Upcycling Formal Specifications for Similar Implementations with Arís

Kuruvilla George Aiyankovil[1], Rosemary Monahan[1][0000-0003-3886-4675], Diarmuid P. O'Donoghue[1] [0000-0002-3680-4217]
[1] Maynooth University, Co. Kildare, Ireland
{Rosemary.Monahan; Diarmuid.ODonoghue}@mu.ie

Abstract. We describe the Arís system for creating new formal specifications for source code by transferring existing specifications to similar implementations. We show the *code graphs* underlying its operation, graph matching supports retrieval, and pattern completion enables transfer of specifications to new implementations. A theorem prover formally verifies the new specifications.

Keywords: formal specifications, source code, graph representation

1 Writing Formal Specifications

While formal verification of source code has become more popular in many real-world applications, successfully verifying the code requires three critical activities: writing formal specifications describing the task to be achieved; writing the source code for that task; and proving the correctness of the source code against this specification (Greengard, 2021). Many verification tools use the design-by-contract approach to annotate implementations with formal specifications and axioms so that they can generate the proof obligations required to verify that the implementation satisfies its specification (Dross et al., 2021). Specifications are written as a formal contract which defines the preconditions (*requires* clauses) that must hold, for the implementation to establish the postconditions (*ensures* clauses). An automated theorem prover verifies correctness of the implementation *wrt* its specification, requiring axioms to assist the prover written as *assertions, invariants,* and *variants* clauses.

Writing specifications and supporting axioms for the proofs require expert training and experience, contributing to poor uptake of verification by industry, unless required to meet safety standards (Huisman, Gurov, & Malkis, 2020). Our work eases the burden of these two activities by retrieving a similar verified implementation from an existing repository of verified source code and reusing it to creating formal specifications and proof support for a target implementation.

1.1 Related Work

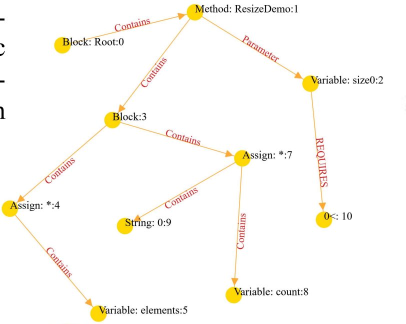
Our work differs from related work on code completion, automatic code generation *etc*. Some of its operation is more akin to code clone detection using conceptual graphs. An image below highlights one specification for a simple C# implementation, using CBR to transfer this specification to functionally similar code. We can describe Arís as: operating on executable source code, working at the statement level of granularity, performing static code analysis with extractive dependency graphs. Similar graphs are used to infer similar specifications.

2 Arís

In the Arís system (Pitu *et al*, 2013) source code is parsed and the resulting Abstract Syntax Tree is analysed to generate a code graph. Arís represents all problems and solution cases as distinct graphs, utilizing 18 categories of nodes and 6 types of relations. Nodes can contain information obtained directly from the source code, such as identifier names, the beginning of a block of code, assignment statements *etc*.

An important part of Arís concerns its representation of cases, focused on semantic graphs generated from examination of the Abstract Syntax Tree of a program, which is in turn generated using the ILSpy decompiler.

```
public ResizeDemo(int size0) {
  Contract.Requires(0 <= size0);
  this.elements = new int[size0];
  this.count = 0; }
```



Retrieval finds the largest common subgraph between a code graph containing specifications and one without. Graph Matching (ISMAGS, VF3) combines the influences of topological similarity with label categories for paired nodes and paired edges.
Reuse. Inter-graph mappings that include paired *variable* nodes are examined and compatible data-types identified, which involve identifying the original C# source code. The locations of the specifications are identified in the problem cases.
Revise. Transferrable specifications are identified and are updated to match their new problem context, including updating the variable names.
Retain. Source code with specifications is added to the project for compilation and verification by the Z3 theorem prover. Successfully verified methods can support subsequent inferences, potentially extending the reach of the initial specifications.

3 References

1. Pitu, M, Grijincu, D, Li, P, Saleem, A, Monahan, R, O'Donoghue, D.P. (2013) Arís: Analogical Reasoning for reuse of Implementation & Specification. *Proc. Artificial Intelligence for Formal Methods (AI4FM)*, 2013.
2. Greengard, S, Formal software verification measures up, *Communications of the ACM*, Volume 64, Issue 7, June 2021, https://doi.org/10.1145/3464933.
3. Dross et al (2021). VerifyThis 2019: a program verification competition. *International Journal on Software Tools for Technology Transfer*, 1-11.
4. Huisman, M., Gurov, D., & Malkis, A. (2020). Formal Methods: From Academia to Industrial Practice. A Travel Guide. *arXiv preprint arXiv:2002.07279*.

ICCBR CBR Demos & Showcases - Aris video at **https://youtu.be/gbbw_LOxoDs**