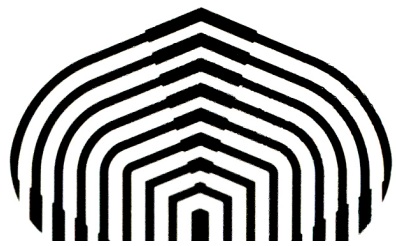
Learning Planning Representations from Traces via SAT

Blai Bonet

Universidad Simón Bolívar, Venezuela

ICAPS. October 26–30, 2020



Collaborators

Learning Planning Representations

Given **set of traces**, where trace is sequence of observation-label pairs

$$o_0, \ell_0, o_1, \ell_1, \ldots, o_n$$

Want **symbolic model for planning** (e.g. STRIPS model) that **explains** the input traces, and is **general** (i.e. works for "bigger instances")

This is general setting; main challenges are:

- What are the assumptions on the input (traces)?
- What's the form of the sought model? and what's it mean to be general?
- How is the model used on new instances?

Motivation

Model-based approach for planning is **successful and robust:**

– steady development of theory, methods and solvers

– models can be understood, analyzed, composed into bigger models, etc.

But models are needed:

– which are often complex and hand-crafted

– and thus planning (technology) is difficult to deploy in real-world settings

(Deep) RL doesn't need full models and has shown impressive results, but **latent representations** are opaque and difficult to understand or analyze
[Groshev, 2018; Garnelo et al., 2016; Marcus, 2018; etc]

Learning planning representations that are general is a step forward in bridging the gap between **model-based solvers** and **model-free learners**

Indeed, learned models can be combined with general solvers based on DL
[Toyer et al., 2018; Bueno et al., 2019; Issakkimuthu et al., 2018; Garg et al., 2018; etc]

Outline

1. Learning first-order STRIPS models for classical planning
2. Qualitative Numerical Planning (QNP): Language for generalized planning
3. Learning QNP abstractions from symbolic traces
4. Learning QNP abstractions from non-symbolic traces provided by teacher
5. Wrap Up

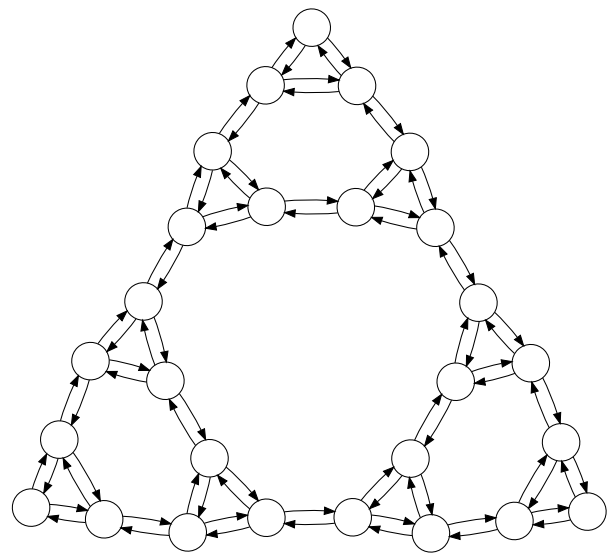
Example: Hanoi

Set of non-symbolic traces on 3-disk and 3-peg instance

Under the assumption:

- states associated with unique obs
- different states associated with diff. obs

Input traces are **summarized** into **single directed graph** (27 nodes, 78 edges)



`MoveDisk(from,to,d):`

Static: `BIGGER(from,d), BIGGER(to,d), NEQ(from,to)`
Prec: `-clear(from), clear(to), clear(d), Non(from,d), -Non(d,from), Non(d,to)`
Effect: `clear(from), -clear(to), Non(d,from), -Non(d,to)`

Partially Observable, Symbolic, and Non-Symbolic Traces

Trace $o_0, \ell_0, o_1, \ell_1, \ldots$ is induced by state trajectory $s_0, a_0, s_1, a_1, \ldots$

– observation o_i corresponds to state s_i: $s_i \rightsquigarrow o_i$

– label ℓ_i corresponds to action a_i: $a_i \rightsquigarrow \ell_i$

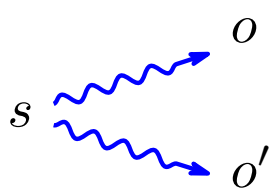
Trace is **non-symbolic** iff each observation and label is **flat** (i.e. has no "structure"), for example, given by **images**

Set of traces is:

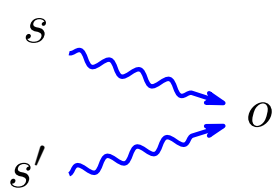
– **complete** if each possible transition (s, a, s') is **represented** by (o, ℓ, o')

– **fully observable** (or free of confusion) if each state yields single observation, and no different states with same observation

Neither



or



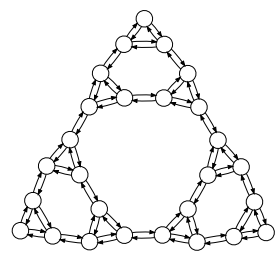
Canonical Representation of Input

Under the following assumptions about the input set T of traces:

- each trace is **non-symbolic** (e.g., images)
- **complete** and **fully observable**

Input T is represented by **directed and labeled** graph $G_T = (V, L, E)$:

- V is set of observations in the traces in T
- L is set of labels in the traces in T
- E is set of **labeled edges** (o, ℓ, o') that appear as transitions in T

Example: In Hanoi, graph $G_T =$  encodes a **complete and fully observable** set T of **non-symbolic** traces

First-Order STRIPS Domains

A (first-order) STRIPS domain is a pair (σ, A) where

- σ is first-order language made of **constants and relations**; e.g., $\{a^0, u^1, r^2)$ denotes language with constant a, and unary and binary predicates u and r
- A is set of **action schemas** $a(\bar{x}) = (\text{Pre}, \text{Eff})$ where $\bar{x}$ is vector of **variables**, and Pre and Eff are sets of σ-literals with free vars in $\bar{x}$

Example: Problem of Hanoi can be encoded in STRIPS with language $\sigma = \{clear^1, on^2, BIGGER^2\}$ and single action:

`MoveDisk(?disk,?src,?dst):`

Prec : $clear(?disk)$, $clear(?dst)$, $on(?disk, ?src)$, $BIGGER(?dst, ?disk)$

Effect : $\neg on(?disk, ?src)$, $\neg clear(?dst)$, $clear(?src)$, $on(?disk, ?dst)$

STRIPS Instances and State Graphs

STRIPS instance for domain $D = (\sigma, A)$ is tuple $I = (O, Init, G)$ where

- O is set of **object names** that extend σ with constants
- $Init$ and G are sets of literals that denote the **initial and goal states**

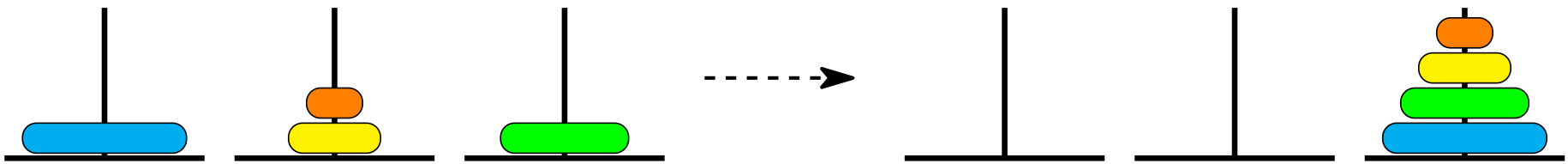
STRIPS model is tuple $M = (D, I)$ made of domain $D = (\sigma, A)$ and instance $I = (O, Init, G)$

M spans **labeled and directed graph** $\mathcal{G}_M$ made of states that are reachable from state denoted by $Init$ (via grounded actions), and labeled edges $(s, a(\bar{o}), s')$ where $a(\bar{o})$ is the ground action that maps state s into s'

Hanoi in STRIPS

Language $\sigma = \{d_0^0, d_1^0, d_2^0, d_3^0, peg_0^0, peg_1^0, peg_2^0, clear^1, on^2, BIGGER^2\}$

Static: $BIGGER(peg_i, d_j)$ and $BIGGER(d_i, d_j)$ for $i < j$



$clear(d_0), on(d_0, peg_0),$
$clear(d_3), on(d_3, d_2), on(d_2, peg_1),$
$clear(d_1), on(d_1, peg_2)$

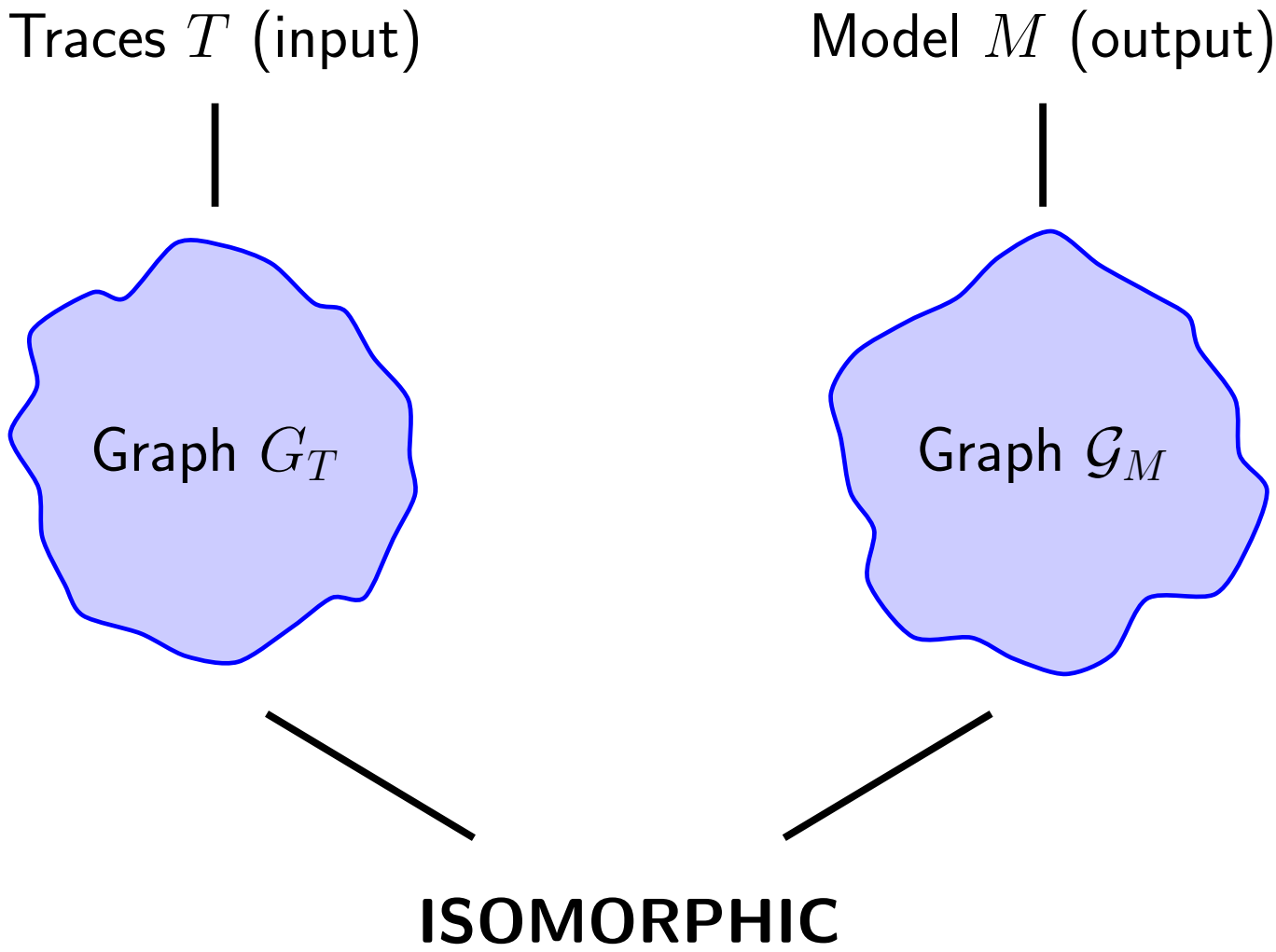
$clear(d_3), on(d_3, d_2), on(d_2, d_1),$
$on(d_1, d_0), on(d_0, peg_2)$

MoveDisk(?disk,?src,?dst):

Prec : $clear($?disk$)$, $clear($?dst$)$, $on($?disk, ?src$)$, $BIGGER($?dst, ?disk$)$

Effect : $\neg on($?disk, ?src$)$, $\neg clear($?dst$)$, $clear($?src$)$, $on($?disk, ?dst$)$

Learning Task for STRIPS Models: Find Isomorphism



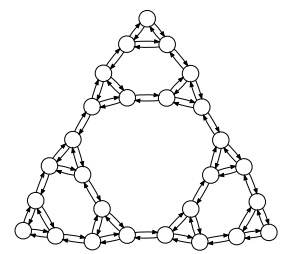
Learning Task for STRIPS Models

For **complete and fully observable** set T of **non-symbolic** traces, represented by G_T over labels L, find $M = (\sigma, A, O, Init, G)$ such that

- each action schema $a(\bar{x})$ in A is associated with label ℓ_a in L
- $\mathcal{G}_M$ and G_T are **isomorphic graphs** with respect to labels $\{\ell_a : a(\bar{x}) \in A\}$

I.e., **1-1 and onto** map f between the vertices of $\mathcal{G}_M$ and G_T such that $(s, a(\bar{o}), s') \in \mathcal{G}_M$ iff $(f(s), \ell_a, f(s')) \in G_T$

In such case, we say that M **solves (or explains)** the input T (or G_T)

Example: STRIPS model for Hanoi **solves** $G_T =$ 

Learning Task for STRIPS Models: General

Given input sets of traces $T_1, T_2, \ldots, T_n$, find STRIPS domain D and instances $I_1, I_2, \ldots, I_n$ for D such that

– each model $M_i = (D, I_i)$ solves the input T_i

If $\mathcal{T} = \{T_1, T_2, \ldots\}$ is class of sets of traces, domain D is **general for $\mathcal{T}$** if for each T_i, there is instance I_i for D such that $M_i = (D, I_i)$ solves T_i

Often, **single and small** T_i enough to learn domain D that is general for an **infinite class** $\mathcal{T}$, yet

– we can't **prove** that D is indeed general for class $\mathcal{T}$

– but we can **verify** that D is general for given **finite subset** of $\mathcal{T}$

Bounded Combinatorial Task

A STRIPS model $M = (\sigma, A, O, Init, G)$ has

- finite but arbitrary large first-order language σ
- finite but arbitrary large set of action schemas A
- finite but arbitrary large set of objects O

that make the space $\mathcal{M}$ of models **unbounded**

However, if α is a vector of **hyperparameters** that bound

- number predicates and max. arity
- number of different atoms used in D
- number of action schemas and max. number of arguments
- number of objects in O

then subclass $\mathcal{M}_\alpha = \{M \in \mathcal{M} : M$ complies with bounds in $\alpha\}$ is **finite**

STRIPS Learner: Learning as Combinatorial Search

For increasing sequence of hyperparameters $\alpha_1 \prec \alpha_2 \prec \alpha_3 \prec \cdots$, the learning task can be solved by

1. $k := 1$
2. **While** true **do**
3. Set $\alpha := \alpha_k$
4. Find domain $D = (\sigma, A)$ and instances $I_1, \ldots, I_n$ such that $M_i = (D, I_i)$ is in $\mathcal{M}_\alpha$ and solves G_{T_j}, for $j = 1, \ldots, n$
5. If successful, return domain D and instances $I_1, \ldots, I_n$
6. Otherwise, set $k := k + 1$

Step 4 can be solved in finite time since $\mathcal{M}_\alpha$ is finite. In practice, outer loop is stopped after sufficiently large α

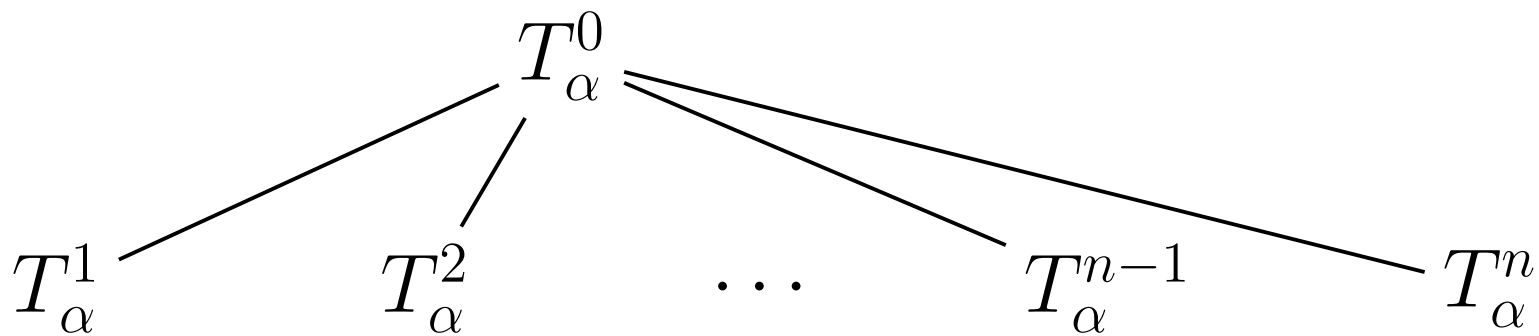
Step 4 is combinatorial task tackled with SAT. Yet, crucial idea is to find model M such that G_T and $\mathcal{G}_M$ are isomorphic!

Encoding in SAT

For vector α of hyperparameters, SAT theory is partitioned into **layers:**

- zero-th layer T^0_α encodes general domain D
- i-th layer T^i_α encodes instance I_i and assures $\mathcal{G}_{M_i}$ is isomorphic to G_{T_i}, $i = 1, \ldots, n$

Propositional theory is $T_\alpha = T^0_\alpha \cup \bigcup_{i=1}^{n} T^i_\alpha$



(edges indicate sharing of variables between subtheories)

Domain Encoding (Sketch): T^0_α

- α sets #preds (max. arity), #actions (max. args), **#atoms,** #static preds
- Atoms for all schemas enumerated as $m_0, m_1, \ldots$; e.g., $on(?x, ?y), clear(?z), \ldots$

Propositions (decision), others (implied) not shown:

- $label(a, l)$: assign label l to schema a
- $p0(a, m)/p1(a, m)$: atom m is neg/pos precondition of schema a
- $e0(a, m)/e1(a, m)$: atom m is neg/pos effect of schema a
- $arity(p, i)$: predicate symbol p has arity i
- $is(m, p)$: atom m is p-atom
- $at(m, i, \nu)$: i-th arg of atom m is (action) argument ν;
  [e.g., $a(?x, ?y)$ and $m = p(?y) \Rightarrow at(m, 1, 2)$]
- $un(u, a, \nu)$: schema a uses **static unary predicate** u on argument ν
- $bin(b, a, \nu, \nu')$: schema a uses **static binary predicate** b on arguments ν and ν'

Theorem

Soundness: *Satisfying assignment μ for T^0_α encodes domain D_μ bounded by α.*
Completeness: *Domain D bounded by α induces satisfying assignment μ_D for T^0_α, and $D_{\mu_D} = D$*

Instance Encoding (Sketch): T^i_α

Encoding of instance I_i that defines model $M_i = (D, I_i)$

- α bounds #objects in instance I_i
- Ground atoms enumerated as $k_0, k_1, \ldots$; e.g. $on(A, B), clear(B), \ldots$

Propositions (decision), others (implied) not shown:

– $gr(k, p)$: ground atom k refers to predicate symbol p

– $gr(k, i, o)$: i-th argument of ground atom k is object o

– $\phi(k, s)$: boolean value of ground atom k at state s

– $r(u, o)$: true if $u(o)$ holds for **static unary predicate** u

– $s(b, o, o')$: true if $b(o, o')$ holds for **static binary predicate** b

Satisfying assigment defines interpretation for σ-literals on all states in G_{T_i}, and gives value to unary and binary **static predicates**

Formulas in T^i_α ensure interpretation is consistent, and different states give different value to at least some grounded atom k

Isomorphism (Sketch): Also in T^i_α

Propositions (decision), others (implied) not shown:

- $mp(t, a)$: transition t is mapped to action schema a
- $mf(t, k, m)$: ground atom k is mapped to atom m in transition t
- $gtuple(a, \bar{o})$: true if $a(\bar{o})$ is a ground instance of a

Edges in G_{T_i} correspond to edges in $\mathcal{G}_M$:

- $mp(t, a)$ map transitions to actions and $mf(t, k, m)$ arguments to objects
- Formulas in T^i_α ensure interpretation of atoms across transitions agree with preconditions and effects of actions

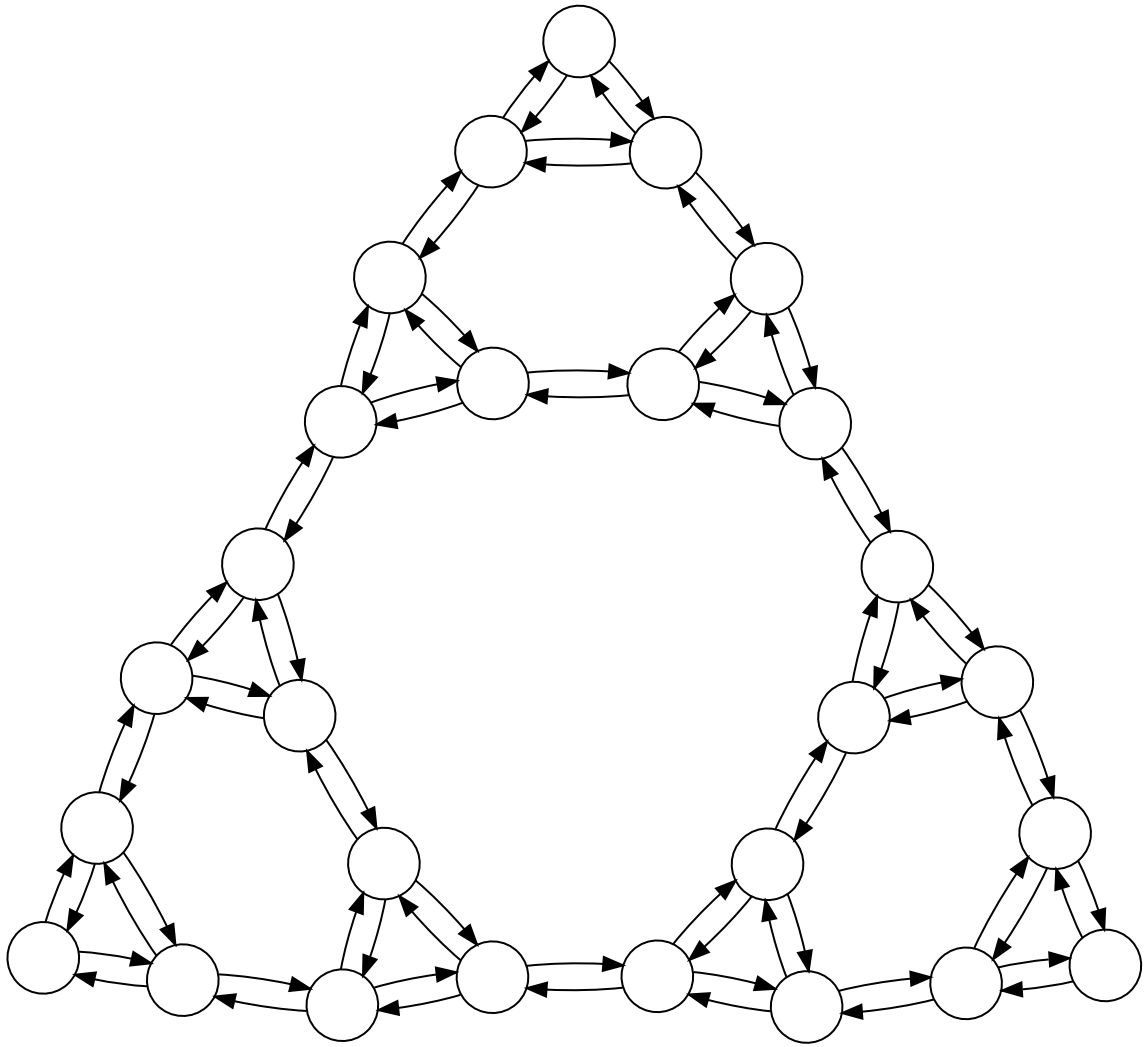
Edges in $\mathcal{G}_M$ correspond to edges in G_{T_i}:

- **Lack of edges** in G_{T_i} explained by model M, and **applicable actions** are applied
- Formulas in T^i_α ensure both

Theorem

(Soundness) *SAT assignment μ for T_α encodes D and $I_1, \ldots, I_n$ bounded by α such that $M_i = (D, I_i)$ solves G_{T_i}.* **(Completeness)** *If D and $I_1, \ldots, I_n$ are bounded by α such that (D, I_i) solves G_{T_i}, there is SAT assignment μ for T_α*

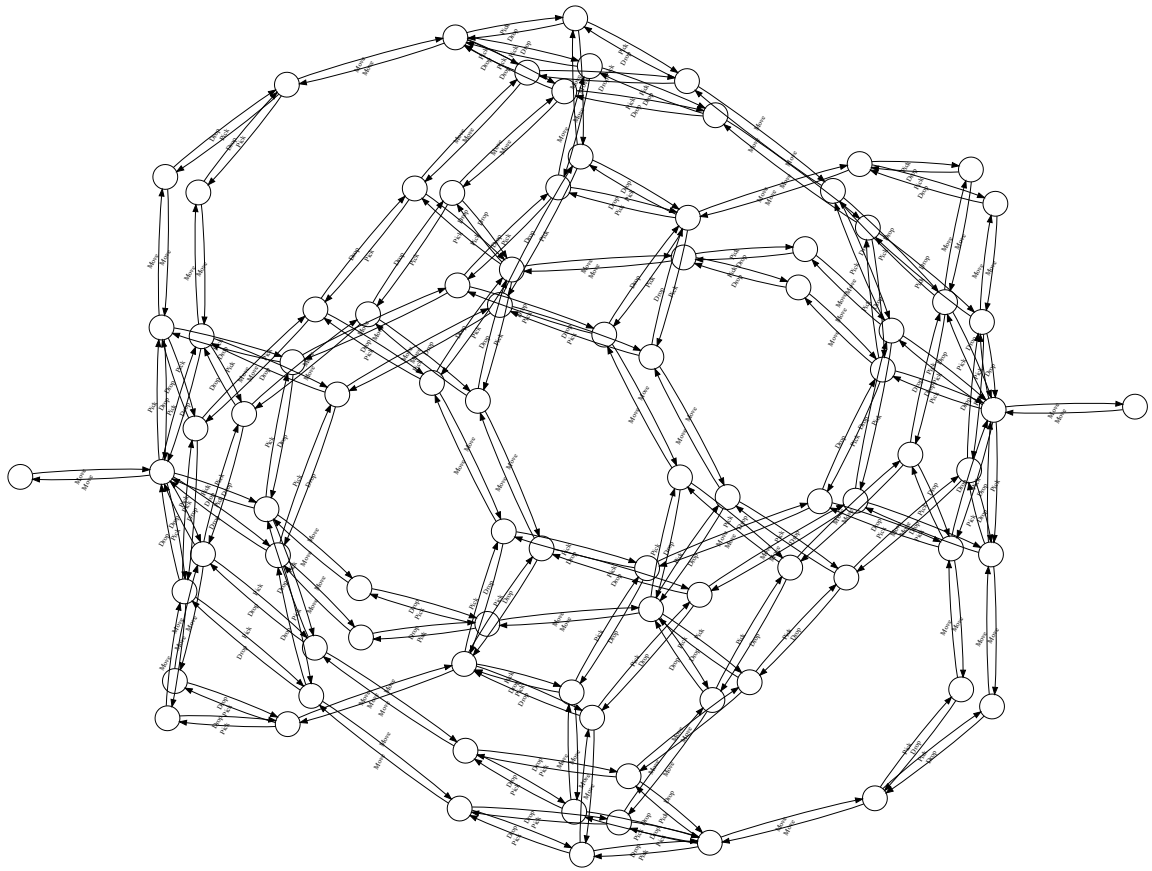
Example: Hanoi: Input and Output



MoveDisk(from,to,d):

Static: BIGGER(from,d), BIGGER(to,d), NEQ(from,to)
Prec: -clear(from), clear(to), clear(d), Non(from,d), -Non(d,from), Non(d,to)
Effect: clear(from), -clear(to), Non(d,from), -Non(d,to)

Example: Gripper: Input and Output



Graph: 88 nodes + 280 edges

Labels: Move, Drop, Pick

Move(from,to):

Static: CONN(from,to)
Prec: at(from), -at(to)
Effect: -at(from), at(to)

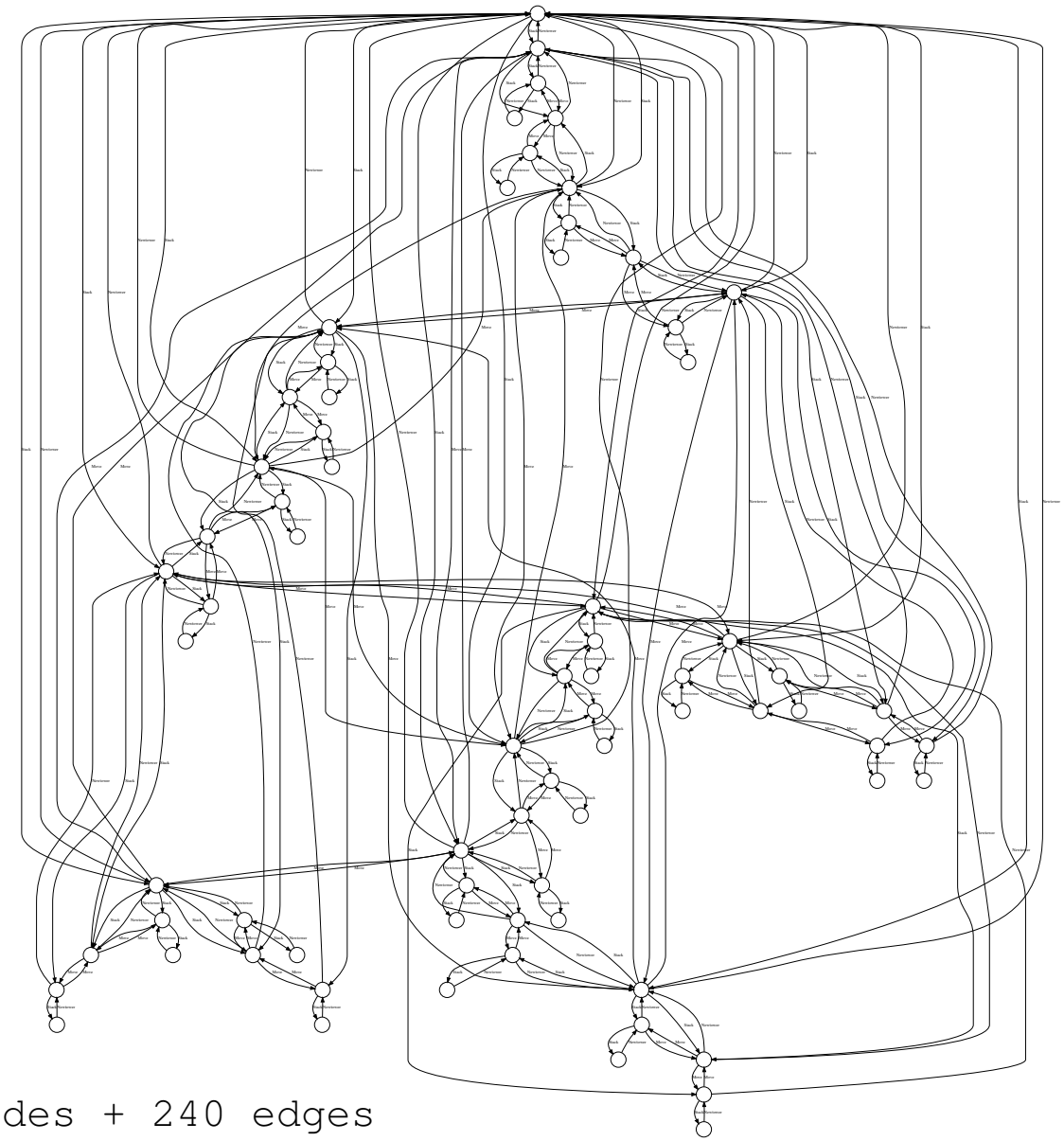
Drop(ball,room,gripper):

Static: PAIR(room,gripper)
Prec: at(room), Nfree(gripper), hold(gripper,ball), Nat(room,ball)
Effect: -Nfree(gripper), -hold(gripper,ball), -Nat(room,ball)

Pick(ball,room,gripper):

Static: PAIR(room,gripper)
Prec: at(room), -Nfree(gripper), -hold(gripper,ball), -Nat(room,ball)
Effect: Nfree(gripper), hold(gripper,ball), Nat(room,ball)

Example: Blocksworld: Input



Graph: `73 nodes + 240 edges`

Labels: `MoveToTable, MoveFromTable, Move`

Example: Blocksworld: Output

Graph: `73 nodes + 240 edges`

Labels: `MoveToTable, MoveFromTable, Move`

`MovetoTable(x,y):`

Static: `NEQ(x,y)`
Prec: `-Nclear(x), Nclear(y), -Ntable-OR-Non(x,y), Ntable-OR-Non(x,x)`
Effect: `-Nclear(y), -Ntable-OR-Non(x,x), Ntable-OR-Non(x,y)`

`MoveFromTable(x,y,d):`

Static: `NEQ(x,y), EQ(y,d)`
Prec: `-Nclear(x), -Nclear(d), -Ntable-OR-Non(x,x), Ntable-OR-Non(x,y)`
Effect: `Nclear(d), Ntable-OR-Non(x,x), -Ntable-OR-Non(x,y)`

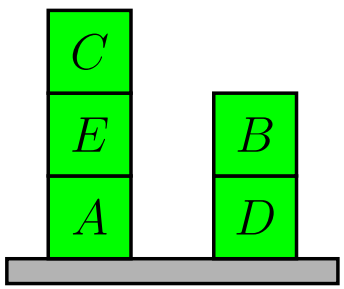
`Move(x,z,y):`

Static: `NEQ(x,z), NEQ(z,y), NEQ(x,y)`
Prec: `-Nclear(x), Nclear(y), -Nclear(z), Ntable-OR-Non(x,x),`
`Ntable-OR-Non(x,z), -Ntable-OR-Non(x,y)`
Effect: `Nclear(z), -Nclear(y), Ntable-OR-Non(x,y), -Ntable-OR-Non(x,z)`

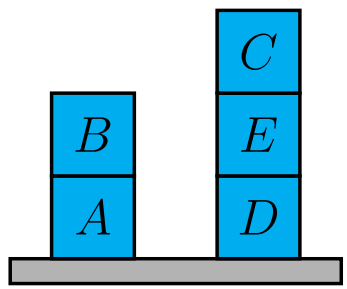
Generalized Planning: The Challenge

Generalized planning is about obtaining a **general plan or strategy** for solving collections of planning problems

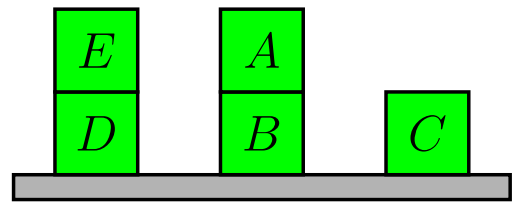
For example, find general strategy to achieve **fixed goal** in all Blocksworld problems, independently of number or initial configuration of blocks



Get A clear



Get A on B



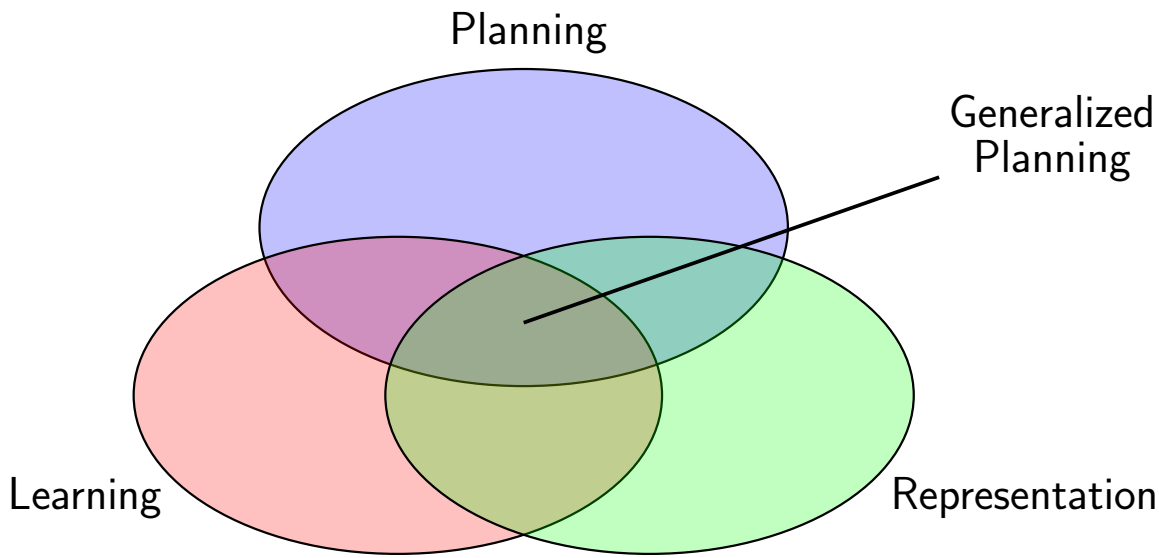
Get tower A, B, C, D, E

Generalized Planning: Motivation

Why solve individual problems from scratch if it's possible to learn general plan in one shot?

Lots of current research in **deep (reinforcement) learning** is about computation of general plans or policies; e.g. [Espeholt et al., 2018; Groshev et al., 2018; Chevalier-Boisvert et al., 2019; François-Lavet et al. 2019]

Generalized planning gives us a crisp vocabulary to talk about general plans [Levesque, 2005; Hu & De Giacomo, 2011; B. & Geffner, 2015; Belle & Levesque, 2016; Jiménez et al., 2019; Illanes & McIlraith, 2019]



Qualitative Numerical Planning

Simple and expressive language for **generalized planning**, introduced by [Srivastava et al., 2011]:

- QNP is **propositional abstraction** for underlying collection $\mathcal{Q}$ of planning instances
- solutions for QNP are policies that solves all planning problems in $\mathcal{Q}$
- after proper reduction to FOND, solved with off-the-shelf FOND planner

QNP problems are similar to STRIPS problems but extended with **numerical variables** that can be incremented or decremented **qualitatively**

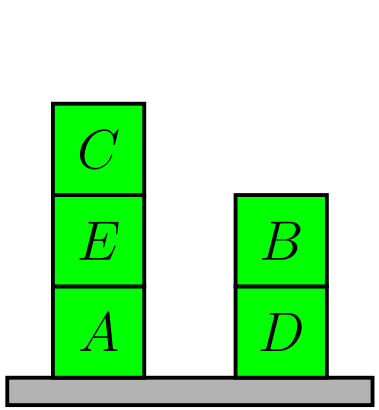
QNP Example: $clear(x)$

Goal: Remove all blocks above fixed block x

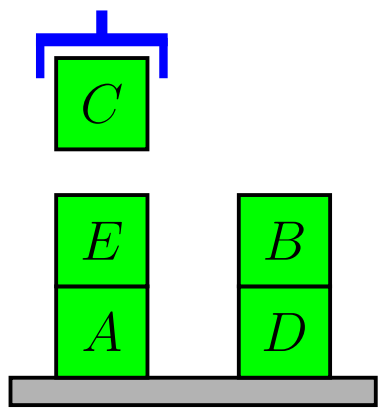
QNP $Q_{clear} = (F, V, I, A, G)$ captures all Blocksworld problems:

- $F = \{H\}$ where H denotes whether gripper holds a block
- $V = \{n\}$ where n **"counts"** blocks above x ($n{>}0$ iff some block above x)
- $I = \{\neg H, n > 0\}$
- $G = \{n{=}0\}$

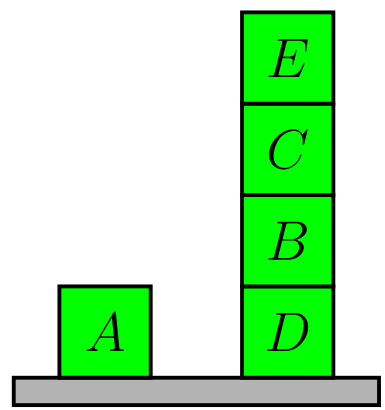
$x =$ block A



$\neg H,\ n = 2$



$H,\ n = 1$

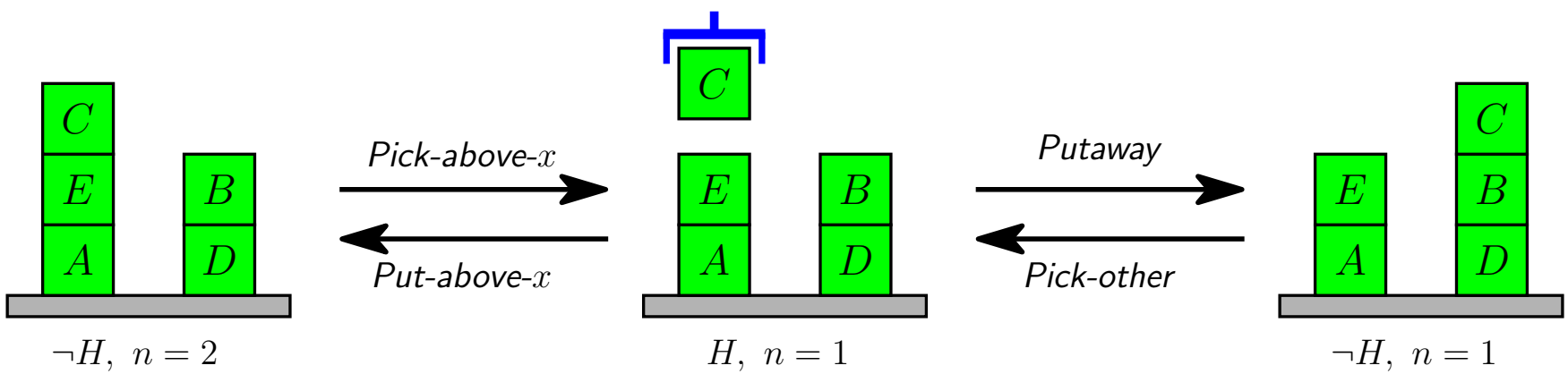


$\neg H,\ n = 0$

QNP Example: Actions for $clear(x)$

- *Putaway* $= \langle H; \neg H \rangle$ puts held block on table or block not above x
- *Pick-above-*x $= \langle \neg H, n > 0; H, n{\downarrow} \rangle$ picks the top block above x
- *Put-above-*x $= \langle H; \neg H, n{\uparrow} \rangle$ puts block being held on top block above x
- *Pick-other* $= \langle \neg H; H \rangle$ picks block not above x

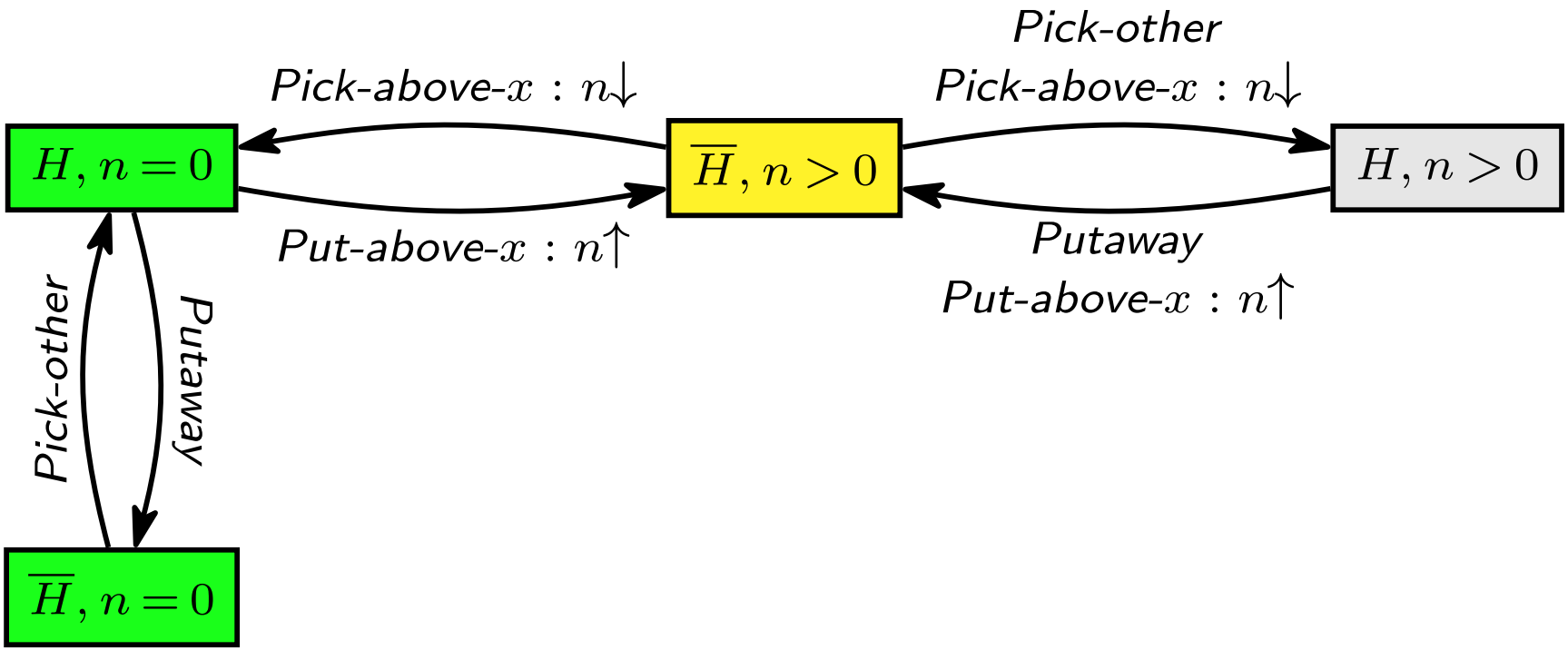
Block x is block A:



Example: QNP Abstraction for $clear(x)$

Observation projection for $Q_{clear} = (F, V, I, A, G)$ where

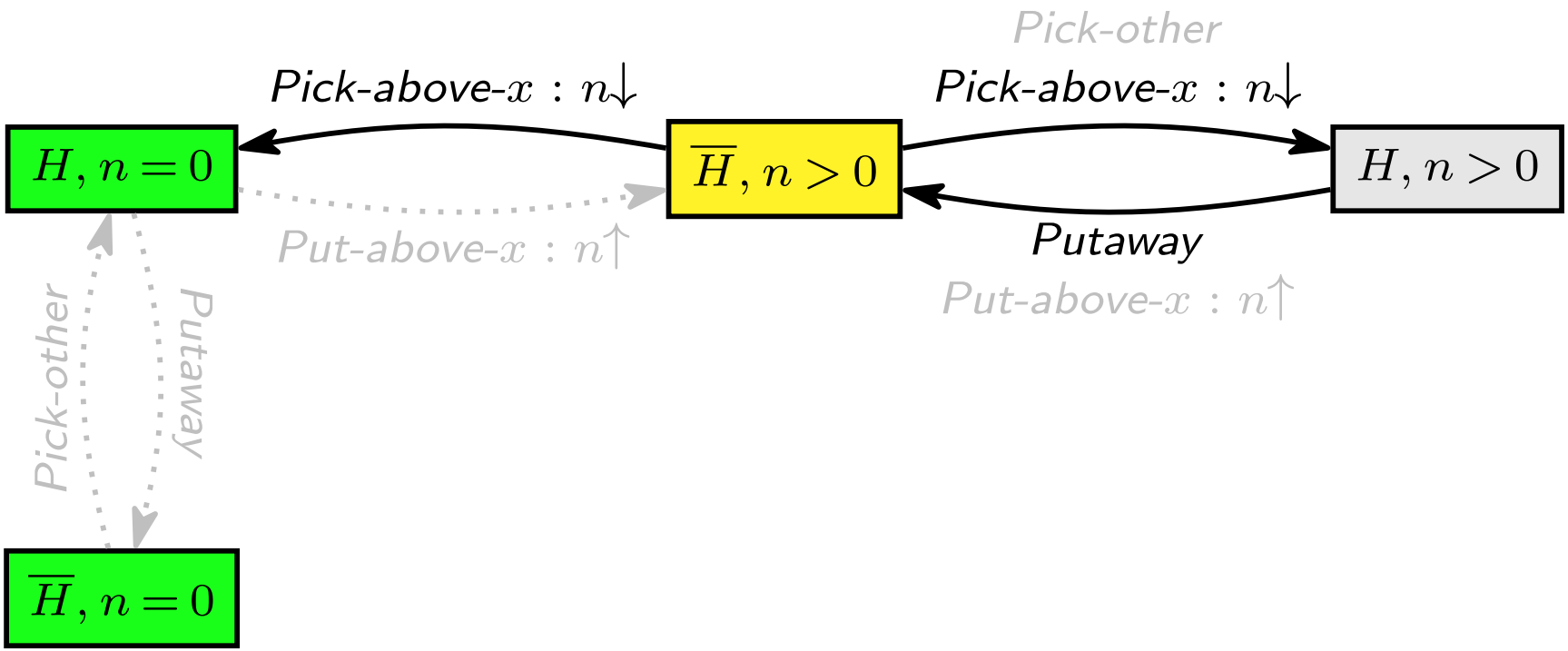
- $F = \{H\}$ and $V = \{n\}$
- $I = \{\overline{H}, n > 0\}$ and $G = \{n = 0\}$
- Actions in A: *Putaway* $= \langle H; \neg H \rangle$, *Pick-above-*$x = \langle \neg H, n > 0; H, n{\downarrow} \rangle$, *Put-above-*$x = \langle H; \neg H, n{\uparrow} \rangle$, and *Pick-other* $= \langle \neg H; H \rangle$



Example: QNP Solution for $clear(x)$

Observation projection for $Q_{clear} = (F, V, I, A, G)$ where

- $F = \{H\}$ and $V = \{n\}$
- $I = \{\overline{H}, n > 0\}$ and $G = \{n = 0\}$
- Actions in A: *Putaway* $= \langle H; \neg H \rangle$, *Pick-above-*$x = \langle \neg H, n > 0; H, n{\downarrow} \rangle$, *Put-above-*$x = \langle H; \neg H, n{\uparrow} \rangle$, and *Pick-other* $= \langle \neg H; H \rangle$



QNP Syntax

QNP is tuple $Q = (F, V, I, A, G)$ where

- F is a finite set of propositions
- V is a finite set of numerical variables
- I is a set of $F{+}V$-literals, where V-literals are $X{=}0$ or $X{>}0$ for X in V
- G is goal condition given by $F{+}V$-literals
- A is set of actions. Each has **precondition** *Pre* ($F{+}V$-literals), **boolean effects** *Eff* (F-literals), and **numerical effects** N (atoms $X{\uparrow}$ or $X{\downarrow}$) with restriction that if $X{\downarrow}$ in N, then $X > 0$ must be in *Pre*

Numerical vars affected only **qualitatively**, and tested for zero

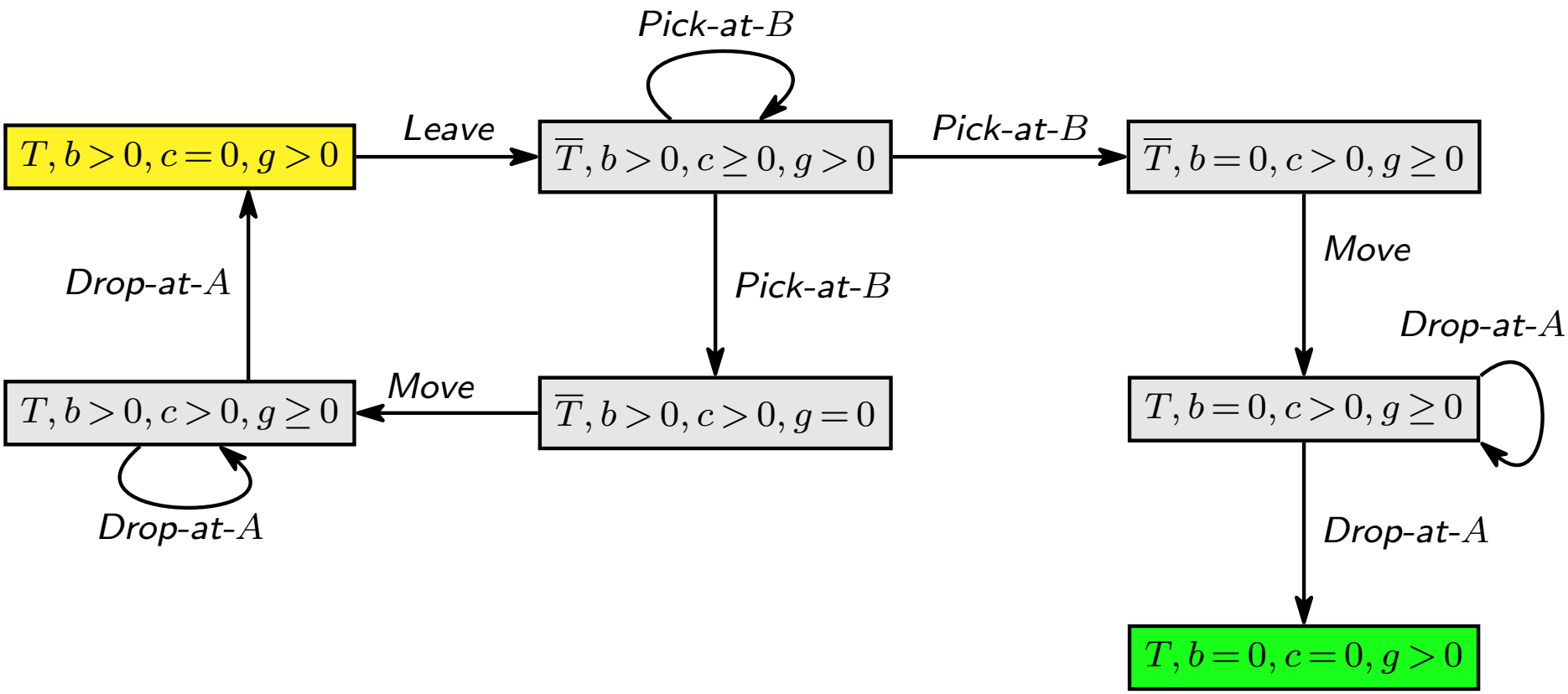
Plan-existence for QNPs **decidable** [Srivastava et al., 2011] whereas it is **undecidable** for numerical planning [Helmert, 2002]

Example: QNP for Gripper

QNP $Q_{gripper} = (F, V, I, A, G)$:

- $F = \{T\}$ where T iff *at*(A)
- $V = \{b, c, g\}$ where b counts balls at B, c balls held, and g free grippers
- $I = \{T, b > 0, c = 0, g > 0\}$ and $G = \{c = 0, b = 0\}$
- Abstract actions are:
  - *Move* $= \langle \neg T; T \rangle$ and *Leave* $= \langle T; \neg T \rangle$
  - *Pick-at-*$B = \langle \neg T, b > 0, g > 0; b\downarrow, c\uparrow, g\downarrow \rangle$
  - *Drop-at-*$B = \langle \neg T, c > 0; b\uparrow, c\downarrow, g\uparrow \rangle$
  - *Drop-at-*$A = \langle T, c > 0; c\downarrow, g\uparrow \rangle$

Example: Solution for Gripper

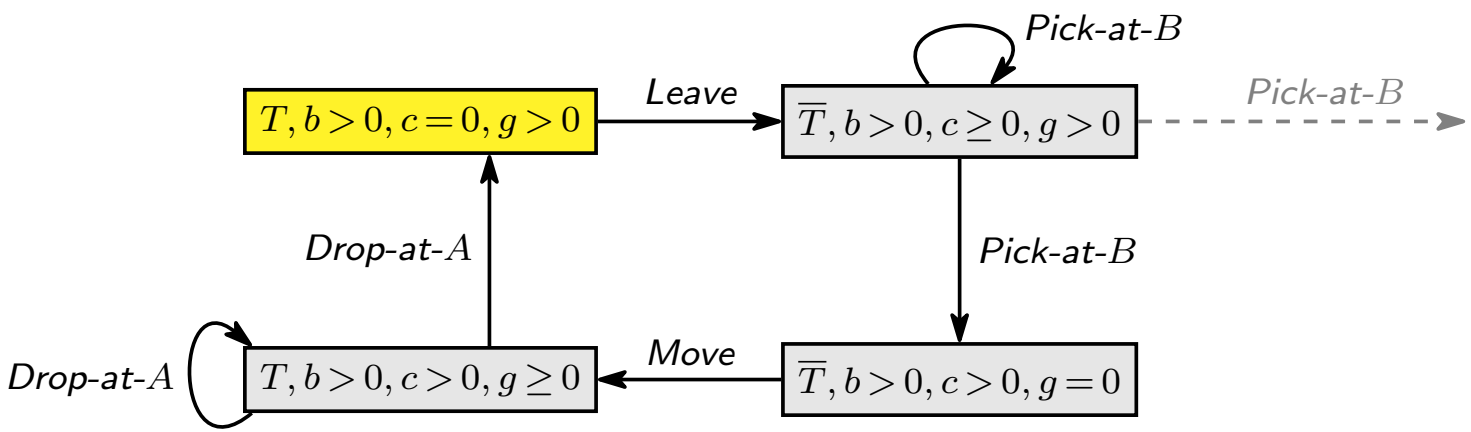


QNP Solutions

QNP solutions are the **strong-cyclic solutions** that **terminate**

- Strong-cyclic means that each reachable state is connected to a goal state
- π terminates if each infinite induced trajectory terminates
- Infinite trajectory denoted by $s_0, s_1, \ldots [s_i, \ldots, s_m]^*$ where $\{s_i, \ldots, s_m\}$ is **set of recurrent states (loop)**
- Such trajectory terminates iff there is variable X that is **decremented but not incremented** in loop

Example: loop terminates because variable b is decremented by *Pick-at-B* but not incremented in loop



Learning QNPs from Symbolic Traces

QNPs are expressive and effective!

QNPs can be **learned from symbolic traces** [B., Francès & Geffner, 2019]

We assume that observations in traces correspond to **full symbolic** representations of states in a planning representation of the task

Main challenge is to learn **concepts** that define features (booleans and numericals); e.g. number of free grippers or blocks above A, distances, etc

From pool of concepts that define pool of features, select **minimal subset** that **explain transitions in traces**

System learns QNPs that are translated into FOND problems and solved

QNP Learner (Symbolic Traces)

Input: STRIPS domain D, set $\mathcal{S}$ of **symbolic traces** over D, and bound N for concept complexity

Output: QNP model Q that explain traces

Method:

- Use atom schemas and **fixed concept grammar** to generate **pool of concepts:** all concepts with complexity $\leq N$
- Interpretation of concept C at state s is **subset of objects** $C(s)$; these define features:
  - **boolean feature** p when $|C(s)| \in \{0, 1\}$ for all states s
  - **numerical feature** $n = |C(s)|$ when $|C(s)| > 1$ for at least some state s
- From **pool of features** $\mathcal{F}$, construct SAT theory $T(\mathcal{S}, \mathcal{F})$ that "separates" states and transitions in sample $\mathcal{S}$
- Recover QNP model from solution of SAT theory $T(\mathcal{S}, \mathcal{F})$

SAT Theory for Learning QNPs

Input:

- $\mathcal{S}$ = transitions (s, s') in set of traces
- $\mathcal{F}$ = pool of features f together with interpretations $f(s)$ at each s in $\mathcal{S}$

Propositional variables:

- $selected(f)$ for each f in $\mathcal{F}$ to select $\mathcal{F}$-subset
- $D_1(s, t)$ iff selected features distinguish states s and t in $\mathcal{S}$
- $D_2(s, s', t, t')$ iff selected features distinguish trans. (s, s') and (t, t') in $\mathcal{S}$

Formulas:

- $D_1(s, t)$ (for states s and t such that only one is goal)
- $\bigwedge_{t'} D_2(s, s', t, t') \implies D_1(s, t)$ (for each (s, s') and t in $\mathcal{S}$)
- $D_1(s, t) \iff \bigvee_f selected(f)$ (for f's that distinguish s and t)
- $D_2(s, s', t, t') \iff \bigvee_f selected(f)$ (for f's that dist. (s, s') and (t, t'))

Recovering QNP from Assignment

Selected features define **boolean and numerical features** in the QNP

Each transition (s, s') is mapped into transition (t, t') over selected features, and defines **QNP action** with precondition t and effect given by the change of value across (t, t')

Multiple actions (t, t') are often collapsed into **simpler action** by removing superfluous preconditions and effects

Theorem

$T(\mathcal{S}, \mathcal{F})$ is SAT iff there is **sound QNP** *abstraction relative to $\mathcal{S}$ and $\mathcal{F}$*

Learned QNPs: Gripper

Training set: traces from 2 instances with 4 and 5 balls each

Learned features (selected) from $|\mathcal{S}| = 403$ and $|\mathcal{F}| = 130$:

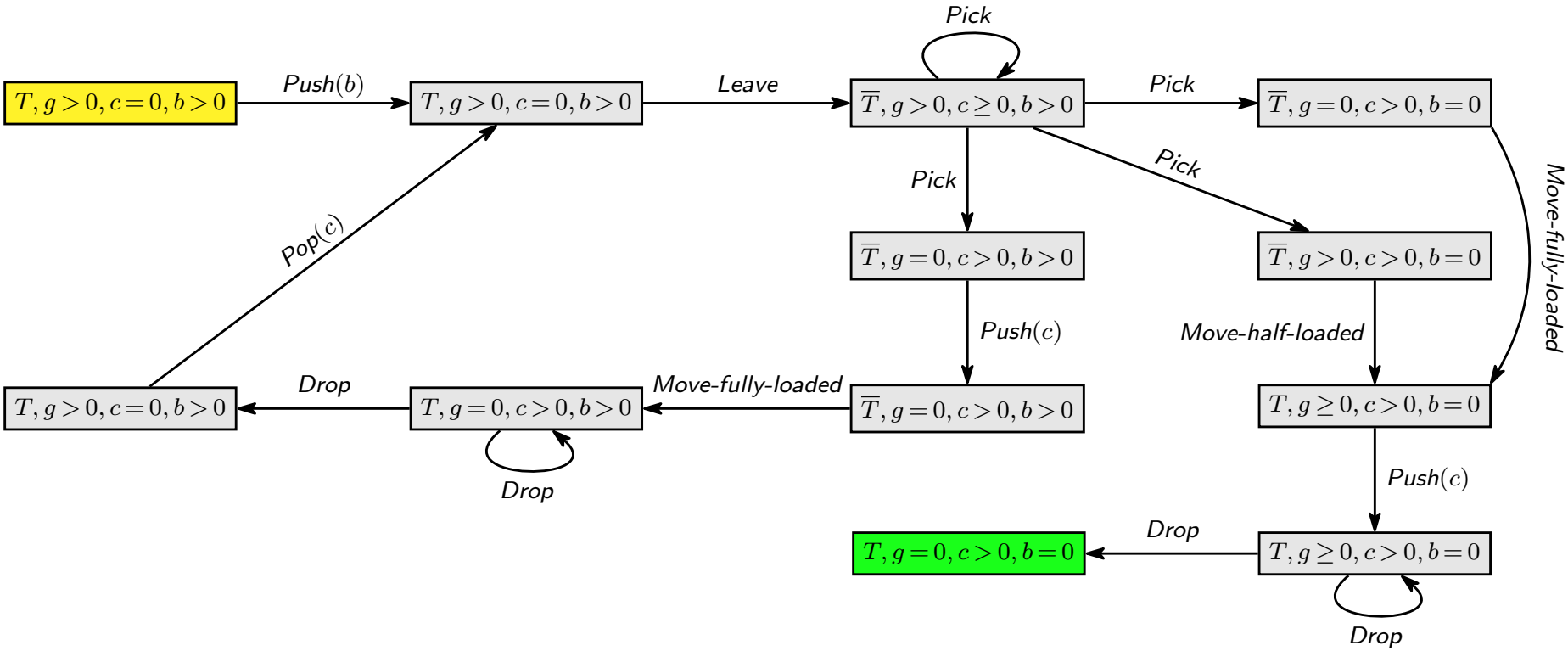
- $T =$ "whether robot is in target room" $= \mathit{at} \sqcap C_A$
- $b =$ "number of balls not in target room" $= |\exists \mathit{at}.\neg C_A|$
- $c =$ "number of balls being held by robot" $= |\exists \mathit{hold}.C_u|$
- $g =$ "number of free grippers (available capacity)" $= |\mathit{empty}|$

Learned abstract actions:

- *Drop* $= \langle T, c > 0; c\downarrow, g\uparrow \rangle$
- *Move-fully-loaded* $= \langle \neg T, c > 0, g = 0; T \rangle$
- *Move-half-loaded* $= \langle \neg T, c > 0, g > 0, b = 0; T \rangle$
- *Pick* $= \langle \neg T, b > 0, g > 0; b\downarrow, g\downarrow, c\uparrow \rangle$
- *Leave* $= \langle T, c = 0, g > 0; \neg T \rangle$

Solution works for any number of balls and grippers!

Example: Solution for (Learned) Gripper



Model Q learned and translated into FOND in less than 1 sec. `FOND-SAT` [Geffner & Geffner, 2018] solves the FOND problem in less than 13 secs after 11 calls to SAT solver

Learned QNPs: Pick Rewards in Grid

Inspired from RL work [Garnelo, Arulkumaran & Shanahan, 2016]

Training set: 2 instances 4×4, 5×5, diff. dist. of blocked cells and rewards

Learned Features (selected) from $|\mathcal{S}| = 568$ and $|\mathcal{F}| = 280$:

– $r =$ "number of remaining rewards" $= |\mathit{reward}|$

– $d =$ "min. dist. to closest reward" $= dist(\mathit{at}, \mathit{adj} : \neg \mathit{blocked}, \mathit{reward})$

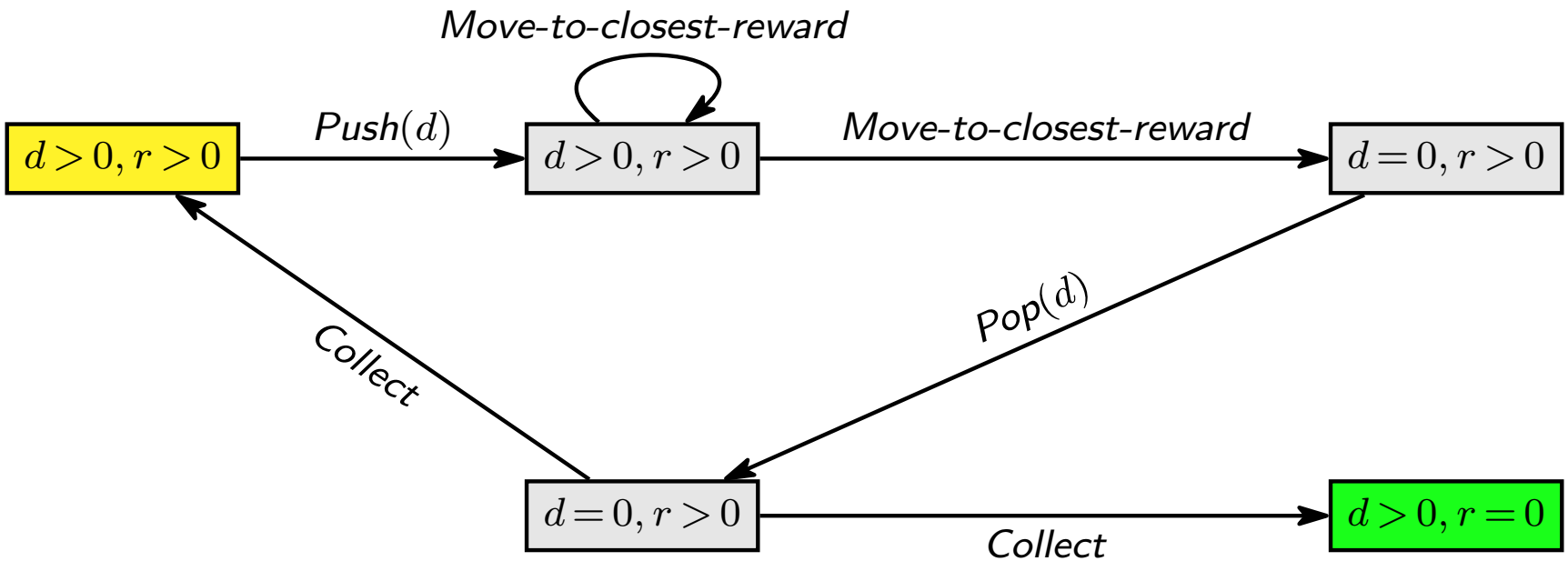
Learned abstract actions:

– *Move-to-closest-reward* $= \langle r > 0, d > 0; d\downarrow \rangle$

– *Collect* $= \langle d = 0, r > 0; r\downarrow, d\uparrow \rangle$

Solution works for any grid dimension, number of rewards, and distribution of blocked cells!

Example: Solution for (Learned) Rewards



Model Q is learned and translated into FOND is less than 1 sec. `FOND-SAT` solves the FOND problem in less than 1 sec after 4 calls to SAT solver

Learning QNPs from Non-Symbolic Traces

Focus on **fully-observable and complete** sets of **non-symbolic** traces

Any planning problem solved with **single numerical feature** n that counts "steps to reach goal" and single QNP action $\langle n > 0; n\downarrow \rangle$

This is valid QNP but:

– this QNP model **lacks structure**

– computing value of n at state s is **intractable** (in general)

To get meaningful models with tractable features, we'll assume the traces are **annotated by teacher**

Goal is to learn QNP model that **explains the teacher**

Annotated Traces

The teacher is responsible for

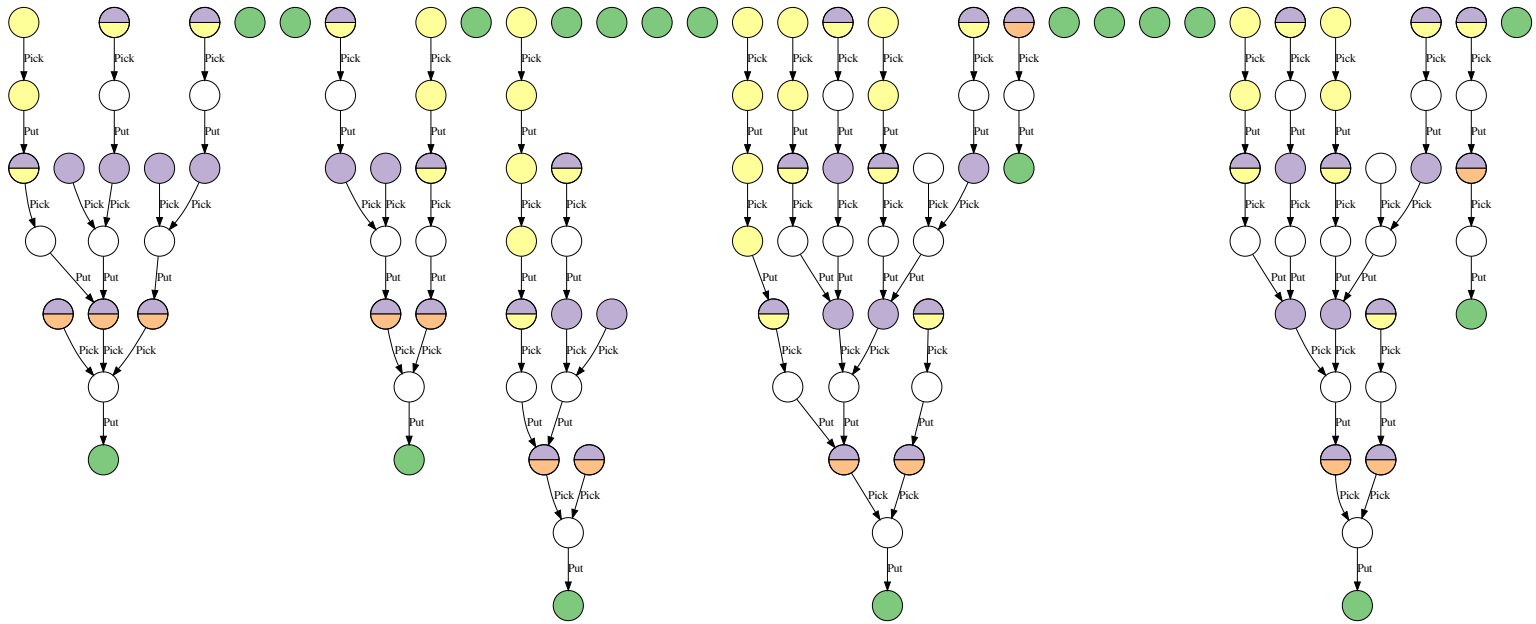
- marking transitions as **good** or **bad** according to his/her preferences
- assigning **labels** to good transitions (possibly single label)
- assigning **colors to states** (disinguishing states, at least goals/non-goals)

The set T of traces then define a graph G_T such that

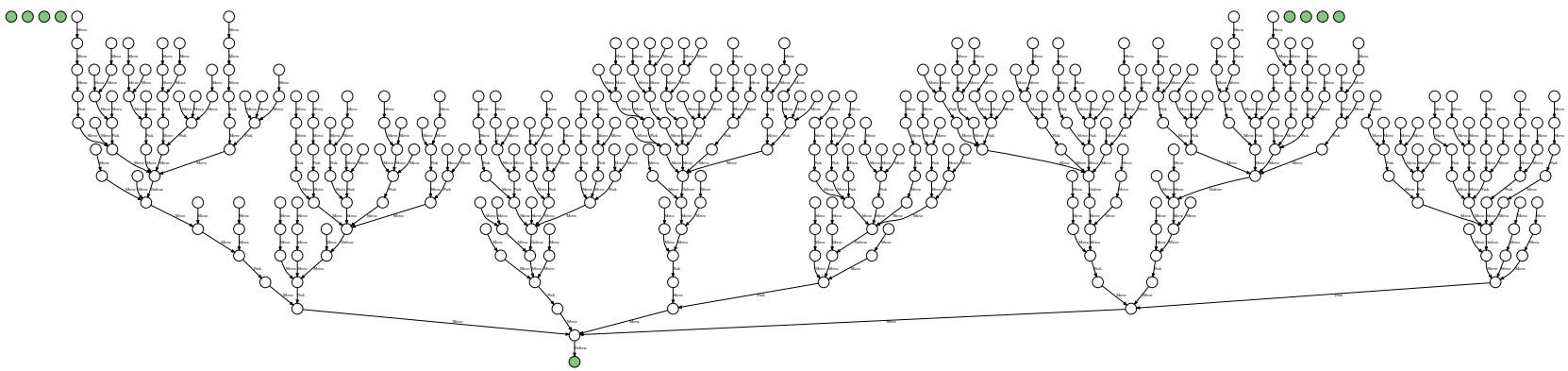
- **subgraph spanned** by good transitions makes up **annotated DAG** where each path leads to a goal state (teacher is responsible for this!)
- bad transitions correspond to **back or cross** edges in DAG
- states are assigned to different colors

Example: Annotated Traces

Blocksworld $on(A, B)$ (4 blocks $=$ 125 states):



Delivery of packages (3×3 and 2 pkgs $=$ 414 states):



Target Class of QNP Models

Policy already hinted by teacher; want **QNP policy** determined by

- set of features (boolean and numerical)
- set of QNP actions
- mapping from abstract states (boolean valuations for features) into actions

Target class of models sliced with vectors α of hyperparameters that tell:

- number of boolean and numerical features
- number of abstract actions

Learning aims at **regular policies** that are guaranteed to **terminate:**

- policy is regular if there is ordering $n_1, \ldots, n_m$ of numerical features such that if some action **increases** n_j, it must **decrease** some n_k with $k > j$

$\mathcal{M}_\alpha$ denotes the (finite) class of QNP models bounded by α

QNP Learner (Non-Symbolic Traces)

Like in task for learning STRIPS models:

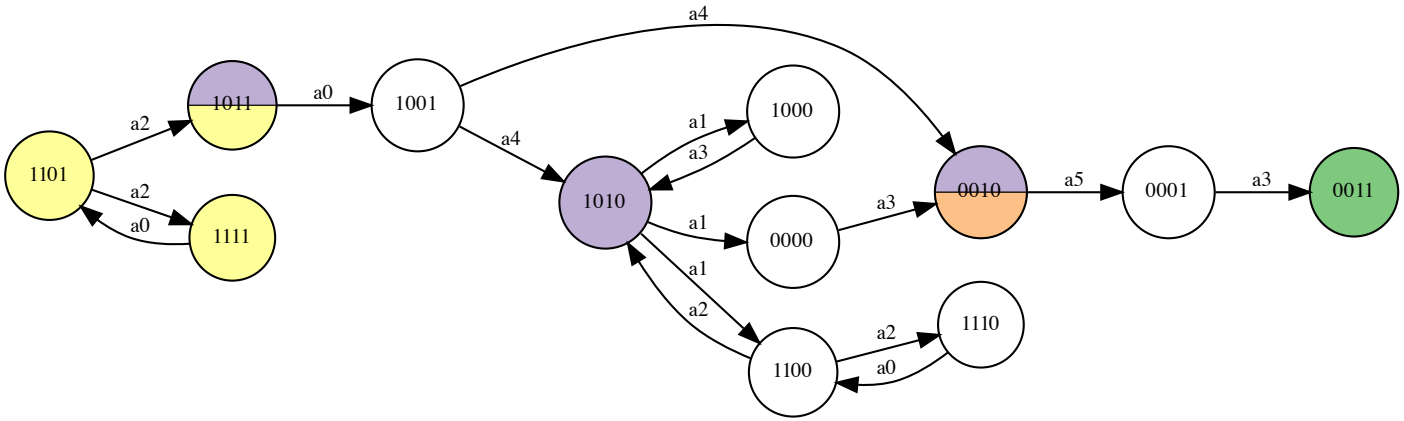
- Input consists of sets $T_1, \ldots, T_n$ of traces defining graphs $G_{T_1}, \ldots, G_{T_n}$
- For vector α of hyperparameters, SAT theory T_α is decomposed as

$$T_\alpha \;=\; T_\alpha^0 \;\cup\; \bigcup_{i=1}^{n} T_\alpha^i$$

  where T_α^0 encodes QNP model and policy, and each T_α^i encodes feature values for states in T_i and conditions to ensure **isomorphism** between model and subtree of **annotated DAG** for G_{T_i}
- Policy encoded in T_α^0 is **strong-cyclic** and **regular**, thus QNP solution
- Abstract states in QNP are colored and bijection from G_{T_i} to QNP model **must respect coloring**

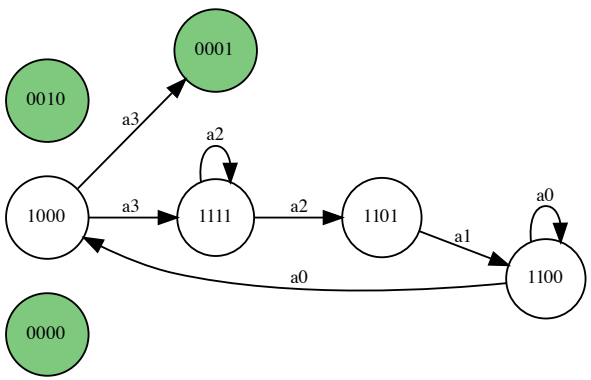
Example: QNP Models and Solutions

Blocksworld $on(A, B)$:




```
Variables: X1 X0 p1 p0
a0/Pick: pre={p1}, eff={-p1}
a1/Pick: pre={p1,X0=0,X1>0}, eff={-p1,reset(X0),dec(X1)}
a2/Put: pre={-p1,X0>0,X1>0}, eff={p1,dec(X0)}
a3/Put: pre={-p1,X0=0}, eff={p1}
a4/Put: pre={p0,-p1,X0=0,X1>0}, eff={-p0,p1,dec(X1)}
a5/Pick: pre={-p0,p1}, eff={p0,-p1}
```

Delivery of packages:




```
Variables: X2 X1 X0 p0
a0/Move: pre={X0=0,X1>0,X2>0}, eff={dec(X1)}
a1/Pick: pre={p0,X2>0}, eff={-p0}
a2/Move: pre={p0,X0>0,X2>0}, eff={dec(X0)}
a3/Deliver: pre={-p0,X0=0,X1=0,X2>0}, eff={p0,reset(X0),reset(X1),dec(X2)}
```

Wrap Up

- Learning planning representations is step towards bridging the gap between model-based solvers and model-free learners
- Learned models are general and can be used for different purposes
- Learning from non-symbolic inputs formulated in terms of crisp and simple principle (graph isomorphism), independent of target class of models
- Learning task reduced to combinatorial task, modeled and solved via SAT
- QNPs for generalized planning can also be learned from symbolic traces and resulting models solved with off-the-shelf FOND planners

What's Not Been Discussed

- **Relax assumptions:** full observability and completeness of traces
- **Target class:** first-order vs. propositional models (e.g., STRIPS vs. Grounded STRIPS or PSVN), other languages beside STRIPS or QNPs
- **Grounding problem;** e.g., how to use learned model in image-based settings where intepretation of atoms is not directly available

Related Work

- Planning/MDP methods that assume symbolic information on the input, either language, objects, number of arguments, etc [Diuk et al., 2008; Yang et al., 2007; Arora et al., 2018; Aineto et al., 2019; Cresswell et al., 2013]
- Inductive logic programming methods [Khardon, 1999; Martin & Geffner, 2004; Fern et al., 2004]
- Learning grounded STRIPS models using autoencoders [Konidaris et al., 2018, Asai, 2019; Asai & Fukunaga, 2018; Asai & Muise, 2020]
- DL of general policies from PDDL models [Toyer et al., 2018; Bueno et al., 2019; Issakkimuthu et al., 2018; Garg et al., 2018]
- Generalized planning and QNPs [Hu & De Giacomo, 2011; Srivastava et al., 2011; B. & Geffner, 2015, 2018, 2020; B. et al., 2017, 2019; Jiménez et al., 2019; Illanes & McIlraith, 2019]
- DRL generate policies without prior symbolic knowledge, but latent repr. aren't general and lack transparency, reusability, and compositionality [Mnih et al., 2015; Groshev et al., 2018; Chevalier-Boisvert, 2019; François-Lavet et al., 2019; Marcus, 2018; Lake & Baroni, 2017; Garnelo et al., 2016]

Current and Future Work

- Learn from **partially observable traces**, where same obs can come from different states
- Apply learned models in **non-symbolic settings:** learn from image-based traces, apply model to image-based setting

Bibliography (1 of 4)

- Aineto, Jiménez, Onaindia & Miquel Ramírez. 2019. Model recognition as planning. In ICAPS, 13-–21.
- Arora, Fiorino, Pellier, Métivier & Pesty. 2018. A review of learning planning action models. The Knowledge Engineering Review 33:e20.
- Asai. 2019. Unsupervised grounding of plannable first-order logic representation from images. In ICAPS, 583–591.
- Asai & Fukunaga. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In AAAI, 6094–6101.
- Asai & Muise. 2020. Learning Neural-Symbolic Descriptive Planning Models via Cube-Space Priors: The Voyage Home (to STRIPS). arXiv 2004:12850.
- Belle & Levesque. 2016. Foundations for generalized planning in unbounded stochastic domains. In KR, pp. 380–389.
- Bonet, De Giacomo, Geffner & Rubin. 2017. Generalized planning: Non-deterministic abstractions and trajectory constraints. In IJCAI, 873–879.
- Bonet, Francès & Geffner. 2019. Learning features and abstract actions for computing generalized plans. In AAAI, 2703–2710.
- Bonet & Geffner. 2015. Policies that generalize: Solving many planning problems with the same policy. In IJCAI, 2798–2804.
- Bonet & Geffner. 2018. Features, projections, and representation change for generalized planning. In IJCAI, 4667–4673.

Bibliography (2 of 4)

- Bonet & Geffner. 2020. Qualitative Numerical Planning: Reductions and Complexity. JAIR. Forthcoming.
- Bueno, de Barros, Mauá & Sanner. 2019. Deep reactive policies for planning in stochastic nonlinear domains. In AAAI, 7530–7537.
- Chevalier-Boisvert, Bahdanau, Lahlou, Willems, Saharia, Nguyen & Bengio. 2019. BabyAI: A platform to study the sample efficiency of grounded language learning. In ICLR.
- Cimatti, Pistore, Roveri & Traverso. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. Artificial Intelligence, 147, 35–84.
- Cresswell, McCluskey & West. 2013. Acquiring planning domain models using LOCM. The Knowledge Engineering Review 28, 195—213.
- Diuk, Cohen & Littman. 2008. An object-oriented representation for efficient reinforcement learning. In ICML, 240–247.
- Espeholt, Soyer, Munos et al. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. arXiv preprint arXiv:1802.01561.
- Fern, Yoon & Givan. 2004. Approximate policy iteration with a policy language bias. In NIPS, 847–854.
- François-Lavet, Bengio, Precup & Pineau. 2019. Combined reinforcement learning via abstract representations. In AAAI, 3582–3589.
- Garg, Bajpai & Mausam. 2020. Symbolic Network: Generalized Neural Policies for Relational MDPs. arXiv:2002.07375.

Bibliography (3 of 4)

- Garnelo, Arulkumaran & Shanahan. 2016. Towards deep symbolic reinforcement learning. arXiv:1609.05518.
- Geffner & Geffner. 2018. Compact policies for fully observable non-deterministic planning as SAT. In ICAPS, 88–96.
- Groshev, Goldstein, Tamar, Srivastava & Abbeel. 2018. Learning generalized reactive policies using deep neural networks. In ICAPS, 408–416.
- Helmert. 2002. Decidability and undecidability results for planning with numerical state variables. In AIPS, 44–53.
- Hu & De Giacomo. 2011. Generalized planning: Synthesizing plans that work for multiple environments. In IJCAI, 918–923.
- Illanes & McIlraith. 2019. Generalized planning via abstraction: Arbitrary numbers of objects. In AAAI, 7610–7618.
- Issakkimuthu, Fern & Tadepalli. 2018. Training deep reactive policies for probabilistic planning problems. In ICAPS, 422–430.
- Jiménez, Segovia-Aguas & Jonsson. 2019. A review of generalized planning. The Knowledge Engineering Review, 34:e5.
- Khardon. 1999. Learning action strategies for planning domains. Artificial Intelligence 113(1-2):125–148.
- Konidaris, Pack Kaelbling & Lozano-Perez. 2018. From skills to symbols: Learning symbolic representations for abstract high-level planning. JAIR 61, 215–289.

Bibliography (4 of 4)

- Lake & Baroni. 2017. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. arXiv:1711.00350.
- Levesque. 2005. Planning with loops. In IJCAI, pp. 509–515.
- Marcus. 2018. Deep learning: A critical appraisal. arXiv:1801.00631.
- Martín & Geffner. 2004. Learning generalized policies from planning examples using concept languages. Applied Intelligence 20(1):9–19.
- Mnih et al. 2015. Human-level control through deep reinforcement learning. Nature, 518(7540).
- Srivastava, Zilberstein, Immerman & Geffner. 2011. Qualitative numeric planning. In AAAI.
- Toyer, Trevizan, Thiébaux & Xie. 2018. Action schema networks: Generalised policies with deep learning. In AAAI.
- Yang, Wu & Jiang. 2007. Learning action models from plan examples using weighted max-sat. Artificial Intelligence, 171(2-3), 107–143.