High Level Planning: Paramtrized symbols, Re-planning and Portable symbols

Chinmay Burgul, Gokul Narayanan, Mihir Deshingkar, Sinan Morcel, Zhaoyuan Ma

I. Introduction

A long standing research in robotics is to make the robots plan in the high level, similar to humans and then act in the low level world. This requires high level reasoning of the low level world. One way is to discretize the continuous state space. However, discretizing the continuous state space often results in the explosion of state space due to high-dimensionality of the real world. Therefore, one needs to have a simpler representation which can represent the range of the low level variables and that's quite challenging. However, with recent advancements in the planning and algorithm domain, new planners [1][2][3] have been developed which address this problem. But, they are not holistic enough to solve the problem efficiently. The symbolic task planners [1][2] which are separated from the motion planners do not understand the geometric and kinematic limitations of the robot which results in failure of plans while executing. The combined task and motion planners [3] tackle this issue but they are computationally expensive as each node in the plan should be checked for feasibility.

Consider a loco-manipulation task of picking objects located in the table and placing them in a bin which is located at the center of the table. Let's consider the case of picking object A and placing it in the bin. For this task, the high level task plan given by symbolic planners would be MovetoA, pickA, MovetoBin and placeA given the preconditions are satisfied. This is a valid plan with respect to high level as the preconditions are satisfied but while executing it in the low level, the plan might fail because of obstructions or other changes in the environment. Since the high level planner is decoupled from the low level motion planners, the high level planner doesn't know when the plan is executable or the reason for the failure. Although earlier combined task and motion planners will be able to solve this problem, they are not scalable as the time complexity increases with increased number of nodes in the task plan. So, to solve this problem effectively, the low level planners should be able to update the failure information to the high level task planners to generate valid plans.

Our main contribution in this paper would be to extend the concept of interface layer[4] to the symbolic task planner[1] to address the low level motion execution failure and high level task plan generation failure by introducing parametrized symbols and re-planning with partial solutions. We are also implementing portable symbols[9] for the loco manipulation task to reduce the variables in the symbolic state space.

II. Related work

The paper [1] uses SMDPs to represent the world state and STRIPS like PDDL as a language for planning in high-level. Each option in PDDL is described by a tuple (precondition, effect+, effect-). The paper presents an algorithm to automatically learn high level states called symbols. However, there are two main drawbacks with this method. Firstly, if the low level planner fails to execute an option in the current high-level plan, the planner returns a failure and there is no provision to expand the tree from the failure point and re-plan. Secondly, in case of unknown obstacles, the execution is likely to fail, as previously learnt symbols are now incorrect and need to be updated.

Max and Heramb [11] proposed the Dynamic option method, to solve the problem: Effects for some options could not be learned from demonstration data in the loco-manipulation task. To tackle with inconsistent effects, one might use parametrized option to address different effect sets. However, that is not sufficient as there are many combinations of start states which are never demonstrated. In such cases, the option cannot be learned directly from demonstration data. To solve this problem, Dynamic options are used rather that effect set based partitioning. For dynamic options, the effect is generated such that it lies in the precondition of the next option to be performed. For dynamic options, positive effects and negative effects are removed from the PDDL description. Dynamic option planning happens at low-level.

Paper [4] by Srivastava et al. introduces a novel interface layer for combined task and motion planners, which, due to the abstraction is independent of the implementations of task or motion planners as well as able to handle the failures in the environment. The abstraction only needs the failures in motion plan to be representable in terms of logical predicates. This interface layer allows the high-level planner to work independent of the motion planner, ignoring the geometry. First, the interface layer calls the high-level planner. Next if it is unable to find any low level pose instantiation corresponding to the high-level action, the interface layer updates the failure cause into the high-level planner. It also stores the partial solution in case of such failure. Lastly, the high-level planner is called with the updated state space and partial solution. However, this paper implements hand coded symbols as opposed to learning symbolic representation from demonstration as in [1].

Another paper [14] by Wells et al. presents a novel concept to improve the scalability and performance of task and motion planning (TMP) by incorporating geometric constraints into the task planning. They train a classifier which can identify

feasibility of motion plans, which is then used as a heuristic to guide the motion planning search towards finding a feasible solution. The training is done only on a few simple scenes and the algorithm is able to extend well to novel complex scenes with more obstacles. Due to heuristic guided search, runtime to find a feasible motion plan is improved by large scale in most of the cases. However, the algorithm has two major limitations. Firstly, it is currently designed and tested only with fixed robots, although can be extended to mobile robots for loco-manipulation. Secondly, it is quite costly to generate the required training data for the classifier. Therefore, this algorithm cannot be directly integrated with our current planner, however, can be incorporated in future for improving runtime performance.

Paper by Adria et al [15] presents an Interface layer between High Level Task Planner and Low level motion planner to update the information about the environment. This framework will allow the task planner to solve complex tasks with basic information of the goal, and re-plan whenever the motion could not be executed by the low level planner. The main goal of the paper is to make the planner to plan robustly, re-planning when the high level plan fails. And develop a framework which allows the robot to work autonomously by executing a set of basic actions, and capable to develop high level tasks defined with clear and simple goals, by just sequencing low-level tasks. The main learnings from this paper was the re-planning approaches a). Pushing the obstacles for re-planning. and b). Planning by moving base at random places and solving IK. And the whole research was focused on implementing planners on a realtime-hardware which gave more insights of the problems faced while implementing.

Paper by Garret et al on FFROB [13] presents Extended Action Specification (EAS) - which models task and motion planning as symbolic planning where the conditions of actions are complex predicates involving geometric and kinematic constraints. At the highest level of Abstraction FFRob iteratively alternates between sampling phase and planning phase until it is able to find a solution. The sampling phase discretizes the PPM problem by creating symbolic actions from a finite sampled set of poses, grasps and configurations. The planning phase performs a discrete search to decide whether a solution exists. If the discrete search fails to find a solution the process repeats with a larger set of samples. Concluding, the conversion of environment state in terms of symbols is a complex task and if the environment is clustered then it is much more complex. The learning from this paper is the unique solution which help us to cluster all the environment efficiently with less cost.

This paper [5] combines the Task and motion planning with Reinforcement learning techniques to achieve robust decision making capabilities. Since this algorithm learns from the environment after every execution, it can give better plan in the long run. Currently, our method focuses on re planning techniques once the task plan execution fails. But combining our method with this algorithm described in this paper will improve the performance of the system. Because the failure rate of executing an option like picking an object is very high in our loco-manipulation task. Similarly, to improve the performance of the combined task and motion planner, [6] uses inverse reinforcement learning techniques to learn heuristics from expert demonstration. This helps in reducing the planning time and the re-planning effort as the planner knows the option with high success rate based on the expert data. This technique combined with our re-planning framework will increase the performance of the whole system especially in the loco-manipulation domain as the plan with shortest length is not always the optimal one because of failure rates of the options.

In Ames et al.'s latest work[7], they extend the ability of learning symbolic representation from analyzing probability distribution of executing different actions. The actions take parameter vector as input and generate corresponding behavior based on given vector value. They have tested this method on an Angry bird game as well as on a real robot to perform a specific option with parameters.

One important concept in Konidaris et al. [1], one that the work was based upon, is that of sub-goal options, which are simply options with compact image sets that can safely be replaced with their effect set. The intuition for this, as [8] explains, go_to_kitchen is a sub-goal option because it doesn't matter where you start in the house; it is also an abstract sub-goal because go_to_kitchen won't change how much gas is in the car. The Konidaris et al. [1] method relies on at least the weak abstract sub-goal assumption to hold, which states that the image of an option being a subset of the precondition of another option that follows is equivalent to having the effect of an option be a subset of the option that follows, in order for their framework that they proposed to hold.

In a follow up work, [9], that aims to accelerate the generation of the symbolic representation or the PDDL description of a task that contains elements that were seen before by the agent, portable symbols are defined. Those are symbols learned in the agent's space, which is a space that remains unchanged when the task is changed and thus the knowledge learned in this space is portable. In order to build the agent space, an observation function that maps the elements from the task space to the agent space has to be chosen, which requires domain knowledge. The observation function has to be non-injective, which means it has to allow for distinct states in the task-space to be considered as one state in the agent-space. For example, the task-space symbols in_front_of_door_A and in_front_of_door_B have to be the same symbol in_front_of_door in the agent-space. After defining the observation function, it is used to map (s, o, s') and other data to the agent space and learn a symbolic representation in the agent-space, like was done in [1] over the task-space.

Figure 1 shows an example agent-state-space and an example task-space. Note how the in_front_of_door symbol is sub-goal in the agent-space, but not in the task-space. The symbol in_front_of_door is thus said to be portable. The information that the portable symbol in_front_of_door is true alone is not enough to tell in which room we end up in by applying the option go_through_door. Thus, the knowledge in which partition the agent is (Figure 1 b) combined with the agent-space symbols determines the effect of the options; the authors refer to lifted symbols like in_front_door(x) where x refers to

the partition. The partitioning is done by separating the options in the agent-space, based on their effect in the task-space. Since the options in the agent-space are already sub-goal, they remain sub-goal after the partitioning, but they become so in both spaces, the agent's and the task's. How the partitioning is done is explained in [9] with reference to [8] for the clustering.

The intuition behind the use of portable symbols is that experience should be used to solve unseen tasks. The key to this is the realisation that the agent's sensory equipment do not change when a new task is at hand; that is, the agent's state-space and the symbols learned in this space do not change and can be used for transfer of knowledge between tasks. As for the need for portable symbols, they accelerate the generation of the symbolic representation once it has been generated before for a previous task. That is, when doing a random exploration of the transition data (i.e. initial-state, option and then new-state), we can stop the exploration within an smaller time interval (up to some extent) and still get a valid PDDL description of the task at hand.

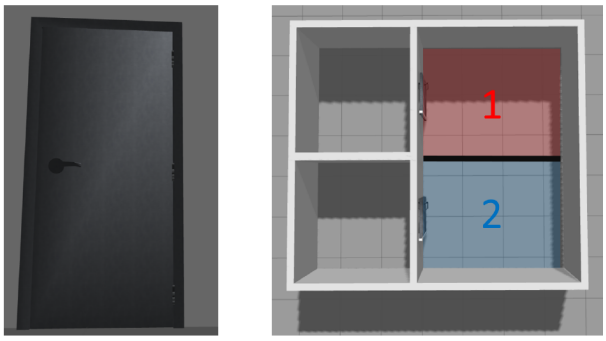

Fig. 1: (a) Agent-state-space and (b) partitioned task-space (source [9])

III. Preliminary Work

In the previous semester, we developed a loco manipulation game to collect data from human subjects. The data collected from the game was used to test the dynamic option partitioning method. The graphical user interface of the game is shown in the Fig. 2. In this game, there is a *Counter work space* (the brown rectangle) with *Objects* (red squares) and a *Bin* (the green square). In the current setup, we divide the table into eleven columns and three rows. There is one *Object* in each column except the center column, which occupied by the *Bin*. The *Object* appears in one of the three row in that columns randomly, with an equivalent chance. A human subject controls the *Agent* (the blue square in front of the *Counter Workspace*) to move left or right. Besides that, the human subject also controls the hand-shape-cursor, which indicate the *Hand* of the *Agent*. The *Hand* is for interacting with *Objects* and the *Bin*. It can only move within the *Reachable Range*, which is shown as the green circular area, moving with the *Agent* as the center. The transparency of the *Reachable Range* reduces from the center to the edge, affects the accuracy of identify the boundary of the *Reachable Range*. This corresponding to the fact that human beings tend to be less accurate when they reach things near the boundary of their reachability.

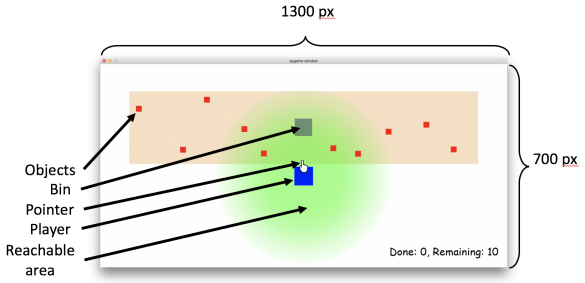

Fig. 2: User interface of the loco-manipulation game

The target for this game is to control the *Agent* by keyboard and its *Hand* by mouse to collect all *Objects* and place them into the *Bin* one by one. There are three options that a human subject can execute:

- Pick: The human subject can pick up an *Object* when the *Hand* is empty and the *Hand* is on that *Object*. This option creates following effects: 1) The *Hand* is occupied, not being able to pick any other *Object*. 2) The position of that *Object* changes from its own position to the top left corner of the interface (fixed position), which indicates the *Object* is in the *Hand*.
- Place: The human subject can place an *Object* only when that particular *Object* is being held and the *Hand* is on the *Bin*. This option creates following effects: 1) The *Hand* becomes empty. 2) The position of that *Object* changes from the top left corner of the interface into the *Bin*. After the place option being executed, the *Object* is no longer available for interacting with.
- Move: The human subject can move the *Agent* and the *Hand* alone the long side of the *Counter Workspace* anytime by any distance.

Based on this game. We designed a user study. We recruited 24 participants (21 males and 3 females). They first received instructions for how to play the simulation game. Then they spent two minutes for practice. The user study is divided into three sessions, each session contains ten trials. Before the start of each session, the participant receives instructions to perform the task with different performance objectives. Three different objectives are:

- Natural mode: to complete the task at a comfortable pace.
- Fast mode: to complete the task as fast as possible.
- Accurate mode: minimizing the errors while completing the task (e.g., missing the click when pick an *Object*.)

The order of three objectives was randomized based on Balanced Latin Square to reduce the affect of possible learning curve. There is two minutes break between sessions for each participants.

Our record including video stream of the simulation game window during the whole game play, and each action in the

format of (s, o, r, s'). s denotes the current state from which the action is performed. o is the performed option. r is the reward of the action (in term of time measured in seconds). s' is the state after executing the action. Both s and s' have 33 variables, which are variables of the *Agent* and *Object*s, three in a group. The three variables of the *Agent* are x and y coordinates in the task space and a boolean variable indicating whether it is holding an *Object*. The three variables of an *Object* are x and y coordinates in the task space and a boolean variable indicating whether it is held by the *Agent*. Among all collected data, there are six sets of data considered as outliers. Therefore we take rest of eighteen sets of data as valid data for further processing.

IV. Problem description

A. Motion planner failure

The loco-manipulation game has ten objects placed on the table. Pick and place options for each of these objects are learned from the user demonstrations. Without addition of any obstacles, the planner successfully plans and executes pick and place for all objects. Now consider a task of picking and placing only one object. Let us assume that another object is between the robot and the object, as shown in Fig. 3 . There are two different possible cases. In the first case, the obstacle is such that it makes some part of the symbol - which represents precondition of robot pose for pick option - invalid. Therefore, if the effect of move option is a one of these invalid points, the motion planner will return a failure and no plan is found. This is because high-level planner is unaware of the obstacles, and it's symbols represent state space without any obstacles. This problem can be solved by executing motion planner for other points in the symbol, until a valid point is found. In the second case, we assume that there is no valid point from the symbol from which motion planner can find a successful plan for picking. Therefore, the low-level motion planner fails during the execution due to object B. Although a solution exists - pick and place object B, and then object A - the previous planner will not be able to provide an alternate solution. This problem can be solved by re-planning with the high-level planner and expanding the remaining tree. We describe this in detail in section V.

B. Incorrect option partioning

Although Dynamic option is introduced, generating partitions for non-dynamic options are still inherited from [1]. This method is problematic for our loco-manipulation task since the position of each object is no longer constant. Consider the example of how it is done in [1]. The merging partition will add two partition together as shown in Fig. 5.

In the loco-manipulation task case, an object may generate three partitions corresponding to the row it belongs to. The further the object is away from the agent, the smaller the partition is. The result of executing the same merging operation, outputs the largest partition as the initiation set of the object, shown as Fig. 6.

If we take this as our option to plan, we may end up with the position shown in Fig. 7 as our "best strategy". If both objects

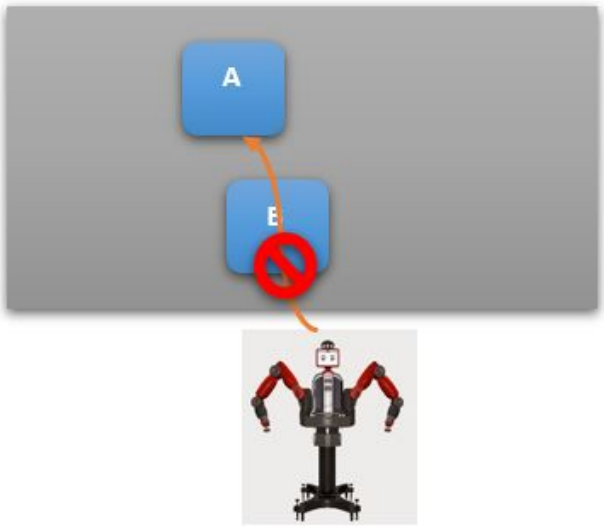

Fig. 3: Plan for Move A in a case of known obstacle as object B

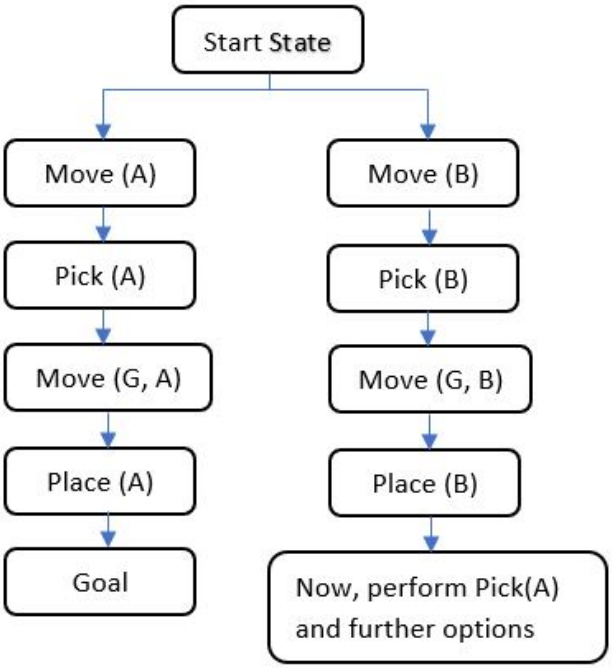

Fig. 4: Replanning in a case of known obstacle

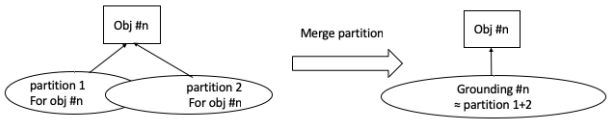

Fig. 5: Previous option partitioning

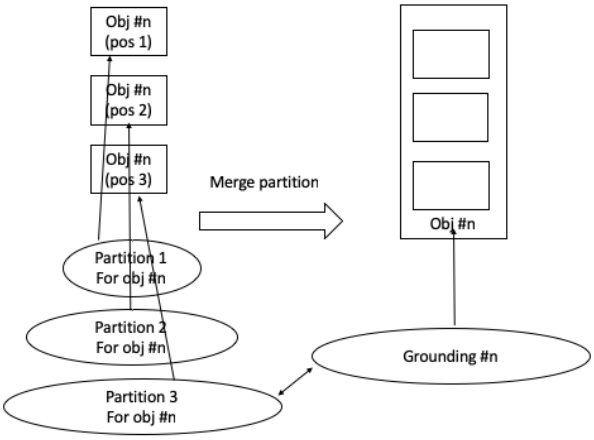

Fig. 6: Previous option partitioning

appear to be at the farthest row, the plan cannot be executed in low-level. Therefore, simply executing "OR" operator cannot fulfill the need in loco-manipulation task. Our initiation set must be able to change according to some states of the object to optimize for the following option as well as to satisfy the reachability.

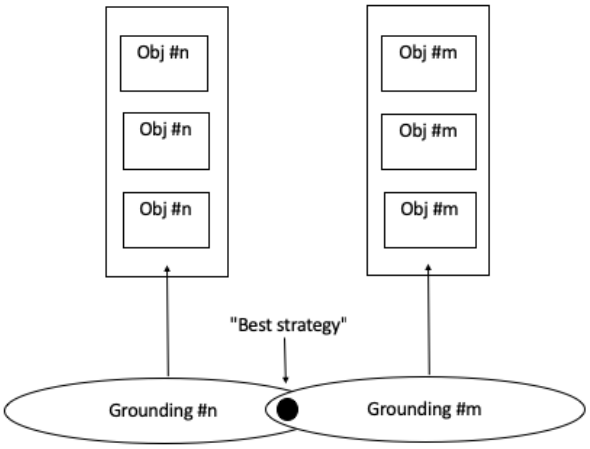

Fig. 7: Planning result based on previous method of option partitioning

C. Portable Symbols

In this section we revisit Portable Symbols. Several issues arise regarding the implementation within the bounds of our application. First, choosing a suitable observation function requires domain knowledge. Second, extending this work to support portable symbols may require incorporating some changes to the method suggested in [9].

This problem may be broken into three different challenges. The first is to define a non-injective observation function; that is, the agent-space state-variables are to be chosen in a way such that multiple symbols in the task-space can map to the same symbol in the agent's space. In short, for the first challenge, we can ask the following question: "How can we define the observations function?" This is also essentially asking: "what variables should the agent space consist of?"

The second challenge is to generate symbols and options in the agent space, such that the resulting options are sub-goal in the agent's space, but not necessarily in the task-space. In here, we can ask: "how can the pipeline we already have be used in the generation of the symbolic representation in the agent space?"

Also, other key questions arise, like for example:

- Are portable symbols suitable for our definition of a task (placing all items in the bin)?
- Are portable symbols suitable for the domain?
- If not inter-task transfer, then how can we apply the concept of portable symbols to a simpler sub-task, like picking an individual object, or placing it in the bin?
- How should the data be collected for portable symbols to be used? That is, should we randomize the location of the items per participant? Should we randomize the locations per trial, that is treat each setting of the objects as a different task?

V. Methodology

A. Replanning and updating symbol parameters

In the previous section, we discussed two cases of failures while executing high-level plan with our previous planner. We implemented A* planner as our low level motion planner to find a path connecting the base of the robot and the object to be picked while executing the pick option. As our game has high pixel resolution, it is quite costly to evaluate A* for points in such a near-continuous space. Therefore, we first discretize our game into square grids with side length of ten pixels. The obstacle space is updated before executing each option, and therefore is able to work with our dynamic environment where state space changes with execution of each option. The motion planner is called for each pick option, and returns success if it finds a path and the path length is less than the predefined hand length. Otherwise, it returns a failure.

As described earlier, the first failure is when part of symbol corresponding to the robot's position for picking an object is invalid. To tackle this problem, we iteratively explore rest of the points in the symbol grounding as follows. Our algorithm first chooses the mid value of the symbol. If this fails, it will check for the two farthest points from the mid point. Here we choose the farthest point from the failure point as we assume high chance of failure for the points near the failure point. Next, if the end points fail too, the algorithm finds farthest points from each pair of consecutive failure points, which are mid-points of the two consecutive failure points. For each mid point, it calls our A* motion planner to check if a plan exists to pick the corresponding object. If there exists such a plan for picking the object, the motion planner returns this path and the corresponding point of execution and the robot is able to execute pick option on the corresponding object by moving to that point. Eventually, this algorithm will evaluate all points in the symbol and return a failure if there is no valid point for executing pick option. Therefore it is probabilistically complete in reference to the provided demonstration data.

If the algorithm described above returns a failure, then it comes to the second failure handling case. Here, the only option is to pick and place the obstacle to clear the path for picking the desired object. In such a case, our low level planner returns an information about which object is causing the failure. This information is then sent to high-level planner, which removes the the failed pick option from the option list for two levels of expansion and re-plans. Use of this additional information reduces the search time and number of expanded nodes by a large amount. Eventually, as the algorithm uses breadth first search, it finds the shortest solution as picking and placing the obstacle, and then picking and placing the object. Next, this high-level plan is executed, which will now succeed as the obstacle is removed. However, if the obstacle is unknown, we cannot execute pick option on it as our high-level planner has not learned pick option on this new object. In such a case, our planner will return a failure. The pseudocode for the whole framework is given below Algorithm 1

Algorithm 1 Combined Task and Motion Planning Framework Pseudo Code

linenosize=

```
1: procedure COMBINED TASK AND MOTION PLANNING
2: T.Plan ← Get High Level Plan (current state, goal state)
3: For each option in T.plan:
5:     if option = pick then
6:         plan, status ← MotionPlan(Robot_Position,Obstacle_Position).
7:         if status then
8:             Execute (plan)
9:         else
10:            status, position ← Feasible Points(option)
11:            if status then
12:                Move(Position)
13:                Execute(Plan)
14:            else
15:                T.Plan, success ← Replan(Failed Option,Obstacle Symbol)
16:                if success then
17:                    Perform Task
18:                    Task Completed
19:                else
20:                    Task Failure
21:
```

Algorithm 2 Function Replan()

```
1: procedure REPLAN(FAILED OPTION, OBSTACLE SYMBOL)
2: {goal_state} ← {goal_state} + obstacle symbol
3: For two levels of expansion,
5:     {option list} ← {option list} - failed option
6: T.Plan ← Get High Level Plan (current_state, goal_state)
8:     return T.Plan, Success
   =0
```

B. Creating Parametrized symbols

Instead of merging option partitions to generate a uniformed precondition for the next option, we want to save all partitioned options' information to help a dynamic option to decide where to land at planning stage.

To achieve this, we need to modify the procedure of processing static options to preserve cluster information first. As mentioned previously, the result of executing a dynamic option lays in the precondition of its following option. After option partitioning of static option, we analyzing these option partitions base on their initiation sets s. We apply DBSCAN to cluster a initiation set into roots. We then save these roots for further processing.

The second stage is generating parameterized symbols for dynamic options. At this stage, once we have a dynamic option, we analyze all possible options happens next. For each "next option", we 1) collect the roots data of the "next option" (name as "outcome"), 2) remove the direct effect of the dynamic option applies to the roots' data (e.g., the agent position when agent perform "move" option), 3) use the number of roots as our expect number of clusters to fit all roots' data to a regression function. The result function enable us to parameterize the dynamic option for the corresponding outcome. Only outcome with more than one roots are parameterized, otherwise standard KDE is applied to model the partition. Removing the direct effect from the roots' data is necessary in the second stage is because a dynamic option such as "move" always result in the changing of the agent position, which is the most significant component during clustering.

By executing the algorithm mentioned above, we can obtain parameterized symbols. To using them in planning, when we need to execute a dynamic option, 1) we look for the corresponding outcome of the option, 2) score the world state of the next target by the parameter function, the one with highest score lands within the initiation set of the outcome.

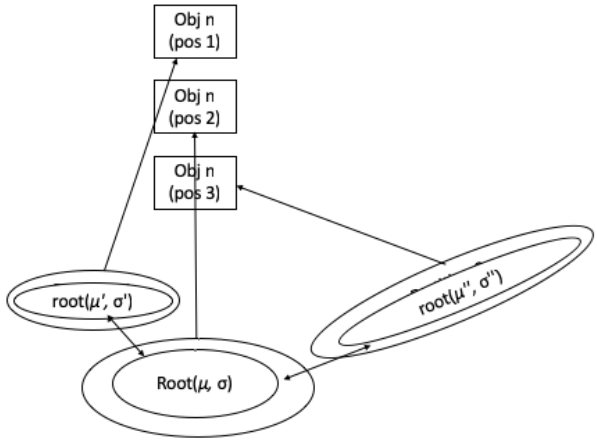

Fig. 8: Parameterized symbol

C. Portable Symbols

For the portable symbols implementation, we first need to define an observation function. Selecting a good observation function involves several considerations:

- The state space size is preferably independent of the number of items to collect.
- It is preferably similar to the typical target robot agent space (i.e. it resembles how today's robots sense the environment, see figure 1 a for an example).
- It has to allow for the options to be sub-goal in the agent-space.
- The function has to be non-injective; preferably, maximally so. There should be multiple symbols in the task-space that map to one symbol in the agent's space, for the method to be useful.

object_in_hand
Offset_from_object
Object_is_can
Object_is_bin
Object_is_void

TABLE I: Agent-space definition

An observation function or agent-space description will make, at least, the pick option sub-goal. The same approach will be used as in [1] for generating the symbols in the agent space. Finally, regarding the last stage where a mapping is required from the agent's space to the task-space for inter-task transfer, more research into the topic is needed, and future work will target this problem. Also, the assumption that we will make is that each new setting of the objects on the table make up a new task, and thus, new data needs to be collected, on a smaller scale, every time a new configuration is at hand. Thus, inter-task transfer becomes defined in our context and thus, the portable symbol implementation would be suitable.

In this term, the focus was on acquiring the PDDL description in agent-space. Towards this goal, the following was done.

1) Random Exploration: First, all work started from a module called max_planner.py that was previously used for starting the options thread and the planner thread, before launching the game. The option-thread emulates events that pygame spools and the game pops and reacts to, while the planner-thread sends the next option to the option-thread. All communication is done through object-handlers (that is, each thread-object has a way of calling the functions of and setting the values of the other thread-object) and memory access lock mechanisms that help prevent data-races. The options-thread did not change throughout this project, however, the planner-thread was replaced with an exploration-thread, which does the same thing but randomly as opposed to following a plan; that is, it generates a random number, and based on that number it sends an option to the option-thread.

The module "max_planner.py" was changed (and also renamed to "portable_symbols_data_collector"). It was turned into a script that runs the game multiple times appending the collected transition-data to a list and writing this list to a file.

2) Editing the Game: The next step was changing the rules of the game such that movement to left or right is made to be relative, as opposed to absolute (the move option doesnt make you jump to a specific location but rather go in a direction with a specific amount of displacement). Limiting the agent to the tables boundaries was taken care of by the update function of the agent/player. Then, the task-space to agent-space observation function was implemented, which takes the task-space state and converts it to an agent space vector, which doesnt depend on the number of elements in the scene and which assumes that only one object, either a can or a bin can be in the scene at any time (unless the agent is carrying a can and another is in front of them). The following table shows the agent-space structure, the is and in_hand cells indicate that the field is a boolean while the offset cell houses the offset from the object (mismatch in the x-position). Table I shows the agent-space structure.

Then, to validate that the observation function is correct, we are plotting the agent space in a small square at the top right corner of the screen. Figure 9 shows the game edits that were implemented. The top-right green square represents the agent-space. Note that the agent-space is completely agnostic of the table itself (the edges are not considered by the agent-space, for the purpose of ensuring non-injectiveness). Finally, the transition data is recorded in both spaces. However, only the agent-space data is stored and propagated through the pipeline, for now.

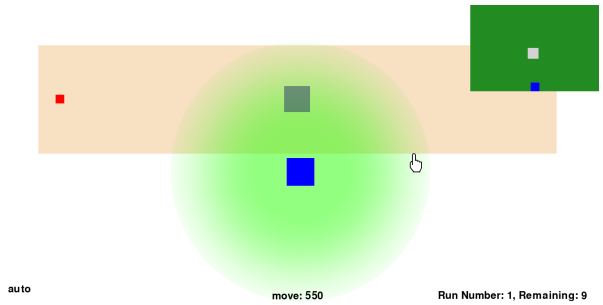

Fig. 9: The game with the agent doing a random exploration.

3) Option Clustering: Once we had the transition data, we needed to perform a clustering to make the options sub-goal. Among the changes that were made to the clustering pipeline was the addition of a custom normalization function, in addition to basic tuning of the clustering algorithms to account for the scarcity of the data collected. In the clustering of the options, in the create_effect_partition routine, the minimum number of elements per cluster is set to 4 and in the create_roots routine, the minimum number of elements per cluster is set to 2. We believe that tuning these and another such value: the number of minimum elements per cluster, currently set to 20, in the merge_partitions routine, would yield a more compact set of symbols.

4) PDDL Generation: Then comes the PDDL generation pipeline: among the notable changes were the enabling of factor-as-features cross-validation as a way to optimize the performance of the models that assign truth-values to symbols. We also removed some of the elements that have to do with another method that we were implementing: dynamic options.

Finally, after obtaining the PDDL representation in the agent space, we validated it by observing the simple high-level option pick, which yielded effect+ and effect- symbol sets that were correct. After the validation, The visualization pipelines were developed from scratch. The reason why pygame was used instead of Matplotlib was because of the learning curve that it was taxing us with. The visualization samples a set of columns each belonging to a field in the agent-space and a set of functions that do the translation back to the deterministic case (turning a list of booleans into one boolean through voting) based on the mask. As a result, we have a function that uses both mask information and the symbol-distributions to generate visualizations. An important but subtle detail is that we should not try to alter the logic to compress the size of the code that visualizes the symbols. For example, if the object is a bin, we cannot just assume that the object is not a can as well, because if we do, we miss important cases that the PDDL relies on. Also, we must make sure that we check

if the samples explicitly indicate that a field is False, and at least write void on the display of the symbol.

VI. Results and Discussion

A. Re-planning framework

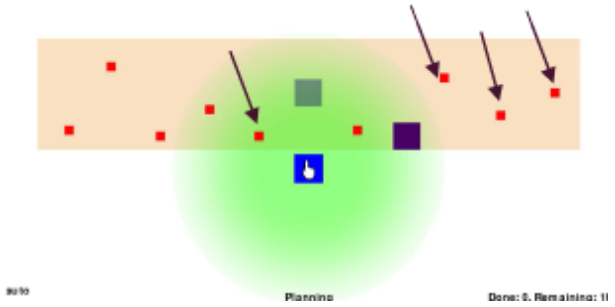

Fig. 10: Start state of the environment

We demonstrate how our combined task and motion planner is able to tackle two types of failures, which cannot be tackled by high-level planner independently. Fig. 10 represents the state of the world and the arrow marks shows the objects to be picked and placed inside the bin. The objects are numbered from left to right starting from 1 and ending in 10.

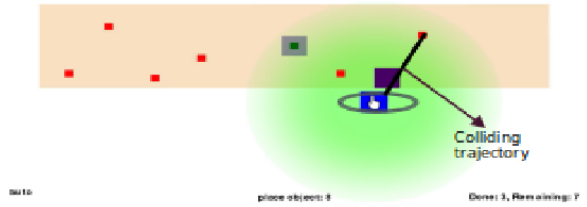

Fig. 11: Trajectory colliding with the obstacle

The high level task planner returns a plan [pick4, place4, move, pick9, move,, place9. move, pick8, place8, pick7, place7]. The first 9 options in the plan were successfully executed but the tenth option which is pick7 is failing because of the obstacle in front of the robot. Fig. 11 demonstrates the first failure case, where a part of the symbol which represents robot state for picking corresponding object is invalid due to the obstacle. Consequently, the motion planner is unable to find a path to execute pick7 from the position returned by effect of move option. However, there exists some points in the symbol from where the robot can pick the object. So, the symbol is sampled as per the method mentioned in the methodology to find a feasible point. Fig. 12 shows the feasible point returned by the sampling step from where the object can be picked.

The second possible failure is when there is no point in the symbol from where the robot can pick the object, as shown in Fig.13. So, the information of the obstacle should be updated to the high level and re- planning should be done. And the planner comes up with a plan to pick and place the obstacle as shown in the 14. Once the obstacle is removed, it can perform pick and place on object 7 to achieve the goal.

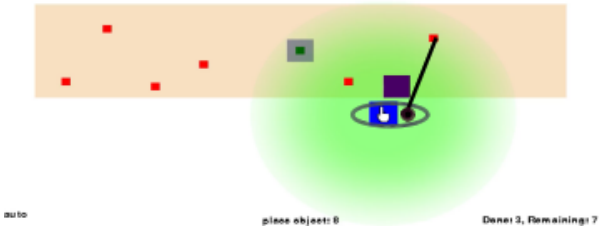

Fig. 12: Sampling points in symbol

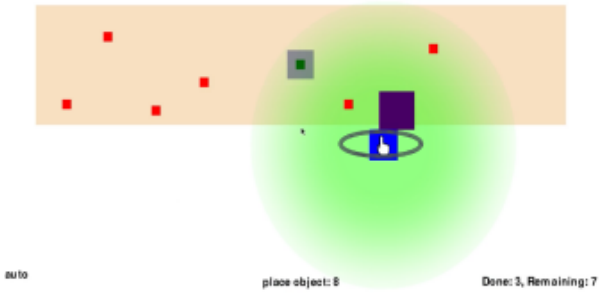

Fig. 13: Bigger obstacle

B. Parametrized Symbols

For the parameterized symbol, we test our method on previous dataset[11]. In this example, since each object has three possible locations, our algorithm generate 3 roots for ”pick” the particular object and 1 root for ”place” that object. Then for a ”move” option partition with ”pick” as its outcome, parameterizing is executed and saved with roots’ label. Our method generates 10 ”move” options with ”pick” one of the objects as their following outcome, which is reasonable. This proves our method so far.

C. Portable Symbols

1) The Symbols: The PDDL generation resulted in 23 symbols. Appendix A shows the whole set of generated symbols. Figure 15 shows the PDDL description of the the high-level option pick1. This option makes the symbols p0, p13 and p14 true. For the precondition, the symbol p3 refers can be named not_void, p7 not_in_front_of_bin, which makes sense. For the effect+, p0 can be called not_in_front_of_object, p13, object_in_hand and p14 in_front_of_void. This means after picking, the object in front of the agent disappears, an object appears in the agents hand and the agent would then be facing nothing. For the effect-, p2 can be called object_not_in_hand, p3, not_void, p23, object_not_in_hand, and p5, in_front_of_object. This means that after picking an the empty hand of the agent will be empty no more, the agent will then face void and the object disappears from in front of the agent. This option is consistent with our understanding of the effect of picking an object and is thus validated.

Figure 16 shows the symbols used by this option. Transparent objects means that the symbol specifies the negation of the type of the object at hand. For example, a transparent object (can) in front of the agent means that the type should not be a can. The rest of the symbols is delivered with the project package.

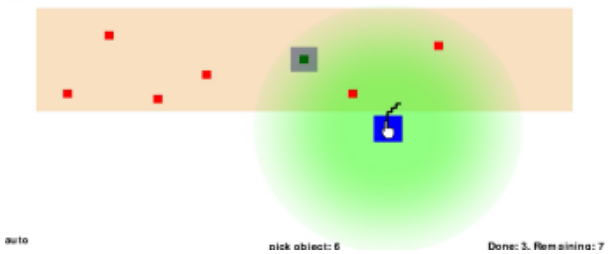

Fig. 14: Picking the obstacle

```
(:action pick1
 :precondition (p3, p7)
 :effect+ (p0, p13, p14)
 :effect- (p2, p3, p23, p5)
)
```

Fig. 15: A sample PDDL option.

A similar analysis shows that the other options make sense too. Another important result is that there are three move-left options and four move-right options. This is consistent with what we should expect because the agent can go left starting in front of the bin or starting from the right-end of the table. Also, when in front of the bin, the agent can go left carrying an object or not (those two cases are enough for the robot to tell that there won't be any object there, because if the robot is carrying an object already, then it would have had picked it from the left-most end of the table).

There are four move-right options, which is also consistent. One from in front of an object towards the bin, another from in front of void with an object in hand towards the bin, a third from in front of the bin towards the void on the right while carrying an object and the final from in front of the bin towards the right but without any object in hand.

2) Gained Intuition: In this section, we will take the opportunity to reflect on how portable symbols can be useful, mainly focusing on the intuition behind their utility. Portable symbols are symbols in the agent space; since the agent space doesn't change when the task is changed, the symbols are said to have been ported from one task to another. Portable symbols reduce the amount of samples needed for PDDL generation of a new task which has elements that the agent recognizes through experience. In other words, portable symbols allow us to leverage experience to solve new tasks. However, it is not clear why their use gives all these benefits. In the remainder of this section, we will illustrate the reason for why PDDL generation requires a smaller number of sample from random exploration.

Lets say the agent observes part of the task space, through some observation function. This function maps multiple distinct state-distributions in the agent space to the same state-distribution in the task space; that is, the function is non-injective. An example of such function is the one shown in the simplified scenario of the table clearing task with separated objects, and the example of a robot standing in front of a door. In either cases, there are multiple distributions of the world state that map to the same distribution in the agent space (that is, they make the same symbol true in the agent space).

For this purpose, because the agent-space doesnt change,

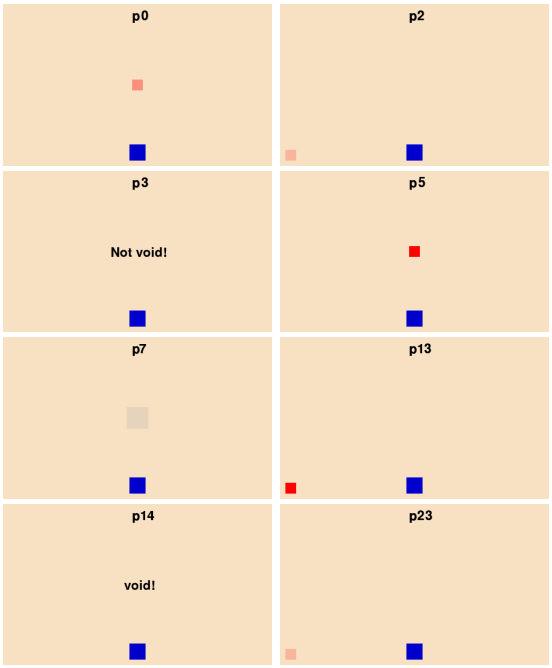

Fig. 16: A set of symbols.

and because standing in front of another object of the same type or door of the same structure sets the same symbol to true, the agent therefore has the ability to recognize that they are in front of a door and reason about what high-level options it has (i.e. open the door). The significance is that every time the agent encounters a door, they are more likely to know what to do with it. They have seen a lot thereof already. All the agent needs to do when exploring a new building is to just interact with each door at least once in order for them to learn the map of the place. In the work that we are reproducing, this is referred to as learning the linking function. The agent can reuse their symbols to deal with any new door, but they need to open the door at least once to learn whats behind it, or what room will they end-up in. More generally, this is saying that in addition to the agent's experience, we need information about the task at hand in order to build a PDDL and plan with it. However, if the agent encounters a door, and they have seen many doors before, and they happen to have collected little data regarding the specific interaction, then, they would still be able to learn that the symbol in_front_of_door means they can initiate the option open_the_door, but they also got the chance to learn more they got to learn that the symbol in_front_of_door_x leads to in_room_y where x corresponds to a partition in the task space that represents the agent being in front of a specific door. The same can be said for the object on the table.

Once the agent has a way to interact with the door, the light-switches and other elements that relate to the specific task, then all of this gets poured (or crystallized through the use of a classifier) into their experience (in the agent space) before they move on to the next task. This is because task-specific information is not useful in other tasks.

In the next task, Its faster for the agent to learn their way around a building because they have seen doors and switches

from all different angles by now. Its also faster for them to learn whether they can pick an object from some angle and distance or not.

Finally, the authors in [9] concede that the use of random exploration weakens the method dramatically. A better approach is to explore based on experience. That is, we let the agent perform the actions that maximize the learning of new symbols or improves the knowledge of ones the agent knows already, but not as much as they know other symbols.

VII. Conclusion and future work

From the above demonstration, we have shown that combining the task and motion planners helps in improving the performance of the system through re-planning in the task plan as well as in the motion plan when the execution fails. This is highly needed for the loco-manipulation domain as the environment is highly dynamic and executing an option has high uncertainties. The current implementation can be extended with some of the reinforcement learning techniques suggested in [5] [6]to make the performance of the system more robust.

We also show that using parameterized symbol could reduce the number of generated symbols. Then we will run a planner to generate and execute a plan in given environment to test our parameterized symbols.

As for portable symbols, we can conclude, from analyzing one sample PDDL high-level option and observing the number of move options, that the symbolic representation is correct. Future work should tackle the propagation of both agent-space and task-space data forward through the pipeline and then creating a Linking function. Also, the random exploration may need to be turned into an informed exploration scheme in which the learning of new experience is maximized per task.

VIII. Contribution of Team Members

A. Sinan

I worked on the generation of the PDDL in the agent-space (the portable symbols section), as well as on the analysis of the results. Of course, I would like to extend a warm thank you to Heramb for helping me get through a lot of issues in the code. I would not have completed much without his help and patience. Thank you, Heramb.

B. Mihir

Implemented low level motion planner. Implemented algorithm for sampling all symbol points. Helped Gokul in integrating high-level and low-level planner.

C. Gokul

Implemented high-level replanning. Integrated low-level planner and high-level planner.

D. Chinmay

Read and understood the main paper by Konidaris and the new paper by Srivastava. Modified the required changes in the game.

E. Zhaoyuan

Implemented parameterized symbol.

References

[1] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez, From Skills to Symbols: Learning Symbolic Representations for Abstract High-level Planning, J. Artif. Int. Res., vol. 61, no. 1, pp. 215289, Jan. 2018.

[2] Multirobot Symbolic Planning under Temporal Uncertainty, Tech. rep. CVC TR98003/DCS TR1165, Oct. 1998.

[3] L. P. Kaelbling and T. Lozano-Prez, "Hierarchical task and motion planning in the now," 2011 IEEE International Conference on Robotics and Automation, Shanghai, 2011, pp. 1470-1477.doi: 10.1109/ICRA.2011.5980391

[4] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," 2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, 2014, pp. 639-646. doi: 10.1109/ICRA.2014.6906922

[5] Integrating Task-Motion Planning with Reinforcement Learning for Robust Decision Making in Mobile Robots Jiang, Yuqian, Yang, Fangkai, Zhang, Shiqi,Stone, Peter arXiv:1811.08955 21Nov2018

[6] Chitnis, Rohan, et al. "Guided search for task and motion plans using learned heuristics." 2016 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2016.

[7] B. Ames, A. Thackston, and G.D. Konidaris. "Learning Symbolic Representations for Planning with Parameterized Skills", To appear, Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, Oct. 2018.

[8] G. Andersen, G. Konidaris (2017). "Active Exploration for Learning Symbolic Representations".

[9] James, S., Rosman, B., & Konidaris, G. (n.d.). Learning to Plan with Portable Symbols.

[10] Shoham, Y., & Leyton-Brown, K. (2008). Multiagent systems: Algorithmic, Game-Theoretic, and logical foundations. Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations, 9780521899437, 1483. https://doi.org/10.1017/CBO9780511811654

[11] Max Merlin, Heramb Nemlekar and Zhi Li (2019). Learning Symbolic Representations for Loco-Manipulation Planning.

[12] Nikolaidis, Stefanos, Swaprava Nath, Ariel D. Procaccia, and Siddhartha Srinivasa. "Game-theoretic modeling of human adaptation in human-robot collaboration." In 2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI, pp. 323-331. IEEE, 2017.

[13] FFRob: Leveraging Symbolic Planning for Efficient Task and Motion Planning Caelan Reed Garrett, Tomas Lozano-Perez, Leslie Pack Kaelbling

[14] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, Learning Feasibility for Task and Motion Planning in Tabletop Environments, IEEE Robotics and Automation Letters, vol. 4, no. 2, pp. 12551262, Apr. 2019.

[15] M. Adria et al, "High level task planning with inference for the TIAGo robot", Masters Thesis at Universitat Politecnica De Cataluunya, September 2016.

[16] Efficient Geometric Predicates for Integrated Task and Motion Planning Andre K. Gaschler

[17] Yuqian Jiang, Fangkai Yang, Shiqi Zhang, Peter Stone. "Integrating Task-Motion Planning with Reinforcement Learning for Robust Decision Making in Mobile Robots", arXiv:1811.08955, Nov 2018.

[18] Adversarial Actor-Critic Method for Task and Motion Planning Problems Using Planning Experience Beomjoon Kim, Leslie Pack Kaelbling and Tomas Lozano-P erez Computer Science and Artificial Intelligence Laboratory Massachusetts Institute of Technology beomjoon,lpk,tlp@mit.edu

Appendix
Agent-space symbols

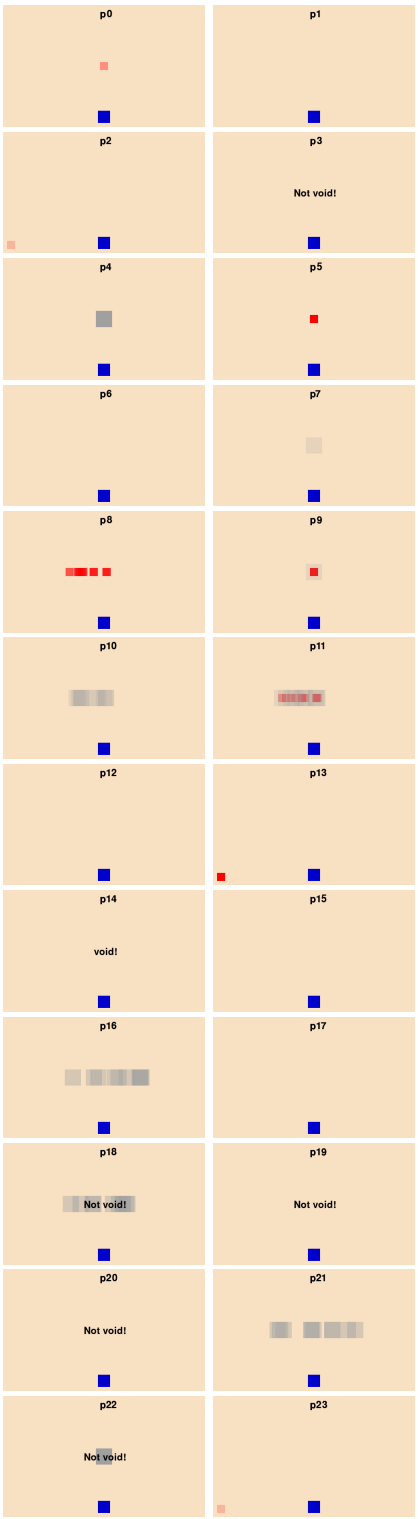

Fig. 17: The whole set of symbols.