IJCAI 2023 Tutorial
Integrated Task and Motion Planning
From Foundations to Research Frontiers
19th August, 2023

Naman Shah
Arizona State University
Sarah Keren
Technion
Siddharth Srivastava
Arizona State University
Outline

1. Background: Why Task and Motion Planning?
2. Abstraction as a Foundation for TMP
3. Modern Abstraction-Based Approaches
4. Research Frontier: Neuro-Symbolic Abstraction Learning for TMP
Fundamental Problem: Long-Horizon Planning

$S$, set of states
$A$, set of actions

$T: S \times A \rightarrow \mu S$, action transition function

$R: S \times A \rightarrow \mathbb{R}$, costs and utility of states, actions (can express goals and some forms of preferences*)

Automated Planning/Sequential Decision Making:

What should the robot do to maximize $R$ (achieve goal) over multiple time steps?
Task and Motion Planning: Longer Horizons, Uncertainty

Formulation as SDM problems: $S = ? A = ?$

[Srivastava, Gupta, Zilberstein, Abbeel and Russell, AAAI 2015]

[Shah, Vasudevan, Kumar, Srivastava, ICRA 2020]
Configuration Space (C-Space): State Space for a Robot in an Environment

Configuration: A complete specification of the position of every point in the system

C-Space: Space of all possible system configurations

2 - Dim

6 - Dim

20 - Dim
Configuration Space (C-Space)

Workspace

(2 DOF: translation only, no rotation)

Configuration Space

free space
obstacles

[Source: CS287, Pieter Abbeel, UC Berkeley]
Configuration Space (C-Space)

Obstacles reduce free space

Computing obstacle boundaries in C-Space provably exponential

[Source: CS287, Pieter Abbeel, UC Berkeley]
Configuration Space (C-Space)

Motion Planning Problem in C-Space $X$

Given $x_i, x_g \in X$
compute a path from $x_i$ to $x_g$

Path: continuous function $\tau: [0,1] \rightarrow X$

s.t. $\tau(0) = x_i$ and $\tau(1) = x_g$ and $\forall x \tau(x)$ not in $X_{obs}$

Obstacles reduce free space

Computing obstacle boundaries in C-Space provably exponential

[Source: CS287, Pieter Abbeel, UC Berkeley]
Examples

PR2: Two 8 DoF arms + 1 DoF height + 3 DoF base

YuMi: Two 7-DoF arms

Formulation as Motion Planning Problems:
State (Config) Space $X = \text{?}$

$x_i, x_g = \text{?}$

What happens to the C-space when the robot picks up a plank?
Pure Motion Planning is Not Enough!

- Motion planning – which path (of waypoints) should the robot take?
  - But which motion planning problem should it solve? different pickups ⇒ different c-spaces
  - Where would motion planning goals in each C-space come from?
  - Clearly, motion planning is not enough
Would Pure Task-Planning Do the Trick?

Can a “higher-level” planner help us compute the strategy?

Then we could refine each action in the plan into a motion plan

Higher-level planning is typically done over states described using features, or properties

E.g., #clothes on table

IsHolding(robot, basket)

... 

+ 

Actions describing how and when robot can change these properties
Would Pure Task-Planning Do the Trick?

Example of a high-level action:

`:action pickup
 :parameters (?obj ?gripper)
 :precondition (and (empty ?gripper)
 (ontable ?obj))
 :effect (and (not (empty ?gripper))
 (not (ontable ?obj)
 (in ?obj, ?gripper))))

SDM problem: which sequence of actions will lead to the goal?
The Shakey Robot

STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving

Richard E. Fikes
Nils J. Nilsson
Stanford Research Institute, Menlo Park, California

Recommended by B. Raphael

How do These High-Level Actions Connect to the C-Space?

Temporal Abstraction ≡ Abstract Actions, Macros, Options...

State Abstraction

GoTo(l)
Pickup(x)
PutDown(x)

At(x, l)
InGripper(x)
AtDestination(x)
How do These High-Level Actions Connect to the C-Space?

GoTo(l)
Pickup(x)
PutDown(x)

Temporal Abstraction ≡ Abstract Actions, Macros, Options…

At(x, l)
InGripper(x)
AtDestination(x)

State Abstraction
Sometimes Intuitive Abstractions are ... Perfect

Pickup(x)
PutDown(x)

OnTable(3)
On(1,2)
...

[action pickup
  :parameters (?obj ?gripper)
  :precondition (and (empty ?gripper)
                     (ontable ?obj) (clear ?obj))
  :effect (and (not (empty ?gripper))
            (not (ontable ?obj)
             (in ?obj, ?gripper) (not (clear ?obj)) ))]]
But Human Intuition has its Limits

Can pickup only from the side
Obstructions depend on choice of movement trajectory

Pickup the red can!

prevailing abstraction

Pickup(x)  Abstract
PutDown(x)  Actions
OnTable(3)  Abstract
On(1,2)     State
OnTable(2)
OnTable(1)
...

[action pickup
 :parameters (?obj ?gripper)
 :precondition (and (empty ?gripper)
        (onTable ?obj) (clear ?obj)
 :effect (and (not (empty ?gripper))
        (not (onTable ?obj)
        (in ?obj, ?gripper) (not (clear ?obj)]]]

[Srivastava et al., ICRA 2014; Srivastava et al., AAAI 2016]
But Human Intuition has its Limits

Can pickup only from the side
Obstructions depend on choice of movement trajectory

Abstract model *thinks* this is a trivial problem
Solutions from abstract model: Mostly infeasible

[Srivastava et al., ICRA 2014; Srivastava et al., AAAI 2016]
But Human Intuition has its Limits

Can pickup only from the side
Obstructions depend on choice of movement trajectory

Task Planning is Not Enough!

Task Planning followed by motion planning is also not enough!

[Srivastava et al., ICRA 2014; Srivastava et al., AAAI 2016]
Summary: We Need to Integrate Task and Motion Planning!

- Task planning – given a task planning problem, computes the high-level action the robot should perform at each step
  - But that action may have no feasible motion plan (recall: cluttered table)
- Motion planning – given a motion planning problem, computes the path that the robot should take
  - But which motion planning problem should it solve?
(Trajectory planning – selects the control inputs that should go to the robot’s motors)
Summary: We Need to Integrate Task and Motion Planning!

- Task planning – given a task planning problem, computes the high-level action the robot should perform at each step
  - But that action may have no feasible motion plan (recall: cluttered table)
- Motion planning – given a motion planning problem, computes the path that the robot should take
  - But which motion planning problem should it solve?
- (Trajectory planning – selects the control inputs that should go to the robot’s motors?)

Technical Problems That Characterize Integrated Task and Motion Planning

High-level (abstract) models are imprecise! They scale to long horizons at the expense of low-level constraints
- Which HL action will have a feasible motion plan at a point in time?

Each HL action (e.g., pickup) defines uncountably infinite Motion Planning Problems!
- Which MP should be solved?

Formalized later in the tutorial
Outline

1. Background: Why Task and Motion Planning?
   Task Planning is Not Enough
   Motion Planning is Not Enough
   Foundations of Motion Planning

2. Abstraction as a Foundation for TMP

3. Abstraction-based Approaches

4. Research Frontier: Neuro-Symbolic Abstraction Learning for TMP
Recall: Configuration Space (C-Space)

Obstacles reduce free space

Computing obstacle boundaries in C-Space provably exponential

[Source: CS287, Pieter Abbeel, UC Berkeley]
Sampling-based Motion Planning

- Sampling-based solutions sample the C-Space instead of explicitly computing it
  - Probabilistic Roadmap (PRM)
  - Rapidly-exploring random tree (RRT)
- Simply need to know if the robot is in collision (workspace query)
Probabilistic Roadmaps (PRM)

Forbidden space

Free/feasible space

[Source: CS287, Pieter Abbeel, UC Berkeley]
Probabilistic Roadmaps (PRM)

Configurations are sampled by picking coordinates at random.
Probabilistic Roadmaps (PRM)

Configurations are sampled by picking coordinates at random

[Source: CS287, Pieter Abbeel, UC Berkeley]
Probabilistic Roadmaps (PRM)

Configurations are sampled by picking coordinates at random
Probabilistic Roadmaps (PRM)

Sampled configurations are tested for collisions

[Source: CS287, Pieter Abbeel, UC Berkeley]
Probabilistic Roadmaps (PRM)

Each milestone is linked to its nearest neighbors by straight paths

[Source: CS287, Pieter Abbeel, UC Berkeley]
Probabilistic Roadmaps (PRM)

PRM is searched for a path from start (s) to goal (g)

[Source: CS287, Pieter Abbeel, UC Berkeley]
Probabilistic Roadmaps (PRM)

Collision-free edges are retained as local paths to form PRM

[Source: CS287, Pieter Abbeel, UC Berkeley]
Probabilistic Roadmaps (PRM)

Start and goal configurations are included as milestones.

[Source: CS287, Pieter Abbeel, UC Berkeley]
Probabilistic Roadmaps (PRM)

Collision-free edges are retained as local paths to form PRM

[Source: CS287, Pieter Abbeel, UC Berkeley]
Probabilistic Roadmaps (PRM)

• Recap
  • Randomly sample configurations from C-Space
  • Connect them to nearest neighbors (if no collisions with obstacles)
  • Two primitive procedures (workspace collision query):
    • Check if configuration is in free space
    • Check if an edge is in free space
  • PRM can be used for multiple queries

How is this different from vanilla search problems?
Probabilistic Roadmaps (PRM)

• Recap
  • Randomly sample configurations from C-Space
  • Connect them to nearest neighbors (if no collisions with obstacles)
  • Two primitive procedures (workspace collision query):
    • Check if configuration is in free space
    • Check if an edge is in free space
  • PRM can be used for multiple queries

How is this different from vanilla search problems?

- Uncountably infinite state space (hence the need for sampling)
- Connectivity needs to be computed on the fly
+ Known metric as a starting point
Rapidly-Exploring Random Trees (RRT)

Data structure: $T = (\text{nodes } V, \text{edges } E)$
Rapidly-Exploring Random Trees (RRT)

1. \( x \leftarrow \text{Sample()} \)
2. \( v \leftarrow \text{Nearest}(T, x) \)
3. \( v' \leftarrow \text{Extend}(v, x) \)
4. If \( \text{ObstacleFree}(v, v') \) then
5. \( V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\} \)

// Sample configuration from C-space
// Find nearest node in the tree to sample
// Try extension nearest node towards sample
// If extension does not collide with obstacles
// Add new node and edge to the tree

[Source: CS287, Pieter Abbeel, UC Berkeley]
Rapidly-Exploring Random Trees (RRT)

1. \( x \leftarrow \text{Sample}() \)
2. \( v \leftarrow \text{Nearest}(T,x) \)
3. \( v' \leftarrow \text{Extend}(v,x) \)
4. If \( \text{ObstacleFree}(v, v') \) then
5. \( V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\} \)

// Sample configuration from C-space
// Find nearest node in the tree to sample
// Try extension nearest node towards sample
// If extension does not collide with obstacles
// Add new node and edge to the tree

[Source: CS287, Pieter Abbeel, UC Berkeley]
Rapidly-Exploring Random Trees (RRT)

1. $x \leftarrow \text{Sample}()$
2. $v \leftarrow \text{Nearest}(T, x)$
3. $v' \leftarrow \text{Extend}(v, x)$
4. If $\text{ObstacleFree}(v, v')$ then
5. \hspace{1cm} $V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\}$

// Sample configuration from C-space
// Find nearest node in the tree to sample
// Try extension nearest node towards sample
// If extension does not collide with obstacles
// Add new node and edge to the tree
Rapidly-Exploring Random Trees (RRT)

1. \( x \leftarrow \text{Sample()} \) 
2. \( v \leftarrow \text{Nearest}(T, x) \)
3. \( v' \leftarrow \text{Extend}(v, x) \)
4. If \( \text{ObstacleFree}(v, v') \) then
5. \( V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\} \)

// Sample configuration from C-space
// Find nearest node in the tree to sample
// Try extension nearest node towards sample
// If extension does not collide with obstacles
// Add new node and edge to the tree

[Source: CS287, Pieter Abbeel, UC Berkeley]
Rapidly-Exploring Random Trees (RRT)

1. \( x \leftarrow \text{Sample}() \)
2. \( v \leftarrow \text{Nearest}(T, x) \)
3. \( v' \leftarrow \text{Extend}(v, x) \)
4. If \( \text{ObstacleFree}(v, v') \) then
5. \( V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\} \)

// Sample configuration from C-space
// Find nearest node in the tree to sample
// Try extension nearest node towards sample
// If extension does not collide with obstacles
// Add new node and edge to the tree

[Source: CS287, Pieter Abbeel, UC Berkeley]
Rapidly-Exploring Random Trees (RRT)

1. $x \gets \text{Sample()}$
2. $v \gets \text{Nearest}(T, x)$
3. $v' \gets \text{Extend}(v, x)$
4. If $\text{ObstacleFree}(v, v')$ then
5. $V \gets V \cup \{v'\}; E \gets E \cup \{(v, v')\}$

// Sample configuration from C-space
// Find nearest node in the tree to sample
// Try extension nearest node towards sample
// If extension does not collide with obstacles
// Add new node and edge to the tree
Early Approaches
Early Characterization of Key Challenges: aSyMov

- Approach built upon PRMs
- *Articulated key technical problem:* picking and placing objects are discrete events, change the “robot”, c-space
  - Picking an object leads to a new, composed robot
- Different composed versions of the robot have different C-spaces
- Solution Idea:
  - Maintain separate (projected) PRMs for different versions of the robot
  - Link them up based on actions such as pick and place
- Motion planning after a pick-up would use the PRM for the composed robot
- Where does task planning come in?
  - Goal specified in PDDL-like language
  - Task plan is used as a heuristic in PRM expansion

aSyMov: Example Problem

- PRM for robot + box is disconnected
  - Components correspond to different grasps

- PRM for robot alone is also disconnected.
  - Components correspond to different box positions

- Need a path through linked points

- Grow a PRM per action; bias expansion by using high-level plan cost as a heuristic for c-states

- High-level model may be abstract (inaccurate) but used in a limited manner – not updated
## High-Level Summary

<table>
<thead>
<tr>
<th>Search Space</th>
<th>High Level Reasoning</th>
<th>Low Level Reasoning</th>
<th>High-level Reasoning</th>
<th>High Level Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>aSyMov</td>
<td>Single</td>
<td>Any TP</td>
<td>PRM/RRT</td>
<td>Symbolic</td>
</tr>
</tbody>
</table>
Outline

1. Background: Why Task and Motion Planning?
2. Abstraction as a Foundation for TMP
3. Modern Abstraction-Based Approaches
4. Research Frontier: Neuro-Symbolic Abstraction Learning for TMP
Recall Problem Hierarchies

• Task planning – given a task planning problem, computes the high-level action the robot should perform at each step
  • Long horizon: each action takes usually a few minutes to complete.
• Motion planning – given a motion planning problem, computes the path that the robot should take
  • short horizon: each waypoint usually takes a few seconds to be achieved
• (Trajectory planning – selects the control inputs that should go to the robot’s motors)
  • extremely short horizon: runs ~50 Hz
Can we better utilize this hierarchy?
Can we somehow exploit this hierarchy? 
Ans: Yes – and the key concept is “abstractions”
(:action Move
  :parameter ?robot ?location ?trajectory
  :precondition
    not At(?robot ?location)
    Collision-free(?trajectory)
  :effect
    At(?robot ?location)
)
Domain for Robot Planning

(:action Move
 :parameter ?robot ?location ?trajectory
 :precondition
   not At(?robot ?location)
   Collision-free(?trajectory)
 :effect
   At(?robot ?location)
 )

Key challenges: infinitely many facts, infinite branching factor
Abstraction: State Abstraction

E.g., $\text{At}(O,\text{Init})$ is True iff $\exists l \in \text{BlueArea} \text{ s.t. } \text{pose}(o,l) = \text{True}$. 
Abstraction: State Abstraction

First-order logic queries from concrete vocabulary $V_l$ to abstract vocabulary $V_h$ where $V_h \subset V_l$.

The query $[r]_{sh}(\bar{o}_1, ..., \bar{o}_n) = True$ iff 

$\exists o_1, ..., o_n$ such that $o_i \in \rho(\bar{o}_i)$ and 

$[\varphi^\rho_r(o_1, ..., o_n)]_{sl} = True$.

E.g., $At(O,Init)$ is True iff $\exists l \in BlueArea$ s.t. $pose(o,l) = True$.

Here, $\rho(Init) = BlueArea = \{p_1, ..., p_n\}$
Abstraction: Symbolic Actions

First-order logic queries from concrete vocabulary $V_l$ to abstract vocabulary $V_h$ where $V_h \subset V_l$.

The query $[r]_{sh}(\bar{o}_1, ..., \bar{o}_n) = True$ iff

$$\exists o_1, ..., o_n \text{ such that } o_i \in \rho(\bar{o}_i) \text{ and } [\varphi^{\rho}_r(o_1, ..., o_n)]_{S_l} = True.$$
Abstraction: State Abstraction

(Move R Init Traj1)
Abstraction: State Abstraction

(Move R Init Traj1)
Abstraction: Refining Symbolic Action

(Move R Init Traj1)
Abstraction: Refining Symbolic Action

(Move R Init Traj1)
Challenge 1 – Which MP to Solve?

(Move R Init Traj1)

Sampler

[0.25,...0.14]

Infinite samples!
Challenge 1 – Which MP to Solve?

C1: Each high-level action defines infinitely many MP problems!
Challenge 2 – Which MP Will be Solvable?

(:action Move
 :parameter ?robot ?location ?trajectory
 :precondition
   not At(?robot ?location)
   Collision-free(?trajectory)
 :effect
   At(?robot ?location)
)

Can pickup only from the side
Obstructions depend on choice of movement trajectory

No way to verify which trajectories are collision-free
Challenge 2 – Which MP Will be Solvable?

(:action Move
 :parameter ?robot ?location ?trajectory
 :precondition
   not At(?robot ?location)
   Collision-free(?trajectory)
 :effect
   At(?robot ?location)
)

Can pickup only from the side
Obstructions depend on choice of movement trajectory

No way to verify which trajectories are collision-free

C2: Loss of information in abstraction
Actions in plan that can not refinable
Task and Motion Planning Problem

$\mathcal{O}$, universe of objects and object poses (implicitly defined)

$\mathcal{P} = \mathcal{P}_\text{sym} \cup \mathcal{P}_h$, set of symbolic predicates and interpretations (definitions in geometric constraints)

Generates $\mathcal{S}$, set of abstract states

$\mathcal{A} = \mathcal{A}_\text{sym} \cup \mathcal{A}_h$, set of abstract actions

$T: \mathcal{S} \times \mathcal{A} \rightarrow \mu\mathcal{S}$, action transition function

$R: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, costs and utility of states, actions (can express goals and some forms of preferences*)

$\gamma$, a concretization function (often in the form of samplers or generators)

Task and motion planning:

Compute a sequence of actions from $\mathcal{A}$ that maximizes the utility $R$ and that can be executed in the given model.
Optimization and SMT Based Approaches
IDTMP

- Core idea: Use advances in SAT/SMT solvers to perform hybrid search for TMP
- State variables for different objects and use poses as values
- Convert each high-level action to low-level motion planning problem.


IDTMP: SMT Planning

- SMT = satisfiability module theories
- Model the problem as Boolean satisfiability problem
IDTMP: SMT Planning

- SMT = satisfiability module theories
- Model the problem as Boolean satisfiability problem

State variables:
- Pose of the object $P_0 \in \mathbb{R}^2$
- Configuration of the robot $C_R \in \mathcal{X}$

Actions:
- Pick
- Place
- Move

Diagram:
- Robot $R$
- Goal Area
- Object $O$
- Init Area
IDTMP: SMT Planning

State variables:
- Pose of the object $P_0 \in \mathbb{R}^2$
- Configuration of the robot $C_R \in \mathcal{X}$

$P = \{p_0, ..., p_n\}$ (a set of state variables)

$A = \{a_0, ..., a_k\}$ (a set of actions)
IDTMP: SMT Planning

State variables:
- Pose of the object $P_0 \in R^2$
- Configuration of the robot $C_R \in \mathcal{X}$

Actions:
- Pick
- Place
- Move

$P = \{p_0, ..., p_n\}$ (a set of state variables)

$A = \{a_0, ..., a_k\}$ (a set of actions)

Three constraints:

1. $a_i^k \Rightarrow Pre(a_i)^k \land Eff(a_i)^{k+1}$  // If an action is taken at the step $k$, the its precondition and effect must hold
IDTMP: SMT Planning

State variables:
Pose of the object $P_0 \in R^2$
Configuration of the robot $C_R \in \mathcal{X}$

Actions:
Pick
Place
Move

$P = \{p_0, ..., p_n\}$ (a set of state variables)
$A = \{a_0, ..., a_k\}$ (a set of actions)

Three constraints:
1. $a_i^k \Rightarrow Pre(a_i)^k \land Eff(a_i)^{k+1}$  // If an action is taken at the step $k$, the its precondition and effect must hold
2. $(p_i^k = p_i^{k+1}) \lor (a_j^k \lor ... \lor a_i^k)$  // Variables that are not changed by the actions remains unchanged
IDTMP: SMT Planning

State variables:
Pose of the object $P_0 \in \mathbb{R}^2$
Configuration of the robot $C_R \in \mathcal{X}$

Actions:
Pick
Place
Move

$P = \{p_0, ..., p_n\}$ (a set of state variables)
$A = \{a_0, ..., a_k\}$ (a set of actions)

Three constraints:
1. $a^k_i \Rightarrow \text{Pre}(a_i)^k \land \text{Eff}(a_i)^{k+1}$  // If an action is taken at the step $k$, the its precondition and effect must hold
2. $(p^k_i = p^{k+1}_i) \lor (a^k_j \lor \ldots \lor a^k_i)$  // Variables that are not changed by the actions remains unchanged
3. $a^k_i \Rightarrow \neg (a^k_0 \lor \ldots \lor a^k_{i-1} \lor \ldots \lor a^k_{i+1} \lor \ldots a^k_i)$  // Only one action can be taken at the given step
IDTMP: SMT Planning

State variables:
- Pose of the object $P_0 \in R^2$
- Configuration of the robot $C_R \in \mathcal{X}$

Actions:
- Pick
- Place
- Move

$P = \{p_0, ..., p_n\}$ (a set of state variables)
$A = \{a_0, ..., a_k\}$ (a set of actions)

Three constraints:
1. $a_i^k \Rightarrow Pre(a_i)^k \land Eff(a_i)^{k+1}$  // If an action is taken at the step $k$, the its precondition and effect must hold
2. $(p_i^k = p_i^{k+1}) \lor (a_j^k \lor \ldots \lor a_i^k)$  // Variables that are not changed by the actions remains unchanged
3. $a_i^k \Rightarrow \neg (a_0^k \lor \ldots \lor a_{i-1}^k \lor \ldots \lor a_{i+1}^k \lor \ldots a_i^k)$  // Only one action can be taken at the given step

SMT formula using $p_i$ and $a_i$  // $p_i \in P$ (a set of state variables) and $a_i \in A$ (a set of actions)
IDTMP: SMT Planning

State variables:
Pose of the object $P_0 \in \mathbb{R}^2$
Configuration of the robot $C_R \in \mathcal{X}$

Actions:
Pick
Place
Move

$P = \{p_0, ..., p_n\}$ (a set of state variables)
$A = \{a_0, ..., a_k\}$ (a set of actions)

Three constraints:

1. $a_i^k \implies \text{Pre}(a_i)^k \land \text{Eff}(a_i)^{k+1}$ // If an action is taken at the step $k$, the its precondition and effect must hold
2. $(p_i^k = p_i^{k+1}) \lor (a_j^k \lor \ldots \lor a_i^k)$ // Variables that are not changed by the actions remains unchanged
3. $a_i^k \implies \neg (a_0^k \lor \ldots \lor a_{i-1}^k \lor \ldots \lor a_{i+1}^k \lor \ldots \lor a_k^k)$ // Only one action can be taken at the given step

SMT formula using $p_i$ and $a_i$ // $p_i \in P$ (a set of state variables) and $a_i \in A$ (a set of actions)

SMT used
1) solve SMT formula consisting of continuous variables and constraints
2) maintain dynamic constraints such as action $a_i$ is not applicable at step $t$
IDTMP: Overall Approach

- Plan_length = 1
- Constraints = initial constraints
- Compute task plan for the current plan length and constraints
IDTMP: Overall Approach

- Plan_length = 1
- Constraints = initial constraints
- Compute task plan for the current plan length and constraints
- If no task plan found: plan_length + 1 and retry
IDTMP: Overall Approach

- Plan_length = 1
- Constraints = initial constraints
- Compute task plan for the current plan length and constraints
- If no task plan found: plan_length + 1 and retry
- If a task plan is found:
  - For every action in task plan:
    - Compute a motion plan
IDTMP: Overall Approach

- Plan_length = 1
- Constraints = initial constraints
- Compute task plan for the current plan length and constraints
- If no task plan found: plan_length + 1 and retry
- If a task plan is found:
  - For every action in task plan:
    - Compute a motion plan
      - If no motion plan:
        - Add new constraints
IDTMP: Overall Approach

- Plan_length = 1
- Constraints = initial constraints
- Compute task plan for the current plan length and constraints
- If no task plan found: plan_length + 1 and retry
- If a task plan is found:
  - For every action in task plan:
    - Compute a motion plan
    - If no motion plan:
      - Add new constraints

If $a_i^k$ does not have a motion plan $\Rightarrow$ disallow $a_i$ at step k.
Incremental Task and Motion Planning: A Constraint-Based Approach

Neil T. Dantam, Zachary K. Kingston, Swarat Chaudhuri, and Lydia E. Kavraki

January 2016
IDTMP: Summary

• Inputs
  • SMT domain with hybrid (continuous and symbolic) variable and actions

• Properties
  • Probabilistically complete – if the low-level motion planner is probabilistically complete.

• How does it handle C1: SMTs reasoning for instantiating continuous variables
• How does it handle C2: SMTs reasoning for instantiations of discrete variables and adding new constraints
## High-Level Summary

<table>
<thead>
<tr>
<th></th>
<th>Search Space</th>
<th>High Level Planner</th>
<th>Low Level Reasoning</th>
<th>High-level</th>
<th>High Level Language</th>
<th>P1: Infinite Motion Plans</th>
<th>P2: Downward Refinability</th>
</tr>
</thead>
<tbody>
<tr>
<td>aSyMov</td>
<td>Single</td>
<td>Any TP</td>
<td>PRM/RRT</td>
<td>Symbolic</td>
<td>PDDL</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>IDTMP</td>
<td>Dual</td>
<td>SMT</td>
<td>Any MP</td>
<td>Variables with continuous domains</td>
<td>SAS</td>
<td>SMT</td>
<td>SMT</td>
</tr>
</tbody>
</table>

- **IDTMP**
  - **Dual**
  - **SMT**
  - **Any MP**
  - **Variables with continuous domains**
  - **SAS**
  - **SMT**
  - **SMT**
Outline

1. Background: Why Task and Motion Planning?
2. Abstraction as a Foundation for TMP
3. Abstraction-based Approaches
4. Research Frontier: Neuro-Symbolic Learning for TMP
Hierarchical Planning in the Now: HPN

Central Idea:

• Don’t abstract the state; plan over fluents & actions with continuous arguments
• For high-level reasoning: Use regression of geometric fluents
• For efficiency: Use an operator-abstraction hierarchy
• For refinement: Interleave refinement with execution
• State represented by a set of fluents with possibly continuous arguments
  • Subroutines used to dynamically evaluate fluents
  • No need to represent complete state descriptions
• Examples of fluents with continuous arguments (1-d):
  • $\text{In}(o, r)$: object $o$ is completely inside region $r$
  • $\text{ObjLoc}(o, l)$: left edge of object $o$ is at location $l$
  • $\text{ClearX}(r, x)$: only objects $\in x$ possibly overlap with region $r$
• Actions:
  • $\text{Place}(o, l_{\text{target}})$ causes $\text{ObjLoc}(o, l_{\text{target}})$;
    • requires $\text{ClearX}(\text{sweptVol}(o, l_{\text{init}}, l_{\text{target}}))$
  • $\text{In}(o, r)$: ramification action for the fluent $\text{In}(o, r)$
  • $\text{Clear}(r, x)$: ramification action for the fluent $\text{Clear}(r, x)$
HPN: Action Specifications

• Action specification without hierarchy:
  • Arguments include current subgoal (maintained during regression)
  • Effects, preconditions including argument-choice constraints

PickPlace((o, l_{target}), s_{now}, γ):

  effect: ObjLoc(o, l_{target})
  choose: l_{start} ∈ \{s_{now}[o].loc\} ∪
          generateLocsInRegions((o,\{warehouse, stove, sink\}),s_{now},γ)

  //operator instantiations have to be considered for each generated value of l_{start}
  //Allows l_{start} to be generated using γ for efficient regression

  pre: ObjLoc(o, l_{start}), ClearX(sweptVol(o, l_{start},l_{target}),\{o\})

• Precondition + choose essentially generates the next subgoal during regression
• Use operator-specific regression subroutines for geometric fluents (provided as input)
HPN: Regression Algorithm

- Carry out goal regression using goal state, preimage computation methods for each operator, geometric fluent
- Search using A*; heuristic = number of goal fluents that are not true in the preimage

Goals
Operations
Ramification actions
Primitive actions

Pick(o,s,γ):
  effect: Holding(o)
  pre: AtGraspPose(o) [1]
  prim: prim_pick

Place(o,s,γ):
  effect: Holding(o) [1]
  pre: ¬Holding(o)
  prim: prim_place

Move(o,l_{end},s,γ):
  choose: l_{start}
  effect: ObjectLoc(l_{end})
  pre: ObjectLoc(l_{start}) [1]
  Holding(o) [2]
  ClearX(sweptVol(o, l_{start}, l_{end}, {o}) [3]
  prim: prim_move
HPN: Hierarchical Action Representation

- Use a hierarchy defined using precondition postponement to reduce the horizon during regression
- Suppose operator \( o \) has preconditions \( p_1, \ldots, p_n \), effect \( r \)
- Operator with \( p_n \) postponed
  - \( o_{\text{postponed}} \):
    - \( \text{precon} = p_1, \ldots, p_{n-1} \)
    - \( \text{expansion:} \)
      - Achieve \( p_n \) while maintaining \( p_1, \ldots, p_{n-1} \);
      - Then execute \( o \)
- Additional side-effects of achieving \( p_n \) may have to be declared
- Define hierarchy by associating abstraction-level with each precondition

\[
\text{Pick}(o,s,\gamma):
\begin{align*}
\text{effect: } & \text{Holding}(o) \\
\text{pre: } & \text{AtGraspPose}(o) [1] \\
\text{prim: } & \text{prim_pick}
\end{align*}
\]

\[
\text{Place}(o,s,\gamma):
\begin{align*}
\text{effect: } & \text{Holding}(o) [1] \\
\text{pre: } & \neg\text{Holding}(o) \\
\text{prim: } & \text{prim_place}
\end{align*}
\]

\[
\text{Move}(o,l_{\text{end}},s,\gamma):
\begin{align*}
\text{choose: } & l_{\text{start}} \\
\text{effect: } & \text{ObjectLoc}(l_{\text{end}}) \\
\text{pre: } & \text{ObjectLoc}(l_{\text{start}}) [1] \\
& \text{Holding}(o) [2] \\
& \text{ClearX(sweptVol}(o, l_{\text{start}}, l_{\text{end}}, (o)) [3] \\
\text{prim: } & \text{prim_move}
\end{align*}
\]
HPN: Hierarchical Action Representation

- Use a hierarchy defined using **precondition postponement** to reduce the horizon during regression
- Suppose operator $o$ has preconditions $p_1, \ldots, p_n$, effect $r$
- Operator with $p_n$ postponed
  - **$o_{\text{postponed}}$**
    - precon = $p_1, \ldots, p_{n-1}$
    - expansion:
      - Achieve $p_n$ while maintaining $p_1, \ldots, p_{n-1}$;
      - Then execute $o$
- Additional side-effects of achieving $p_n$ may have to be declared
- **Define hierarchy** by associating abstraction-level with each precondition

Pick($o, s, y$):
- effect: Holding($o$)
- pre: AtGraspPose($o$) [1]
- prim: prim_pick

Place($o, s, y$):
- effect: Holding($o$) [1]
- pre: $\neg$Holding($o$)
- prim: prim_place

Move($o, l_{\text{end}}, s, y$):
- choose: $l_{\text{start}}$
- effect: ObjectLoc($l_{\text{end}}$)
- pre: ObjectLoc($l_{\text{start}}$) [1]
  - Holding($o$) [2]
  - ClearX(sweptVol($o, l_{\text{start}}, l_{\text{end}}, \{o\}$)) [3]
- prim: prim_move
HPN: Main Algorithm

• Repeat depth-first refinement + execution
  • Primitives are executed as they are generated ("planning in the now")

```plaintext
HPN(s\text{now}, \gamma, \alpha, \text{world}):
    p = PLAN(s\text{now}, \gamma, \alpha)
    for (\omega_i, g_i) in p
        if isPrim(\omega_i)
            world.EXECUTE(\omega_i, s\text{now})
        else
            HPN(s\text{now}, g_i, NEXTLEVEL(\alpha, \omega_i), \text{world})
```
3.1 hpn architecture

The hpn process is invoked by

\[\text{hpn}(s_{\text{now}}, \gamma, \alpha, \text{world})\]

where

- \(s_{\text{now}}\) is a description of the state of \(\text{world}\) when the planner is called;
- \(\gamma\) is the goal, which describes a set of world states;
- \(\alpha\) is a structure that controls the abstraction level, which we discuss in more detail in section 3.2.4; and
- \(\text{world}\) is an actual robot or a simulator on which primitive actions can be executed. In the prototype system described in this paper, \(\text{world}\) is actually a geometric motion planner coupled with a simulated or physical robot.

hpn calls the regression-based plan procedure, which returns a whole plan at the specified level of abstraction, ((\(\omega_0, g_0\)), (\(\omega_1, g_1\)), \ldots, (\(\omega_n, g_n\))). The pre-images, \(g_i\), will serve as the goals for the planning problems at the next level down in the hierarchy.

\[\text{hpn}(s_{\text{now}}, \gamma, \alpha, \text{world}):\]

\[p = \text{PLAN}(s_{\text{now}}, \gamma, \alpha)\]

\[\text{for } (\omega_i, g_i) \text{ in } p\]

\[\text{if isPrim}(\omega_i)\]

\[\text{world.EXECUTE}(\omega_i, s_{\text{now}})\]

\[\text{else}\]

\[\text{hpn}(s_{\text{now}}, g_i, \text{NEXTLEVEL}(\alpha, \omega_i), \text{world})\]

Figure 7 shows a hierarchical version of the plan in figure 6. Blue nodes are goals for planning problems; pink nodes are operations associated with a concrete action; gray nodes are definitional operations. Operation nodes are prefixed with An where \(n\) is an integer representing the abstraction value at which that operator is being applied.

Instead of one large problem with an 11-step plan, we now have 5 planning problems, with solutions of length 1, 2, 3, 4, and 5. Because planning time is generally exponential in the length of the plan, the reduction in length of the longest plan is significant.
HPN: Overall Approach

Plan 1
\( A_0: \text{In}(o,G) \)

Plan 2
\( \text{ObjectLoc}(o,l_{\text{end}}) \)

Plan 2
\( \neg \text{Holding}(o) \)

Plan 3
\( \text{ClearX}(\text{sweptVol}(\ldots, \{o\})) \)

Plan 4
\( \text{Holding}(o) \)

\( \text{prim\_move} \)

\( \text{prim\_pick} \)

\( \text{A0: Move}(o, l_{\text{end}}) \)

\( \text{A1: Move}(o, l_{\text{end}}) \)

\( \text{A0: Clear} \)

\( \text{A0: Pick}(o) \)

\( \text{Plan 1} \)

\( \text{Plan 2} \)

\( \text{Plan 3} \)

\( \text{Plan 4} \)

\( \text{In}(o, s, \gamma) \):
\( \text{effect: Holding}(o) \)
\( \text{pre: AtGraspPose}(o) \) [1]
\( \text{prim: prim\_pick} \)

\( \text{Place}(o, s, \gamma) \):
\( \text{effect: } \neg \text{Holding}(o) \) [1]
\( \text{pre: Holding}(o) \)
\( \text{prim: prim\_place} \)

\( \text{Move}(o, l_{\text{end}}, s, \gamma) \):
\( \text{choose: } l_{\text{start}} \)
\( \text{effect: ObjectLoc}(l_{\text{end}}) \)
\( \text{pre: ObjectLoc}(l_{\text{start}}) \) [1]
\( \text{Holding}(o) \) [2]
\( \text{ClearX}(\text{sweptVol}(o, l_{\text{start}}, l_{\text{end}}, \{o\})) \) [3]
\( \text{prim: prim\_move} \)

\( \text{In}(o, R, s, \gamma) \):
\( \text{choose: } l \)
\( \text{effect: In}(o, R) \)
\( \text{pre: ObjectLoc}(l) \) [1]
\( \neg \text{Holding}(o) \) [2]
HPN: Overall Approach

HPN\( (s_{\text{now}}, \gamma, \alpha, \text{world}) \):
\[
p = \text{PLAN}(s_{\text{now}}, \gamma, \alpha)
\]
for \( (\omega_i, g_i) \) in \( p \)
if \( \text{isPrim}(\omega_i) \)
\[
\text{world}.\text{EXECUTE}(\omega_i, s_{\text{now}})
\]
else
\[
\text{HPN}(s_{\text{now}}, g_i, \text{NEXTLEVEL}(\alpha, \omega_i), \text{world})
\]

Plan 1
\( A0: \text{In}(o, G) \)

Plan 2
ObjectLoc\( (o, l_{\text{end}}) \)

Plan 3
ClearX\( (\text{sweptVol}(\ldots, \{o\}) ) \)

Plan 4
Holding\( (o) \)

A0: Clear
A1: Move\( (o, l_{\text{end}}) \)
Prim_move
A0: Pick\( (o) \)

Prim_pick

Place\( (o, l, \gamma) \):
\[
\text{effect: Holding}(o)
\]
\[
\text{pre: AtGraspPose}(o) \quad [1]
\]
\[
\text{prim: prim_pick}
\]

Move\( (o, l_{\text{end}}, s, \gamma) \):
\[
\text{choose: } l_{\text{start}}
\]
\[
\text{effect: ObjectLoc}(l_{\text{end}})
\]
\[
\text{pre: ObjectLoc}(l_{\text{start}}) \quad [1]
\]
\[
\text{Holding}(o) \quad [2]
\]
\[
\text{ClearX}(\text{sweptVol}(o, l_{\text{start}}, l_{\text{end}}, \{o\}) \quad [3]
\]
\[
\text{prim: prim_move}
\]

In\( (o, R, s, \gamma) \):
\[
\text{choose: } l
\]
\[
\text{effect: In}(o, R)
\]
\[
\text{pre: ObjectLoc}(l) \quad [1]
\]
\[
\text{Holding}(o) \quad [2]
\]
HPN: Overall Approach

HPN\(\left(s_{\text{now}}, \gamma, \alpha, \text{world}\right)\):
\[
p = \text{PLAN}\left(s_{\text{now}}, \gamma, \alpha\right)
\]
for \((\omega_i, g_i)\) in \(p\)
if isPrim\(\left(\omega_i\right)\)
\[
\text{world.EXECUTE}\left(\omega_i, s_{\text{now}}\right)
\]
else
\[
\text{HPN}\left(s_{\text{now}}, g_i, \text{NEXTLEVEL}\left(\alpha, \omega_i\right), \text{world}\right)
\]

Plan 1
\(\text{A0: } \text{In}(o,G)\)

Plan 2
\(\text{ObjectLoc}(o, l_{\text{end}})\)

Plan 2
\(\text{Place}(G)\)

Plan 3
\(\text{ClearX}(\text{sweptVol}(\ldots, \{o\}))\)

Plan 4
\(\text{Holding}(o)\)

\(\text{A0: Move}(o, l_{\text{end}})\)

\(\text{A1: Move}(o, l_{\text{end}})\)

\(\text{prim_pick}\)

\(\text{prim_move}\)

\(\text{A0: Clear}\)

\(\text{A0: Pick}(o)\)

\(\text{prim_pick}\)

Plan 3
\(\text{Plan 1}

\(\text{Plan 2}

\(\text{Plan 2}

\(\text{Plan 3}

\(\text{Plan 4}

\(\text{A0: Clear}\)

\(\text{A0: Pick}(o)\)

\(\text{prim_pick}\)

In((o, R), s, \gamma):
\[
\text{choose: } l
\]
\[
\text{effect: } \text{In}(o, R)
\]
\[
\text{pre: } \text{ObjectLoc}(l) \quad [1]
\]
\[
\neg \text{Holding}(o) \quad [2]
\]

Place((o), s, \gamma):
\[
\text{effect: } \neg \text{Holding}(o) \quad [1]
\]
\[
\text{pre: } \text{Holding}(o)
\]
\[
\text{prim: } \text{prim_place}
\]

Move((o), l_{\text{end}}, s, \gamma):
\[
\text{choose: } l_{\text{start}}
\]
\[
\text{effect: } \text{ObjectLoc}(l_{\text{end}})
\]
\[
\text{pre: } \text{ObjectLoc}(l_{\text{start}}) \quad [1]
\]
\[
\text{Holding}(o) \quad [2]
\]
\[
\text{ClearX}(\text{sweptVol}(o, l_{\text{start}}, l_{\text{end}}, \{o\}) \quad [3]
\]
\[
\text{prim: } \text{prim_move}
\]

Pick((o), s, \gamma):
\[
\text{effect: } \text{Holding}(o)
\]
\[
\text{pre: } \text{AtGraspPose}(o) \quad [1]
\]
\[
\text{prim: } \text{prim_pick}
\]
HPN: Algorithmic Details & Optimizations

- Use generators to make choice of arguments more efficient
- Algorithm commits to subgoals generated at higher level of abstraction when refining
  - Need to ensure subgoal feasibility
  - Approximate the generation of feasible results of “choose” operations using limited logical reasoning
    - Ensures logical consistency of fluents based on domain-specific integrity constraints
- Achieving postponed preconditions can lead to additional effects in abstract operators
  - Can declare approximations/conservative versions of side-effects with actions
HPN: Experiments

(a) A plate (intended for the first food item) has been placed on the table, and the robot is getting food from refrigerator.

(d) Starting to tidy up; both pans are in the sink.

(f) In order to enable picking up the pink cup in the right warehouse, the objects in the warehouse are re-arranged.

(h) All the objects in the right warehouse have been placed in refrigerator; two dirty cups left out on various tables are in the sink.
HPN: Summary

• Input:
  • Correct and complete primitive action definitions using geometric predicates
  • Operator-specific regression functions for geometric properties
  • Pose generators
  • For efficiency, can use additional input:
    • Pose generators that make use of current subgoal
    • Precondition levels to obtain a hierarchy, declaration of operator side-effects
    • Limited logical reasoning in pose generators

• Properties:
  • Complete if the problem has no dead-ends, action preconditions are accurate
    • Motion planners terminate and return solutions when preconditions hold
  • Approach for C1: through generators and regression (backward-search)
  • Approach for C2: Regression over geometric fluents

## High-Level Summary

<table>
<thead>
<tr>
<th>Search Space</th>
<th>High Level Planner</th>
<th>Low Level Reasoning</th>
<th>High-level Reasoning</th>
<th>High Level Language</th>
<th>P1: Infinite Motion Plans</th>
<th>P2: Downward Refinability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>aSyMov</strong></td>
<td>Single</td>
<td>Any TP</td>
<td>PRM/RRT</td>
<td>Symbolic</td>
<td>PDDL</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>IDTMP</strong></td>
<td>Dual</td>
<td>SMT</td>
<td>Any MP</td>
<td>Variables with continuous domains</td>
<td>SAS</td>
<td>SMT</td>
</tr>
<tr>
<td><strong>HPN</strong></td>
<td>Dual</td>
<td>HPN-specific regression planner</td>
<td>Any MP</td>
<td>Hybrid</td>
<td>HPN-specific</td>
<td>generators</td>
</tr>
</tbody>
</table>
TMP through an Interface Layer

- Geometric planning is hard $\rightarrow$ Symbolic high-level
- Off-the-shelf task planner
- Off-the-shelf motion planner
- Interface layer that communicates between task planner and motion planner
  - Converts each task level action to a motion planning problem
  - Converts motion planning failures as facts over symbols & refines abstract information

TMP through an Interface Layer: Example

PickUp(obj o1, pose p1, pose p2, pose p3, path p):
precondition: Empty(gripper) \land \text{GripperAt}(p1) \land
\text{At}(o1, p3) \land \text{IsGraspingPose}(p2, o1, p3)
\land \text{path}(p, p1, p2) \land \forall o2 \neg \text{Obstructs}(o2, p)
effect: \text{Holding}(o1) \land \neg \text{At}(o1, p3) \land
\neg \text{Empty(gripper)} \land \text{GripperAt}(p2)
TMP through an Interface Layer: Example

PickUp(obj o1, pose p1, pose p2, pose p3, path p):
precondition: Empty(gripper) \land GripperAt(p1) \land
At(o1, p3) \land IsGraspingPose(p2, o1, p3)
\land path(p, p1, p2) \land \forall o2 \neg Obstructs(o2, p)
effect: Holding(o1) \land \neg At(o1, p3) \land
\neg Empty(gripper) \land GripperAt(p2)
TMP through an Interface Layer: Example

PickUp(obj o1, pose p1, pose p2, pose p3, path p):
precondition: Empty(gripper) ∧ GripperAt(p1) ∧
At(o1, p3) ∧ IsGraspingPose(p2, o1, p3)
∧ path(p, p1, p2) ∧ ∀o2 ¬ Obstructs(o2, p)
effect: Holding(o1) ∧ ¬ At(o1, p3) ∧
¬ Empty(gripper) ∧ GripperAt(p2)

• High level intuitive plan:
  • pick block1 after going to block1’s grasping pose along a trajectory
TMP through an Interface Layer: Example

Interface level:
Searches for an instantiation of block1’s grasping pose that is reachable via a feasible (collision-free) trajectory

...finds no feasible trajectory

Symbolic references

• High level intuitive plan:
  • pick block1 after going to block1’s grasping pose along a trajectory
TMP through an Interface Layer: Example

Interface level:
Searches for an instantiation of block1’s grasping pose that is reachable via a feasible (collision-free) trajectory

...finds no feasible trajectory

• High level intuitive plan:
  • pick block1 after going to block1’s grasping pose along a trajectory

Symbolic references

Fix values for references, report reason for failure:
“block2 obstructs block1’s grasping pose along a trajectory”
TMP through an Interface Layer: Example

PickUp(obj o1, pose p1, pose p2, pose p3, path p):
precondition: Empty(gripper) \& GripperAt(p1) \&
At(o1, p3) \& IsGraspingPose(p2, o1, p3)
\& path(p, p1, p2) \& \forall o2 \neg Obstructs(o2, p)
ext: Holding(o1) \& \neg At(o1, p3) \&
\neg Empty(gripper) \& GripperAt(p2)

• High level intuitive plan:
  • pick block1 after going to block1’s grasping pose along a trajectory

Fix values for references, report reason for failure:
“block2 obstructs block1’s grasping pose along a trajectory”
TMP through an Interface Layer: Example

PickUp(obj o1, pose p1, pose p2, pose p3, path p):
precondition: Empty(gripper) ∧ GripperAt(p1) ∧
At(o1, p3) ∧ IsGraspingPose(p2, o1, p3)
∧ path(p, p1, p2) ∧ ∀o2 ¬ Obstructs(o2, p)
effect: Holding(o1) ∧ ¬ At(o1, p3) ∧
¬ Empty(gripper) ∧ GripperAt(p2)

Discrete state += Obstructs(block2, path(initLoc, gp(block1)))
PickUp(obj o1, pose p1, pose p2, pose p3, path p):
precondition: Empty(gripper) ∧ GripperAt(p1) ∧
                At(o1, p3) ∧ IsGraspingPose(p2, o1, p3)
                ∧ path(p, p1, p2) ∧ ∀o2 ¬ Obstructs(o2, p)
effect: Holding(o1) ∧ ¬ At(o1, p3) ∧
            ¬ Empty(gripper) ∧ GripperAt(p2)

Discrete state += Obstructs(block2, path(initLoc, gp(block1)))
**TMP through an Interface Layer: Example**

PickUp(obj o1, pose p1, pose p2, pose p3, path p):
precondition: Empty(gripper) ∧ GripperAt(p1) ∧
    At(o1, p3) ∧ IsGraspingPose(p2, o1, p3)
    ∧ path(p, p1, p2) ∧ ∀o2 ¬ Obstructs(o2, p)

effect: Holding(o1) ∧ ¬ At(o1, p3) ∧
    ¬ Empty(gripper) ∧ GripperAt(p2)

- High level intuitive plan:
  - pick block1 after going to *block1’s grasping pose*...
- REPLAN
  - pick block2 after going to *block2’s grasping pose*...
  - release block2 after going to *release pose for free area*...
  - pick block1 after going to *block1’s grasping pose*...
TMP through an Interface Layer: Example

PickUp(obj o1, pose p1, pose p2, pose p3, path p):
precondition: Empty(gripper) \land GripperAt(p1) \land 
          At(o1, p3) \land IsGraspingPose(p2, o1, p3) 
          \land path(p, p1, p2) \land \forall o2 \not\in Obstructs(o2, p)
effect: Holding(o1) \land \neg At(o1, p3) \land 
       \neg Empty(gripper) \land GripperAt(p2)

• High level intuitive plan:
  • pick block1 after going to block1's grasping pose...
• REPLAN
  • pick block2 after going to block2's grasping pose...
  • release block2 after going to release pose for free area...
  • pick block1 after going to block1's grasping pose...

Goal Reached!!
TMP through an Interface Layer: Complete Algorithm

Task Planner

Plan with pose references

Updated initial state

Pose instantiation

Error generation, state update

Motion planning goals

Trajectories or errors

Interface

Motion Planner

Environment Model

PDDL Domain
TMP through an Interface Layer: Complete Algorithm

Iteratively try all instantiations

Stop when a valid instantiation is found for all the symbolic references.
TMP through an Interface Layer: Complete Algorithm

Iteratively try all instantiations

What if that fails?
TMP through an Interface Layer: Complete Algorithm

Iteratively try all instantiations

What if that fails?

Fix the symbolic state using failure reason

Move(b₁, bpgf_b₁)

Pick(b₁, gpfg_b₁)
TMP through an Interface Layer: Complete Algorithm

Iteratively try all instantiations

What if that fails?

Fix the symbolic state using failure reason

Compute a new plan
TMP through an Interface Layer: Complete Algorithm

Iteratively try all instantiations

What if that fails?

Fix the symbolic state using failure reason

Compute a new plan
TMP through an Interface Layer: Experiments

- Several objects obstruct the target object
- Most of these objects are themselves obstructed by other objects
- No designated free space
- Geometric predicates constructed by the interface layer: `obstructs`(pose, obj1, obj2)
TMP through an Interface Layer: Experiments

- 3 pairs of noodle & soup bowls, predefined destinations
- Tray available, but utilization not
- Task planner has to decide whether to use the tray based on plan costs
- Geometric predicates constructed by interface layer:
  - smaller(obj1, obj2);
  - wrong_side(gripper, pose) — used for determining which hand to use
TMP through an Interface Layer: Summary

- **Inputs**
  - PDDL domain with references instead of continuous values
  - Generators for instantiating references
  - Subroutines for infeasibility detection and expression, given a motion plan

- **Properties**
  - Probabilistically complete
  - Each task planner invocation gets
    - Small branching factor
    - States have only a relevant subset of facts for current instantiation

- How does it handle P1: through symbolic references, forward-search, and lazily invoking the motion planner

- How does it handle P2: by communicating errors to the high-level planner and computing new plans

## High-level Summary

<table>
<thead>
<tr>
<th></th>
<th>Search Space</th>
<th>High Level Planner</th>
<th>Low Level Reasoning</th>
<th>High-level Reasoning</th>
<th>High Level Language</th>
<th>P1: Infinite Motion Plans</th>
<th>P2: Downward Refinability</th>
</tr>
</thead>
<tbody>
<tr>
<td>aSyMov</td>
<td>Single</td>
<td>Any TP</td>
<td>PRM/RRT</td>
<td>Symbolic</td>
<td>PDDL</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>IDTMP</td>
<td>IDTMP</td>
<td>Dual</td>
<td>SMT</td>
<td>Any MP</td>
<td>Variables with continuous domains</td>
<td>SAS</td>
<td>SMT</td>
</tr>
<tr>
<td>HPN</td>
<td>Dual</td>
<td>HPN-specific regression planner</td>
<td>Any MP</td>
<td>Hybrid</td>
<td>HPN-specific generators</td>
<td>Regression over geometric fluents</td>
<td></td>
</tr>
<tr>
<td>Symbolic Interface</td>
<td>Dual</td>
<td>Any TP</td>
<td>Any MP</td>
<td>Symbolic</td>
<td>PDDL</td>
<td>Pose generators</td>
<td>Identifying errors</td>
</tr>
</tbody>
</table>

136
PDDLStream

• Forward-search over hybrid representation
• Modifies the PDDL representation

PDDLStream

- Forward-search over hybrid representation
- Modifies the PDDL representation
PDDLStream: Streams

- Forward-search over hybrid representation
- Modifies the PDDL representation

```prolog
(:action move
 :param (?q1 ?t ?q2)
 :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1)
 (forall(?b) (imply (Block ?b) (Safe ?t ?b))))
 :eff (and (AtConf ?q2) (not (AtConf ?q1))
 (incr (total-cost) (Dist ?t))))
```
PDDLStream: Streams

- Forward-search over hybrid representation
- Modifies the PDDL representation

```
(:action move
 :param (?q1 ?t ?q2)
 :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1)
 (forall(?b)(imply (Block ?b) (Safe ?t ?b))))
 :eff (and (AtConf ?q2) (not (AtConf ?q1))
 (incr (total-cost) (Dist ?t)))
```

Move([0,0],[t,[-1,1]])
t = some trajectory
PDDLStream: Streams

- Forward-search over hybrid representation
- Modifies the PDDL representation

```
(:action move
 :param (?q1 ?t ?q2)
 :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1)
 (forall(?b) (imply (Block ?b) (Safe ?t ?b))))
 :eff (and (AtConf ?q2) (not (AtConf ?q1))
 (incr (total-cost) (Dist ?t)))
```

Requires continuous parameters
PDDLStream: Streams

- Forward-search over hybrid representation
- Modifies the PDDL representation
- Main two concepts:
  - Streams

```
(:action move
 :param (?q1 ?t ?q2)
 :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1))
 :eff (and (AtConf ?q2) (not (AtConf ?q1))
         (incr (total-cost) (Dist ?t)))
```

```
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
            (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
            (Motion ?q1 ?t ?q2)))
 (:function (Dist ?t)
 :dom (Traj ?t))
```
PDDLStream: Streams

- Forward-search over hybrid representation
- Modifies the PDDL representation
- Main two concepts:
  - Streams ~ **Samplers or Generators**

```
(:action move
 :param (?q1 ?t ?q2)
 :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1))
 :eff (and (AtConf ?q2) (not (AtConf ?q1))
 (incr (total-cost) (Dist ?t)))

(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
 (Motion ?q1 ?t ?q2)))
 (:function (Dist ?t)
 :dom (Traj ?t))
```
PDDLStream: Streams

- Forward-search over hybrid representation
- Modifies the PDDL representation
- Main two concepts:
  - Streams ~ **Samplers or Generators**
    - Procedural component: A conditional generator

```python
(:action move
 :param (?q1 ?t ?q2)
 :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1))
 :eff (and (AtConf ?q2) (not (AtConf ?q1))
       (incr (total-cost) (Dist ?t))))
```

```python
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1) (Conf ?q2)))
 :out (?t)
 :cert (and (Traj ?t) (Motion ?q1 ?t ?q2)))
 (:function (Dist ?t)
 :dom (Traj ?t))
```

```python
def motion(q1, q2):
    # code to sample a pose
    ...
    return traj
```

![Diagram showing a robot moving from initial area to goal area](image)

- Goal Area
- Init Area
- Robot R
- Object O
- Move([0,0],t,[-1,1])
- t = some trajectory
PDDLStream: Streams

- Forward-search over hybrid representation
- Modifies the PDDL representation
- Main two concepts:
  - Streams ~ **Samplers or Generators**
    - Procedural component: a conditional generator
    - Declarative component: add facts that can be guaranteed

```
(:action move
  :param (?q1 ?t ?q2)
  :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1))
  :eff (and (AtConf ?q2) (not (AtConf ?q1))
    (incr (total-cost) (Dist ?t)))
```

```
(:stream motion
  :inp (?q1 ?q2)
  :dom (and (Conf ?q1)
    (Conf ?q2))
  :out (?t)
  :cert (and (Traj ?t)
    (Motion ?q1 ?t ?q2)))
```

Move([0,0],t,[-1,1])

Object O

Init Area

Goal Area

Robot R

t = some trajectory

G

Object O

Init Area

Goal Area

Robot R

t = some trajectory

G
PDDLStream: Streams

• Forward-search over hybrid representation
• Modifies the PDDL representation
• Main two concepts:
  • Streams ~ **Samplers or Generators**
    • Procedural component: a conditional generator
    • Declarative component: add facts that can be guaranteed

```
(:action move
  :param (?q1 ?t ?q2)
  :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1))
  :eff (and (AtConf ?q2) (not (AtConf ?q1))
    (incr (total-cost) (Dist ?t))))
```

```
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1) (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
   (Motion ?q1 ?t ?q2)))
 (:function (Dist ?t)
   :dom (Traj ?t))
```

Move((0,0),t,[-1,1])

G

Robot R

Object O

Init Area

Goal Area

146
PDDLStream: C1

- Forward-search over hybrid representation
- Modifies the PDDL representation
- Main two concepts:
  - Streams ~ **Samplers or Generators**
  - Procedural component: a conditional generator
  - Declarative component: add facts that can be guaranteed

C1: Infinite groundings → Infinitely many motion planning problems!!
PDDLStream: Optimistic Samples

- Forward-search over hybrid representation
- Modifies the PDDL representation
- Main two concepts:
  - Streams ~ **Samplers or Generators**
    - Procedural component: a conditional generator
    - Declarative component: add facts that can be guaranteed
  - Optimistic samples ~ **Symbolic References**
    - Evaluating streams to generate samples is expensive
    - High-level planning would need a thousands of these calls for even simple problems
    - So solution? → Generate “optimistic” placeholder samples..
      Assumed to be valid

\[
\text{Move}([0,0], t, [-1,1])
\]
PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples (Tackling C1)
PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples

Only depends on objects that does not require any stream \( \implies \) Required depth = 0

Achievable optimistic samples at level 0:
- \#p1_0 \([(O, Init)]\)
- \#p2_0\([(O, Goal)]\)
- \#g1_0\([(G, Goal)]\)
PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples

\[ \text{Required depth} = 1 \]

At level 1:
- \#q1_1 [(O,#p1_0, #g_0)]
- \#q2_1 [(O,#p2_0, #g_0)]
- \ldots
PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples

Required input that is generated using streams of depth = 0

\[
\text{At level 1: } \#q1_1 [(O,#p1_0, #g_0)] \\
\#q2_1 [(O,#p2_0, #g_0)] \\
\ldots
\]

→ Required depth = 1

\[
\begin{align*}
\text{(stream poses)} \\
\text{:inp (?b ?r)} \\
\text{:dom (and (Block ?b)} \\
\text{(Region ?r))} \\
\text{:out (?p)} \\
\text{:cert (and (Pose ?b ?p, ?r)) :out (?g)} \\
\text{(Contains ?b ?p ?r)} \\
\text{:cert (and (Conf ?g) (stream grasps))} \\
\text{:inp (?b)} \\
\text{:dom (Block ?b)} \\
\text{:out (?g)} \\
\text{:cert (and (Conf ?g) (stream motion))} \\
\text{:inp (?g)} \\
\text{:dom (and (Conf ?qg) (stream cfree))} \\
\text{:out (?g)} \\
\text{(:cert (Grasp ?b ?g) (Conf ?qg))} \\
\text{:cert (and (Traj ?t) (stream cfree))} \\
\text{:dom (and (Traj ?t) (Function (Dist ?t) ?g))} \\
\text{:cert (CFree ?t ?b ?p) :dom (Traj ?t))}
\end{align*}
\]
PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples
  - Compute an abstract plan using the optimistic samples

We need:
(Motion ...) and (Kin ...) 

Min depth = 2
PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples
  - Compute an abstract plan using the optimistic samples
  - If no plan found
    - Increase depth and repeat.

We need:
(Motion ...) and (Kin ... )

Min depth = 2

```
(:stream poses
 :inp (?b ?r)
 :dom (and (Block ?b) (Region ?r))
 :out (?p)
 :cert (and (Pose ?b ?p) (Out ?p))
)
(:stream grasps
 :inp (?b)
 :dom (Block ?b) (Grasp ?b)
 :out (?p)
 :cert (Grasp ?b ?g)
)
(:stream cfree
 :inp (?b ?p)
 :dom (and (Traj ?t))
 :cert (and (CFree ?t ?b ?p))
)
(:stream ik
 :inp (?b ?r)
 :dom (and (Pos ?b ?p ?g))
 :out (?g)
 :cert (and (Pose ?b ?p ?g))
)
```

Robot R

Goal Area

Object O

Init Area
PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples
  - Compute an abstract plan using the optimistic samples
  - If no plan found
    - Increase dept and repeat.
  - If a plan is found
    - Use streams to instantiate optimistic samples with real values

```python
def ik():
    # code to compute IK
    ....
```

(Move #q0 #t0 #q1)
(Pick b1 #p0 g #q1)
(Move #q1 #t2 #q2)
(Place b #p2 g #q2)
PDDLStream: Overall Approach

• Depth = 0
• For the current dept:
  • Generate achievable optimistic samples
  • Compute an abstract plan using the optimistic samples
  • If no plan found
    • Increase dept and repeat.
  • If a plan is found
    • Use streams to instantiate optimistic samples with real values
    • If no instantiation found
      • Disable stream at the current depth and replan with the same depth

```python
def motion():
    # code to compute IK
    ...
```

(Move #q0 #t0 #q1)
(Pick b1 #p0 g #q1)
(Move #q1 #t2 #q2)
(Place b #p2 g #q2)
PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples
  - Compute an abstract plan using the optimistic samples
  - If no plan found
    - Increase dept and repeat.
  - If a plan is found
    - Use streams to instantiate optimistic samples with real values
    - If no instantiation found
      - Disable stream at the current depth and replan with the same depth

Forces task planner to give a new plan!!

\[ \text{def motion():}
\]
\[ \text{#code to compute IK}
\]
\[ \text{.....}
\]
PDDLStream: Overall Approach

- **Depth = 0**
- For the current dept:
  - Generate achievable optimistic samples
  - Compute an abstract plan using the optimistic samples
  - If no plan found
    - Increase dept and repeat.
  - If a plan is found
    - Use streams to instantiate optimistic samples with real values
    - If no instantiation found
      - Disable stream at the current depth and replan with the same depth

**Def motion():**
#code to compute IK

(Move #q0 #t0 #q1)
(Pick b1 #p0 g #q1)
(Move #q1 #t2 #q2)
(Place b #p2 g #q2)

Forces task planner to give a new plan!!
PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples
  - Compute an abstract plan using the optimistic samples
  - If no plan found
    - Increase dept and repeat.
  - If a plan is found
    - Use streams to instantiate optimistic samples with real values
    - If no instantiation found
      - Disable stream at the current depth and replan with the same depth

To ensure completeness:
Adaptively switch between computing new plan and refinements.

```python
def motion():
    # code to compute MP
    ...
```

(Move #q0 #t0 #q1)  
(Pick b1 #p0 g #q1)  
(Move #q1 #t2 #q2)  
(Place b #p2 g #q2)
PDDLStream: Experiments
PDDLStream: Summary

• Inputs
  • PDDL domain with streams
  • Procedural function for streams -- conditional generators

• Properties
  • Forward search using hybrid representation
  • Probabilistically complete – guarantees it by switching between computing refinements and finding new plans

• Key Ideas:
  • C1: through optimistic samples and lazy-querying streams
  • C2: by forcing the planner to generate new plans until it finds a plan that is refinaible.
# High-Level Summary

<table>
<thead>
<tr>
<th></th>
<th>Search Space</th>
<th>High Level Planner</th>
<th>Low Level Reasoning</th>
<th>High-level Reasoning</th>
<th>High Level Language</th>
<th>P1: Infinite Motion Plans</th>
<th>P2: Downward Refinability</th>
</tr>
</thead>
<tbody>
<tr>
<td>aSyMov</td>
<td>Single</td>
<td>Any TP</td>
<td>PRM/RRT</td>
<td>Symbolic</td>
<td>PDDL</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>IDTMP</td>
<td>Dual</td>
<td>SMT</td>
<td>Any MP</td>
<td>Variables with continuous domains</td>
<td>SAS</td>
<td>SMT</td>
<td>SMT</td>
</tr>
<tr>
<td>HPN</td>
<td>Dual</td>
<td>HPN-specific regression planner</td>
<td>Any MP</td>
<td>Hybrid</td>
<td>HPN-specific generators</td>
<td>Regression over geometric fluents</td>
<td></td>
</tr>
<tr>
<td>Symbolic Interface</td>
<td>Dual</td>
<td>Any TP</td>
<td>Any MP</td>
<td>Symbolic</td>
<td>PDDL</td>
<td>Pose generators</td>
<td>Identifying errors</td>
</tr>
<tr>
<td>PDDLStream</td>
<td>Dual</td>
<td>Any TP</td>
<td>Any MP</td>
<td>Hybrid</td>
<td>PDDL</td>
<td>optimistic samples</td>
<td>Iterating over all task plans</td>
</tr>
</tbody>
</table>
Limitations

TMP through an extension layer [Srivastava et al. 2014]
Incremental TMP [Dantam et al. 2018]
PDDLStream [Garrett et al. 2020]

Works only for deterministic problems
Limitations

TMP through an extension layer [Srivastava et al. 2014]
Incremental TMP [Dantam et al. 2018]
PDDLStream [Garrett et al. 2020]

Works only for deterministic problems

What if robot’s actions are stochastic?
PickUp(obj o1, pose p1, pose p2, pose p3, path p):
precondition:
Empty(gripper) ≔ GripperAt(p1) ∧
At(o1, p3) ∧ IsGraspingPose(p2, o1, p3)
∧ path(p, p1, p2) ∧ ∀o2 ¬ Obstructs(o2, p)

effect:
0.8 Holding(o1) ∧ ¬ At(o1, p3) ∧
¬ Empty(gripper) ∧ GripperAt(p2)
0.2 [No change]
Task and Motion Planning Under Uncertainty

Abstract Model → Symbolic Planner → High-level plan

1. Move $R_{\text{O}} \text{ traj}_1$
2. Pick $R_{\text{C}} \text{ traj}_2$
3. Move $R_{\text{G}} \text{ traj}_3$
4. Place $R_{\text{O}} \text{ traj}_4$
Task and Motion Planning Under Uncertainty

Abstract Model → Symbolic Planner → High-level plan

- 1. Move $R_a \text{traj}_1$
- 2. Pick $R_b \text{traj}_2$
- 3. Move $R_c \text{traj}_3$
- 4. Place $R_d \text{traj}_4$

High-level policy

Move → Move → Pick → Move → Place → Move
• High-level domain: classical planning PDDL domain

PPDDL domain for a stochastic shortest path problem

• High-level domain: classical planning PDDL domain
  PPDDL domain for a stochastic shortest path problem

• Use an SSP solver to compute a branching policy
STAMP

1. High-level domain: classical planning PDDL domain
   PPDDL domain for a stochastic shortest path problem

2. Use an SSP solver to compute a branching policy

3. Refine the plan to compute task and motion plan:
   Refine the **entire policy** to compute task and motion policy
STAMP: Dealing with #branches

- Too many branches: Waiting to refine the entire policy tree would be inefficient
• Too many branches: Waiting to refine the entire policy tree would be inefficient

• Intuitive idea: knapsack problem with computation as cost and probability of encountering as value
• Too many branches: Waiting to refine the entire policy tree would be inefficient

• Intuitive idea: knapsack problem with computation as cost and probability of encountering as value

Theorem: Let $t$ be the time since the start of the algorithm at which the refinement of any RTL path is completed. If path costs are accurate and constant then the total probability of unrefined paths at time $t$ is at most $1 - \frac{opt(t)}{2}$, where $opt(t)$ is the best possible refinement that could have been achieved in time $t$. 
STAMP: HPlan Algorithm

PRN 1
- Partially refined policy
- Current Low-level state
- Current High-level state

Error $e_1$
Error $e_2$
Error $e_3$

PRN 2
PRN 3
PRN 4
The overall algorithm works as follows:

- Select a node from PRG.
- Compute an abstract policy.
- Select one of the following:
  - Explore
  - Expand the PRG

Repeat until a policy is fully refined in one of the PRG nodes.
Time-based switching between nodes allows maintaining multiple abstract models and prevents getting stuck into a single abstract model.

**Theorem:** If there exists a proper policy that reaches the goal within horizon $h$ with probability $p$, and has feasible low-level refinement, then the algorithm will find it with probability $1.0$ in the limit of infinite samples.
STAMP: Experiments

- Problem: build a desired structure using Keva planks.
  - Target design is expressed as a goal condition

- Stochasticity:
  - User may place the plank on one of two different locations

- Robot: Yumi IRB 14000
STAMP: Summary

• Inputs
  • PPDDL domain with references instead of continuous values and possibly stochastic actions
  • Generators for instantiating references
  • Subroutines for infeasibility detection and expression, given a motion plan

• Properties
  • **Handles Stochasticity**
    • + all the properties of “TMP using Interface layer”
  • How does it handle P1: through symbolic references, forward-search, and lazily invoking the motion planner
  • How does it handle P2: by communicating errors to the high-level planner and computing new plans

<table>
<thead>
<tr>
<th></th>
<th>Search Space</th>
<th>High Level Planner</th>
<th>Low Level Reasoning</th>
<th>High-level</th>
<th>High Level Language</th>
<th>P1: Infinite Motion Plans</th>
<th>P2: Downward Refinability</th>
<th>Stochastic</th>
</tr>
</thead>
<tbody>
<tr>
<td>aSyMov</td>
<td>Single</td>
<td>Any TP</td>
<td>PRM/RRT</td>
<td>Symbolic</td>
<td>PDDL</td>
<td>N/A</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>HPN</td>
<td>Dual</td>
<td>HPN-specific</td>
<td>Any MP</td>
<td>Hybrid</td>
<td>HPN-specific</td>
<td>generators</td>
<td>Regression over geometric fluents</td>
<td>No</td>
</tr>
<tr>
<td>IDTMP</td>
<td>Dual</td>
<td>SMT</td>
<td>Any MP</td>
<td>Variables with continuous domains</td>
<td>SAS</td>
<td>SMT</td>
<td>SMT</td>
<td>No</td>
</tr>
<tr>
<td>Symbolic Interface</td>
<td>Dual</td>
<td>Any TP</td>
<td>Any MP</td>
<td>Symbolic</td>
<td>PDDL</td>
<td>Pose generators</td>
<td>Identifying errors</td>
<td>No</td>
</tr>
<tr>
<td>PDDLStream</td>
<td>Dual</td>
<td>Any TP</td>
<td>Any MP</td>
<td>Hybrid</td>
<td>PDDL</td>
<td>optimistic samples</td>
<td>Iterating over all task plans</td>
<td>No</td>
</tr>
<tr>
<td>STAMP</td>
<td>Dual</td>
<td>Any TP</td>
<td>Any MP</td>
<td>Symbolic</td>
<td>PPDDL</td>
<td>Pose generators</td>
<td>Identifying errors</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Limitations

<table>
<thead>
<tr>
<th>Approach</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>aSyMov</td>
<td>PDDL</td>
</tr>
<tr>
<td>IDTMP</td>
<td>PDDL + module to convert symbolic action to motion planning problems</td>
</tr>
<tr>
<td>HPN</td>
<td>Action descriptions + preimage for each action</td>
</tr>
<tr>
<td>Symbolic Interface</td>
<td>PDDL + generators</td>
</tr>
<tr>
<td>PDDLStream</td>
<td>PDDL + streams (generators)</td>
</tr>
<tr>
<td>STAMP</td>
<td>PDDL + generators</td>
</tr>
</tbody>
</table>
Limitations

<table>
<thead>
<tr>
<th>Approach</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>aSyMov</td>
<td>PDDL</td>
</tr>
<tr>
<td>IDTMP</td>
<td>PDDL + module to convert symbolic action to motion planning problems</td>
</tr>
<tr>
<td>HPN</td>
<td>Action descriptions + preimage for each action</td>
</tr>
<tr>
<td>Symbolic Interface</td>
<td>PDDL + generators</td>
</tr>
<tr>
<td>PDDLStream</td>
<td>PDDL + streams (generators)</td>
</tr>
<tr>
<td>STAMP</td>
<td>PDDL + generators</td>
</tr>
</tbody>
</table>

Requires

1) State abstractions
2) Action abstractions
3) Action descriptions
4) Samples or generators
Outline

1. Background: Why Task and Motion Planning?
2. Abstraction as a Foundation for TMP
3. Abstraction-based Approaches
4. Research Frontier: Neuro-Symbolic Learning for TMP
What needs to be learned?

1. State abstractions
2. Temporal abstractions
   1. Identifying actions
   2. Learning action descriptions
3. Learning samplers / generators for action refinements
Learning State Abstractions Given High-level Actions
Skills-to-Symbols

- Core idea: Learned symbolic model in a PDDL representation

- Input: State variables with low-level continuous values

- Output: A symbolic PDDL model

- What is given:
  - Options masks -- set of low-level state variables relevant to an option
  - Abstract goal options -- Options that achieves termination sets with probability 1.0 expressed using only relevant state variables

Skills-to-symbols

given options → partition → abstract subgoal options → construct classifiers → characterizing sets and masks

domain description (set theoretic / PDDL) → compute operator descriptions → complete symbolic vocabulary → compute factors and projections
Skills-to-symbols: Computing Factors

Variables

\[ s_1 = \text{Pose}_r = \text{Robot pose} \]
\[ s_2 = \text{Pose}_o = \text{Object pose} \]
\[ s_3 = \text{Attached} = \text{Whether the object is picked or not} \]
Skills-to-symbols: Computing Factors

Variables

\[ s_1 = \text{Pose}_r = \text{Robot pose} \]
\[ s_2 = \text{Pose}_o = \text{Object pose} \]
\[ s_3 = \text{Attached} = \text{Whether the object is picked or not} \]

Options

\[ o_1 = \text{Move (r)} : s_1 \quad // \text{moves the robot} \]
\[ o_2 = \text{Grab (o).} : s_1, s_2 \quad // \text{grabs the object} \]
\[ o_3 = \text{UnGrab(o).} : s_1, s_2 \quad // \text{releases the object} \]
\[ o_4 = \text{Transport(o)} : s_1, s_3 \quad // \text{moves the object} \]
Skills-to-symbols: Computing Factors

Variables

\[ s_1 = \text{Pose}_r = \text{Robot pose} \]
\[ s_2 = \text{Pose}_o = \text{Object pose} \]
\[ s_3 = \text{Attached} = \text{Whether the object is picked or not} \]

Options

\[ o_1 = \text{Move (r)} : s_1 \quad \text{// moves the robot} \]
\[ o_2 = \text{Grab (o)} : s_1, s_2 \quad \text{// grabs the object} \]
\[ o_3 = \text{UnGrab(o)} : s_1, s_2 \quad \text{// releases the object} \]
\[ o_4 = \text{Transport(o)} : s_1, s_3 \quad \text{// moves the object} \]
Skills-to-symbols: Computing Factors

Variables

\[ s_1 = \text{Pose}_r = \text{Robot pose} \]
\[ s_2 = \text{Pose}_o = \text{Object pose} \]
\[ s_3 = \text{Attached} = \text{Whether the object is picked or not} \]

Options

\[ o_1 = \text{Move (r)} : s_1 \quad // \text{moves the robot} \]
\[ o_2 = \text{Grab (o).} : s_1, s_2 \quad // \text{grabs the object} \]
\[ o_3 = \text{UnGrab(o).} : s_1, s_2 \quad // \text{releases the object} \]
\[ o_4 = \text{Transport(o)} : s_1, s_3 \quad // \text{moves the object} \]
Skills-to-symbols: Computing Factors

Variables

\( s_1 = \text{Pose}_r = \text{Robot pose} \)
\( s_2 = \text{Pose}_o = \text{Object pose} \)
\( s_3 = \text{Attached} = \text{Whether the object is picked or not} \)

Options

\( o_1 = \text{Move (r)} : s_1 \quad \text{// moves the robot} \)
\( o_2 = \text{Grab (o).} : s_1, s_2 \quad \text{// grabs the object} \)
\( o_3 = \text{UnGrab(o).} : s_1, s_2 \quad \text{// releases the object} \)
\( o_4 = \text{Transport(o)} : s_1, s_3 \quad \text{// moves the object} \)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>( s_1 )</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>( s_2 )</td>
</tr>
<tr>
<td>( f_3 )</td>
<td>( s_3 )</td>
</tr>
</tbody>
</table>

Simple case!!
Skills-to-symbols: Computing Factors

<table>
<thead>
<tr>
<th>Factor</th>
<th>State Variables</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>$s_1, s_2$</td>
<td>$o_1$</td>
</tr>
<tr>
<td>$f_2$</td>
<td>$s_3$</td>
<td>$o_{1, 2}$</td>
</tr>
<tr>
<td>$f_3$</td>
<td>$s_4$</td>
<td>$o_2$</td>
</tr>
<tr>
<td>$f_4$</td>
<td>$s_5$</td>
<td>$o_{2, 3}$</td>
</tr>
<tr>
<td>$f_5$</td>
<td>$s_6, s_7$</td>
<td>$o_3$</td>
</tr>
</tbody>
</table>
Skills-to-symbols: Assigning Propositions

Identify propositions:

1. Identify independent factors -- factors that can be used to decompose the effect
   create a proposition for each independent factor

2. For the remaining collection of factor -- create a proposition

<table>
<thead>
<tr>
<th>Factor</th>
<th>State Variables</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>$s_1, s_2$</td>
<td>$o_1$</td>
</tr>
<tr>
<td>$f_2$</td>
<td>$s_3$</td>
<td>$o_1, o_2$</td>
</tr>
<tr>
<td>$f_3$</td>
<td>$s_4$</td>
<td>$o_2$</td>
</tr>
<tr>
<td>$f_4$</td>
<td>$s_5$</td>
<td>$o_2, o_3$</td>
</tr>
<tr>
<td>$f_5$</td>
<td>$s_6, s_7$</td>
<td>$o_3$</td>
</tr>
</tbody>
</table>
Skills-to-symbols: Learning Symbolic Operators

- For each option:
  - Identify the added and removed propositions in the effect set (termination set)
  - Identify the propositions in the propositions in the initiation set
Skills-to-symbols: Example

(:action cupboard_open1
 :parameters ()
 :precondition (and (symbol1) (symbol3) (symbol4))
 :effect (and (symbol5) (not (symbol4))
 (decrease (reward) 67.44))
)

symbol1  symbol3  symbol4  symbol5
Skills-to-symbols: Example

(:action cupboard_open
 :parameters ()
 :precondition (and (symbol1) (symbol3) (symbol4))
 :effect (and (symbol5) (not (symbol4)))
 (decrease (reward) 67.44))
## High-Level Summary

<table>
<thead>
<tr>
<th>Skills to Symbols</th>
<th>Abstract States</th>
<th>Abstract Actions</th>
<th>Low-level behaviors</th>
<th>Samplers</th>
<th>C-Space changes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Learned</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Learning High-Level Actions Given State Abstractions
Learning Symbolic Operators

• Core idea: Learn descriptions (precondition and effects) of high-level actions for task and motion planning

• What is provided:
  • State abstractions -- predicates and interpretation of predicates
  • Controllers -- low-level behaviors parameterized by typed objects
  • Samplers -- generators for discretizing high-level arguments
  • Data -- \{x_i, a_i, x_{i+1}\} for a set of training tasks

Learning Symbolic Operators: Approach

What is given?

Data

$x_0, c_1, x_1$

... 

$x_{n-1}, c_n, x_n$

Controllers

Move(o)

Pick(o)

Place(o)

Predicates

At(?o – obj ?a – area)

RobotAt(?a – area)

Holding(?o – obj)
Learning Symbolic Operators: Approach

What is given?

Data
- \( x_0, c_1, x_1 \)
- \( \ldots \)
- \( x_{n-1}, c_n, x_n \)

Controllers
- Move(o)
- Pick(o)
- Place(o)

Predicates
- At(?o – obj ?a – area)
- RobotAt(?a – area)
- Holding(?o – obj)

1. Convert low-level states into abstract states in the data
Learning Symbolic Operators: Approach

What is given?

**Data**
- $x_0, c_1, x_1$
- ...
- $x_{n-1}, c_n, x_n$

**Controllers**
- Move($o$)
- Pick($o$)
- Place($o$)

**Predicates**
- At(?$o$ – obj ?a – area)
- RobotAt(?a – area)
- Holding(?$o$ – obj)

1. Convert low-level states into abstract states in the data
2. Cluster transitions using the lifted effects

Abstract Data
- $s_0, c_1, s_1$
- ...
- $s_{n-1}, c_n, s_n$
Learning Symbolic Operators: Approach

**What is given?**

<table>
<thead>
<tr>
<th>Data</th>
<th>Controllers</th>
<th>Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0, c_1, x_1$</td>
<td>Move(o)</td>
<td>At(?o – obj ?a – area)</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>Pick(o)</td>
<td>RobotAt(?a – area)</td>
</tr>
<tr>
<td>$x_{n-1}, c_n, x_n$</td>
<td>Place(o)</td>
<td>Holding(?o – obj)</td>
</tr>
</tbody>
</table>

1. Convert low-level states into abstract states in the data
2. Cluster transitions using the lifted effects
3. Learn preconditions for each cluster
Learning Symbolic Operators: Approach

1. Convert low-level states into abstract states in the data
2. Cluster transitions using the lifted effects
3. Learn preconditions for each cluster
4. Make an operator for each cluster

What is given?

Data

$x_0, c_1, x_1$

....

$x_{n-1}, c_n, x_n$

Controllers

Move(o)

Pick(o)

Place(o)

Predicates

At(?o – obj ?a – area)

RobotAt(?a – area)

Holding(?o – obj)

Abstract Data

$s_0, c_1, s_1$

....

$s_{n-1}, c_n, s_n$
TMP with Learned High-level Actions

TMP framework requires:
- Abstract states
- Abstract actions
- Low-level behaviors / motion planner
- Samplers / generators
TMP with Learned High-level Actions

TMP framework requires:

Abstract states -- Input
Abstract actions -- Learned
Low-level behaviors / motion planner -- Input
Samplers / generators -- Input
### High-Level Summary

<table>
<thead>
<tr>
<th>Skills to Symbols</th>
<th>Abstract States</th>
<th>Abstract Actions</th>
<th>Low-level behaviors</th>
<th>Samplers</th>
<th>C-Space changes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Learned</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Yes</td>
</tr>
</tbody>
</table>

| Silver et al. 2021 | Input           | Learned          | Input               | Input    | Yes             |
Learning NSRTs

• Core idea: learn high-level abstract actions in the form of NSRTs
• NSRTs = Neuro-symbolic relational transitions

• NSRTs include:
  • High-level action components: parameters, preconditions, effects
  • Low-level reactive policy: $\pi(a|x)$

• What is provided:
  • Predicates -- state abstraction
  • $f: X \times A \rightarrow X$ a known low-level deterministic transition function

Learning NSRTs: Overall Approach

Data: \{(s, a, s')\}

Partition the data (Section 6.1)

Data for Partition 1
- Symbolic Learning (Section 6.2)
  - NSRT 1 (pick with side grasp)
    - Params: (?robot, ?obj)
    - Pre: Graspable(?obj)
      - HandEmpty(?robot)
    - Eff: HoldingSide(?robot, ?obj)
  - Neural low-level transition model
  - Neural action sampler

Data for Partition 2
- Symbolic Learning (Section 6.2)
  - NSRT 2 (place into shelf)
    - Params: (?robot, ?obj)
    - Pre: HoldingSide(?robot, ?obj)
    - Eff: HandEmpty(?robot)
      - InShelf(?obj)
  - Neural low-level transition model
  - Neural action sampler

Symbolic planning
1. NSRT1(robby, obj1)
2. NSRT2(robby, obj1)

A* Search

Continuous planning with neural networks
- obj1.pose
- obj1.mass
- obj1.held
- robby.conf

.action 7.2, 3.5, 2.3
- [2, 2, 2] 50 False [0, 0, 0]

.action 5.8, 6.3, 9.1
- [3, 4, 3.5] 50 True [1.8, 6, 2]

.action 5.2, 6.7
- [5.2, 6, 7] 50 False [0, 0, 0]
Learning NSRTs: Creating Data Partitions and Effects

What is given?

Data
\( x_0, a_1, x_1 \)
\( \ldots \)
\( x_{n-1}, a_n, x_n \)

Predicates
\( \text{At}(?o \rightarrow \text{obj} \ ?a \rightarrow \text{area}) \)
\( \text{RobotAt}(?a \rightarrow \text{area}) \)
\( \text{Holding}(?o \rightarrow \text{obj}) \)

Data
\( x_0, a_1, x_1 \)
\( \ldots \)
\( x_{n-1}, a_n, x_n \)

Abstract Data
\( s_0, a_1, s_0 \)
\( \ldots \)
\( s_{n-1}, a_n, s_n \)

Partition 1
\( s_0, a_1, s_0 \)
\( \ldots \)
\( s_0, a_n, s_1 \)
What is given?

Data

\[ x_0, a_1, x_1 \]

\[ \ldots \]

\[ x_{n-1}, a_n, x_n \]

Predicates

At(o – obj ?a – area)

RobotAt(?a – area)

Holding(?o – obj)

Data

\[ s_0, a_1, a \]

\[ \ldots \]

\[ s_{n-1}, a_n, s_n \]

Abstract Data

Partition 1

Partition 2

Partition 3
Learning NSRTs: Learning Preconditions

What is given?

Data
\[ x_0, a_1, x_1 \]
\[ \ldots \]
\[ x_{n-1}, a_n, x_n \]

Predicates
\[ \text{At(?o – obj ?a –area)} \]
\[ \text{RobotAt(?a – area)} \]
\[ \text{Holding(?o – obj)} \]

Abstract Data
\[ s_0, a_1, s_0 \]
\[ \ldots \]
\[ s_{n-1}, a_n, s_n \]
Learning NSRTs: Reactive Low-Level Policy

What is given?

Data

\( x_0, a_1, x_1 \)

\( \ldots \)

\( x_{n-1}, a_n, x_n \)

Predicates

At(?o – obj ?a – area)

RobotAt(?a – area)

Holding(?o – obj)

Partition 1

Low-level Data

\( s_0, a_1, s_0 \)

\( \ldots \)

\( s_0, a_n, s_1 \)

\( x_0, a_1, x_1 \)

\( \ldots \)

\( x_{n-1}, a_n, s_n \)

Learn a regression model \( \pi(a|x) \)
## High-Level Summary

<table>
<thead>
<tr>
<th>Skills to Symbols</th>
<th>Abstract States</th>
<th>Abstract Actions</th>
<th>Low-level behaviors</th>
<th>Samplers</th>
<th>C-Space changes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Learned</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Yes</td>
</tr>
<tr>
<td>Silver et al. 2021</td>
<td>Input</td>
<td>Learned</td>
<td>Input</td>
<td>Input</td>
<td>Yes</td>
</tr>
<tr>
<td>Silver et al. 2022</td>
<td>Input</td>
<td>Learned</td>
<td>Learned</td>
<td>Learned</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Silver et al. 2021 and Silver et al. 2022 refer to the works by Silver et al. in 2021 and 2022, respectively.
Learning State and Action Abstractions
HARP: A Neuro-Symbolic Motion Planner

- Core idea: Learn to zero-shot create state and action abstractions simultaneously using critical regions

- What is provided?
  - A set of training environments
  - A random problem generators (random initial and goal configuration)
  - A motion planner

- What is learned?
  - A method to zero-shot create state and action abstractions for unseen environments
Critical Regions

Given a class of motion planning problem $M$, criticality of an open set $r$ in the C-space:

$$\mu(r) = \lim_{s_n \to +\nu} \frac{f(r)}{v(s_n)}$$

$f(x)$ = fraction of solution plans that pass through $x$ // captures hubs
$v(x)$ = measure of $x$ under uniform sampling density // captures bottlenecks.

[Molina et al., 2020, ICRA]
HARP: Learning to Zero-Shot Predict CRs

\[ n = \#\text{degrees of freedom (DOFs)} \]
\[ k = \#\text{DOFs that are not determined by the end-effector's location in the workspace} \]

end-effector = base link for navigation
end-effector = gripper link for manipulation

\[ I_1 \]

Occupancy Matrix

Critical regions for each DOF

UNet\(^1\)

\[ L_1, L_2, L_3, \ldots, L_m, \ldots, L_{k+1} \]

\[ k \text{ channels} \]

---

HARP: Training Data

Training Data

Input

CRs for base location

CRs for base rotation

Input

CRs for base location

CRs for base rotation

CRs for hinged angle
HARP: Learning to Zero-Shot Predict CRs

\[ n = \# \text{degrees of freedom (DOFs)} \]
\[ k = \# \text{DOFs that are not determined by the end-effector's location in the workspace} \]

end-effector = base link for navigation
end-effector = gripper link for manipulation

HARP: Constructing Zero-Shot Abstractions

• Given a robot and new environment,
  Predict CRs
HARP: Constructing Zero-Shot Abstractions

- Given a robot and new environment,
  Predict CRs
  Construct Voronoi diagrams around CRs
HARP: Constructing Zero-Shot Abstractions

- Given a robot and new environment,
  - Predict CRs
  - Construct Voronoi diagrams around CRs

Abstract states = Voronoi cells

2D projection of RBVDs
HARP: Constructing Zero-Shot Abstractions

• Given a robot and new environment,
  Predict CRs
  Construct Voronoi diagrams around CRs

Abstract states = Voronoi cells
Abstract actions = transitions between abstract states

2D projection of RBVDs
HARP: Hierarchical Motion Planning

- Given a robot and new environment,
  Predict CRs
  Construct Voronoi diagrams around CRs

  Abstract states = Voronoi cells
  Abstract actions = transitions between abstract states

  Hierarchical motion planning using:
  a high-level multi-source bi-directional beam search
  a multi-source multi-directional LLP mp

2D projection of RBVDs
HARP: Experiments

Rectangular robot
Hinged robot
Car-Rectangular robot
Fetch 8-DOF manipulator
HARP: Experiments

Uses CRs + abstractions

Uses CRs

Formal results: downward refinability for holonomic robots; soundness; probabilistic completeness

Rectangular robot
Hinged robot
Car-R Rectangular robot
Fetch 8-DOF manipulator
SHARP: What if Robot Dynamics are Stochastic?

Objective: Compute a motion plan
    Compute a motion policy
Objective: Compute a motion plan
  Compute a motion policy

High-level actions = Options
SHARP: What if Robot Dynamics are Stochastic?

Objective: Compute a motion plan
  Compute a motion policy

High-level actions = Options

Option guide = autogenerated dense pseudo-reward
SHARP: What if Robot Dynamics are Stochastic?

Objective: Compute a motion plan

  Compute a motion policy

High-level actions = Options

Option guide = autogenerated dense pseudo-reward

  Autogenerated reward shaping
SHARP: What if Robot Dynamics are Stochastic?

Objective: 

Compute a motion plan
Compute a motion policy

High-level actions = Options

Option guide = autogenerated dense pseudo-reward
Autogenerated reward shaping
SHARP: Experiments

Test Environments

Navigation
15m x 15m
S1, S2

75m x 75m
L1, L2, L3

Manipulation
M1, M2

Test Robots

Husky
Fetch

Limo
SHARP performs well (times include creation of state and action abstractions).

Next-best: RRT-replan! Other baselines struggle to learn

Hypothesis: not suited for stochasticity, long horizons, sparse rewards
SHARP: Results

Bars indicate solution length in number of steps (lower is better)
Pies indicate % success (darker is better)
## High-Level Summary

<table>
<thead>
<tr>
<th>Skills to Symbols</th>
<th>Abstract States</th>
<th>Abstract Actions</th>
<th>Low-level behaviors</th>
<th>Samplers</th>
<th>C-Space changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learned</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Yes</td>
</tr>
<tr>
<td>Silver et al. 2021</td>
<td>Input</td>
<td>Learned</td>
<td>Input</td>
<td>Input</td>
<td>Yes</td>
</tr>
<tr>
<td>Silver et al. 2022</td>
<td>Input</td>
<td>Learned</td>
<td>Learned</td>
<td>Learned</td>
<td>Yes</td>
</tr>
<tr>
<td>HARP/SHARP</td>
<td>Learned</td>
<td>Learned</td>
<td>Learned / Computed</td>
<td>Learned</td>
<td>No</td>
</tr>
</tbody>
</table>
Outline

1. Background: Why Task and Motion Planning?
2. Abstraction as a Foundation for TMP
3. Abstraction-based Approaches
4. Research Frontier: Neuro-Symbolic Learning for TMP

What next?
Open Questions

• Stochasticity at the low-level
Open Questions

• Stochasticity at the low-level
• Partial observability
Open Questions

- Stochasticity at the low-level
- Partial observability
- Learning abstractions for TMP -- What if *neither state nor actions* are not provided?

Temporal Abstraction
≡ Abstract Actions, Macros, Options…

Task N’ Motion

- GoTo(l)
- Pickup(x)
- PutDown(x)

State Abstraction

- At(x, l)
- InGripper(x)
- AtDestination(x)
Open Questions

- Stochasticity at the low-level
- Partial observability
- Learning abstractions for TMP -- What if state and actions are not provided?
- Orthogonal direction: Pose estimation (understanding low-level observational data)