Authoring Human Simulators via Probabilistic Functional Reactive Program Synthesis

Michael Jae-Yoon Chung
Computer Science & Engineering
University of Washington
Seattle, Washington
mjyc@cs.washington.edu

Maya Cakmak
Computer Science & Engineering
University of Washington
Seattle, Washington
mcakmak@cs.washington.edu

***Abstract*—One of the core challenges in creating interactive behaviors for social robots is testing. Programs implementing the interactive behaviors require real humans to test and this requirement makes testing of the programs extremely expensive. To address this problem, human-robot interaction researchers in the past proposed using human simulators. However, human simulators are tedious to set up and context-dependent and therefore are not widely used in practice. We propose a program synthesis approach to building human simulators for the purpose of testing interactive robot programs. Our key ideas are (1) representing human simulators as probabilistic functional reactive programming programs and (2) using probabilistic inference for synthesizing human simulator programs. Programmers then will be able to build human simulators by providing interaction traces between a robot and a human or two humans which they can later use to test interactive robot programs and improve or tweak as needed.**

I. Introduction

The past decade has been marked by the rise of social robots, with soaring interest from start-ups, large corporations, and researchers alike. Purchasing and building social robot hardware have been democratized through declining price points and the availability of fabrication and physical computing tools. However, programming a social robot for a particular application still requires specialized software development skills. As a result, the ability to explore new applications is limited to a small population.

Many companies building social robot platforms aim to address this problem by outsourcing application development to 3rd party developers by releasing software development kits (SDK) [1]–[4]. Such SDK often consists of (1) an application programming interface (API) for programming the company's robot platform which is implemented in a particular programming language and (2) other tools and utilities, e.g., for debugging and configuration purposes.

While SDK enables developers to program social robots, creating and maintaining robot applications is still difficult. The core problem is testing the interactive robot behaviors. To achieve natural and robust interaction, developers need to test programs representing the robot behaviors with target human users, which is an extremely expensive task. Recognizing this problem, human-robot interaction researchers have employed human simulators to systematically evaluate robot behavior [5]–[7]. Still, human simulators are not widely used as they are laborious to build and difficult to reuse because of their context and user-dependent nature.

In this paper, we propose a program synthesis approach to building task-dependent human simulators for testing purposes. We propose to represent human simulators as programs in a probabilistic functional reactive programming language to enable sampling interaction traces and make reasoning about time possible. To enable programmers to rapidly prototype human simulators, we propose a program synthesis that allows users to provide an incomplete program and a set of input/output pairs and then completes the program using probabilistic inference.

II. Related Work

Human-robot interaction researchers have a long history of employing human simulators [6], [8]–[12]. Notably, Steinfeld and colleagues explicitly emphasized (1) the distinction between rigorous human modeling and simplified simulations for evaluation purposes [5] and (2) the importance of systematically evaluating interaction methods and systems with the latter, i.e., human simulators. To this end, the community developed multiple simulation libraries and frameworks for evaluating service robot applications [13], [14] and control algorithms designed for providing physical assistance [15]. Our work is the most closely related to the work of Chao and colleagues that proposes timed Petri nets for representing socially intelligent robot behaviors and evaluated them using hand-built human simulators also implemented as TPNs [6]. Similar to our work, they emphasized the importance of the representation's ability to model time and asynchronous events Unlike Chao and colleagues' work, we use a probabilistic functional reactive programming language to represent both robot and human behaviors and focus on exploring ways to quickly create human simulators.

Recently, human-robot interaction researchers investigated applications of program synthesis. *Synthè* is a tool that enables non-programmers like designers to author social robot behaviors by acting out multiple demonstrations of an interaction [16]. The creators of *Synthè* treated the problem of authoring from high-level and discrete interaction trances as a program synthesis with input-output specifications and used a MaxSMT solver to solve the synthesis problem. A program synthesis approach also has been employed to ease the work of creating a low-level controller for a human-robot interaction task like handover [17]; users specify desired behaviors in Signal Temporal Logic, which a Mixed-Integer Linear Problem solver takes to output a controller. Hammond and colleagues applied a program repair approach, a variant of program synthesis, to automatically recover failures while running end-user programs [18]. A probabilistic inference method was used to detect and recover failures on runtime. While there is more work of applying program synthesis in the human-robot interaction domain, our work is unique in its purpose, i.e., testing interactive robot behaviors.

Outside of the human-robot interaction research literature, our work is related to work investigating techniques to use a model to guide test generation for reactive systems [19], spoken dialogue systems [20], computer vision [21], and autonomous driving [22] and modeling temporal processes via probabilistic programming [23], [24]. Our work is also inspired by a popular reactive programming framework in the web development community such as RxJS[1] and an existing method for synthesizing functional reactive programs [25], [26].

III. Approach

A. Probabilistic Functional Reactive Programming

Probabilistic functional reactive programming is a tool for designing and using complex stochastic asynchronous event streams where the streams refer to variables that can change their values over time and the events refer to the occurrence of a new value in such streams. Our key idea is to borrow features from an existing programming language that is great at creating complex event streams, i.e., functional reactive programming, and another language that is great at creating statistical models, i.e., probabilistic programming, and apply them for modeling stochastic asynchronous event streams, e.g., human behaviors.

Let us try to model an extremely simple speaking behavior of a person, e.g., of speaking and being silent alternatively. Here is an example functional reactive program using RxJS's observables like `interval` and operators like `map`:[2]

```
var human = interval(1000).pipe(
  map(function (number) {
    return number + 1;
  }),  // periodically emits from 1
  startWith(0),  // emits 0 immediately
  map(function (number) {
    return number % 2 === 0
      ? "speak"  // if number is even
      : "silent";  // if number is odd
  }),  // maps the number to a string
);

// human emits:
// "silent" at 0ms
// "speak" at 1000ms
// "silent" at 2000ms
// "speak" at 3000ms
// ...
```

The program creates an event stream that alternatively emits strings "silent" and "speaking" at a fixed interval of 1000 milliseconds. It essentially implements a simple timed state machine with two states.

It is likely that the fixed interval is too restrictive to model a speaking behavior of a person. Here is an example of the program that creates an event stream that emits strings "silent" and "speaking" at two different fixed intervals, 2000 milliseconds and 1000 milliseconds.

```
var makeHuman = function(state) {
  return merge(
    of(state),  // emits state immediately
    of(state).pipe(
      // Delays an event
      delay(
        // Select a delay duration per state
        state === "speak" ? 2000 : 1000
      ),
      // Maps an event to a stream, resulting
      // in a stream of streams
      map(function (s) {
        // This body gets called in future,
        // i.e., after delay
        return makeHuman(
          // State transition function
          s === "speak"
            ? "silent"
            : "speak"
        );
      }),
      // Flattens the stream of streams
      switchAll(),
    )
  );
};
var human = makeHuman("silent");

// human emits:
// "silent" at 0ms
// "speak" at 1000ms
// "silent" at 3000ms
// "speak" at 4000ms
// "silent" at 6000ms
// ...
```

To create a stream that infinitely emits events with alternating intervals, this program employs a higher order stream to call the recursive function in future (i.e., after delay).

While the new program is better, a real person is likely to speak (or be silent) for a stochastic amount of duration, and therefore still too restrictive to model the desired human behavior. The following example introduces some randomness in the previous example program by employing probabilistic programming's elementary random primitives like `gaussian`:[3]

```
var makeHuman = function(state) {
  return merge(
    of(state),
```

[1] https://rxjs.dev/

[2] We assume readers' familiarity with RxJS and JavaScript. For a tutorial on reactive programming involving RxJS in JavaScript, see https://gist.github.com/staltz/868e7e9bc2a7b8c1f754.

[3] We used the syntax of WebPPL (http://webppl.org/). For a tutorial on probabilistic programming involving WebPPL, see http://adriansampson.net/doc/ppl.html.

```
      of(state).pipe(
        // Sample durations at each occurrence
        var speakDuration = gaussian(2000, 1000);
        var silentDuration = gaussian(1000, 500);
        delay(state === "speak"
          ? speakDuration
          : silentDuration
        ),
        map(function (s) {
          // State transition function
          return makeHuman(s === "speak"
            ? "silent"
            : "speak"
          );
        }),
        switchAll()
      )
    );
  };
  var human = makeHuman("silent");

  // human emits:
  // "silent" at 0ms
  // "speak" at a sampled milliseconds from
  //   gaussian(1000, 500)
  // "silent" at the previous event timestamp
  //   plus a sampled milliseconds from
  //   gaussian(2000, 1000)
  // "speak" at the previous event timestamp
  //   plus a sampled milliseconds from
  //   gaussian(1000, 500)
  // ...
```

This program creates a stream that emits two events alternatively with sampled intervals from two normal distributions. Once again, the use of recursion and higher order stream enables sampling from the two distributions at every event occurrence, which makes the overall event emission pattern more random.

Our final program is still too rudimentary to model an interesting human behavior, however, we believe creating a more complex program, e.g., consisting of more states and multiple streams, is straightforward.

B. Program Synthesis via Probabilistic Inference

Program synthesis is the idea of automatically generating a program that implements the desired specification. We employ the specification style of using a set of input/output pairs, or rather a set of input/output traces where a trace is a set of recorded event value and timestamp pairs. The probabilistic functional reactive program synthesis relies on sketching and probabilistic inference. Sketching is a synthesis technique that first asks the user for a template program (i.e., sketch) that implements the high-level structure of the desired program with a specification and then fills in the details automatically.

For example, in the previous scenario of modeling a speaking behavior, it might be difficult or tedious for a programmer to select distribution parameters for the duration random variables. Using a sketch, the programmer can leave such an uncertain part of the program with holes.

```
  ...
        // Sample durations at each occurrence
        var h1 = uniform(0, 10000);
        var h2 = uniform(0, 10000);
        var speakDuration = gaussian(h1, 1000);
        var silentDuration = gaussian(h2, 500);
  ...
```

In this example, variables `h1` and `h2` and their exact values will be determined by the synthesizer.

Programmers can use holes to express uncertainty in control flow. For example, a programmer can create a sketch that expresses uncertainty in state transition.

```
  ...
          // State transition function
          h3 = flip(0.5);
          return makeHuman(h3
            ? // 1st transition function
              s === "speak"
              ? "silent"
              : "speak"
            : // 2nd transition function
              s === "speak"
              ? "hesitate"
              : s === "hesitate"
              ? "silent"
              : "speak"
          );
          // should define hesitateDuration
          //   for the 2nd transition function
  ...
```

In this example, depends on the value of `h3`, different state transition function will be used.

To turn the synthesis problem into a probabilistic inference problem, we propose defining holes as random variables. With our choice of the specification style, a set of input and output traces, as a dataset, typical probabilistic inference techniques like MCMC can be used for determining the most likely values for the holes.

C. Human Simulator and Robot Behavior Authoring Workflow

The goal of the proposed synthesis approach was to enable programmers to easily create and maintain human simulators for testing interactive robot behaviors. To that end, we propose the following workflow for programmers when developing interactive robot behaviors.

1) Define a target human-robot interaction and create an initial robot program and a human simulator sketch.
2) Collate input and output traces from human-robot or human-human interactions.
3) Synthesize the human simulator program with the collected traces.
4) Update the robot behavior.
5) Repeat 2)-4) until satisfied.

When the output human simulator needs to be tweaked, e.g., to model a similar human behavior in a different context, the programmer can manually set the values for the holes or by repeating steps 2) through 4). In addition, the initial sketch can be improved and refactored over time as the programmer gains experience in working with the synthesizer for to make the re-using process more effective.

IV. Conclusion

In this paper, we presented (1) the idea of using probabilistic functional reactive programming for representing human behaviors, (2) a program synthesis technique based on probabilistic inference for building human simulators in the context of testing interactive programs, (3) and a potential workflow for using the synthesizer for developing interactive robot behaviors. We believe the problem of building human simulators for testing purposes is important and the proposed approach for representing human behaviors and creating human simulator programs opens up new and exciting research directions such as other synthesis techniques, human simulator domain-specific language design, and further applications.

References

[1] E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier, "Choregraphe: a graphical tool for humanoid robot programming," in *Proceedings of the 18th IEEE International Symposium on Robot and Human Interactive Communication*, pp. 46–51.

[2] A. Naveed, "Jibo SDK - The Beginnings," ttps://youtu.be/POv4bYS3eRQ, 2016, accessed: 2020-12-14.

[3] Anki, "Cozmo SDK," http://cozmosdk.anki.com/docs/, 2016, accessed: 2020-12-14.

[4] MYSTYROBOTICS, "MYSTYROBOTICS SDK," http://sdk.mistyrobotics.com/, 2020, accessed: 2020-12-14.

[5] A. Steinfeld, O. C. Jenkins, and B. Scassellati, "The oz of wizard: simulating the human for interaction research," in *Proceedings of the 4th ACM/IEEE International Conference on Human-robot Interaction*, 2009, pp. 101–108.

[6] C. Chao and A. L. Thomaz, "Timing in multimodal turn-taking interactions: Control and analysis using timed petri nets," *Journal of Human-Robot Interaction*, vol. 1, no. 1, pp. 4–25, 2012.

[7] J. G. Trafton, L. M. Hiatt, A. M. Harrison, F. P. Tamborello, S. S. Khemlani, and A. C. Schultz, "Act-r/e: An embodied cognitive architecture for human-robot interaction," *Journal of Human-Robot Interaction*, vol. 2, no. 1, pp. 30–55, 2013.

[8] J. G. Trafton, A. C. Schultz, D. Perznowski, M. D. Bugajska, W. Adams, N. L. Cassimatis, and D. P. Brock, "Children and robots learning to play hide and seek," in *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-robot Interaction*, 2006, pp. 242–249.

[9] D. Sadigh, S. Sastry, S. A. Seshia, and A. D. Dragan, "Planning for autonomous cars that leverage effects on human actions." in *Robotics: Science and Systems*, vol. 2, 2016.

[10] S. Nikolaidis, Y. X. Zhu, D. Hsu, and S. Srinivasa, "Human-robot mutual adaptation in shared autonomy," in *Proceedings of the 12th ACM/IEEE International Conference on Human-robot Interaction*, 2017, pp. 294–302.

[11] R. Choudhury, G. Swamy, D. Hadfield-Menell, and A. D. Dragan, "On the utility of model learning in hri," in *Proceedings of the 14th ACM/IEEE International Conference on Human-robot Interaction*, 2019, pp. 317–325.

[12] M. J.-Y. Chung and M. Cakmak, "Iterative repair of social robot programs from implicit user feedback via bayesian inference," *interaction*, vol. 1, p. 2.

[13] S. Lemaignan, G. Echeverria, M. Karg, J. Mainprice, A. Kirsch, and R. Alami, "Human-robot interaction in the morse simulator," in *Proceedings of the 7th ACM/IEEE International Conference on Human-robot Interaction*, 2012, pp. 181–182.

[14] O. Robotics, "Service Robot Simulator," https://www.openrobotics.org/blog/2018/5/22/service-robot-simulator, 2018, accessed: 2020-12-14.

[15] Z. Erickson, V. Gangaram, A. Kapusta, C. K. Liu, and C. C. Kemp, "Assistive gym: A physics simulation framework for assistive robotics," in *IEEE International Conference on Robotics and Automation*, 2020, pp. 10 169–10 176.

[16] D. Porfirio, E. Fisher, A. Sauppé, A. Albarghouthi, and B. Mutlu, "Bodystorming human-robot interactions," in *Symposium on User Interface Software and Technology*, 2019.

[17] A. Kshirsagar, H. Kress-Gazit, and G. Hoffman, "Specifying and synthesizing human-robot handovers," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019.

[18] J. C. Hammond, J. Biswas, and A. Guha, "Automatic failure recovery for end-user programs on service mobile robots," *arXiv preprint arXiv:1909.02778*, 2019.

[19] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, "Model-based testing of reactive systems: Advanced lectures," in *Springer LNCS*, vol. 3472, 2005.

[20] J. Schatzmann, K. Weilhammer, M. Stuttle, and S. Young, "A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies," *Knowledge Engineering Review*, vol. 21, no. 2, pp. 97–126, 2006.

[21] C. Jiang, S. Qi, Y. Zhu, S. Huang, J. Lin, L.-F. Yu, D. Terzopoulos, and S.-C. Zhu, "Configurable 3d scene synthesis and 2d image rendering with per-pixel ground truth using stochastic grammars," *International Journal of Computer Vision*, vol. 126, no. 9, pp. 920–941, 2018.

[22] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: a language for scenario specification and scene generation," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 63–78.

[23] N. D. Goodman and A. Stuhlmüller, "The Design and Implementation of Probabilistic Programming Languages," http://dippl.org, 2014, accessed: 2020-12-9.

[24] G. Baudart, L. Mandel, E. Atkinson, B. Sherman, M. Pouzet, and M. Carbin, "Reactive probabilistic programming," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 898–912.

[25] J. L. Newcomb and R. Bodik, "Using human-in-the-loop synthesis to author functional reactive programs," *arXiv preprint arXiv:1909.11206*, 2019.

[26] B. Finkbeiner, F. Klein, R. Piskac, and M. Santolucito, "Synthesizing functional reactive programs," in *Proceedings of the 12th ACM SIGPLAN International Symposium on Haskell*, 2019, pp. 162–175.