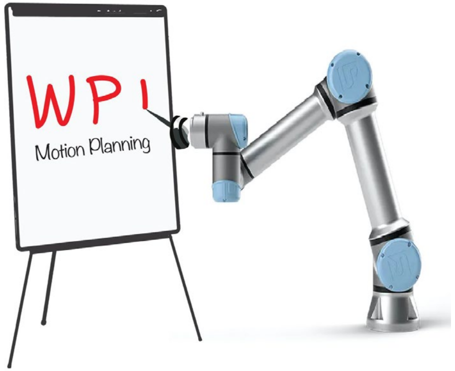
RBE550

Motion Planning

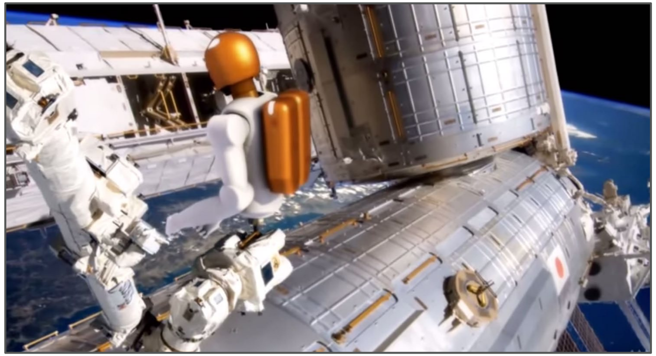
Task and Motion Planning



Constantinos Chamzas
www.cchamzas.com
www.elpislab.org

Motivation

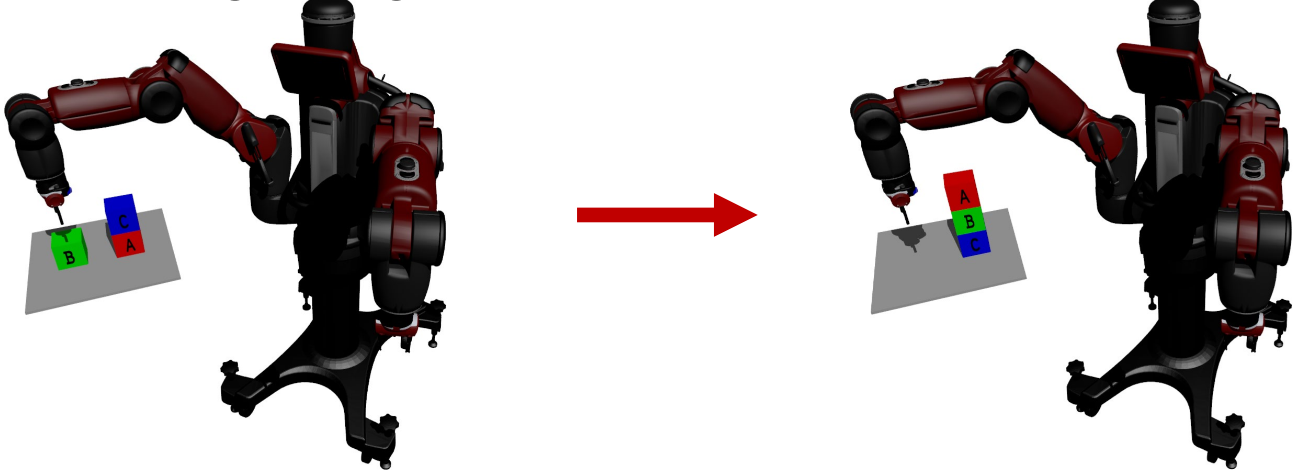
- Want robot to perform complex tasks autonomously:
  - e.g., Household robot to clean the house
  - e.g., Robonaut to fix and replace components on the exterior of ISS
- **Given high-level task description, have the robot achieve this task!**



Task and Motion Planning

Informal	General Case
Tell robot where everything is	Initial state
Tell it where everything needs to end up	Goal state(s)
Let the robot figure it out	Transitions between states

Design an algorithm to find a solution if one exists.



Previous Lecture

- Focused on task planning
- Example of task planning with discrete state-space search
- Overview of how we can describe task planning problems using formal logics
- Heuristic search methods for task planning

Overview

- Review of task planning
- SAT Planning for task planning
- Task and Motion Planning problem
- Approach to Task and Motion Planning
- Case study using TMKit

Task and Motion Planning

- **The problem of searching for a robot trajectory:**
  - Trajectory should correspond to a discrete sequence of actions that satisfies the task specification.
  - Continuous trajectory should be collision-free.

Task Planning Problem with Boolean Logic

- **Given:**
  - First-Order Logic Relations
  - First-Order Logic Functions
  - Actions (with preconditions and effects)
  - Start State
  - Goal State(s)
- **Find:**
  - Sequence of actions from start to goal

Planning Domain Definition Language

- Planning Domain Definition Language (PDDL) is a way to express a task planning problem
- Use prefix notation:
  - `(+ 1 2)` returns `3`; `(+ 1 2 (* 3 4))` returns `15`
- Define:
  - Predicates: `on(?x,?y)`, `clear(?x)`, etc.
  - Actions: `stack(?x,?y)`, etc.
  - Initial state: `(ontable a)`, etc.
  - Goal state: `(on a b)`, `(on b c)`, `(ontable a)`

Planning Domain Definition Language

Operators

```
(define (domain blocks)
  (:predicates (on ?x ?y) (ontable ?x)
               (clear ?x) (handempty) (holding ?x))
  (:action pick-up  :parameters (?x)
           :precondition (and (clear ?x) (ontable ?x)
                              (handempty))
           :effect (and (not (ontable ?x))(not (clear ?x))
                        (not (handempty)) (holding ?x)))
  (:action put-down  :parameters (?x)
           :precondition (holding ?x)
           :effect (and (not (holding ?x)) (clear ?x)
                        (handempty)(ontable ?x)))
  (:action stack  :parameters (?x ?y)
           :precondition (and (holding ?x) (clear ?y))
           :effect (and (not (holding ?x))(not (clear ?y))
                        (clear ?x)(handempty) (on ?x ?y)))
  (:action unstack  :parameters (?x ?y)
           :precondition (and (on ?x ?y) (clear ?x)
                              (handempty))
           :effect (and (holding ?x) (clear ?y)
                        (not (clear ?x))
                        (not (handempty))
                        (not (on ?x ?y)))))
```

Facts

```
(define
 (problem sussman-anomaly)
  (:domain blocks)
  (:objects a b c)
  (:init (on c a)
         (ontable a)
         (ontable b)
         (clear c)
         (clear b)
         (handempty))
  (:goal (and (on b c)
              (on a b))))
```

Grounding on First-Order Logic

- Rely on the fact that there are finite:
  - Constants
  - Objects
  - locations
  - steps
- Construct a list of things that could be true or false for each time step
  - `At(Obj?, Loc?) → PLATE_2_AT_LOC_3`
  - `At(Obj?, Loc?) → PLATE_1_AT_LOC_5`
  - Etc.

Graph Search Approach

- Search state space to find a sequence of actions from start to goal
- Find successor states by looking at all actions that can be applied from a given state:
  - this means preconditions are met
- Search until we reach the goal:
  - this search space is typically very large
  - general heuristic search for task planning is an open research problem
  - use heuristic search
    - e.g., A*, Dijkstra's, FF, ...

Task Planning Approaches

- **Graph approach (employed so far)**
  - From PDDL description build a graph
  - Apply heuristic search algorithms
    - Search in graph implied by logical description
    - Heuristics are essential for good performance

- **Another approach that can be very efficient**
  - Reduce planning to a SAT problem instance (Boolean logic)
  - Solve with off-the-shelf solver
  - Avoid relying on task planning heuristics

Overview

- Review of task planning
- SAT Planning for task planning
- Task and Motion Planning problem
- Approach to Task and Motion Planning
- Case study using TMKit

SAT

- **Boolean satisfiability** problem, also known as **SAT**:
  - **Given**
    - Boolean formula $f(x_1, \dots, x_n): 2^n \rightarrow \{0,1\}$
  - **Find**
    - An assignment $x_1, \dots, x_n$ such that $f(x_1, \dots, x_n) = 1$
- Complexity: NP-Complete
  - SAT is the first problem that was proven to be NP-complete
  - all problems in the complexity class NP are at most as difficult as SAT
- Fast solvers with general SAT heuristics, e.g., Z3

SAT example

1. If it is raining, then the ground is wet.
2. The ground is wet.

---

3. *Therefore, it is raining. (Invalid inference)*

- R = it is raining; W = the ground is wet

```
1.  R → W
2.  W
```

```
3.  R (Invalid Inference)
```

$$(((R \rightarrow W) \wedge W) \wedge (\neg R))$$

- Can test validity by checking if the conclusion can be false when the premises are true
- If the argument is valid the following formula must always be false, otherwise we have falsified the argument

What assignment will satisfy(falsify argument) this formula

- R=0, W=1
- R=1, W=1
- R=0 W=0
- R=1, W=0

What assignment will satisfy(falsify argument) this formula

$$(((R \rightarrow W) \wedge W) \wedge (\neg R))$$

R=0, W=1

0%

R=1, W=1

0%

R=0 W=0

0%

R=1, W=0

0%

What assignment will satisfy(falsify argument) this formula

R=0, W=1

0%

R=1, W=1

0%

R=0 W=0

0%

R=1, W=0

0%

Boolean Satisfiability

- Example: (test)
  - Given an assignment, plug it into the formula

$$
\begin{gathered}
\mathrm{R} = 0,\ \mathrm{W} = 1 \\
(((\mathrm{R} \rightarrow W) \wedge W) \wedge (\neg R)) \\
(((0 \rightarrow 1) \wedge 1) \wedge (\neg 0)) \\
((1 \wedge 1) \wedge (\neg 0)) \\
(1 \wedge (\neg 0)) \\
(1 \wedge 1) \\
1
\end{gathered}
$$

Assignment to the variables

Evaluation of formula (0 or 1)

Boolean Satisfiability

- Example: (proof)
  - Given an assignment, plug it into the formula

$$\mathrm{R} = 0,\ \mathrm{W} = 1$$

$$(((\mathrm{R} \rightarrow W) \wedge W) \wedge (\neg R))$$

$$(((0 \rightarrow 1) \wedge 1) \wedge (\neg 0))$$

$$((1 \wedge 1) \wedge (\neg 0))$$

$$(1 \wedge (\neg 0))$$

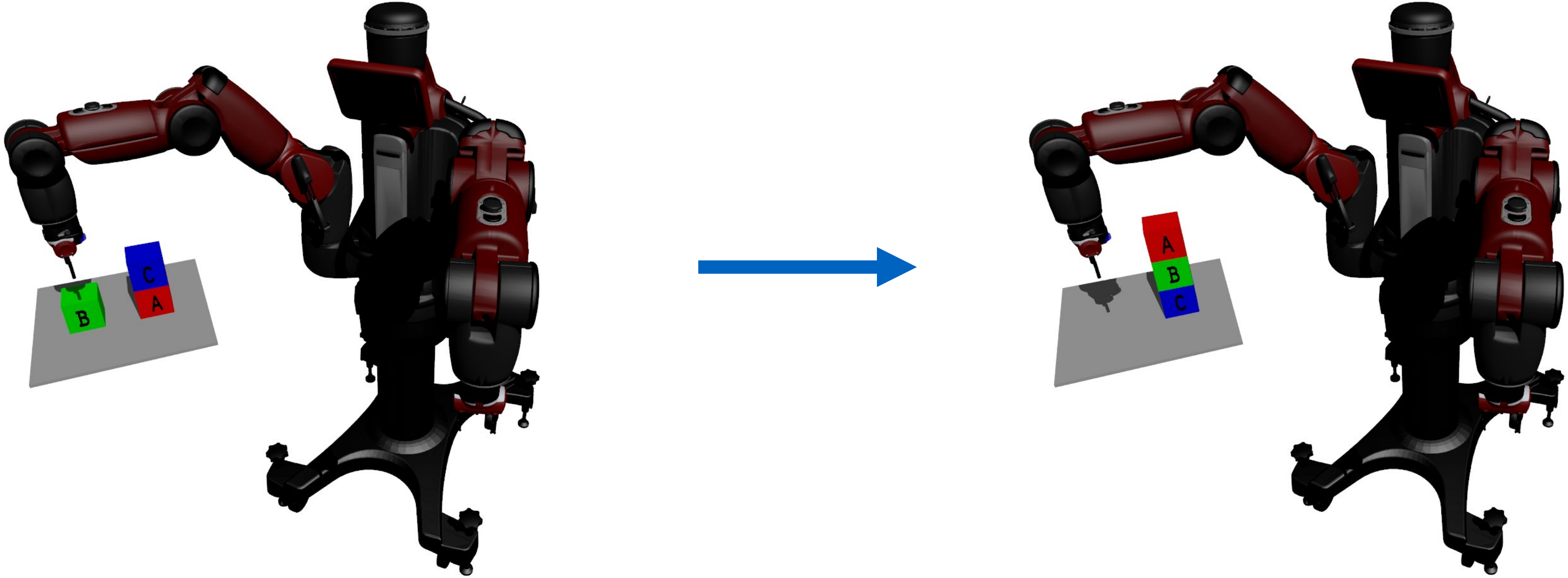
$$(1 \wedge 1)$$

$$1$$

Assignment to the variables

Evaluation of formula (0 or 1)

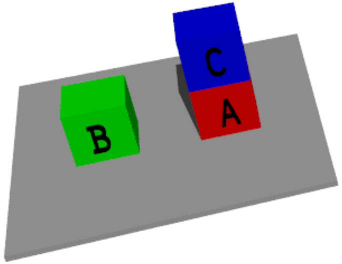
Task Specification

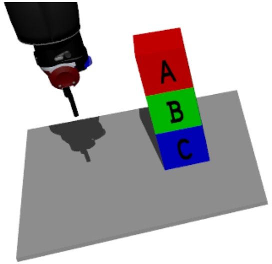


SAT- Plan

- Encode planning problem as Boolean formula
  - Consider a bounded number of steps
  - Formula over per-step includes state variables plus selected action
- Call a SAT-solver on the formula
  - If SAT: return action assignment
  - If UNSAT: increment step horizon and repeat

Task Specification





B-is-on-Table AND
C-is-on-A AND
A-is-on-Table

(NOT A-is-on-B) AND
(NOT B-is-on-C) AND
(NOT C-is-on-TABLE)

all Boolean variables

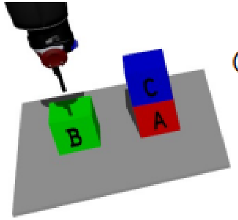
A-is-on-B AND
B-is-on-C AND
C-is-on-Table

(NOT B-is-on-Table) AND
(NOT C-is-on-A) AND
(NOT A-is-on-Table)

Task Specification

- Task operators
  - State: defined with logical formulas
  - Actions (operators): define changes in state, e.g., `unstack(?x, ?y)`
- Example:
  - Unstacking blocks

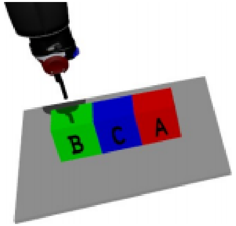
How can we represent things like this with Boolean variables?



On(B,Table) ∧
On(C,A) ∧
On(A,Table) ∧
Clear(B) ∧
Clear(C)

Unstack(C,A)

On(B,Table) ∧
On(C,Table) ∧
On(A,Table) ∧
Clear(A) ∧
Clear(B) ∧
Clear(C)



SAT-Plan Encoding

- We want to feed a problem into the SAT solver
  - If the problem is SAT,
    - We get an assignment we can convert to a task plan
  - If the problem is UNSAT,
    - it means there is no task plan within that time horizon
- We need to eliminate quantifiers $\forall$ and $\exists$
  - Quantifiers are used with variables
  - so we will ground variables and time steps

SAT-Plan Encoding

1. (Initial State) Need to be in the start state at step 0:

    - Start State: `start-step-0`

2. (Goal State) Need to get to the goal by step k (a no-op can be used if we get there too early):

    - Goal State: `goal-step-k`

3. (Operator Encodings) Applying an operator at step i implies preconditions and, on the next step, effects

    - `op-step-i` → `pre(op)-step-i` ∧ `eff(op)-step-(i+1)`

SAT-Plan Encoding

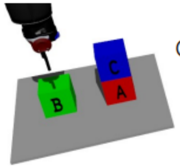
- 4. (Frame Axioms) Nothing changes unless we change it:
  - Frame Axioms: `(P-step-i == P-step-(i+1)) ∨ (op-P-0 ∨… ∨ op-P-n)`
- 5. (Complete Exclusion Axion) Can take only one action per step:
  - Operator Exclusion: `op-0 → ((¬ op-1) ∧ … ∧ (¬ op-n))`
- We ground all possible operators and predicates:
  - Relations: `rel_0-arg_0, …, rel_0-arg_K, …, rel_M-arg_0, …, rel_M-arg_K`
  - Operators: `op_0-arg_0, …, op_0-arg_K, …, op_M-arg_0, …, op_M-arg_K`

SAT-Plan Encoding

- Assume a value k for the number of steps in the solution
- Construct a huge formula with is the conjunction of 1-5 above
- A solution plan exists if and only if the resulting Boolean formula is satisfiable
- Give formula to a SAT solver
- Get the values that need to be assigned truth values and these will indicate the steps than need to be taken (steps is indicated in the encoding)

SAT-Plan Encoding

- Start State: `start-step-0`
- Goal State: `goal-step-k`
- Operator:
  - `Unstack_C_A_step_0` → `pre_unstack_C_A_step_0` ∧ `eff_unstack_C_A_step_1`
- Frame axiom:
  - `(on_C_A_step_0 == on_C_A_step_1)` ∨ `(unstack_C_A_step_0)` ∨ …
- Operator exclusion:
  - `Unstack_C_A_step_0` → `(¬ stack_C_B_step_0 )` ∧ `(¬ stack_B_C_step_0)` ∧ …

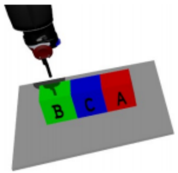


start

```
On(B,Table) ∧
On(C,A) ∧
On(A,Table)∧
Clear(B) ∧
Clear(C)
```

`Unstack(C,A)` →

```
On(B,Table) ∧
On(C,Table) ∧
On(A,Table)∧
Clear(A) ∧
Clear(B) ∧
Clear(C)
```



goal

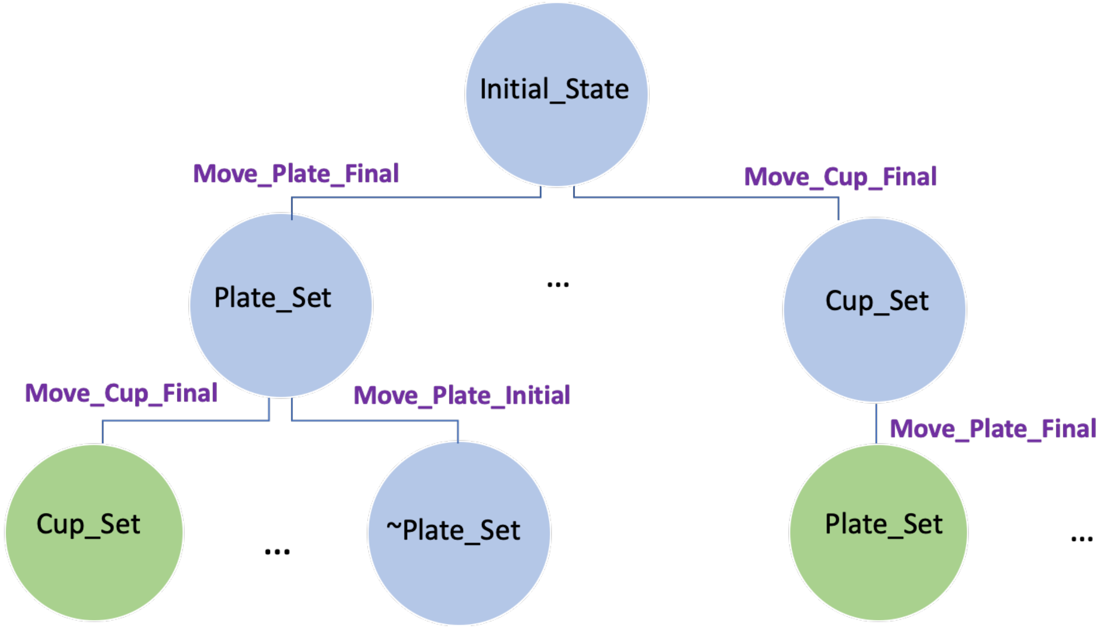
Overview

- Review of task planning
- SAT Planning for task planning
- Task and Motion Planning problem
- Approach to Task and Motion Planning
- Case study using TMKit

Task and Motion Planning

- A naïve approach
  - Find a task plan (using e.g., A* search or SAT plan)
  - For each action in the task plan, solve the motion planning problem
  - Concatenate these motion plans together to solve the TMP problem

Task and Motion Planning



Initial_State:

¬ Plate_Set

¬ Cup_Set

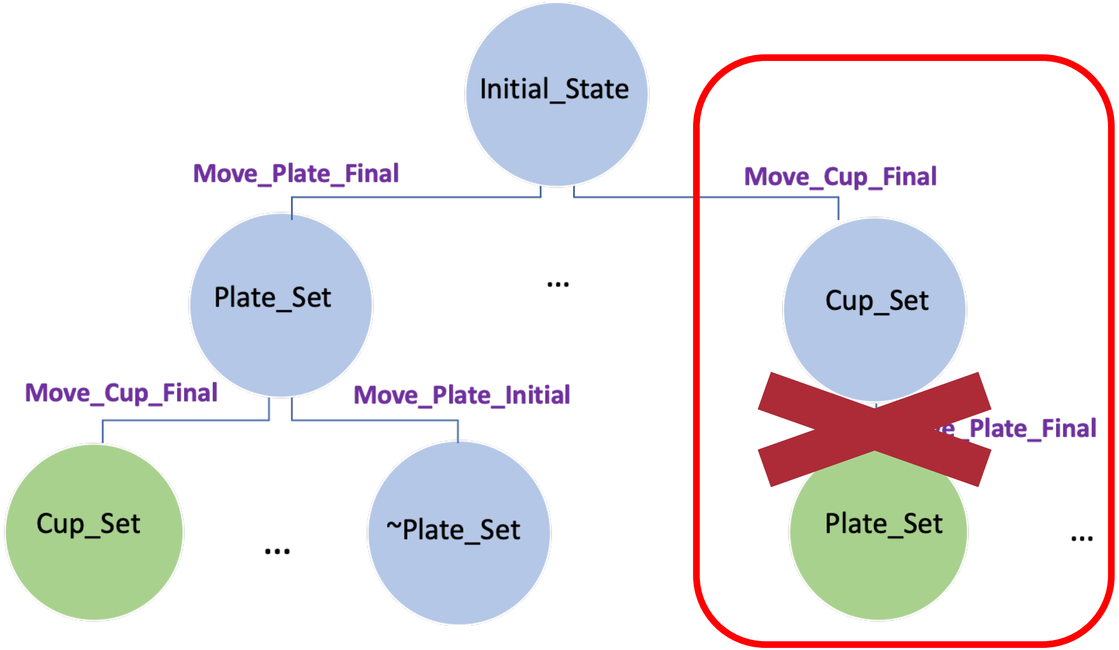
Plate_Initial

Cup_Initial

Hand_Free

…

Task and Motion Planning



Initial_State:

¬ Plate_Set

¬ Cup_Set

Plate_Initial

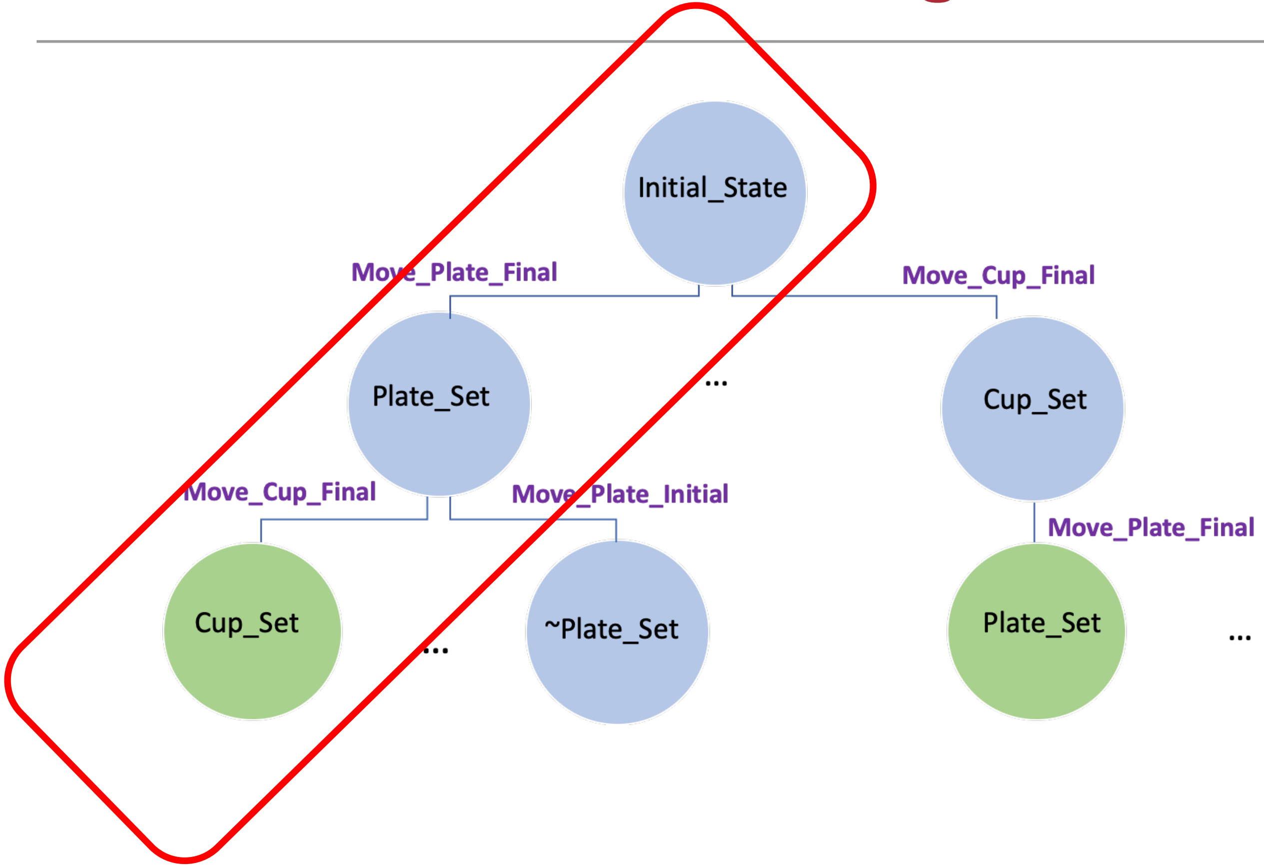
Cup_Initial

Hand_Free

…

We need a way to recover from this !

Task and Motion Planning



Initial_State:

¬ Plate_Set

¬ Cup_Set

Plate_Initial

Cup_Initial

Hand_Free

…

Overview

- Review of task planning
- SAT Planning for task planning
- Task and Motion Planning problem
- Approach to Task and Motion Planning
- Case study using TMKit

Task and Motion Planning

- A naïve approach
  - Find a task plan (using e.g., FF search or SAT plan)
  - For each action in the task plan, solve the motion planning problem
  - Concatenate these motion plans together to solve the TMP problem
- Problems with naïve approach:
  - If the motion plan is infeasible (no solution exists), the algorithm fails
  - Instead, we want to try a different task plan
  - Typical algorithms for motion planning are probabilistically complete (we know when we succeed, but we don’t know if the instance is infeasible)
  - We want some sort of feedback between the task planning and the motion planning algorithms

Task and Motion Planning

- An alternative approach:
  - Find a task plan
  - For each action in the task plan, *solve the motion planning problem*
  - If motion planning fails, try another task plan
  - If we can find motion plans for all actions, concatenate them together and return this plan as the solution
  - Continue until we succeed or timeout

When do we stop motion planning?

Task and Motion Planning

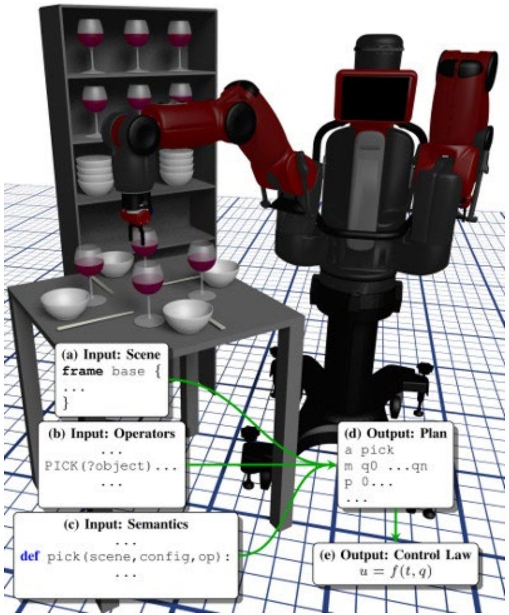
- A probabilistically-complete approach can be obtained by
  - Find a task plan
  - For each action in the task plan, *attempt to solve the motion planning problem*
  - If motion planning fails, try another task plan
  - Eventually revisit motion planning (with a longer time horizon)
  - If we can find motion plans for all actions, concatenate them together and return this plan as the solution
  - Continue until we succeed or timeout

Overview

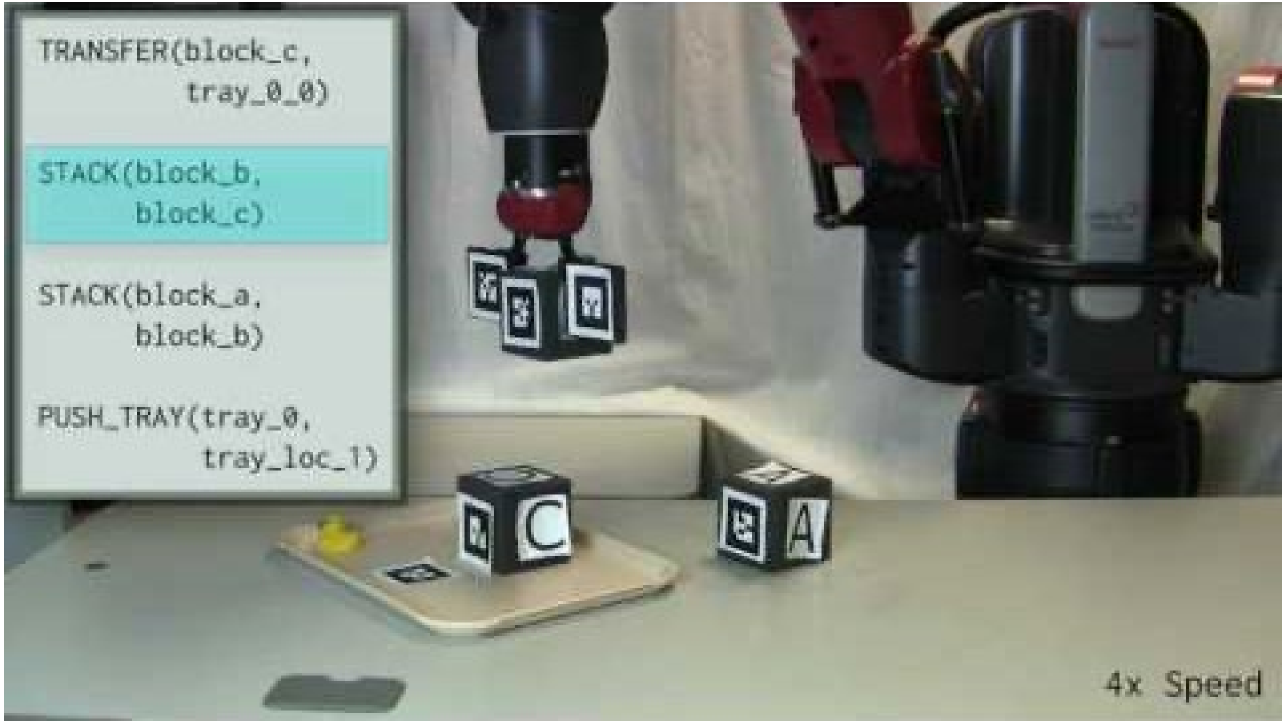
- Review of task planning
- SAT Planning for task planning
- Task and Motion Planning problem
- Approach to Task and Motion Planning
- Case study using TMKit

Case Study

- TMKit
  - Probabilistically complete tool for task and motion planning.
  - Developed by Neil Dantam at Colorado School of Mines
  - Available for download:
    - http://tmkit.kavrakilab.org/

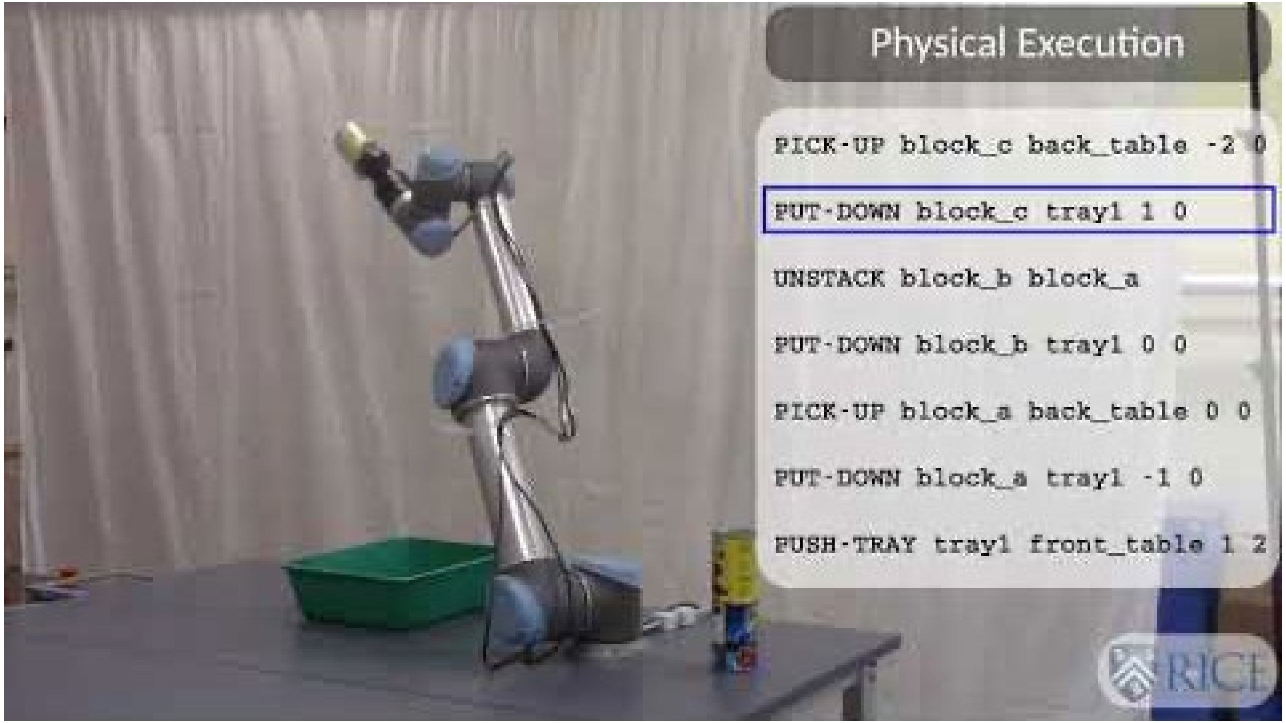


Case Study

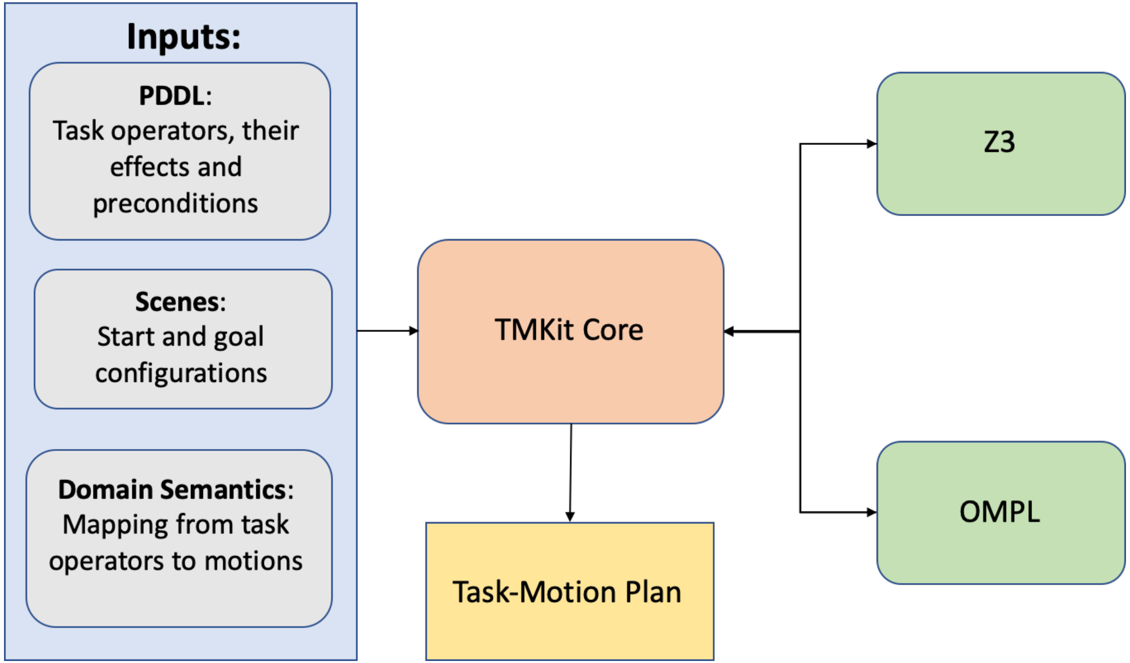


TRANSFER(block_c,
tray_0_0)
STACK(block_b,
block_c)
STACK(block_a,
block_b)
PUSH_TRAY(tray_0,
tray_loc_1)
C
A
4x Speed

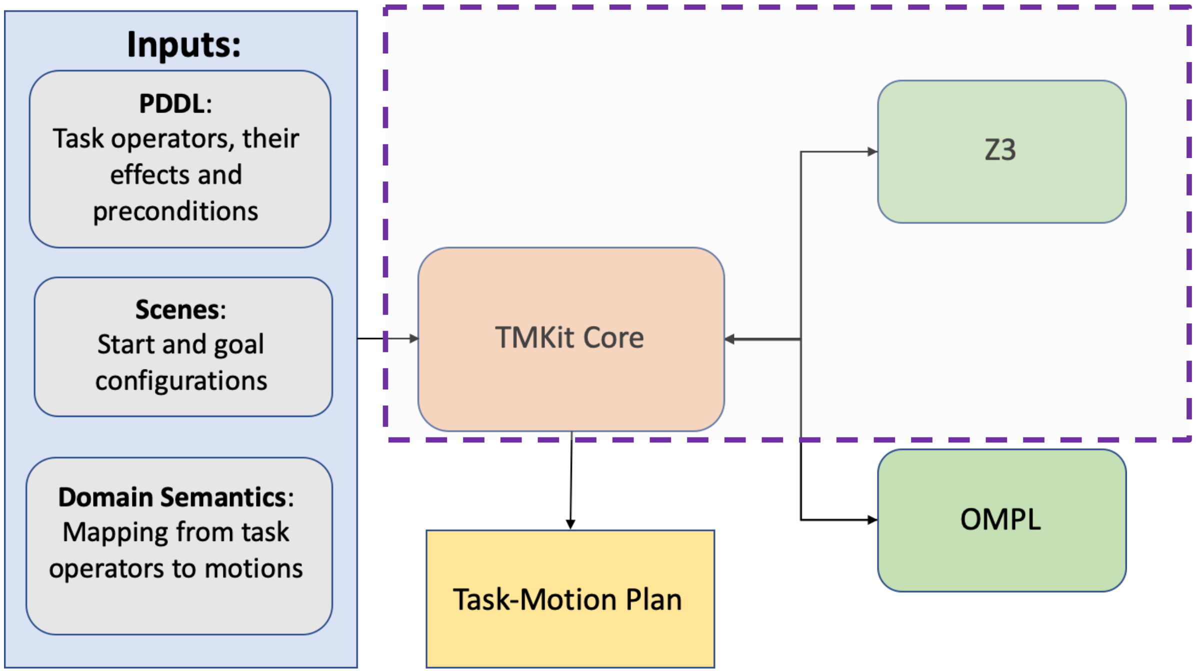
Case Study


Physical Execution
PICK-UP block_c back_table -2 0
PUT-DOWN block_c tray1 1 0
UNSTACK block_b block_a
PUT-DOWN block_b tray1 0 0
PICK-UP block_a back_table 0 0
PUT-DOWN block_a tray1 -1 0
PUSH-TRAY tray1 front_table 1 2
RICE

Case Study – TMKit architecture



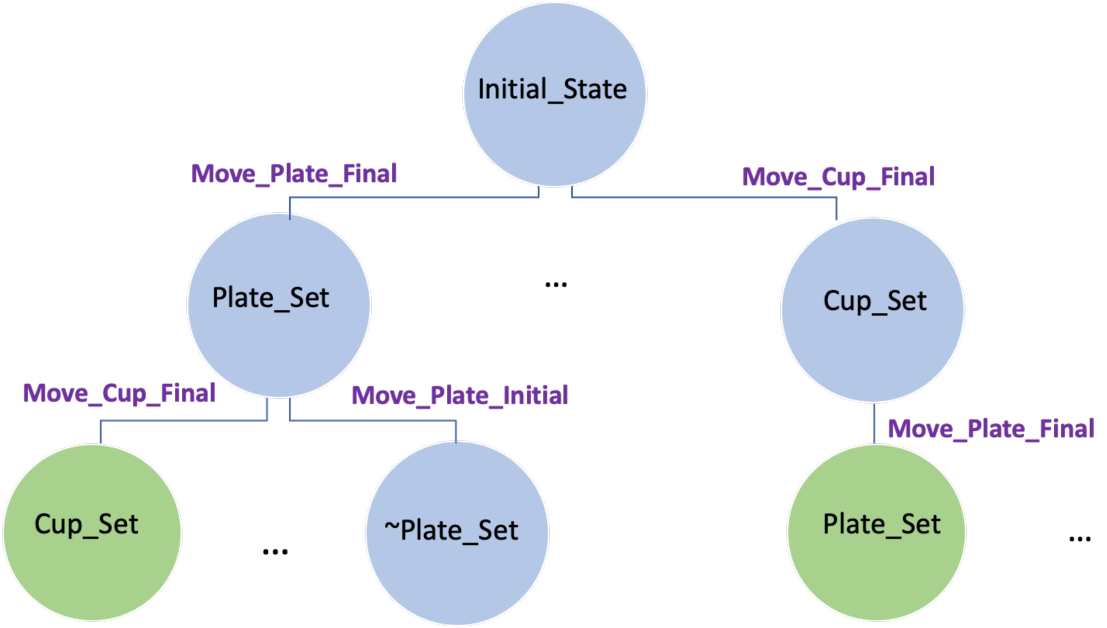
TMKit - Task Planning



TMKit – Task Planning

- Conducts discrete search with SAT planning using Z3.
- Sets a time horizon and search for a plan within that time horizon
- Finds an assignment satisfying the following conjunction:
  - Formula describing initial state
  - Formula describing goal state
  - Formulas describing preconditions and effects of actions
  - Formulas describing complete exclusion (one action per time step)
  - Formulas providing a solution to the frame problem (changes actions)
- Utilizes incremental solving of Z3 to find additional plans

Task and Motion Planning



Initial_State:

¬ Plate_Set

¬ Cup_Set

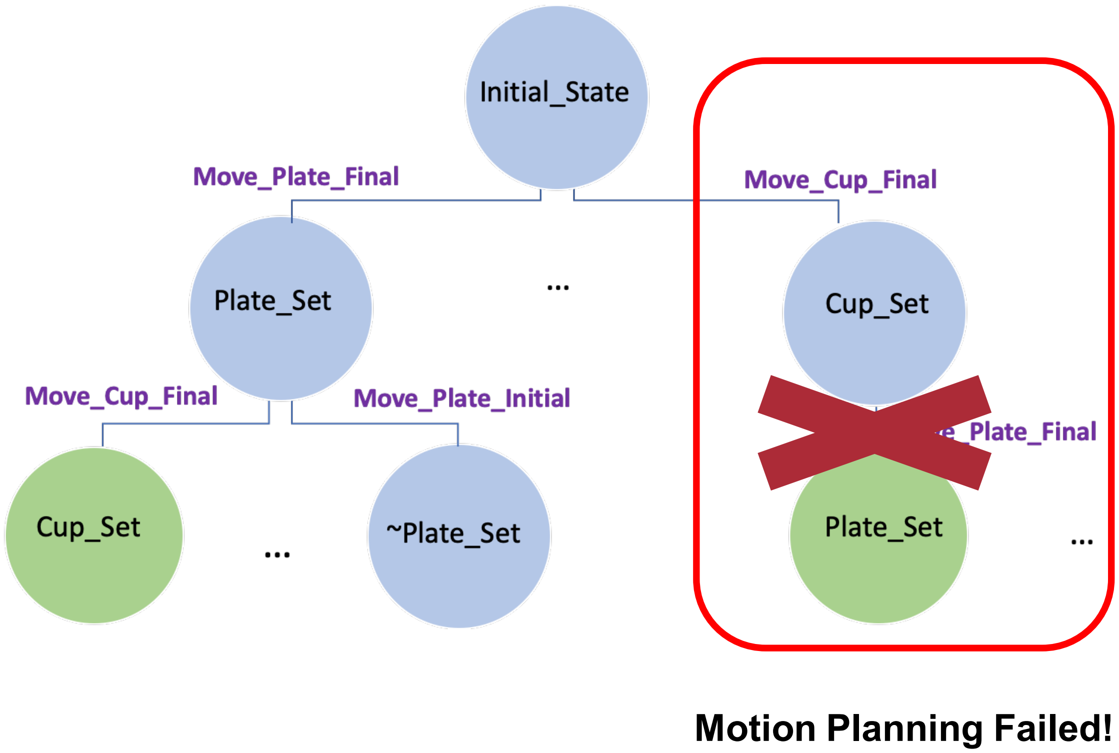
Plate_Initial

Cup_Initial

Hand_Free

…

Task and Motion Planning



Initial_State:

¬ Plate_Set

¬ Cup_Set

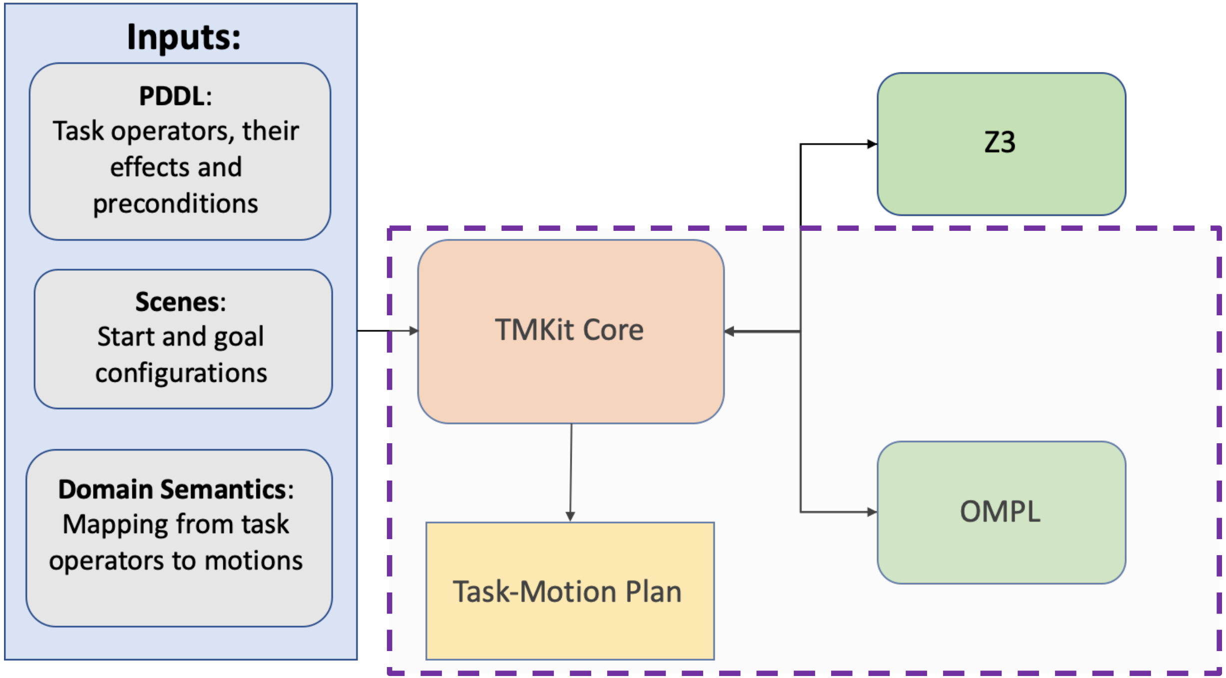
Plate_Initial

Cup_Initial

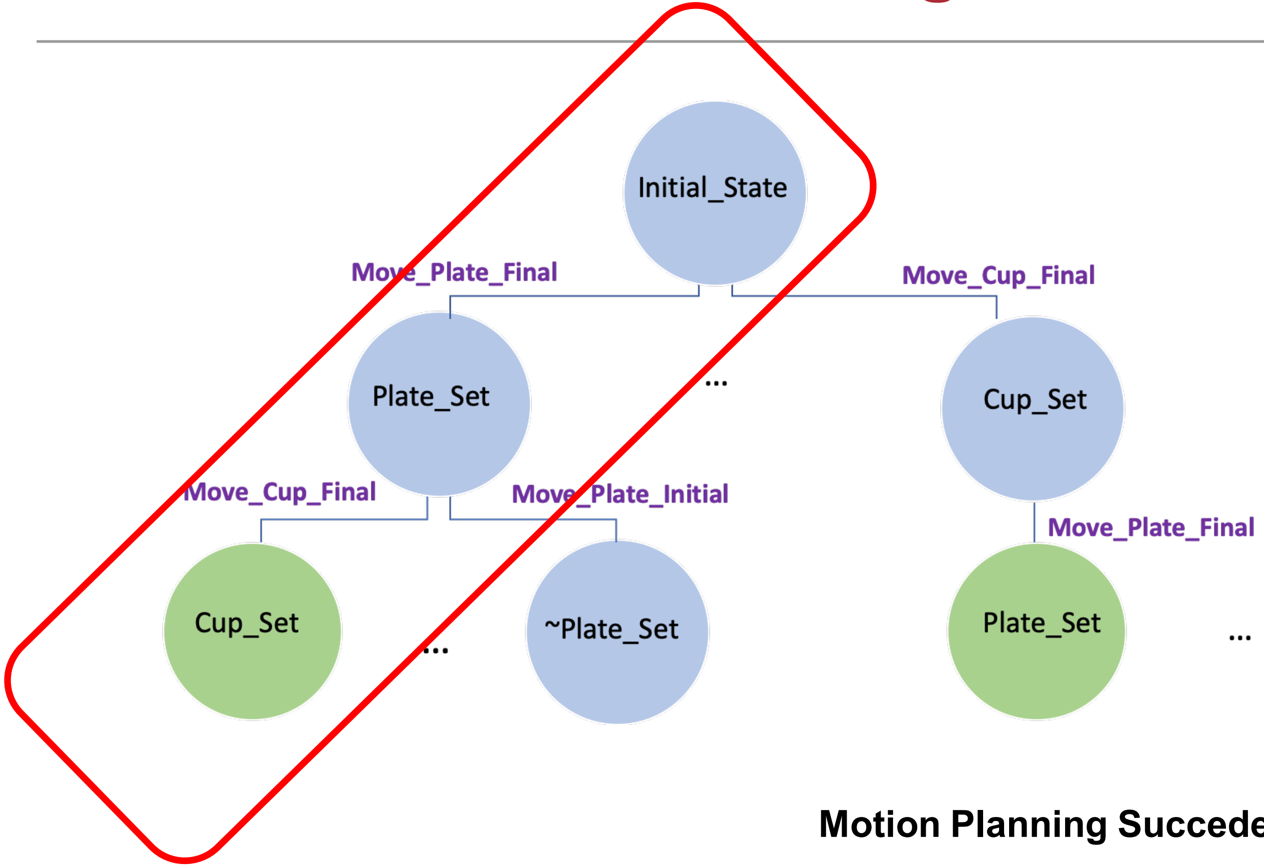
Hand_Free

…

TMKit - Motion Planning



Task and Motion Planning



Initial_State:

¬ Plate_Set

¬ Cup_Set

Plate_Initial

Cup_Initial

Hand_Free

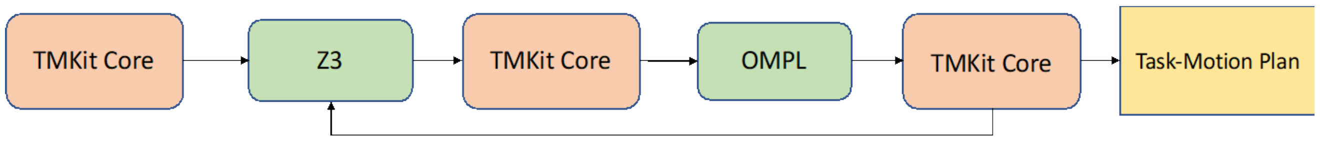
…

Motion Planning Succeded!

TMKit- Motion Planning

- Sampling-based motion planning
  - We can use different motion planners
  - Which ones make the most sense?
    - Domain knowledge!
- Issues:
  - Because the algorithms are non-deterministic and probabilistically complete, choosing a motion planning timeout is messy
  - TMKit starts with a 10 second timeout and increase it whenever the planning horizon is increased

TMKit-Overview



Resources

- Paper:
    - Dantam Neil T., Kingston, Zachary K., Chaudhuri, Swarat and Kavraki, Lydia E. Incremental Task and Motion Planning: A Constraint-Based Approach. IJRR 2018
- Recommended reading:
    - Ffrob: An efficient heuristic for task and motion planning (can be found at https://dspace.mit.edu/bitstream/handle/1721.1/112348/Lozano-Perez_FFRob.pdf?sequence=1&isAllowed=y)
    - Integrated task and motion planning (can be found at https://www.annualreviews.org/doi/full/10.1146/annurev-control-091420-084139)