Maximizing Independence in Auction-Based Scheduling via Successive Refinement

Guy Avni[1], Kaushik Mallik[2], Suman Sadhukhan[3], and Tomer Yarkoni[4]

[1] Department of Computer Science, University of Haifa, Israel
gavni@cs.haifa.ac.il
[2] IMDEA Software Institute, Spain
kaushik.mallik@imdea.org
[3] Clausthal University of Technology, Germany
suman.sadhukhan@tu-clausthal.de
[4] Technion, Israel
tomeryar06@gmail.com

Here we present one of our works in our ongoing research on *decoupled synthesis*, currently under submission.

We propose a decoupled approach to synthesizing a word, letter by letter, that is in the conjunction of a given pair of regular objectives. A key application is multi-objective robotic path planning, where each letter corresponds to a robot action and the goal is to find a plan that satisfies both objectives. The traditional monolithic solution would construct the product automaton, and obtains a "generator" that outputs an accepting word. Instead, we synthesize two independent generators and compose them at runtime via an auction-based mechanism: at each time step, the generators bid for who chooses the next symbol. Advantages of this approach include design in parallel or by different vendors, and reusability, namely when an objective changes only the relevant generator is updated and the other is reused. We design, for the first time, a framework in which each generator is designed on a separate automaton, which enables specifying objectives as logical formulas. For cases in which a feasible solution is not found, we develop a successive refinement algorithm that searches for a pair of assumptions that regain feasibility. Weaker assumptions lead to increased modularity. Our algorithm is based on a novel automata-learning algorithm that can be of independent interest. We design proof-of-concept experiments where we implement our algorithm, and demonstrate the effectiveness.

We concretize our problem of interest using an example of multi-objective path planning. Consider a cleaning robot that must collect litter pieces as well as visit a charging station at regular intervals. Here, Σ is the set of available action primitives (move forward, etc.), L_1 is the set of finite action sequences that visit all litter locations, and L_2 is the set of finite action sequences that visit the charging station at the required frequency. Thus, a word in $L_1 \cap L_2$ constitutes a *plan* for the robot that satisfies both its tasks. The goal is to independently (i.e, in a *decoupled* manner) synthesize two programs M_1 and M_2 which generate a letter from Σ at each step to fulfill their respective tasks. The main difficulty in decoupling is composition, which needs to resolve conflicts between the generators. For example, let $\Sigma = \{a, b\}$, L_1 consists of words with a as the first

letter, i.e, $L_1 = a(a+b)^*$, L_2 consists of the words with b as the second letter, i.e, $L_2 = (a+b)b(a+b)^*$. Consider generators M_1 and M_2 that output a^* and b^*, respectively, thereby satisfying their own objectives when operating alone. A mechanism composes M_1 and M_2 by choosing which generator's output to pick at each step. Note that not all compositions lead to global correctness: if M_2 chooses a letter first, followed by M_1, the composition generates ba, which violates both L_1 and L_2. Rather, M_1 should choose first followed by M_2, to generate $ab \in L_1 \cap L_2$. Moreover, to maximize modularity, a composition mechanism should be independent of L_1 and L_2 so that changes in either L_1 or L_2 do not require changes to the other generator [2].

We build upon our framework called auction-based scheduling (ABS) [3], in which each generator M_i starts with a positive budget B_i^0 such that $B_1^0 + B_2^0 = 1$. At each step, the mechanism auctions the right to choose the next symbol: each generator chooses a bid that does not exceed their available budget, the higher bidder selects the symbol and pays their bid to the other generator. Intuitively, a generator bids more when it is urgent to control the next symbol. The computational task is to choose initial budgets and local strategies so that both objectives can be enforced. Returning to L_1 and L_2 above, consider the allocation $B_1^0 = 0.75$ and $B_2^0 = 0.25$. In the first step, M_1 bids $b_1 = 0.25 + \epsilon$, and necessarily wins the bidding since $b_1 > B_2^0$, thus M_1 chooses the first symbol to be a. The bid is paid to M_2 so $B_1^1 = 0.5 - \epsilon$ and $B_2^1 = 0.5 + \epsilon$. Next M_2 secures b by bidding all in.

Global correctness of the composition is guaranteed as follows. Each generator M_i is equipped with a *budget requirement* for satisfying L_i; formally, a *budget threshold* $t_i \in [0,1]$ for L_i is the smallest value such that if M_i is allocated a budget greater than t_i, it can enforce L_i even in the extreme case that M_i is composed with an adversary. If $t_1 + t_2 < 1$, there is a budget allocation with $B_1^0 > t_1$ and $B_2^0 > t_2$, and $M_1 \bowtie M_2$ generates a word in $L_1 \cap L_2$.

We address a central limitation of the above ABS framework [3] in the current work; ABS assumes that the generators operate on a shared graph with separate reachability objectives specified on the vertices of the graph. This imposes a strict restriction on the specifications that the framework can handle. To see this, recall that it is a common practice to specify planning tasks in logic, e.g., [4, 5, 8, 6]. To illustrate, in the robot example, (1) the requirement to charge in every T time units, by visiting a charging station located at C, can be formalized in LTLf as $\varphi_1 = G\big(\bigvee_{1 \le i \le T} X^i C\big)$, and (2) collecting litter that is located in locations L as $\varphi_2 = \bigwedge_{\ell \in L} F\ell$. Given specifications φ_1 and φ_2, one first translates the two to automata $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$ that recognize the language of words that satisfy the corresponding specification. Once $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$ are constructed, the goal is to find a word in their intersection. However, the original ABS framework [3] cannot be applied, because it requires $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$ to have the same underlying transition graph, which is unrealistic. A workaround is to construct the product transition system $\mathcal{A}_{\varphi_1} \times \mathcal{A}_{\varphi_2}$, but this undermines modularity and produces solutions that are not reusable. Our framework, on the other hand, directly applies to the separate automata.

We proceed as follows. Let $\mathcal{A}_i$ be an automaton that recognizes the language L_i, for $i \in \{1, 2\}$. We solve a *bidding game* [7] on $\mathcal{A}_i$, which produces a generator M_i and the threshold budget $t_i \in [0, 1]$ that is required to ensure that the output word is in L_i, even when composed with an adversary. If the composition is feasible, i.e., $t_1 + t_2 < 1$, we are done, namely $M_1 \bowtie M_2$ is a word in $L_1 \cap L_2$. Otherwise, we successively refine the adversarial assumption in threshold computation by successively introducing assumptions. To maximize modularity, we seek weakest assumptions that lead to feasible solutions, as we demonstrate using an example.

Our successive refinement procedure is based on the observation that for $i \in \{1, 2\}$, a generator M_i for L_i will never choose a *bad prefix*, namely x such that for every suffix y, we have $x \cdot y \notin L_i$. Thus, when designing M_i, the set of bad prefixes of the other generator, denoted $\mathtt{bad}(L_{3-i})$, can serve as an *assumption* to decrease the budget requirement. Our successive refinement algorithm is based on automata learning and could be of independent interest. We note that the classic L* [1] cannot be applied in our setting since it produces languages that are not contained in the target language $\mathtt{bad}(L_i)$ and thus compromises soundness. Our novel automata-learning algorithm produces a sequence of languages that are contained in the target language.

From a known result [3, Thm. 5], it follows that if $|\Sigma| = 2$, our refinement scheme is *guaranteed* to find a feasible pair of generators. To put this into perspective, if the robot were instead placed in a one-directional maze where, at each location, it can choose between at most two alternative directions, it is *guaranteed* that a pair of feasible generators exist. Our successive refinement terminates in finite steps, and in the limit, it returns the full assumption set $\mathtt{bad}(L_{3-i})$. Therefore, our refinement algorithm is no more incomplete than the baseline solution of using $\mathtt{bad}(L_{3-i})$ for designing each generator M_i, which has been already studied (under the name "assume-admissible synthesis") for ABS on shared graphs [3]. Moreover, our successive refinement finds the weakest assumptions that attain feasibility, which provides us maximal modularity and reusability.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987)
2. Avni, G., Henzinger, T.A., Mallik, K., Sadhukhan, S., Thejaswini, K.: Decoupled planning for multiple omega-regular objectives. In: Proc. 38th CAV (2026)
3. Avni, G., Mallik, K., Sadhukhan, S.: Auction-based scheduling. In: Proc 30th TACAS. Lecture Notes in Computer Science, vol. 14572, pp. 153–172. Springer (2024)
4. Bacchus, F., Kabanza, F.: Planning for temporally extended goals. Ann. Math. Artif. Intell. **22**(1-2), 5–27 (1998)
5. Giacomo, G.D., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Rossi, F. (ed.) Proc. 23rd IJCAI. pp. 854–860. IJCAI/AAAI (2013)

6. Lahijanian, M., Almagor, S., Fried, D., Kavraki, L.E., Vardi, M.Y.: This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction. In: Proc. 29th AAAI. pp. 3664–3671 (2015)
7. Lazarus, A.J., Loeb, D.E., Propp, J.G., Stromquist, W.R., Ullman, D.H.: Combinatorial games under auction play. Games and Economic Behavior **27**(2), 229–264 (1999)
8. Patrizi, F., Lipovetzky, N., Giacomo, G.D., Geffner, H.: Computing infinite plans for LTL goals using a classical planner. In: Proc. 22nd IJCAI. pp. 2003–2008. IJCAI/AAAI (2011)