




# Hoogle $\star$ : Constants and $\lambda$ -abstractions in Petri-net-based Synthesis using Symbolic Execution (Artifact)

Henrique Botelho Guerra  

INESC-ID and IST, University of Lisbon, Portugal

João F. Ferreira   

INESC-ID and IST, University of Lisbon, Portugal

João Costa Seco   

NOVA LINCS, NOVA School of Science and Technology, Caparica, Portugal

## Abstract

Type-directed component-based program synthesis is the task of automatically building a function with applications of available components and whose type matches a given goal type. Existing approaches to component-based synthesis, based on classical proof search, cannot deal with large sets of components. Recently, HOOGLE+, a component-based synthesizer for Haskell, overcomes this issue by reducing the search problem to a Petri-net reachability problem. However, HOOGLE+ cannot synthesize constants nor  $\lambda$ -abstractions, which limits the problems that it can solve.

We present HOOGLE $\star$ , an extension to HOOGLE+ that brings constants and  $\lambda$ -abstractions to the search space, in two independent steps. First, we introduce the notion of *wildcard* component, a component that matches all types. This enables the al-

gorithm to produce *incomplete functions*, i.e., functions containing occurrences of the wildcard component. Second, we complete those functions, by replacing each occurrence with constants or custom-defined  $\lambda$ -abstractions. We have chosen to find constants by means of an inference algorithm: we present a new unification algorithm based on symbolic execution that uses the input-output examples supplied by the user to compute substitutions for the occurrences of the wildcard.

When compared to HOOGLE+, HOOGLE $\star$  can solve more kinds of problems, especially problems that require the generation of constants and  $\lambda$ -abstractions, without performance degradation.

The artifact contains the source code of HOOGLE $\star$ , as well as scripts to reproduce the evaluation done in the paper.

**2012 ACM Subject Classification** Software and its engineering  $\rightarrow$  Automatic programming; Theory of computation  $\rightarrow$  Automated reasoning

**Keywords and phrases** Type-directed, component-based, program synthesis, symbolic execution, unification, Haskell

**Digital Object Identifier** 10.4230/DARTS.9.2.20

**Funding** FCT UIDB/04516/2020, FCT UIDB/50021/2020, and ANI Lisboa-01-0247-Feder-045917

**Related Article** Henrique Botelho Guerra, João F. Ferreira, and João Costa Seco, “Hoogle $\star$ : Constants and  $\lambda$ -abstractions in Petri-net-based Synthesis using Symbolic Execution”, in 37th European Conference on Object-Oriented Programming (ECOOP 2023), LIPIcs, Vol. 263, pp. 4:1–4:28, 2023.  
<https://doi.org/10.4230/LIPIcs.ECOOP.2023.4>

**Related Conference** 37th European Conference on Object-Oriented Programming (ECOOP 2023), July 17–21, 2023, Seattle, Washington, United States

**Evaluation Policy** The artifact has been evaluated as described in the ECOOP 2023 Call for Artifacts and the ACM Artifact Review and Badging Policy.



© Henrique Botelho Guerra, João F. Ferreira, and João Costa Seco;  
licensed under Creative Commons License CC-BY 4.0  
Dagstuhl Artifacts Series, Vol. 9, Issue 2, Artifact No. 20, pp. 20:1–20:3  
Dagstuhl Artifacts Series  
Schloss Dagstuhl – Leibniz-Zentrum für Informatik,  
Dagstuhl Publishing, Germany



**1 Scope**

The artifact includes the source code of HOOGLE $\star$ , and allows the reproduction of the experiments done in the evaluation section of the paper, automatically generating tables 3 and 4 of the paper.

Specifically, it supports the answers to the two research questions in the paper:

**RQ1** Can HOOGLE $\star$  solve all the problems that HOOGLE+ solves, without performance degradation?

**RQ2** Can HOOGLE $\star$  solve more problems than HOOGLE+?

**Answer to RQ1** The addition of the wildcard component did not lead to performance degradations.

Instead, the removal of constants resulted in performance improvements. From the original HOOGLE+ benchmarks, there is a single benchmark that HOOGLE+ solves and HOOGLE $\star$  cannot solve within the timeout, but it solves two that HOOGLE+ does not solve.

**Answer to RQ2** HOOGLE $\star$  can solve many more new problems than HOOGLE+, especially when constants or  $\lambda$ -abstractions are required, which makes it able to solve new classes of problems.

We also found that in the cases that both synthesizers produce solutions, the solutions of HOOGLE $\star$  are simpler, since they use fewer components.

**2 Content**

The artifact package consists of a zip file with:

1. A docker image with a minimal installation on Ubuntu 22.04, GHC 8.8.4, stack 2.11.1, and:
  - the source code of HOOGLE $\star$ , HOOGLE+ and HOOGLE+ with examples;
  - HOOGLE $\star$ , HOOGLE+ and HOOGLE+ with examples pre-installed and ready to use;
  - a script that runs the experiments of the paper and generates tables 3 and 4 of the paper.
2. A README.md file containing:
  - a description of the files and directories present in the docker images;
  - instructions on how to run the evaluation, and run particular problems on each version;
  - a mapping of the algorithms in the paper to the functions in the source code.

**3 Getting the artifact**

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact is also available at: [https://github.com/sr-lab/hoogle\\_plus](https://github.com/sr-lab/hoogle_plus).

**4 Tested platforms**

The artifact only requires docker to run. Table 1 shows the platforms in which we successfully ran the evaluation script.

**Table 1** Tested platforms.

CPU	RAM	OS	Docker Server Version
AMD Ryzen 5 5600G	16 GB	Ubuntu 22.04	20.10.21
Intel Core i5-4200U	8 GB	Windows 7 Professional	19.03.12
AMD Ryzen 5 5500U	12 GB	Ubuntu 22.04	20.10.24

**5 License**

The artifact is available under license Creative Commons license.

**6 MD5 sum of the artifact**

5eaf7ec500bb0795aaf8925ce874ebdf

**7 Size of the artifact**

1.83 GiB

**A Documentation**

Table 2 shows the mapping between the algorithms described in the paper and the functions in the source code.

To change the way the occurrences of the wildcard component are replaced, simply redefine the function `runExampleChecks` (`/home/hoogle_plus_ext/src/HooglePlus/GHCChecker.hs`). Note that this function takes as argument the function synthesized by the Petri net (possibly with wildcards), as well as the query type and the set of input-output examples, and returns a list of functions in which the occurrences of the wildcard are replaced, because we may find different replacements for the same wildcard.

The file `README.md` contains more information, as explained in Section 2.

**Table 2** Mapping between algorithms of the paper and the source code. The paths are relative to the directory `/home/hoogle_plus_ext`.

Algorithm	Source code
Algorithm 1	function <code>eval</code> in <code>src/SymbolicMatch/Eval.hs</code>
Algorithm 2	functions <code>main</code> and <code>executeSearch</code> in <code>app/HooglePlus.hs</code>
Algorithm 3	functions <code>executeCheck</code> and <code>runExampleChecks</code> in <code>src/HooglePlus/GHCChecker.hs</code>
Algorithm 4	function <code>synthLamba</code> in <code>src/HooglePlus/GHCChecker.hs</code> and function <code>linearSynth</code> in <code>src/HooglePlus/LinearSynth.hs</code>
Algorithm 5	function <code>completeExpr</code> in <code>src/HooglePlus/LinearSynth.hs</code>
Algorithm 6	function <code>applyMatch</code> in <code>src/HooglePlus/LinearSynth.hs</code>
Addition of wildcard	function <code>generateEnv</code> in <code>src/Database/Environment.hs</code>
Unification algorithm	function <code>match</code> in <code>src/SymbolicMatch/Match.hs</code> ; each case of this function is related to an inference rule presented in the paper.
Conversion to Haskell notation	function <code>showExpr</code> in <code>src/SymbolicMatch/Expr.hs</code>