Programs Synthesis Roadmap

Leandro E. Nahabedian

1 Introduction

Programs synthesis Program synthesis is the systematic derivation of a program from a given specification. For this reason, over the years, different formalisms to represent the specifications were presented. Each of them shows a large variety of strengths and weaknesses, making the area of programs synthesis a controversial one. Probably, the most common way to write specifications is by using logics expressing, in a declarative manner, the goals of the system to be synthesised. For instance, in [16, 1, 2], the authors used temporal logic to defined the specifications. Building software automatically showed many advantages in contrast to building them manually, attracting researchers to produce new program synthesis techniques until now.

The main strength of synthesis techniques are the absence of human mistakes that a programmer introduce to a code while working. Resulting software generated by synthesis are correct in terms of the requirements as it is built avoiding violations on the given requirements. The *correct by construction* philosophy is used in both controllers synthesis problems, as in [4, 14, 6] and also, in code synthesis problems, as in [19, 18].

On the other hand, producing concurrent programs are a challenging task for a programmer because interleaving executions must be taken into account. Many advantages of making programs by synthesis in concurrent domains, are presented in [8, 1, 2] where in [8] the authors proposed to map high-level requirements as a supervisory control specification. However, in [1, 2], temporal logics are used.

Controllers synthesis This roadmap is strongly connected with techniques that builds automatically behaviour strategies. For example, we can identify different areas similar to programs synthesis, like model-driven development in [7], planning in [17, 3, 20], supervisory control in [9] and controller synthesis in [13]

Controller synthesis area started with the paper of Büchi, Landweber, and Rabin in [5] and its evolution to modern techniques [15]. The general problem of synthesis is computationally expensive, but recently works tries to tackle it by defying sub-logics which allows handling more restrictive specifications with polynomial complexity (i.e. [13, 10]). Originally, research community has been focused in embedded systems and digital circuits, leading synthesis to other domains such as communications through shared memory and concurrency. However, in areas like requirements engineering, architectural design, and concurrency, a communication through messages model are more appropriate, and more recently, synthesis of behavioural models based in events had been explored. [6]

2 Background

The beginning of the programs synthesis has its origin 30 years ago, with [12, 16, 11].

At the beginning, [16], use a special Logic to define the requirements, called Linear Temporal Logic (LTL).

3 Discussion

In spite of the benefits of the synthesis technique introduced before, we have to pay an expensive price for this.

3.1 Controllers Synthesis vs Programs Synthesis

References

[1] P. C. Attie and E. A. Emerson. Synthesis of concurrent systems with many similar processes. *ACM Transactions on Programming Languages and Systems*, 20:51–115, 1998.

[2] P. C. Attie and E. A. Emerson. Synthesis of concurrent programs for an atomic read/write model of computation. *ACM Transactions on Programming Languages and Systems*, 23:187–242, 2001.

[3] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: a model based planner. In *Proceedings of the IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information*, 2001.

[4] A. Bertolino, P. Inverardi, P. Pelliccione, and M. Tivoli. Automatic synthesis of behavior protocols for composable web-services. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC/FSE '09, pages 141–150, New York, NY, USA, 2009. ACM.

[5] J. Buchi and L. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, pages 295–311, 1969.

[6] N. R. D'Ippolito, V. Braberman, N. Piterman, and S. Uchitel. Synthesis of live behaviour models. In *Proceedings of the Int. Symp. on Foundations of Soft. Eng.*, FSE '10, pages 77–86. ACM, 2010.

[7] H. Giese and W. Schaefer. Model-driven development of safe self-optimizing mechatronic systems with mechatronicUML. Technical Report tr-ri-12-322, Heinz Nixdorf Institute, University of Paderborn, Apr. 2012.

[8] M. V. Iordache and P. J. Antsaklis. Synthesis of Concurrent Programs Based on Supervisory Control.

[9] S. Jiang and R. Kumar. Supervisory control of discrete event systems with ctl* temporal logic specifications. In *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, volume 5, pages 4122–4127, 2001.

[10] O. Kupferman and M. Y. Vardi. Safraless decision procedures. In *Proceedings of the IEEE Symp. on Found. of Computer Science*, FOCS '05, pages 531–542, Washington, DC, USA, 2005. IEEE CS.

[11] Z. Manna and R. Waldinger. A deductive approach to program synthesis. *ACM Trans. Program. Lang. Syst.*, 2(1):90–121, Jan. 1980.

[12] Z. Manna and R. J. Waldinger. Toward automatic program synthesis. *Commun. ACM*, 14(3):151–165, Mar. 1971.

[13] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive (1) designs. *Lecture notes in computer science*, 3855:364–380, 2006.

[14] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '89, pages 179–190, New York, NY, USA, 1989. ACM.

[15] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the Symp. on Principles of Programming Languages*, pages 179–190, New York, NY, USA, 1989. ACM.

[16] K. Ramamritham. Synthesizing code for resource controllers. *IEEE Transactions on Software Engineering*, 11(8):774–783, 1985.

[17] M. Schoppers. Universal plans for reactive robots in unpredictable environments. In *IJCAI*, volume 87, pages 1039–1046. Citeseer, 1987.

[18] A. Solar-lezama, L. Tancau, R. BodÃk, S. A. Seshia, and V. A. Saraswat. Combinatorial sketching for finite programs. In *Architectural Support for Programming Languages and Operating Systems*, pages 404–415, 2006.

[19] S. Srivastava, S. Gulwani, and J. S. Foster. From program verification to program synthesis. In *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '10, pages 313–326, New York, NY, USA, 2010. ACM.

[20] D. Sykes, W. Heaven, J. Magee, and J. Kramer. From Goals to Components: A Combined Approach to Self-Management. In *Proc. of the Workshop on Soft. Eng. for Adaptive and Self-Managing Systems*, 2008.