Software-Synthesis via LL(*) for Context-Free Robot Programs

Arash Rouhani, Neil Dantam, and Mike Stilman

I. Introduction

Producing reliable software for robotic systems requires formal techniques to ensure correctness. Some popular approaches model the discrete dynamics and computation of the robot using finite state automata or linear temporal logic. We can represent more complicated systems and tasks, and still retain key guarantees on verifiability and runtime performance, by modeling the system instead with a context-free grammar. The challenge with a context-free model is the need for a more advanced software synthesis algorithm. We address this challenge by adapting the LL(*) parser generation algorithm, originally developed for program translation, to the domain of online robot control. We demonstrate this LL(*) parser generation implementation in the Motion Grammar Kit, [1] permitting synthesis for robot control software for complex, hierarchical, and recursive tasks.

This tool promotes development of reliable robot software by automating production of executable code from powerful, verifiable models. Context-free grammars can model robot behavior, providing advantages in hierarchical task decomposition, verification, and supervisory control [4, 5]. To directly use the grammar for robot control, it must be translated to machine code, i.e, a computer program; this is *parser generation*. This is a well studied problem in the field of compiler development [1], but online robot control presents a unique set of challenges. There are many parser types which handle different subsets of the context-free set. Here we present the LL(*) parser type [10] adapted to robot control. This parser is strictly more powerful than the one used in [6]; therefore, it generates software from a broader set of grammars.

The *Motion Grammar* is a model for robotic systems that augments a context-free grammar with semantic rules for continuous system dynamics. The terminal symbols of the grammar represent system states or sensor readings. The productions of the grammar form top-down task decomposition, defining an online control *policy* for the robot. During operation, selected productions will execute the semantic rules to compute input commands for the robot. Thus, controlling the robot corresponds to parsing sensor readings online according to the rules of the grammar.

There are several related modeling and verification approaches for robots. Language and Automata Theory [9] was first applied to Discrete Event Systems (DES) by [11]. Hybrid systems extend this approach by including continuous dynamics. Typically, the discrete dynamics are modeled with finite state [2, 3, 8]. Linear Temporal Logic is a popular finite state model [12, 7] which corresponds to Büchi Automata. Our tool operates on Context-Free grammars, which are a strict superset of finite automata. However, it is still possible to verify safety properties of these systems [4].

[1] http://github.com/golems/motion-grammar-kit

II. LL(*) and the Motion Grammar Kit

The tool presented in this paper synthesizes a C-program for a Motion Grammar using the LL(*) algorithm. This generated *Motion Parser* controls the online operation of the robot. For grammar symbols representing semantic rules, C-function stubs are produced. These semantic rules represent computation performed outside of the discrete grammar, e.g., multiplying a matrix or sending a Controller Area Network message, and the user must fill in the stub appropriately. This approach also decouples the parser generation tool from the physical hardware, which improves flexibility. The generated parser thus runs independently of the Motion Grammar Kit, permitting efficient, real-time performance.

The LL(*) algorithm extends LL(1) and LL(k), which perform fixed lookahead, by permitting arbitrarily long lookahead based on a finite automaton [10]. Because LL(*) was initially developed for program translation, we introduce some modifications to apply it to online control. The critical difference between program translation and online control is time. In program translation, tokens come from a static file available a priori. In robot control, tokens come from sensor readings in real-time. While a compiler need only give its output at the end of the file or statement, a robot must continually respond to its environment. We describe this constraint as LL(1) Semantics [4], meaning that a robot must be able to compute the immediate input without looking ahead to future tokens or backtracking on previous input commands.

We statically ensure that the synthesized LL(*) parser is Semantically LL(1). Whenever we reach a point in the parser where there is a semantic rule available to execute, we ensure that there is only a single possible semantic rule so that the parser can decide the proper action to take. While this is a restriction on the parser, it also provides an advantage. The prediction step of LL(*) normally does not permit semantic rule execution because these rules are not rewindable [1]. However, this is not a problem for a Semantically LL(1) grammar, where any potentially executed rules following some prefix will be identical. Thus, we can execute actions during the LL(*) prediction, allowing code generation for more grammars than otherwise possible.

$$
\begin{array}{rcl}
\langle \text{SERVE} \rangle & \rightarrow & [\neg \text{CANEMPTY}]\,\{PICK\}\,\langle \text{POUR} \rangle \langle \text{S}_1 \rangle \\
& | & [\neg \text{CANEMPTY}]\,\{PICK\}\,\langle \text{POUR} \rangle \langle \text{S}_2 \rangle \\
& | & [\text{CANEMPTY}]\,\{TRYNEWCAN\}\,[\neg \text{CANEMPTY}]\,\langle \text{SERVE} \rangle \\
& | & [\text{CANEMPTY}]\,\{TRYNEWCAN\}\,[\text{CANEMPTY}] \\
\langle \text{S}_1 \rangle & \rightarrow & [\text{GLASSFULL}]\,\{PLACE\}\,\{GIVEGLASS\}\,\langle \text{SERVE} \rangle\,\{\text{CHARGE}\} \\
\langle \text{S}_2 \rangle & \rightarrow & [\neg \text{GLASSFULL} \wedge \text{CANEMPTY}]\,\{PLACE\}\,\langle \text{SERVE} \rangle \\
\langle \text{POUR} \rangle & \rightarrow & [\neg \text{GLASSFULL} \wedge \neg \text{CANEMPTY}]\,\{POUR\}\,\langle \text{POUR} \rangle \\
& | & \varepsilon
\end{array}
$$

Fig. 1: An example of a grammar

The robot should start serving. When serving, the robot should pour cans of soda into glasses as long as there are nonempty cans remaining. When pouring, the robot should keep pouring until either the glass is full or the can is empty. If the glass becomes full, we should serve the glass and then charge for the glass once we're done serving. Further, whenever pouring is completed, the can must be placed down again.

Fig. 2: An declarative description of how to serve drinks

III. Demonstration

We propose to demonstrate a software verification and synthesis example for a robot waiting tables. In this scenario, customers order drinks and the robot serves a predefined liquid in glasses from a stash of cans the robot is carrying around. In similar human interactions, a waiter would come and serve each patron and then collect the bill from each patron, thus the robot should serve all patrons and collect the bills at the end. This requires memory. A finite automata cannot model this task for arbitrary number of customers. However, memory in the form of a context-free stack compactly represents the task. Figure 2 describes the task in natural language, and figure 1 describes the task as a context-free grammar. A parse tree for a context-free production is shown in figure 3. Lastly, figure 4 shows the actual syntax used by the tool to represent that production.

Though the actions in figure 1, i.e., $\{pick\}$, $\{place\}$, are high-level, these can also be hierarchically decomposed within the same grammar down to a discrete-time control loop to track a trajectory [4].

Additionally, we will automatically verify the following property, given as a regular expression, that the robot pours no more drinks after charging for the order (ensuring safety of the cash-register).

$$S = (\neg \{charge\})^* \, (\{charge\} \, (\neg \{pour\})^*)? \qquad (1)$$

Finally, we note that the grammar in figure 2 cannot be parsed by a LL(1) parser. The biggest obstacle is the lookahead required to choose between the first two productions, as the $\langle \text{POUR} \rangle$ nonterminal can be infinitely long, meaning that no fixed-lookahead LL(k) parser can parse it either.

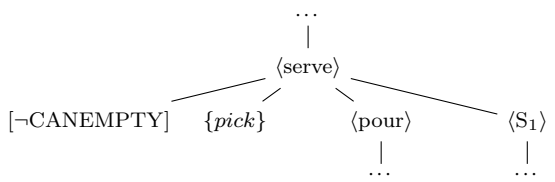

Fig. 3: The first production from the grammar in figure 1 represented as part of a parse tree

```
(SERVE (pred (not can-empty)) (mu pick) POUR S1)
```

Fig. 4: The production in figure 3 written in the syntax recognized by the motion grammar kit software

References

[1] A. Aho, M. Lam, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, & Tools*. Pearson, 2nd edition, 2007.

[2] R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. *Hybrid Systems*, pages 209–229, 1993.

[3] C.G. Cassandras and Stéphane Lafortune. *Introduction to Discrete-Event Systems*. Springer, 2nd edition, 2008.

[4] N. Dantam and M. Stilman. The motion grammar: Analysis of a linguistic method for robot control (accepted). *Trans. on Robotics*, 2013.

[5] N. Dantam, C. Nieto-Granda, H. Christensen, and M. Stilman. Linguistic composition of semantic mapping and hybrid control. In *ISER*, 2012.

[6] N. Dantam, A. Hereid, and M. Ames, A. Stilman. Correct software synthesis for stable speed-controlled robotic walking (accepted). In *RSS*. IEEE, 2013.

[7] Georgios Fainekos, Sriram Sankaranarayanan, Koichi Ueda, and Hakan Yazarel. Verification of automotive control applications using s-taliro. In *Proceedings of the American Control Conference*, 2012.

[8] T.A. Henzinger. The theory of hybrid automata. In *Symposium on Logic in Computer Science*, pages 278–292. IEEE, 1996.

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

[10] T. Parr and K. Fisher. LL (*): the foundation of the antlr parser generator. In *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*, pages 425–436. ACM, 2011.

[11] 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.

[12] S. Sarid, B. Xu, and H. Kress-Gazit. Guaranteeing high-level behaviors while exploring partially known maps. In *RSS*. IEEE, 2012.