Task and Motion Policy Synthesis for Mobile Manipulation

Yue Wang, Neil T. Dantam, Swarat Chaudhuri, and Lydia E. Kavraki

I. Introduction

Robots should safely and correctly operate alongside uncontrollable agents such as humans. In domains involving uncontrollable agents, such as the scenario shown in Fig. 1, robots must react to changes *online* to ensure safety and accomplish desired tasks. Thus, rather than pre-programming finite instructions or pre-computing a single, linear plan, robots need a *policy* that determines the correct response over the infinite-horizon interaction with a changing environment, while also guaranteeing safety and achieving task goals. Task and Motion Synthesis (TMS) constructs such policy that satisfies both high-level task requirements and low-level motion constraints.

In this abstract, we extend the TMS approach presented in [19] (henceforth referred as RoboSynthReact), to solve tasks with *safety* and *recurrence* goals (see Section II for definitions of safety and recurrence). At a high level, RoboSynthReact formulates TMS as a concurrent two-player game [6, 1] between the robot and the environment, and synthesizes a winning policy for the robot by iteratively generating a candidate and verifying its correctness [16]. This iterative policy synthesis procedure converges either to a winning policy or a proof that no such policy exists. We extend RoboSynthReact with two additional policy verification modules for safety and recurrence tasks. Like what we did in RoboSynthReact, we adapt the proof rules of [2] to develop these two policy verification modules, which provide compact, symbolic constraints that characterize the correctness of policies.

Recently, there has been a growing interest in the integration of Task and Motion Planning (TMP) [9, 20, 3, 13, 10, 17, 4, 12]. These previous TMP approaches assume the environment is static, while we consider in this abstract a changing environment with uncontrollable agents where we require not just a plan describing a single execution path, but instead a policy describing the online response to uncontrollable events. Other recent work has focused on reactive synthesis for robots and hybrid systems [8, 21, 5, 18]. These approaches consider the differential dynamics of hybrid systems but do not incorporate efficient path planning. In contrast, we focus on the mobile manipulation domain where high dimensionality makes efficient, collision-free path planning crucial, and we therefore apply fast, sampling-based motion planning methods [14]. Furthermore, these previous works often perform a combinatorial search over large state spaces. In contrast, we avoid expensive combinatorial search using SMT-based symbolic methods, improving the scalability of policy synthesis.

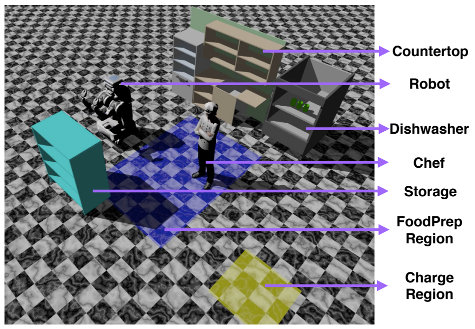


Fig. 1: The robot must navigate through the kitchen and pick a cleaned dish from the dishwasher, while avoiding collisions with the chef. Since the chef's movement is uncontrollable, the robot needs a *policy* to accomplish the task no matter how the chef moves.

We validate our approach in a mobile manipulation domain with human-robot interaction. Our results show that for the tested benchmarks, our method scales better than an alternate synthesizer [15].

II. Problem Formulation

We consider TMS for a controllable robot operating in an environment with uncontrollable agents such as humans. The robot and the environment have both continuous, kinematic state and purely discrete state. The robot can select actions to modify its own state, but it cannot control the state of the adversarial environment. Finally, we specify the desired task as a set of valid state sequences. Our goal is to synthesize a *policy* that guides the robot during execution to accomplish the task.

We formulate TMS as a discrete, concurrent game between the robot and the environment:

Definition 1 (Concurrent Game)**.**
A concurrent game is a tuple $G = (\Sigma, \theta, \Gamma_e, \Gamma_r, \delta, \varphi)$:

- Σ is a state space of the game.
- $\theta \subseteq \Sigma$ is the set of initial game states.
- Γ_e, Γ_r are valid-move functions. For each state s, $\Gamma_e(s)$ and $\Gamma_r(s)$ represents the set of valid *moves* for the environment and the robot at state s, respectively.
- δ is the transition function of the game. For every state $s \in \Sigma$, *move* $a_e \in \Gamma_e(s)$ and *move* $a_r \in \Gamma_r(s)$, $\delta(s, a_e, a_r) \in \Sigma$ is the corresponding *successor* state.

A *play* σ is an infinite sequence of states: $s_0, s_1, \ldots$, such that $s_0 \in \theta$ is an initial state and for $i \geq 0$, s_{i+1} is a successor state of s_i as defined by δ.
- φ is a set of *winning* plays for the robot.

In [19], we have presented an algorithm for solving concurrent games with *liveness* winning conditions: the robot needs to eventually reach a certain state. In this work, we consider solving two other kinds of concurrent games: *safety games*, and *recurrence games*. The wining play set φ for each kind of games is defined formally as follows:

Definition 2 (Safety Games)**.**
In a safety concurrent game $G = (\Sigma, \theta, M_e, M_r, \Gamma_e, \Gamma_r, \delta, \varphi)$, there is a set *safe* of safe states. A play σ is a *winning* play $\sigma \in \varphi$ if every state s in the play σ is a safe sate $s \in$ *safe*.

Definition 3 (Recurrence Specification)**.**
In a recurrence concurrent game $G = (\Sigma, \theta, M_e, M_r, \Gamma_e, \Gamma_r, \delta, \varphi)$, there is a set *recur* of states. A play σ is a *winning* play $\sigma \in \varphi$ if the play σ visits every state s of the set *recur* infinitely often.

III. Policy Synthesis

We extend the policy synthesis algorithm in RoboSynthReact (see Fig. 2) to solve safety and recurrence games. First, the *candidate generator* in Fig. 2 generates a policy candidate based on a given grammar. Then, the *policy verifier* in Fig. 2 verifies the correctness of this candidate using an SMT solver to check symbolic constraints that characterize winning policies. Finally, we generalize verification failures via domain-specific heuristics to help subsequent policy candidate generation, reducing the number of necessary iterations and improving efficiency. This iterative policy synthesis procedure converges either to a winning policy or a proof that no such policy exists. For more details about the algorithms used in RoboSynthReact (see Fig. 2), please refer to [19].

In this work, we consider synthesizing policies for two kinds of concurrent games: *safety* games and *recurrence* games, as defined in section II. We extend RoboSynthReact with two additional policy verification modules for safety and recurrence games.

A. Policy Verification: Safety Games

A safety concurrent game $G = (\Sigma, \theta, M_e, M_r, \Gamma_e, \Gamma_r, \delta, \varphi)$ is associated with a set *safe* of safe states. In a winning play, every state should be a safe state $s \in$ *safe*.

Consider a set *inv* that contains every state in a winning play. The *winning* policies should guarantee that, for every state $s \in$ *inv* and every environment *move* $a_e \in \Gamma_e(s)$, the robot *move* $a_r \in \Gamma_r(s)$ selected by the policy $a_r = p(s)$ should lead to a successor state $s' = \delta(s, a_e, a_r)$ that is also a member of the set *inv*.

The following formulas summarize the above constraints on *winning policies*:

$$\begin{aligned} \forall s, a_e.\ \ & (s \in inv) \wedge (a_e \in \Gamma_e(s)) \ \rightarrow \\ & (\delta(s, a_e, p(s) \in inv) \end{aligned} \quad (1)$$

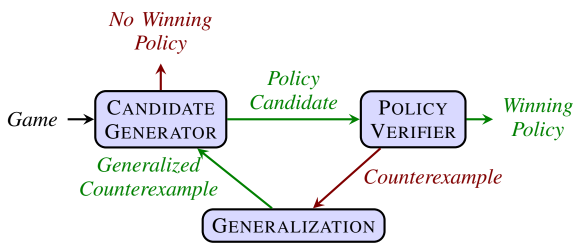


Fig. 2: The core steps of the policy synthesis algorithm used in RoboSynthReact

Our policy synthesizer iteratively constructs the auxiliary set *inv* following the steps described below:

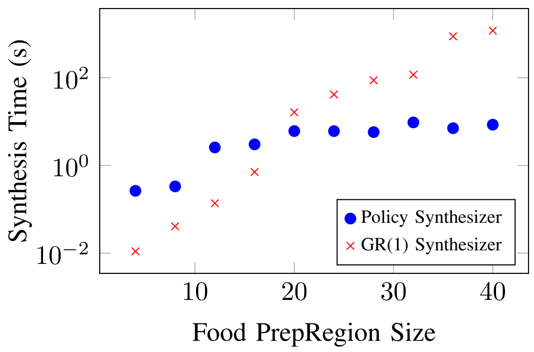
1) Initially, we set *inv* = *safe* since every state in a winning play should be a safe state $s \in$ *safe*.
2) During the policy synthesis process, when we found a *invalid* state s that always leads to an unsafe successor state no matter what robot *move* the policy selects, we shrink the set *inv* by removing this *invalid* state s. To speed up the synthesis process, we also remove similar *invalid* states from the set *inv* using the *generalization* described in [19].

If the policy synthesizer terminates, i.e., finds a policy p such that the constraints in Formula 1 hold, with an *non-empty* set *inv*, then p is a *winning* policy that can always maintain a safe successor state for every state $s \in$ *inv*. However, if the policy synthesizer terminates with an *empty* set *inv*, we can conclude that there is no winning policy for the robot because every state $s \in \Sigma$ will result in an unsafe successor state, no matter what robot *move* the policy selects.

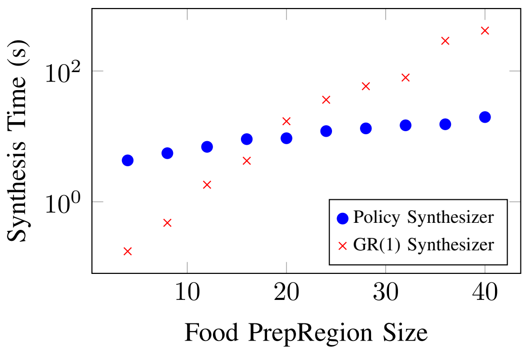
1) Policy Verification: Recurrence Games: A recurrence concurrent game $G = (\Sigma, \theta, M_e, M_r, \Gamma_e, \Gamma_r, \delta, \varphi)$ is associated with a set *recur* of states. A winning play should visit every state $s \in$ *recur* infinitely often.

We treat this recurrence game as a combination of liveness games, where every state $s \in$ *recur* corresponds to a liveness game with a goal state set *goal* $= \{s\}$ that contains only one goal state s. If the recurrence game has a winning play σ, there exist a subsequence σ' of states in the winning play σ that visits every state $s \in$ *recur* in a certain order. Thus, for every state $s \in$ *recur*, our policy synthesizer solve the corresponding liveness game following the algorithm for liveness games [19] and then combined the policies of these liveness games in an appropriate order to obtain the winning policy for the recurrence game.

If the policy synthesizer fails to solve one of these liveness games, or for every order of the states in the set *recur*, it is impossible to combine the polices of these liveness games (i.e., for some state $s \in$ *recur*, it is impossible to reach the next state $s' \in$ *recur*), then we can conclude that there is no winning policy for the robot in this recurrence game. Otherwise, our policy synthesizer will construct a winning policy for the robot in this recurrence game by combining the winning policies of these liveness games in an appropriate order.



(a) Safety Game Benchmark Results



(b) Recurrence Game Benchmark Results

Fig. 3: Performance of our *policy synthesizer* and LTLMoP back-end GR(1) synthesizer as the size of the *FoodPrep Region* in the scene increases. Our *policy synthesizer* scales better in these tested benchmarks.

IV. Experiments

We evaluate our policy synthesis approach in a kitchen environment (Fig. 1) with a simulated PR2 robot and uncontrollable agents (e.g., chefs) moving within the *FoodPrep Region*. We compare our policy synthesizer with the back-end of *LTL MissiOn Planner* (LTLMoP), a state-of-the-art synthesis tool for robotic applications [11]. The back-end of LTLMoP is based on the GR(1) synthesis algorithm presented in [15]. We note also that LTLMoP contains many front-end features, such as natural language processing, that are orthogonal or even complementary to the policy synthesis work we present in this paper.

All experiments were carried out on an 8 core 3.4 GHz machine with 8 GB memory. We use Z3 [7] as our backend SMT solver and utilize the *linear arithmetic* and *uninterpreted functions* theory solvers of Z3. For all benchmarks, the size of the workspace is fixed, and there are 2 chefs in the kitchen.

A. Benchmark: Safety Games

In the safety game benchmark, there are two task requirements for the robot:

- Always avoid collisions with the people.
- When the current region is dirty, clean the current region.

The performance results for the safety property benchmark are shown in Figure 3a. Note that the result graph is plotted on a semi-log scale. For small size problems (*FoodPrep Region* size ≤ 16), the GR(1) Synthesizer performs better than our *policy synthesizer*. However, for problems where *FoodPrep Region* size is greater than 16, our method performs better. Moreover, for problems with size above 32, the GR(1) Synthesizer took more than 10 minutes while our *policy synthesizer* solves all problems within 10 seconds.

B. Benchmark: Recurrence Games

In the recurrence game benchmark, there are two task requirements for the robot:

- Always avoid collisions with the people.
- Visit the marked regions infinitely often.

The performance results for the recurrence property benchmark are shown in Figure 3b. This result graph is also plotted on a semi-log scale. The scalability results of this benchmark are similar to the results of the safety benchmark. For small size problems, the GR(1) synthesizer performs slightly better than our *policy synthesizer* while for problems where *FoodPrep Region* size is greater than 16, our method starts to perform better.

V. Conclusion

We have presented a symbolic approach to address Task and Motion Synthesis (TMS) problems where the robot must accomplish tasks in environments involving uncontrollable agents such as humans, with safety and recurrence goals. Our results show that, compared to an existing robotic synthesis tool – the GR(1) back-end of LTLMoP – our approach scales better for the benchmark problems.

In this work, we assume that the robot has perfect sensing with no observation uncertainty. However, physical robots often have significant noise and error in sensing. An important ongoing question is how to extend the current policy synthesis implementation to handle uncertainty in the environment, including sensor noise or other probabilistic beliefs. Investigating this direction would improve our policy synthesizer's ability to handle problems with physical uncertainties.

References

[1] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5): 672–713, September 2002. ISSN 0004-5411. doi: 10.1145/585265.585270. URL http://doi.acm.org/10.1145/585265.585270.

[2] Tewodros Beyene, Swarat Chaudhuri, Corneliu Popeea, and Andrey Rybalchenko. A constraint-based approach to solving games on infinite graphs. In *POPL*, pages 221–233, 2014.

[3] A. Bhatia, M.R. Maly, L.E. Kavraki, and M.Y. Vardi. Motion planning with complex goals. *Robotics & Automation Magazine, IEEE*, 18(3):55–64, Sept 2011. ISSN 1070-9932. doi: 10.1109/MRA.2011.942115.

[4] Marcello Cirillo, Federico Pecora, Henrik Andreasson, Tansel Uras, and Sven Koenig. Integrated motion planning and coordination for industrial vehicles. In *ICAPS*, 2014.

[5] N. Dantam and M. Stilman. The motion grammar: Analysis of a linguistic method for robot control. *Robotics, IEEE Transactions on*, 29(3):704–718, June 2013. ISSN 1552-3098. doi: 10.1109/TRO.2013.2239553.

[6] L. de Alfaro and T.A. Henzinger. Concurrent omega-regular games. In *Logic in Computer Science, 2000. Proceedings. 15th Annual IEEE Symposium on*, pages 141–154, 2000.

[7] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *TACAS*, pages 337–340, 2008.

[8] Jonathan A. Decastro and Hadas Kress-Gazit. Synthesis of nonlinear continuous controllers for verifiably correct high-level, reactive behaviors. *Int. J. Rob. Res.*, 34(3):378–394, March 2015. ISSN 0278-3649. doi: 10.1177/0278364914557736. URL http://dx.doi.org/10.1177/0278364914557736.

[9] C. Dornhege, M. Gissler, M. Teschner, and B. Nebel. Integrating symbolic and geometric planning for mobile manipulation. In *Safety, Security & Rescue Robotics (SSRR), 2009 IEEE International Workshop on*, pages 1–6, 2009.

[10] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *ICRA*, pages 4575–4581, 2011.

[11] C. Finucane, Gangyuan Jing, and H. Kress-Gazit. Ltlmop: Experimenting with language, temporal logic and robot control. In *IROS*, pages 1988–1993, 2010.

[12] Keliang He, Morteza Lahijanian, Lydia E Kavraki, and Moshe Y Vardi. Towards manipulation planning with temporal logic specifications. In *ICRA*, pages 346–352. IEEE, 2015.

[13] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *ICRA*, pages 1470–1477, 2011.

[14] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, Aug 1996. ISSN 1042-296X. doi: 10.1109/70.508439.

[15] Nir Piterman, Amir Pnueli, and Yaniv Saar. Synthesis of reactive (1) designs. In *Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer, 2006.

[16] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. In *ASPLOS*, pages 404–415, 2006.

[17] Sanjeev Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stephen Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *ICRA*, pages 639–646, 2014.

[18] Alphan Ulusoy, Michael Marrazzo, and Calin Belta. Receding horizon control in dynamic environments from temporal logic specifications. In *Robotics: Science and Systems*, 2013.

[19] Yue Wang, Neil Dantam, Swarat Chaudhuri, and Lydia Kavraki. Task and motion policy synthesis as liveness games. In *ICAPS*, 2016.

[20] Jason Wolfe, Bhaskara Marthi, and Stuart J. Russell. Combined task and motion planning for mobile manipulation. In *ICAPS*, 2010.

[21] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon control for temporal logic specifications. In *HSCC*, pages 101–110, 2010.