Integrating Resource Analyses via Resource Decomposition

Long Pham, Yue Niu, Nathan Glover, Feras Saad, Jan Hoffmann
Carnegie Mellon University

June 25, 2025

1 Introduction

What is included From Zenodo, you should obtain the following items:

- This `README.pdf` document; and
- Archived and compressed Docker image `resource-decomposition.tar.gz`.

Artifact overview *Hybrid resource analysis* integrates two resource-analysis techniques with complementary strengths and weaknesses. Given a program, the goal of resource analysis is to infer its worst-case symbolic bound on resource usage (e.g., running time and memory). Three existing families of resource-analysis techniques, namely static, data-driven, and interactive analyses, complement each other. For example, static resource analysis is sound but incomplete due to the undecidability of resource analysis in general. By contrast, data-driven resource analysis, which statistically infers a cost bound from a dataset of cost measurements, is complete but unsound. Additionally, compared to static and data-driven analyses, interactive resource analysis, where human intervention is required, is not fully automated but has higher expressiveness and more reasoning power. By integrating two analysis techniques, hybrid resource analysis retains their strengths while mitigating their respective weaknesses.

Resource decomposition is a new approach to hybrid resource analysis proposed in our paper. It works as follows. First, given an input program $P(x)$, the user annotates the source code to specify a quantity called *resource component* (e.g., the recursion depth of a helper function inside the program $P(x)$). The resulting annotated program $P_{\text{rd}}(x)$ is called a *resource-decomposed program*. The first resource analysis is performed on the resource-decomposed program $P_{\text{rd}}(x)$, inferring a symbolic bound $g(x)$ of the resource component. Next, the resource-decomposed program $P_{\text{rd}}(x)$ is extended with a numeric variable r, called a resource guard, which tracks the user-specified resource component. The resulting program $P_{\text{rd}}(x, r)$ is called a *resource-guarded program*. The second resource analysis is performed on the resource-guarded program to infer its overall cost bound $f(x, r)$. Finally, an overall cost bound of the original program $P(x)$ is obtained by substituting the resource-component bound $r \leq g(x)$ for the resource guard r in the bound $f(x, r)$, yielding $f(x, g(x))$.

Our artifact contains code for evaluating three concrete instantiations of the resource-decomposition technique (§4–6 in the paper). These instantiations each combine the following pairs of resource-analysis techniques:

Inst 1 Static analysis (AARA [2, 4]) and Bayesian data-driven analysis (§4);

Inst 2 Static analysis (AARA [2, 4]) and proof-assistant-based interactive resource analysis (Iris with time credits [1]) (§5); and

Inst 3 SMT-based semi-automatic analysis (TiML [5]) and Bayesian data-driven analysis (§6).

Inst 1 is evaluated on 13 benchmarks listed in Table 1 of the paper. Inst 2 is evaluated on Kruskal's algorithm for minimum spanning trees. Inst 3 is evaluated on a version of quicksort where integer comparison has logarithmic cost in the maximum size of integers (i.e., linear cost in the number of bits).

Out of the six analyses (i.e., two resource analyses employed in each instantiation), the artifact performs five of them: all except the interactive resource analysis in the second instantiation (§5). The inference result of the interactive analysis is available in Charguéraud and Pottier [1] and hence is not included in our artifact.

Supported claims The artifact supports the following claims in the paper:

- Inst 1: Percentages of sound bounds and analysis time, which are listed in the last four columns of Table 1.
- Inst 1: Plots of posterior distributions of the 13 benchmarks in the first instantiation (Figure 3,4, 7–20).
- Inst 2: Cost bound inferred by static analysis AARA [2, 4] for the resource-guarded program of Kruskal's algorithm. The bound is displayed in Eq. (5.3) in the paper.
- Inst 2: Plots of inferred symbolic bounds (Figure 5).
- Inst 3: Cost bound inferred by TiML [5] for quicksort, which is displayed in Eq. (6.1) in the paper.
- Inst 3: Plots of inferred symbolic bounds (Figure 6).

2 Hardware Dependencies

The artifact has no particular hardware dependencies.

3 Getting Started Guide

Docker is required to run the artifact. First, install Docker Engine on your machine as instructed in `https://docs.docker.com/engine/install/`. To check if Docker has been installed properly, run

```
$ docker --version
Docker version 28.1.1, build 4eba377
```

Load the Docker image `resource-decomposition.tar.gz` by running

```
$ docker load --input resource-decomposition.tar.gz
```

It creates an image named `resource-decomposition` and stores it locally on your machine. Docker may create an image with a slightly different name from `resource-decomposition`. To check the name of the image, display all Docker images on your local machine by running

```
$ docker images
```

To run the image `resource-decomposition`, run

```
$ docker run --name resource-decomposition -it --rm resource-decomposition
root@25bfacdb517e:/home/ocaml-benchmarks#
```

It creates a Docker container (i.e., a runnable instance of the Docker image), which has the same name `resource-decomposition` as the Docker image. The command also starts a shell inside the container. If the command does not run properly, you can instead build the image locally on your machine as instructed in §4.

Throughout this document, any command line starting with # is executed inside the Docker container, and any command line starting with $ is executed in your local machine's terminal.

3.1 Demonstration of Resource Decomposition's First Instantiation

This section demonstrates the first instantiation of the resource-decomposition technique (Inst 1), which integrates static resource analysis and Bayesian data-driven resource analysis. The static part runs RaML [3], which is an implementation of static resource analysis AARA [2, 4]. RaML automatically infer polynomial cost bounds of input OCaml programs. Meanwhile, the data-driven part runs Bayesian inference to statistically infer a symbolic bound of a resource component using a dataset of the resource component's measurements.

We consider the benchmark MergeSort implemented in OCaml. The resource metric of interest is the number of function calls (including all recursive calls and helper functions). Our goal is to infer an asymptotically tight $O(n \log n)$ (with concrete coefficients) cost bound of MergeSort.

AARA Inside the Docker container, the current working directory is

```
# pwd
/home/ocaml-benchmarks
```

Let $P(x)$ denote an original program of MergeSort. To view the source code of the original program $P(x)$, run

```
# cat lib/merge_sort/merge_sort.ml
```

This OCaml code is annotated with the construct `Raml.tick 1.0` to specify resource usage: it increments a cost counter by 1.0. This construct is inserted at the start of each function in the source code.

RaML can infer a polynomial bound for MergeSort. To invoke RaML on MergeSort, run a python script:

```
# python3 raml/run_raml.py benchmark merge_sort standard
```

It prints out the inference result:

```
== merge_sort :

  int list -> int list

  Non-zero annotations of the argument:
         7  <--  [::(*); ::(*)]
         1  <--  [::(*)]
         1  <--  []

  Non-zero annotations of result:

  Simplified bound:
     1 - 2.5*M + 3.5*M^2
   where
     M is the number of ::-nodes of the argument

--
  Mode:          upper
  Metric:        ticks
  Degree:        2
  Run time:      0.13 seconds
  #Constraints:  777

====
```

In the middle of this displayed inference result, we can find a quadratic cost bound of MergeSort inferred by RaML:

$$1 - 2.5m + 3.5m^2, \tag{3.1}$$

where m is the length of the input list. This quadratic bound is sound (i.e., it is a valid worst-case cost bound for all input sizes). But it is not an asymptotically tight $O(n \log n)$ bound of MergeSort.

Resource decomposition To obtain an asymptotically tighter cost bound of

4 Reusability Guide

References

[1] A. Charguéraud and F. Pottier. Verifying the Correctness and Amortized Complexity of a Union-Find Implementation in Separation Logic with Time Credits. *Journal of Automated Reasoning*, 62(3):331–365, Mar. 2019. ISSN 1573-0670. doi: 10.1007/s10817-017-9431-7. 2

[2] J. Hoffmann. *Types with Potential: Polynomial Resource Bounds via Automatic Amortized Analysis*. PhD thesis, Ludwig-Maximilians-Universität München, Oct. 2011. URL https://edoc.ub.uni-muenchen.de/13955/. 2, 3

[3] J. Hoffmann, K. Aehlig, and M. Hofmann. Resource aware ML. In P. Madhusudan and S. A. Seshia, editors, *Proc. 24 International Conference on Computer Aided Verification*, volume 7358 of *Lecture Notes in Computer Science*, pages 781–786, Berlin, 2012. Springer. doi: 10.1007/978-3-642-31424-7_64. 3

[4] J. Hoffmann, A. Das, and S.-C. Weng. Towards automatic resource bound analysis for OCaml. In *Proc. 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, pages 359–373, New York, NY, USA, Jan. 2017. ACM. doi: 10.1145/3009837.3009842. 2, 3

[5] P. Wang, D. Wang, and A. Chlipala. Timl: a functional language for practical complexity analysis with invariants. *Proc. ACM Program. Lang.*, 1(OOPSLA), oct 2017. doi: 10.1145/3133903. URL https://doi.org/10.1145/3133903. 2