PhD thesis

Domain specific software development in natural language

1 Context

As a number domains and industries go through a digital transformation, one can observe a constant creation of demand for programmers. While these industries face a shortage of available software developers, the programming tasks are very specific : they require specific domain knowledge and only a modest level of programmings skills. Even this is an important phenomenon, and a basic programming skill would be desirable for the majority of professions of the 21st century, the public education curricula do not address this problem sufficiently. Certain industries face a shortage of available software developers and this problem is likely to increase.

A number tools [1] -often based of artificial intelligence- are available to address this problem and enable people with no or little programming skills to become productive developers. Recently, a number of AI-based tools -ChatGPT, Copilot, CodeClippy [2], etc. - emerged that enable to generate code in different programming languages, out of natural language. These tools could largely improve the productivity of software developers, but to make use of these tools, one still needs competences in programming languages and an understanding of the generated code.

This thesis aims to develop methodologies and tools that can enable or support domain specialists to engage in activities that result executable software. Specifically, we envisage that they not only describe their programming tasks in natural language but they test, and debug their software in natural language, without interacting with the code itself. We would like to develop tools and methodologies to realise this vision in two different use cases.

2 Objectives

We would like to develop a methodology to develop domain specific applications in natural language. The methodology should include the following aspects :

— Program synthesis : Generating code out of natural language descriptions to a specific target environment

— Guiding the developer in the writing phase : We would like to develop methods to guide the developer to improve the provided textual description of the task if the provided text description is not sufficiently precise, to generate a code.

1. https://cacm.acm.org/news/263950-no-code-ai-platforms-and-tools/fulltext
2. https://github.com/CodedotAl/gpt-code-clippy/wiki

— Guiding the developer in the testing/debugging phase : We will develop methodologies to correct the generated program, without specific coding skills. In particular if the the developers discover some unexpected behaviour of the executed code, they should be able to modify their description. For this they also need guidance how to change the original text. Potentially, they interact with a visual representation of the code rather than the original text, but they should be able to change the code to correct the behaviour of their software.

We plan to develop methodologies and tools for two use cases : autonomous vehicles simulation software testing. In both of the scenarios the goal is to develop simple software with low complexity that requires only basic programming skills, but specific domain knowledge.

2.1 Autonomous vehicles test scenarios

In this use case, we will focus on the use case of autonomous vehicles, where one needs to develop test scenarios for the driving licence of the autonomous vehicles. These scenarios are described in a well-defined, standard language the OpenScenario [3] and OpenRoad [4]. These scenarios can be executed using a scenario execution software, that generates a visual presentation of the defined scenario.

2.2 Software testing

In this use case we would like to develop methods and tools to support software testing. The goal is to obtain executable test scripts out of natural language descriptions of test scenarios. If the resulting test script does not correspond to the intended scenario, we user should have guidance and suggestions how to modify the text input describing the task to get the desired results.

3 Research questions

Program synthesis [8] is a research domain that aims to develop methods that can synthesize executable code out of high level descriptions and domain specific languages (DSLs). Researchers have proposed a variety of methods, including the use of satisfiability or SMT solvers, reasoners, and also evolutionary computing. The most recent and advanced methods are based on the technique of neurosymbolic programming [4]. These techniques enable to combine symbolic methods to assure that the hard (and soft) constrains that correct synthesized software are satisfied, with (neural network-based) machine learning. Some important contributions in this area include [2], [5], [6], [13], [14], [16], [1] , [10]. Some of the neurosymbolic programming systems are available as open source projets, including Dreamcoder (Ellis et al. [7]).

Our planned work will use neurosymbolic techniques. While these methods enable to realise powerful tools, they do not address several points that are very important in our context :

3. `https://www.asam.net/standards/detail/openscenario/`
4. `https://github.com/The-OpenROAD-Project`

— Interaction. We would like that the user can interactively influence the generation process. While some papers propose interactive synthesis, such as [18], they assume that the developer understands the synthesised code, while we would like that the interaction is based on natural language. Phrases in natural language could have to much ambiguity to define programming tasks. When we would like to guide the programmer we might need to rely on a different representation. This could be for example a description of the scenario in a controlled language [12], or other representation that is easy to understand. We would like to avoid that the developer has to ready the code itself.

— Guiding the expert in the programming phase can require a number of methods, including the identification of ambiguous parts of the programs. Other techniques could involve proposing auto-completion techniques. Auto-completion techniques are widely used in different areas such as in information retrieval [3], in (graph) databases [17]. We propose specific auto-completion mechanisms for this form of software development. In this context, auto-completion should take into account the specific constraints of the domain. In our work we would like to enable developers define certain domain knowledge in form of constraints. We would like exploit these constraints to generate the auto-completion options. Methods for generating auto-completion suggestions -in the presence of constraints- might include probabilistic reasoning [15] or machine learning based techniques. Examples for the use of these techniques in other domains include [9] or [11].

Advisors Zoltan Miklos, Annie Foret, Olivier Ridoux (`first.last@irisa.fr`). Contact : `zoltan.miklos@irisa.fr`

Références

[1] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, and C. Sutton. Program synthesis with large language models, 2021.

[2] R. Bunel, M. J. Hausknecht, J. Devlin, R. Singh, and P. Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[3] F. Cai and M. de Rijke. *A Survey of Query Auto Completion in Information Retrieval*. Now Publishers Inc., Hanover, MA, USA, 2016.

[4] S. Chaudhuri, K. Ellis, O. Polozov, R. Singh, A. Solar-Lezama, and Y. Yue. Neurosymbolic programming. *Foundations and Trends® in Programming Languages*, 7(3) :158–243, 2021.

[5] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[6] M. D. Cranmer, A. Sanchez-Gonzalez, P. W. Battaglia, R. Xu, K. Cranmer, D. N. Spergel, and S. Ho. Discovering symbolic models from deep learning with inductive biases. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33 : Annual Conference on*

Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.

[7] K. Ellis, C. Wong, M. I. Nye, M. Sablé-Meyer, L. Morales, L. B. Hewitt, L. Cary, A. Solar-Lezama, and J. B. Tenenbaum. Dreamcoder : bootstrapping inductive program synthesis with wake-sleep library learning. In S. N. Freund and E. Yahav, editors, *PLDI '21 : 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, pages 835–850. ACM, 2021.

[8] S. Gulwani, O. Polozov, and R. Singh. Program synthesis. *Foundations and Trends® in Programming Languages*, 4(1-2) :1–119, 2017.

[9] N. Q. V. Hung, M. Weidlich, N. T. Tam, Z. Miklós, K. Aberer, A. Gal, and B. Stantic. Handling probabilistic integrity constraints in pay-as-you-go reconciliation of data models. *Information Systems*, 83 :166 – 180, 2019. `http://www.sciencedirect.com/science/article/pii/S030643791830320X`.

[10] N. Jain, S. Vaidyanath, A. Iyer, N. Natarajan, S. Parthasarathy, S. Rajamani, and R. Sharma. Jigsaw : Large language models meet program synthesis. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 1219–1231, New York, NY, USA, 2022. Association for Computing Machinery.

[11] K. Kikuchi, E. Simo-Serra, M. Otani, and K. Yamaguchi. Constrained graphic layout generation via latent optimization. In *Proceedings of the 29th ACM International Conference on Multimedia*, MM '21, page 88–96, New York, NY, USA, 2021. Association for Computing Machinery.

[12] T. Kuhn. A Survey and Classification of Controlled Natural Languages. *Computational Linguistics*, 40(1) :121–170, 03 2014.

[13] A. Murali, A. Sehgal, P. Krogmeier, and P. Madhusudan. Composing neural learning and symbolic reasoning with an application to visual discrimination. In L. D. Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 3358–3365. ijcai.org, 2022.

[14] E. Parisotto, A. Mohamed, R. Singh, L. Li, D. Zhou, and P. Kohli. Neuro-symbolic program synthesis. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[15] J. Pearl. *Probabilistic reasoning in intelligent systems : networks of plausible inference*. Morgan Kaufmann, San Francisco, Calif., 2009. Example for Explaning away.

[16] R. Shin, M. Allamanis, M. Brockschmidt, and O. Polozov. *Program Synthesis and Semantic Parsing with Learned Code Idioms*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[17] P. Yi, B. Choi, S. S. Bhowmick, and J. Xu. Autog : A visual query autocompletion framework for graph databases. *Proc. VLDB Endow.*, 9(13) :1505–1508, sep 2016.

[18] T. Zhang, L. Lowmanstone, X. Wang, and E. L. Glassman. Interactive program synthesis by augmented examples. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20, page 627–648, New York, NY, USA, 2020. Association for Computing Machinery.