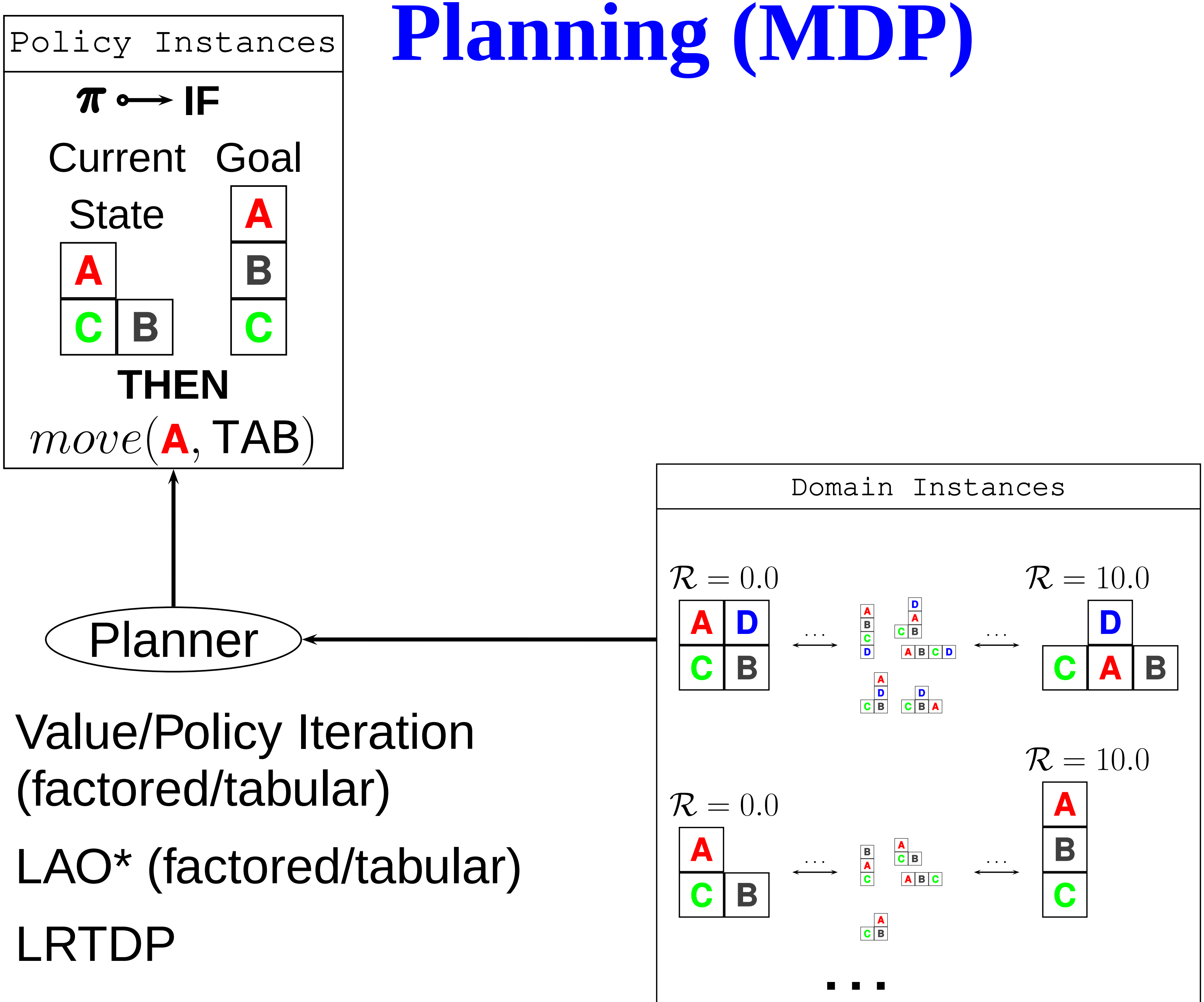




Planning (MDP)



Planning (MDP)

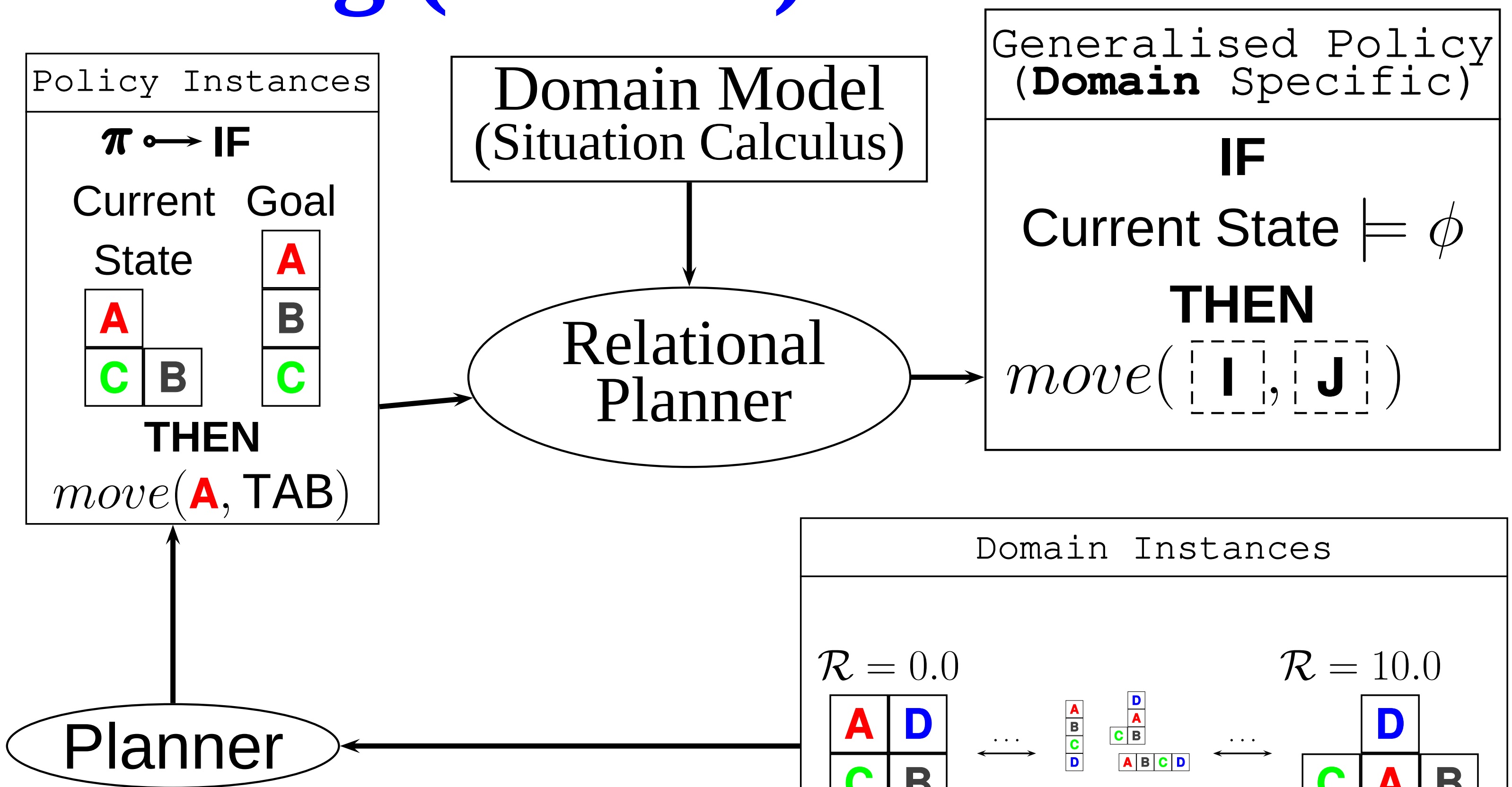


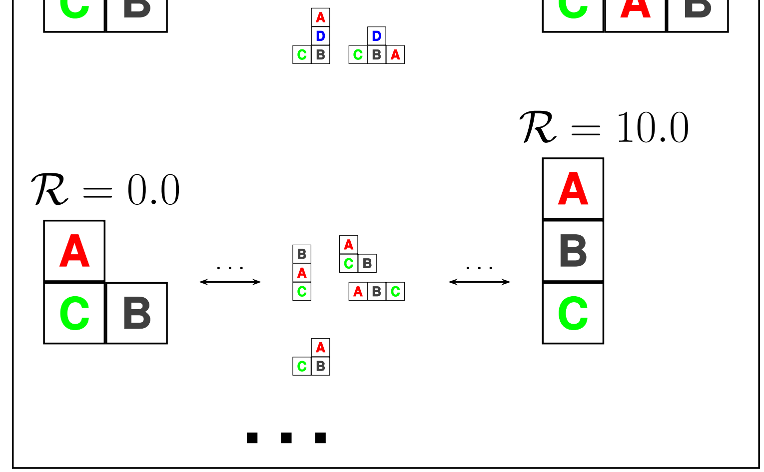
- Value/Policy Iteration (factored/tabular)
- LAO* (factored/tabular)
- LRTDP
- Q-Learning, TD(λ), API

Overview

- Decision-theoretic planning
  - MDP, ***the*** model for decision-theoretic planning
  - What about the ***relational structure*** of domains (***situation-calculus***, PPDDL, Prob-STRIPS)?
- RMDPs, computing a generalised policy
  - Previous approaches:
    - Reasoning – decision theoretic regression
    - Learning – policy and/or value function
  - Situation-calculus specification
  - Algorithm
  - Results
- Future work

Planning (RMDP)





- Value/Policy Iteration (factored/tabular)
- LAO* (factored/tabular)
- LRTDP
- Q-Learning, TD(λ), API

Overview

- Decision-theoretic planning
  - MDP, *the* model for decision-theoretic planning
  - What about the ***relational structure*** of domains (***situation-calculus***, PPDDL, Prob-STRIPS)?
- RMDPs, computing a generalised policy
  - Previous approaches:
    - Reasoning – decision theoretic regression
    - Learning – policy and/or value function
  - Situation-calculus specification
  - Algorithm
  - Results
- Future work

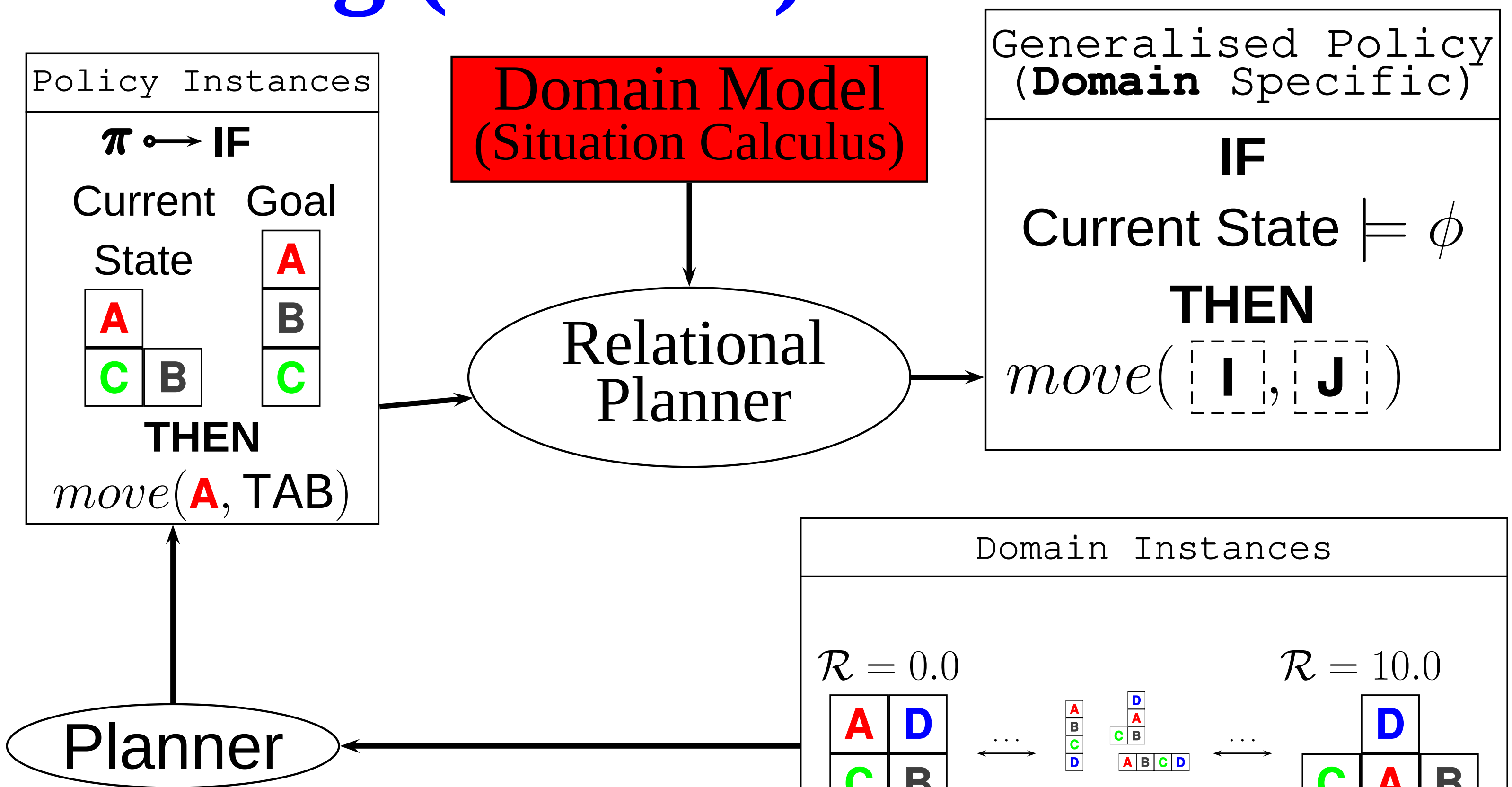
Previous Approaches – (Reasoning)

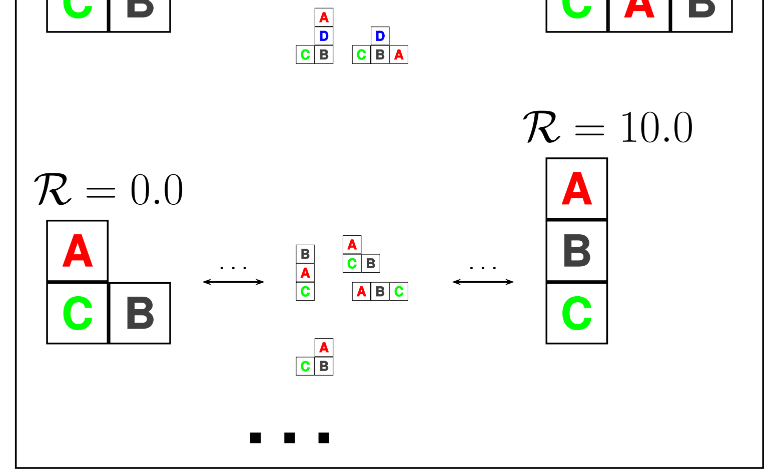
- Use pure reasoning to compute a generalised policy
  - [Boutilier *et al.*, 2001]
  - Requires theorem proving
  - Smart data structures
  - Not particularly practical

Previous Approaches – (Learning)

- Policy focused
  - Use pure induction, given a fairly arbitrary hypotheses space
    - [Fern *et al.*, 2004], [Mausam and Weld, 2003], [Yoon *et al.*, 2002], [Dzeroski and Raedt, 2001], [Martin and Geffner, 2000], [Khardon, 1999]
    - Hypotheses space is either a user enumerated list of concepts **or**
    - Sentences, in a taxonomic language bias
- Value focused
  - Multi agent planning problems[Guestrin *et al.*, 2003]
- Our plan is to combine the best attributes of learning and reasoning

Planning (RMDP)





- Value/Policy Iteration (factored/tabular)
- LAO* (factored/tabular)
- LRTDP
- Q-Learning, TD(λ), API

Situation Calculus – as an RMDP Specification Language

- Usual quantifiers and connectives :: $\{\exists, \forall, \wedge, \vee, \neg, \rightarrow\}$
- 3 disjoint sorts:

1. *Objects* :: Blocks-World (block)

   Logistics (box, truck, city)
2. *Actions* :: first-order terms built from an action function symbol of sort $object^n \rightarrow action$ and its arguments (i.e. $move(a, b)$).
3. *Situations* :: are lists of actions:
   - Constant symbol S_0 denotes the initial situation (empty list)
   - Function symbol
     $do : action \times situation \rightarrow situation$
     lists of length greater than 0.

RMDP Specification (cont)

- *Relational Fluents* :: relations that have truth values which vary from situation to situation.
  - Built using predicate symbols of sort $object^n \times situation$ (i.e. $On(b1, b2, do(move(a, b), s))$).
- *Precondition* :: for each deterministic action $A(\vec{x})$, we need to write one axiom of the form: $poss(A(\vec{x}), s) \equiv \Psi_A(\vec{x}, s)$.

$$
\begin{aligned}
& poss(moveS(b1, b2), s) \equiv poss(moveF(b1, b2), s) \equiv \\
& \quad b1 \neq table \wedge b1 \neq b2 \wedge \not\exists b3\ On(b3, b1, s) \wedge \\
& \quad (b2 = table \vee \not\exists b3\ On(b3, b2, s))
\end{aligned}
$$

RMDP Specification (cont)

- $t = case[f_1, t_1; \ldots ; f_n, t_n]$ abbreviates $\vee_{i=1}^{n}(f_i \wedge t = t_i)$.
- *Possibilities (natures choices)* ::

$choice(a, A(\vec{x})) \equiv \vee_{j=1}^{k}(a = D_j(\vec{x}))$

$prob(D_j(\vec{x}), A(\vec{x}), s) = case[\phi_j^1(\vec{x}, s), p_j^1; \ldots ; \phi_j^m(\vec{x}, s), p_j^m]$

$$choice(a, move(b1, b2)) \equiv$$
$$a = moveS(b1, b2) \vee a = moveF(b1, b2)$$

$$prob(moveS(b1, b2), move(b1, b2), s) =$$
$$case[Rain(s), 0.7; \neg Rain(s), 0.9]$$

$$prob(moveF(b1, b2), move(b1, b2), s) =$$
$$case[Rain(s), 0.3; \neg Rain(s), 0.1]$$

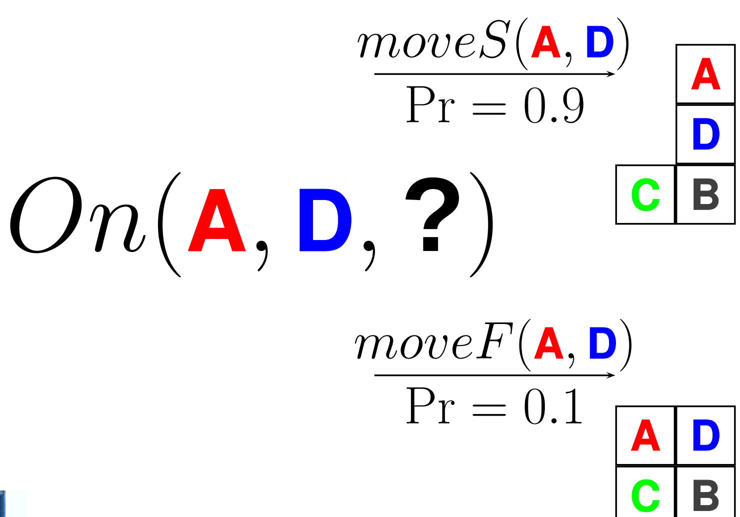
State Formulae

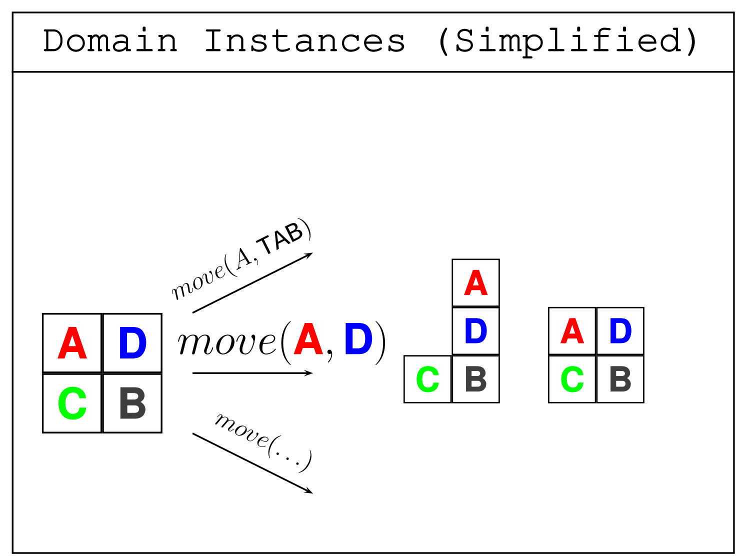
- $f(\vec{x}, s)$, whose only free variables are non-situation variables $\vec{x}$ and situation variable s, and in which no other situation term occurs.
- State formulae do not contain statements involving predicates $poss$ and $choice$, and functions $prob$.
- ϕ is a state formula whose only free variable is s.

RMDP Specification (cont)

- *Successor state axiom* :: For each relational fluent $F(\vec{x}, s)$, there is one axiom of the form: $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$, where $\Phi_F(\vec{x}, a, s)$ is a state formula characterising the truth value of F in the situation resulting from performing a in s.

$$On(b1, b2, do(a, s)) \equiv a = moveS(b1, b2) \vee$$
$$(On(b1, b2, s) \wedge \nexists b3\ (b3 \neq b2 \wedge a = moveS(b1, b3)))$$

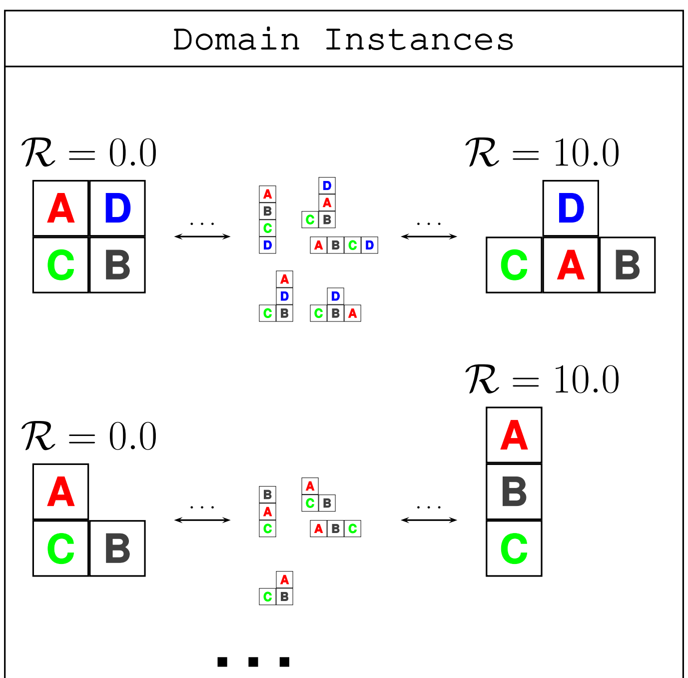




RMDP Specification (cont)

- $t = case[f_1, t_1; \ldots; f_n, t_n]$ abbreviates $\vee_{i=1}^{n}(f_i \wedge t = t_i)$.
- *Reward* :: $R(s) = case[\phi_1(s), r_1; \ldots; \phi_n(s), r_n]$, where the r_is are reals and the ϕ_is are state formulae.

$$R(s) \equiv$$
$$case[\forall b1\ \forall b2\ (OnG(b1, b2) \rightarrow On(b1, b2, s)), 10.0;$$
$$\exists b1\ \exists b2\ (OnG(b1, b2) \wedge \neg On(b1, b2, s)), 0.0]$$



Regression gives a Hypotheses Language

- The regression of a state formula ϕ through a deterministic action α (i.e. $\mathtt{regr}(\phi, \alpha)$) is a state formula that holds before α is executed iff ϕ holds after the execution.
- Consider the set $\{\phi_j^0\}$ consisting of the state formulae in the *reward axiom* case statement.
- We can compute $\{\phi_j^1\}$ from $\{\phi_j^0\}$ by regressing the ϕ_j^0 over all the domain's deterministic actions.
- A state in a subset of MDP states $I \subseteq \mathcal{E}$ that are one action application from a rewarding state, "models" $\bigvee_j \phi_j^1$.

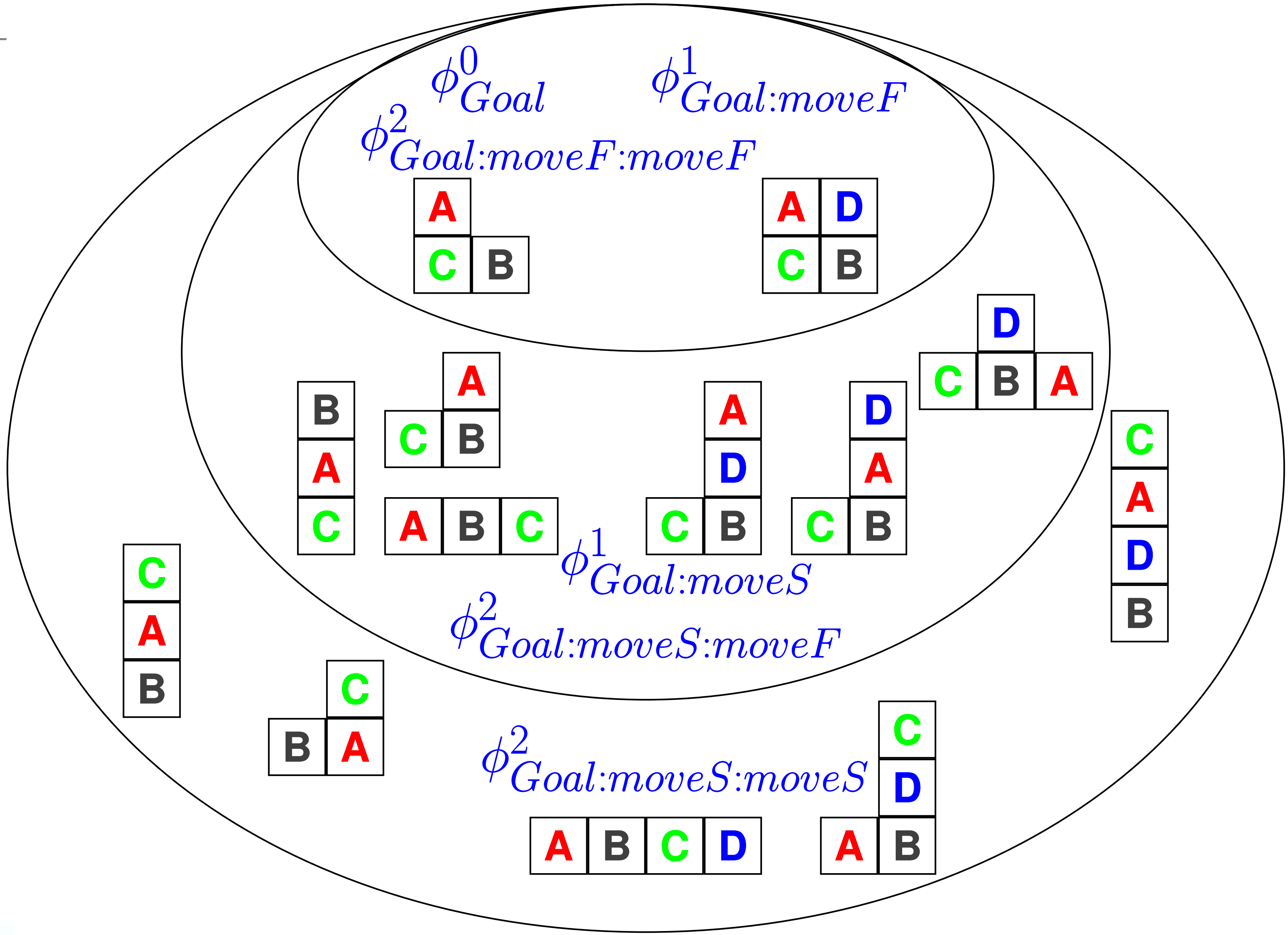
Regression gives a Hypotheses Language

- A state formula characterising pre-action states for each stochastic action, can be formed by considering disjunctions over $\{\phi_j^1\}$.
- We can encapsulate longer trajectories facilitated by stochastic actions, by computing $\{\phi_j^n\}$ for larger n.
- Formulae relevant to n-step trajectories are found in:

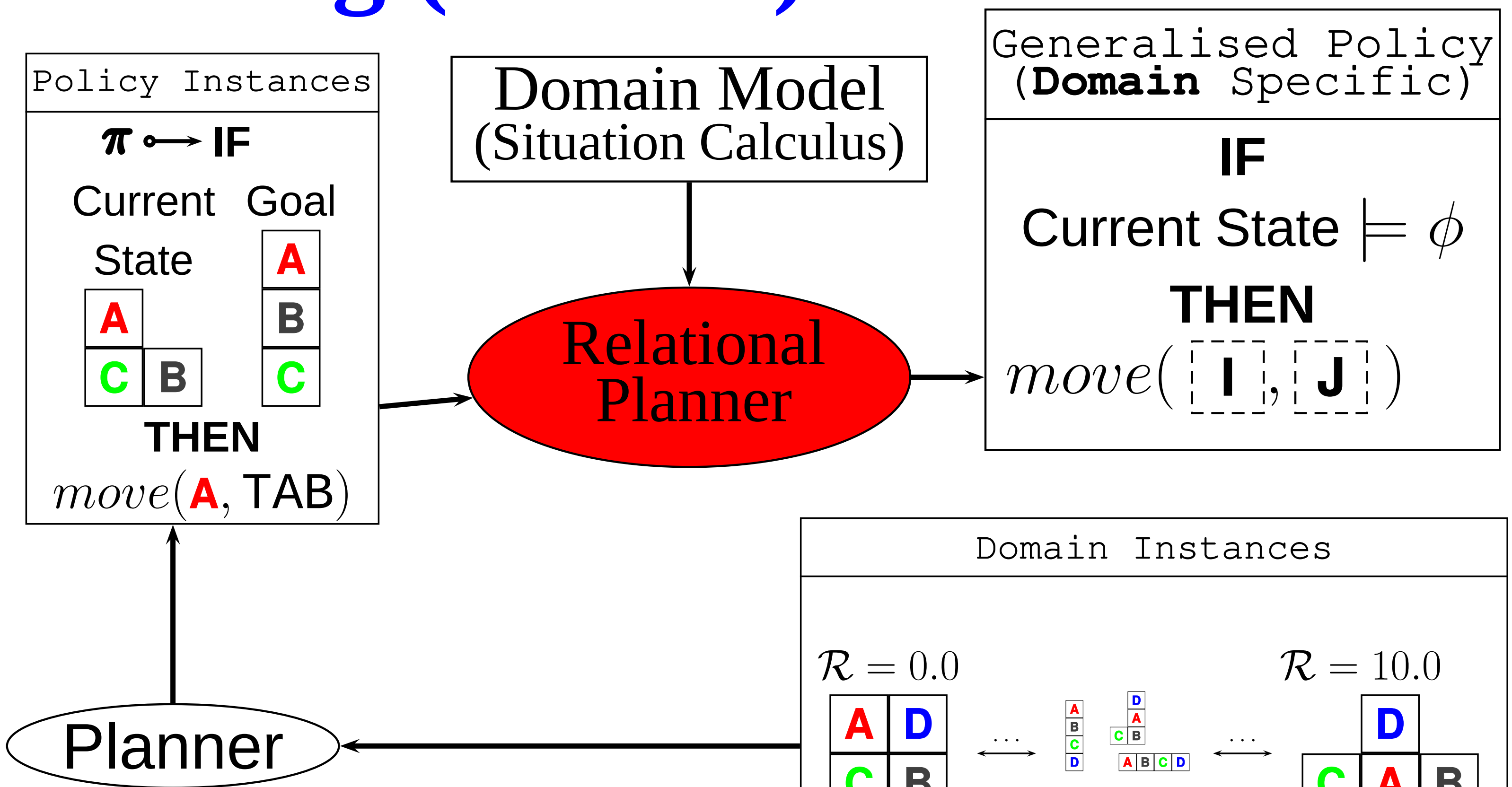
$$F^n \equiv \bigcup_{i=0\ldots n} \{\phi_j^i\}$$

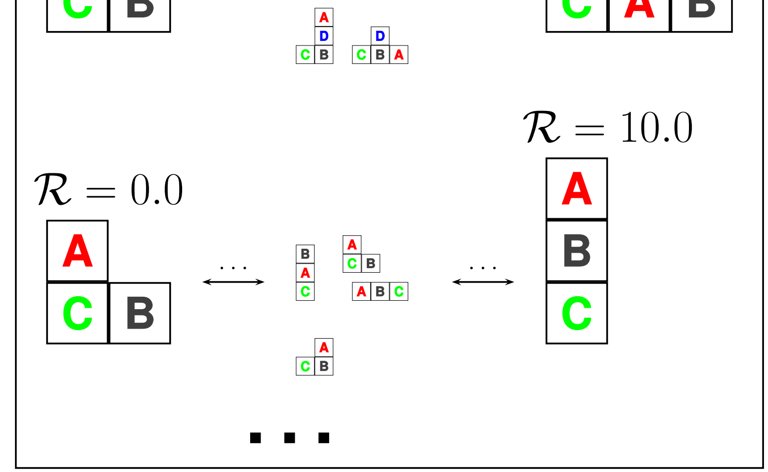
- We shall always be able to induce a classification of state-space regions by value and/or policy using state-formulae given by regression.

Picture of First-Order Regression



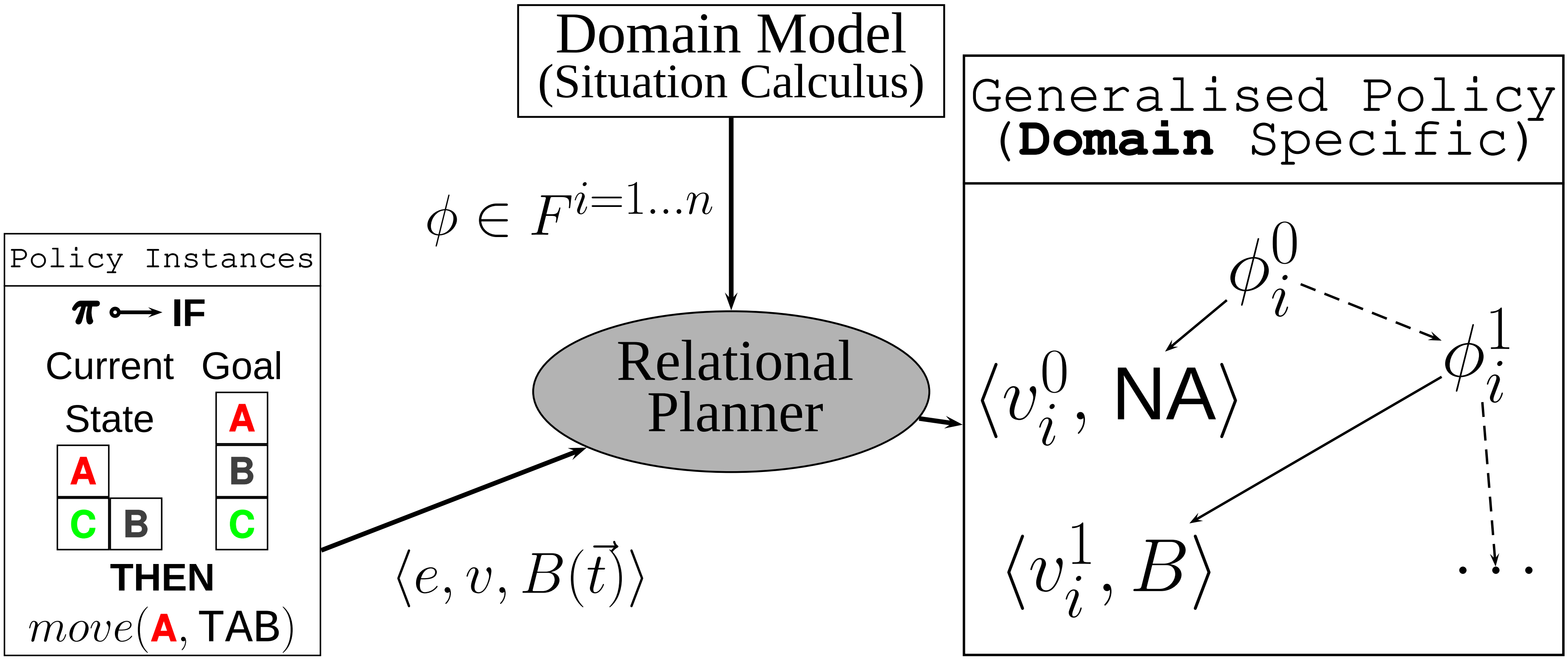
Planning (RMDP)





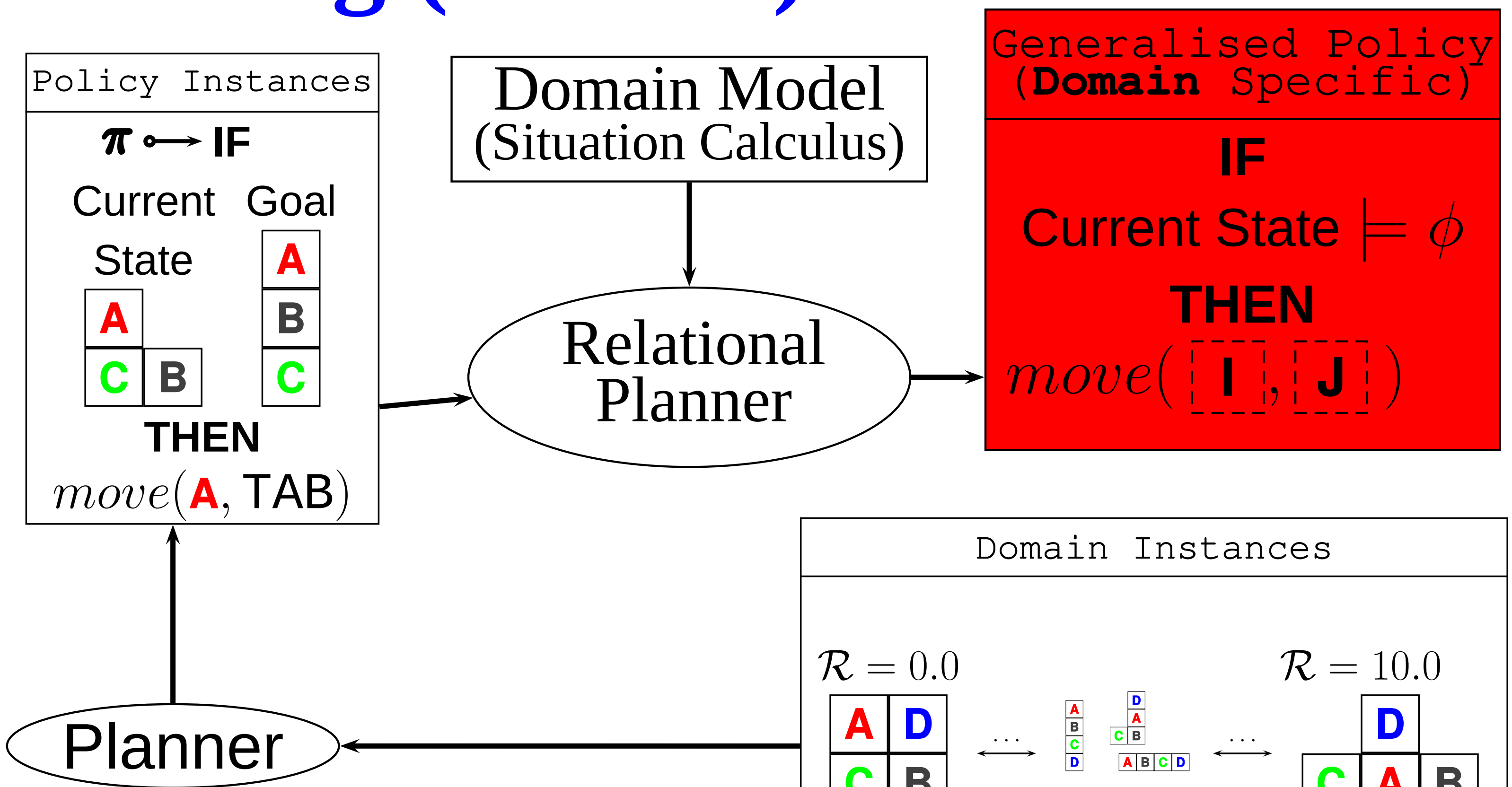
- Value/Policy Iteration (factored/tabular)
- LAO* (factored/tabular)
- LRTDP
- Q-Learning, TD(λ), API

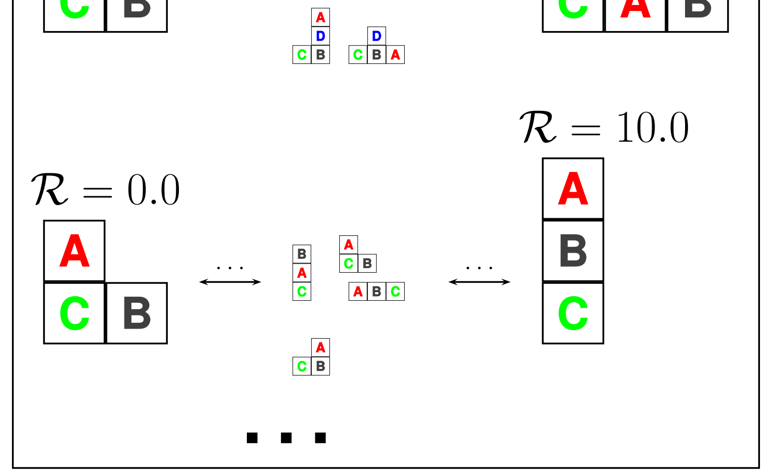
Algorithm



- e is an MDP state
- v is the value of e
- $B(\vec{t})$ is the optimal ground stochastic action

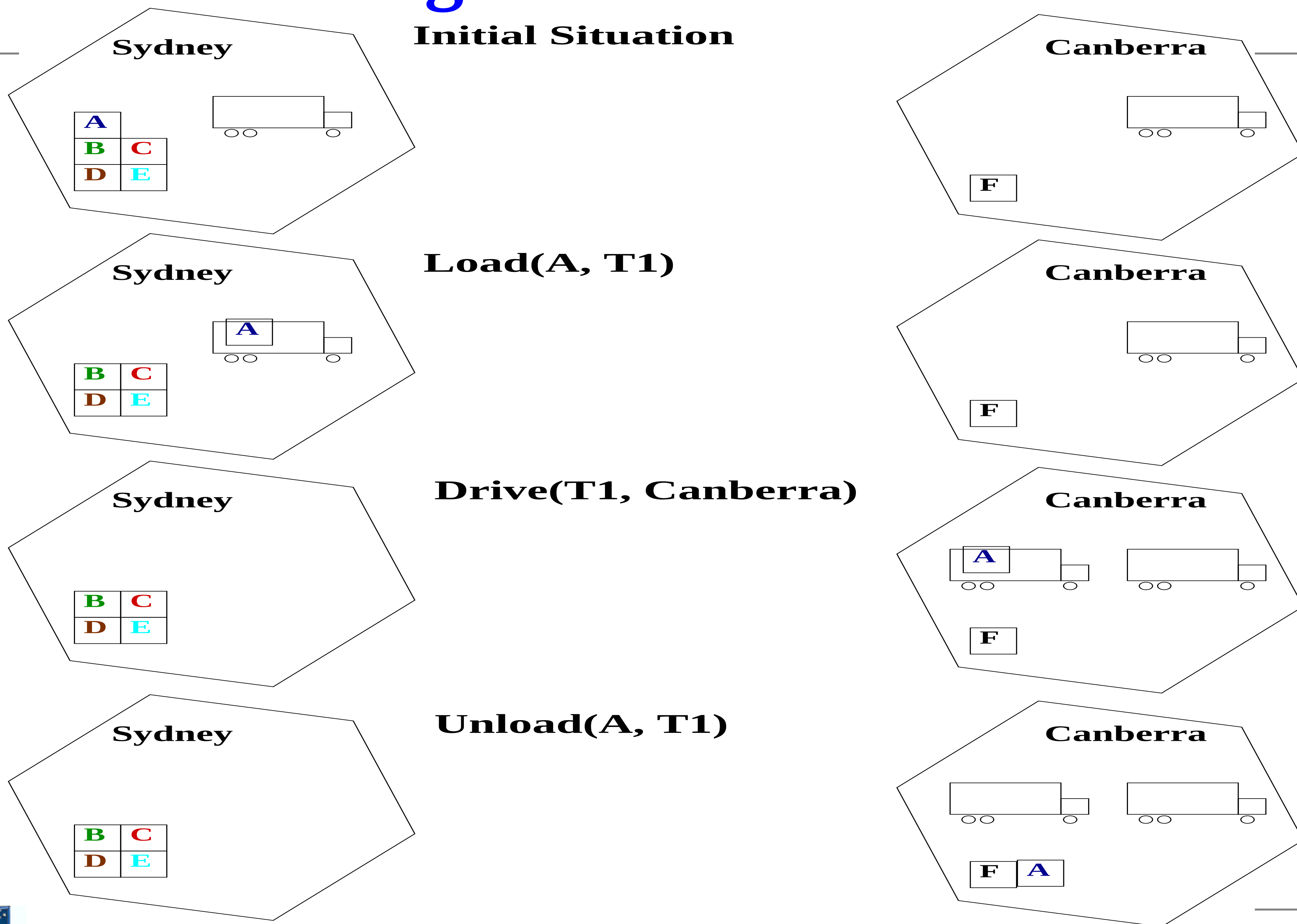
Planning (RMDP)





- Value/Policy Iteration (factored/tabular)
- LAO* (factored/tabular)
- LRTDP
- Q-Learning, TD(λ), API

Logistics [Boutilier *et al.*, 2001]



Policy – Logistics

IF $\exists b \; (Box(b) \wedge Bin(b, Syd))$

THEN act = NA, val = 2000

ELSE

IF $\exists b \exists t \; (Box(b) \wedge Truck(t) \wedge Tin(t, Syd) \wedge On(b, t))$

THEN $act = unload(b, t)$, val = 1900

ELSE

IF $\exists b \exists t \exists c \; (Box(b) \wedge Truck(t) \wedge City(c) \wedge$

$Tin(t, c) \wedge On(b, t) \wedge c \neq Syd)$

THEN $act = drive(t, Syd)$, val = 1805

ELSE

IF $\exists b \exists t \exists c \; (Box(b) \wedge Truck(t) \wedge City(c) \wedge$

$Tin(t, c) \wedge Bin(b, c) \wedge c \neq Syd)$

THEN $act = load(b, t)$, val = 1714.75

ELSE

IF $\exists b \exists t \exists c \; (Box(b) \wedge Truck(t) \wedge City(c) \wedge$

$\neg Tin(t, c) \wedge Bin(b, c))$

THEN $act = drive(t, c)$, val = 1629.01

Results – Deterministic

Domain	max_n	size	$\|E\|$	type	time	scope
LG-EX	4	2	56	P	0.2	∞
LG-EX	4	3	4536	P	14.41	∞
BW-EX	2	3	13	P	0.2	∞
BW-EX	2	4	73	P	2.2	∞
BW-EX	2	5	501	P	23.5	∞
BW-ALL	5	4	73	T	33.9	5
BW-ALL	6	4	73	T	136.8	6
BW-ALL	5	10	10	T	131.9	5
BW-ALL	6	10	10	T	2558.5	6
LG-ALL	8	2	56	P	1.8	8
LG-ALL	8	2	56	P	*0.5	8
LG-ALL	12	3	4536	P	#17630.3	5
LG-ALL	12	3	4536	P	#*263.4	6
LG-ALL	12	3	4536	P	#*1034.2	9

Results – Stochastic

Domain	max_n	size	$\|E\|$	type	time	scope
LG-EX$_s$	5	2	56	P	0.2	∞
LG-EX$_s$	5	3	4536	P	16.19	∞
BW-EX$_s$	3	3	13	P	0.3	∞
BW-EX$_s$	3	4	73	P	2.8	∞
BW-EX$_s$	3	5	501	P	29.3	∞
BW-ALL$_s$	4	4	73	P	*0.4	4
BW-ALL$_s$	7	4	73	P	*11.5	7
BW-ALL$_s$	8	4	73	P	*58.0	8
BW-ALL$_s$	9	4	73	P	*1389.6	9
LG-ALL$_s$	12	2	56	P	2.1	12
LG-ALL$_s$	12	2	56	P	*0.7	12
LG-ALL$_s$	22	3	4536	P	#1990.8	12
LG-ALL$_s$	22	3	4536	P	#*574.4	14
LG-ALL$_s$	22	3	4536	P	#*1074.5	15

Conclusions

- **GOOD ::** Given domains for which the optimal generalised value function has finite range
- **BAD ::** With infinite objects, the value function can have an infinite range
- Model checking is a bottle neck

Future work

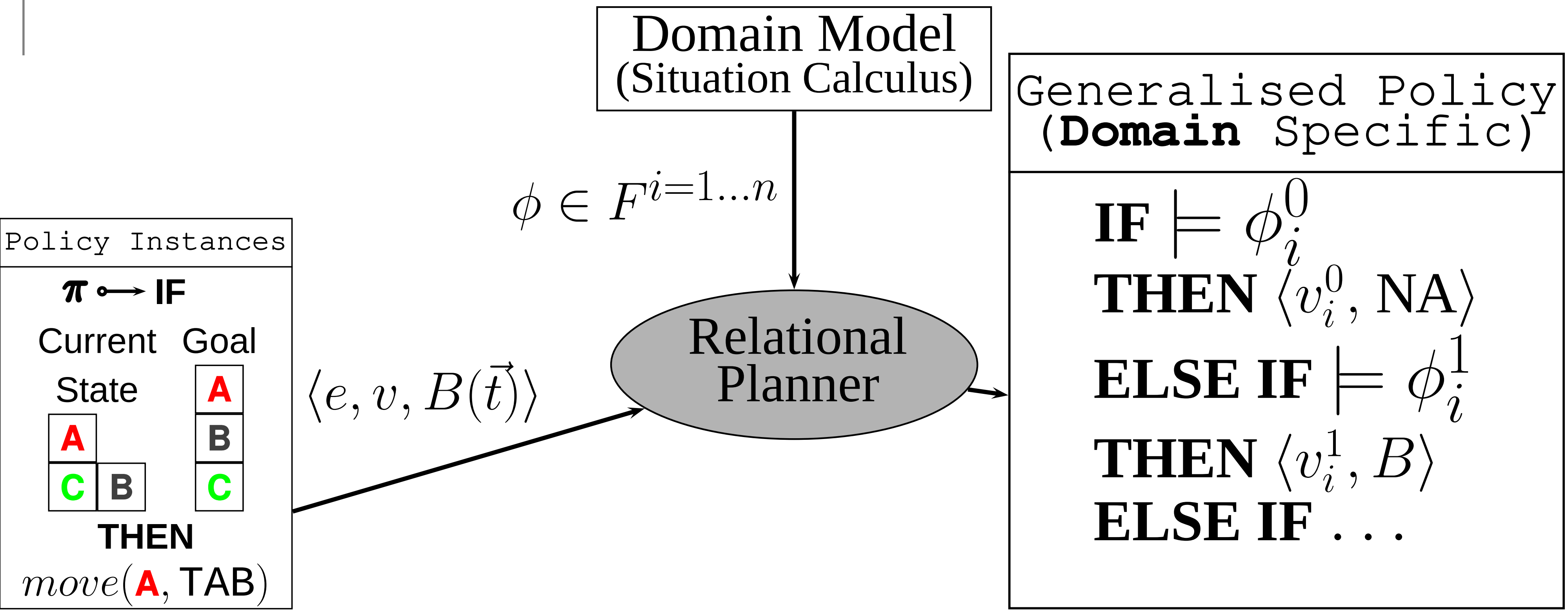
- Prune more via **control knowledge**

  Do not *try* unload after a load

  $$\models \ominus(a = load(\vec{x})) \rightarrow (a \neq unload(\vec{y}))$$

- Avoid implicit and explicit universal quantification at all costs
  - May have to **sacrifice optimality**
- Concatenate n-step-to-go optimal policies
  - **Macro actions**

Algorithm



- e is an MDP state
- v is the value of e
- $B(\vec{t})$ is the optimal ground stochastic action

Algorithm (pseudo code)

Initialise {max_n, $\{\phi^0\}$, F^0}; Compute set of examples E; Call $\textsc{Build\_tree}(0, E)$

function $\textsc{Build\_tree}$(n : integer, E : examples)

if $\textsc{pure}$(E) **then**

  return `success_leaf`

end if

$\phi \leftarrow$ good classifi er in F^n for E. **NULL** if none exists

if $\phi \equiv$ **NULL then**

  $n \leftarrow n + 1$

  if $n >$ max_n **then**

    return `failure_leaf`

  end if

  $\{\phi^n\} \leftarrow \textsc{update\_hypotheses\_space}(\{\phi^{n-1}\})$

  $F^n \leftarrow \{\phi^n\} \cup F^{n-1}$

  return $\textsc{Build\_tree}(n, E)$

else

  $positive \leftarrow \{\eta \in E \mid \eta$ satisfi es $\phi\}$

  $negative \leftarrow E \backslash positive$

  $positive\_tree \leftarrow \textsc{Build\_tree}(n, positive)$

  $negative\_tree \leftarrow \textsc{Build\_tree}(n, negative)$

  return $\textsc{Tree}(\phi, positive\_tree, negative\_tree)$

end if

References

[Boutilier *et al.*, 2001] C. Boutilier, R. Reiter, and B. Price. Symbolic Dynamic Programming for First-Order MDPs. In *Proc. IJCAI*, 2001.

[Dzeroski and Raedt, 2001] S. Dzeroski and L. De Raedt. Relational reinforcement learning. *Machine Learning*, 43:7–52, 2001.

[Fern *et al.*, 2004] A. Fern, S. Yoon, and R. Givan. Learning Domain-Specific Knowledge from Random Walks. In *Proc. ICAPS*, 2004.

[Guestrin *et al.*, 2003] C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia. Generalising Plans to New Environments in Relational MDPs. In *Proc. IJCAI*, 2003.

[Khardon, 1999] R. Khardon. Learning action strategies for planning domains. *Artificial Intelligence*, 113(1-2):125–148, 1999.

[Martin and Geffner, 2000] M. Martin and H. Geffner. Learning generalized policies in planning using concept languages. In *Proc. KR*, 2000.

[Mausam and Weld, 2003] Mausam and D. Weld. Solving Relational MDPs with First-Order Machine Learning. In *Proc. ICAPS Workshop on Planning under Uncertainty and Incomplete Information*, 2003.

[Yoon *et al.*, 2002] S.W. Yoon, A. Fern, and R. Givan. Inductive Policy Selection for First-Order MDPs. In *Proc. UAI*, 2002.