Linguistic Fault Handling for Logical Domains

Neil T. Dantam Heni Ben Amor Henrik I. Christensen Mike Stilman

I. Introduction

In real-world systems, failures happen. Reliably executing manipulation tasks depends on handling errors and faults. Typical approaches for logical task planning identify a single execution path, without considering faults. This eases computation, but presents a challenge when the robot must respond to unexpected conditions. The principal challenge in explicitly representing policies for logical domains is handling the state space that is potentially in the number of propositions. We can address this challenge by using a linguistic, hierarchical representation for manipulation task policies. Using a language-based policy representation, we can compactly encode desired execution, potential faults, and the appropriate response.

We consider two types of errors and how they may be handled. First, we consider subtask or action failure, such as a missed grasp, where a chosen action does not produce the desired effect. Second, we consider uncontrollable events, such a force limits, where some transition occurs unpredictable or unavoidably. In both of these cases, we can use a language-based approach to continue and recover.

This approach integrates formal language theory, logical planning, and discrete event system control. Formal language can model both computation and discrete dynamics [5], [9]. A formal language is a set of strings, and a string is a sequence of atomic symbols. This approach has been applied to robots [7], [8], [3]. Different aspects of the relationship between planing, temporal logics, and language have also been explored. SAT-solvers have proven effective for both planning [6] and model checking [1]. Generating plans with loops is analyzed in [10]. [4] considers infinite-string Büchi automata for solving planning problems, and [2] shows a translation from Linear Temporal Logic to the Planning Domain Description Language (PDDL). In this work, we apply language and automata methods to policy generation for fault handling.

II. Planning Domain Languages

Logical planning domains correspond to formal languages over propositions and actions A planning domain defines interleaved sequences of state assignments and action symbols. Starting from some initial state S, we can select actions which lead to subsequent states. The domain defines a set of these sequences which are the potential plans. This corresponds to the definition of a formal language, which is also a set of sequences of symbols. *A planning domain is a formal language,* whose strings are the permissible plans.

The formal language view provides a few techniques to compact the representation: (1) state minimization of a finite automaton, which my run directly from the set of logical actions, (2) compact, context-free forms for hierarchical domains, and (3) independence conditions to enable separate solutions to subgoals.

To minimize state, we modify Hopcroft's algorithm to operate directly on the logical planning domain, avoiding an intermediate, exponentially sized automaton.

To automatically compact hierarchical tasks, we identify repeated *submachines* in the automaton. A submachine is an automaton containing a subset Q' of overall states Q and the edges corresponding only to Q'. A submachine appearing more than once is replaced at each occurrence in the initial automaton with a nonterminal symbol reference to the submachine, compacting the representation by storing equivalent submachines only once.

To further compact the automata, we consider the necessary conditions to independently achieve two different subgoals. Informally, subgoals are independent if we can develop plans and policies for one subgoal without negating other subgoals. Independent subgoals can thus be solved by concatenating plans and policies, turning a potential product of automata states into merely a sum.

III. Error Representation and Recovery

Subtask Failure: Manipulation subtasks may not always be executed correctly. For example, grasps may miss, objects may be dropped, and parts may not align. To reliably execute an overall task, a robot should gracefully handle these errors. We consider the issue of subtask failure by extending the logical planning domain to include alternative effects of actions.

Definition 1 (Alternative Propositional Planning Domain): $D = (\Phi, K, S_0, \Gamma)$, where Φ is the finite set of propositions, and K is the set of actions. Each $K_i = (\alpha, \beta, \kappa, E)$, where $\alpha \subseteq \Phi$ is the set of propositions which must be true before execution, $\beta \subseteq \Phi$ is the set of propositions which must be false before execution, κ is the action symbol, and E is the set of potential effects of the action (a, d) where $a \subseteq \Phi$ is the set of propositions made true by execution, and $d \subseteq \Phi$ is the set of propositions made false by execution. A state S is an assignment of propositions Φ. A set of states Θ corresponds to a boolean formula over Φ. S_0 is the initial state. The goal condition Γ is a list of subgoals in $\Phi \times \{\text{true}, \text{false}\}$.

This variation on planning domains can be readily used with the techniques introduced in Sect. II. The key difference is to consider the alternative effects when using the variation of Hopcroft's algorithm to generate the initial minimum state automaton. When computing the predecessor states of some set, we must consider whether any alternative effect of each action leads to the current set. If so, then that action and

Fig. 1. Potential Maniplations faults. (left) Missed grasp. (middle) Heavy object. (right) Uncontrollable obstacle.

effect denotes a valid predecessor. Once we have this initial automaton, then we can directly apply the previous methods for computing hierarchical decompositions and solving for independent subgoals.

Uncontrollable Events: While a robot can choose which discrete action to attempt, some discrete events may be uncontrollable. These events may represent system faults, hardware limits, or actions of other agents or humans. Controllable events may be blocked by a supervisor, while uncontrollable events may not. Control in the presense of uncontrollable events is well studied in the context of discrete event systems. Now, we relate this approach to logical task planning.

In general, we can test if system G is controllable with regard to specification S by considering prefixes on the controlled system G' which may be followed by an uncontrolled event. The prefixes of X are given as $\widetilde{X}$. All prefixes of the controlled system G' followed by an uncontrollable event which are prefixes of the original system G must also exist in G':

$$G' = G \cap S \qquad \widetilde{G'} Z_{uc} \cap \widetilde{G} \subseteq \widetilde{G'} \tag{1}$$

By representing logical planning domains using language, we can directly apply this result to uncontrollable events within those domains. The planning domain itself corresponds to system G and the goal corresponds to a specification S which is the set of all strings leading to the desired state. Then, we can apply (1) to check if this goal is achievable given the uncontrollable events.

IV. Conclusion

We have proposed a language-based approach for representing manipulation policies and error-recovery. The language form enables both more compact representation and additional analyses to handle failures. Crucially, languages explicitly describe the multiple paths that occur due to failures. This accounts for alternative outcomes of manipulation actions. In addition, we show how to apply the controllablity results from discrete event system control to manipulation domains using the language representation. These results enable more reliable manipulation through correct fault response.

```
(define (domain bimanual-move)
  (:action grasp-left
    (:precondition (not holding-left))
    (:effect (or (not holding-left) holding-left)))
  (:action grasp-right
    (:precondition (not holding-right))
    (:effect (or (not holding-right) holding-right)))
  (:action lift-left
    (:precondition (not heavy)
                   (not holding-right) holding-left)
    (:effect (or lifted heavy)))
  (:action lift-left-right
    (:precondition holding-left) (:effect lifted))
  (:action move
    (:precondition lifted) (:effect moving))
  (:action limit
    (:precondition moving) (:effect limit))
  (:action destination
    (:precondition moving) (:effect destination))
  (:action retract
    (:precondition limit) (:effect (not limit))))
```

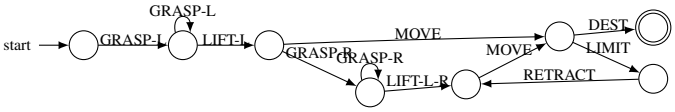

Fig. 2. Example grasping domain for bimanual manipulation and the corresponding policy automaton. Faults include failed grasps, heavy objects require lifting with two hands, and limits which require stopping the motion.

References

[1] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Computer Aided Verification*, pages 359–364. Springer, 2002.

[2] Stephen Cresswell and Alexandra M Coddington. Compilation of ltl goal formulas into pddl. In *European Conference on Artificial Intelligence*, 2004.

[3] Neil T. Dantam and Mike Stilman. The motion grammar: Analysis of a linguistic method for robot control. *IEEE/RAS Transactions on Robotics*, 29(3):704–718, 2013.

[4] Giuseppe De Giacomo and Moshe Y Vardi. Automata-theoretic approach to planning for temporally extended goals. In *Recent Advances in AI Planning*, pages 226–238. Springer, 2000.

[5] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, Reading, MA, 1979.

[6] Henry Kautz and Bart Selman. Blackbox: A new approach to the application of theorem proving to problem solving. In *AIPS98 Workshop on Planning as Combinatorial Search*, volume 58260, pages 58–60, 1998.

[7] Jana Košecká and Ruzena Bajcsy. Discrete event systems for autonomous mobile agents. *Robotics and Autonomous Systems*, 12(3):187–198, 1994.

[8] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Temporal-logic-based reactive mission and motion planning. *Robotics, IEEE Transactions on*, 25(6):1370–1381, 2009.

[9] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *Analysis and Optimization of Systems*, 25(1):206–230, January 1987.

[10] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. Applicability conditions for plans with loops: Computability results and algorithms. *Artificial Intelligence*, 191:1–19, 2012.