# Lifted Static Analysis of Dynamic Program Families by Abstract Interpretation (Artifact)

**Aleksandar S. Dimovski**
Mother Teresa University, Skopje, North Macedonia

**Sven Apel**
Saarland University, Saarland Informatics Campus, 66123 Saarbrücken, Germany

## Abstract

In this article, we describe the usage and evaluation results of the tool DSPLNum$^2$Analyzer introduced by the paper "Lifted Static Analysis of Dynamic Program Families by Abstract Interpretation". We provide step-by-step instructions on how to download, install, run, and compare the tool's outputs to outputs described in the paper. DSPLNum$^2$Analyzer is a research prototype lifted static analyzer based on abstract interpretation designed for performing numerical static analysis of dynamic C program families.

**2012 ACM Subject Classification** Software and its engineering → Software functional properties; Software and its engineering → Software creation and management; Theory of computation → Logic

**Keywords and phrases** Dynamic program families, Static analysis, Abstract interpretation, Decision tree lifted domain

**Digital Object Identifier** 10.4230/DARTS.7.2.6

**Related Article** Aleksandar S. Dimovski and Sven Apel, "Lifted Static Analysis of Dynamic Program Families by Abstract Interpretation", in 35th European Conference on Object-Oriented Programming (ECOOP 2021), LIPIcs, Vol. 194, pp. 14:1–14:28, 2021.
https://doi.org/10.4230/LIPIcs.ECOOP.2021.14

**Related Conference** 35th European Conference on Object-Oriented Programming (ECOOP 2021), July 12–16, 2021, Aarhus, Denmark (Virtual Conference)

## 1 Scope

In this work, we present a tool, called DSPLNum$^2$Analyzer, for lifted static analysis (á-la abstract interpretation) of dynamic program families in C [1]. Our proof-of-concept implementation is written in OCaml and consists of around 8K lines of code. The tool uses the lifted domain of decision trees $\mathbb{T}(\mathbb{C}_{\mathbb{D}}, \mathbb{D})$, in which numerical domains $\mathbb{D}$ (e.g., intervals, octagons, and polyhedra) from the APRON library [4] are used as parameters. Definitions of abstract operations and transfer functions of the decision tree lifted domain can be found in [1, 3]. We compare precision and time performances of our decision tree-based lifted analysis with the single-program analysis, where feature variables are considered as ordinary program variables and the resulting single program is analyzed using off-the-shelf numerical domains from the APRON library [4]. This artifact confirms that our lifted analysis provides an acceptable precision/cost tradeoff [1]: we obtain invariants with a higher degree of precision within a reasonable amount of time than when using single-program analysis.

## 2 Content

The artifact package includes:

- ecoop27.ova is a Virtual Machine image containing the tool already installed. Username: ecoop27, Password: ecoop27. Enter 'DSPLNUM2Analyzer' subfolder of the 'home' folder and follow instructions for using the tool.

- DSPLNUM2Analyzer.tar.gz contains the tool and instructions how to install and use it. To install it using a single script see README-Script.txt. To install it using step-by-step written commands see README-StepByStep.txt

## 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 [2]: `https://zenodo.org/record/4718697#.YJrDzagzbIU`.

## 4 Tested platforms

All experiments are executed on a 64-bit Intel®Core$^{TM}$ i7-8700 CPU@3.20GHz × 12, Ubuntu 18.04.5 LTS, with 8 GB memory. We report times measured via Sys.time function of OCaml needed only for the actual static analysis task to be performed.

## 5 License

The artifact is available under license "CC-BY";
`http://creativecommons.org/licenses/by/3.0/`.

## 6 MD5 sum of the artifact

9b7967b7be95ddf100a645830eea1b2b

## 7 Size of the artifact

1.66 GiB

## A Performance results

All experiments are executed on a 64-bit Intel®Core$^{TM}$ i7-8700 CPU@3.20GHz × 12, Ubuntu 18.04.5 LTS, with 8 GB memory. All times are reported as average over five independent executions. We compare performances (time and precision) of our decision-tree based lifted analysis with the single-program analysis. All benchmarks from the paper are in "tests" and "spl-tests" subfolders of "DSPLNUM2Analyzer" folder. We report times measured via Sys.time function of OCaml needed only for the actual static analysis task to be performed. Possible validity answers to assertions are: "unreachable", "correct", "erroneous", "i don't know", and "mixed". For detailed description of their meaning and the obtained invariants for each benchmark, we refer to the paper [1].

### A.1 Warming-up benchmarks

#### Motivating Example.

dFAMILY example given in Fig.2 (on pp. 4), "Motivating Example" section:
`$ ./Main.native -single -domain boxes tests/dfamily-single.c` | [0.001], "i don't know"=2
`$ ./Main.native -single -domain polyhedra tests/dfamily-single.c` | [0.004], "i don't know"=2

**Table 1** Performance results for single analysis $\mathcal{A}(\mathbb{D})$ vs. lifted analysis $\overline{\mathcal{A}}_{\mathbb{T}}(\mathbb{D})$ and $\overline{\mathcal{A}}_{\mathbb{T}}(O)$ on selected benchmarks from SV-COMP. All times are in seconds.

| Benchmark | folder | $\lvert\mathbb{F}\rvert$ | LOC | $\mathcal{A}(P)$ | | $\overline{\mathcal{A}}_{\mathbb{T}}(O)$ | | $\overline{\mathcal{A}}_{\mathbb{T}}(P)$ | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | TIME | ANS. | TIME | ANS. | TIME | ANS. |
| `half_2.c` | `invgen` | 1 | 25 | 0.008 | × | 0.014 | ≃ | 0.017 | ✓ |
| `seq.c` | `invgen` | 2 | 30 | 0.015 | × | 0.084 | ✓ | 0.045 | ✓ |
| `sum01*.c` | `loops` | 1 | 15 | 0.008 | × | 0.009 | ✓ | 0.041 | ✓ |
| `count_up_d*.c` | `loops` | 1 | 15 | 0.002 | × | 0.008 | ≃ | 0.011 | ✓ |
| `hhk2008.c` | `lit` | 2 | 20 | 0.003 | × | 0.073 | ≃ | 0.032 | ✓ |
| `gsv2008.c` | `lit` | 1 | 20 | 0.002 | × | 0.007 | ✓ | 0.015 | ✓ |
| `Mysore.c` | `crafted` | 1 | 30 | 0.0008 | × | 0.002 | ✓ | 0.004 | ✓ |
| `Copenhagen.c` | `crafted` | 2 | 30 | 0.002 | × | 0.012 | ≃ | 0.021 | ✓ |

`$ ./Main.native -tree -domain boxes tests/dfamily-tree.c` | [0.007], "mixed"=2
`$ ./Main.native -tree -domain polyhedra tests/dfamily-tree.c` | [0.011], "mixed"=2

**Example 5 (on pp.13).**

`$ ./Main.native -single -domain polyhedra tests/example5-single.c` | [0.001], see invariants at locations [9:], [10:], [11:]
`$ ./Main.native -tree -domain polyhedra tests/example5-tree.c` | [0.003], see invariants at locations [9:], [10:], [11:] in Fig.5 on pp.14

**Example 8 (on pp.17).**

`$ ./Main.native -single -domain polyhedra tests/example8-single.c` | [0.003], "i don't know"=1, see invariant at location [17:]
`$ ./Main.native -tree -domain polyhedra tests/example8-tree.c` | [0.042], "mixed"=1, see invariant at location [17:] in Fig.8 on pp.18

## A.2 Benchmarks from "Other benchmarks" paragraph

We first present results shown in Table 2 on pp. 22 in the paper [1] (here reproduced in Table 1), since those benchmarks are smaller and less time consuming. For each row, we present results for 1st column ($\mathcal{A}(P)$ single-program analysis with polyhedra), 2nd column ($\overline{\mathcal{A}}_{\mathbb{T}}(O)$ decision-tree lifted analysis with octagons), and 3rd column ($\overline{\mathcal{A}}_{\mathbb{T}}(P)$ decision-tree lifted analysis with polyhedra). For more detailed description of the obtained invariants, see the paper [1] (pp. 21-23).

**`half_2.c`.**

`$ ./Main.native -single -domain polyhedra tests/half_2-single.c` | [0.008], "i don't know"=1 | 1st column of Table 1
`$ ./Main.native -tree -domain octagons tests/half_2-tree.c` | [0.014], "mixed"=1 | 2nd column of Table 1
`$ ./Main.native -tree -domain polyhedra tests/half_2-tree.c` | [0.017], "mixed"=1 | 3rd column of Table 1

`seq.c`.

`$ ./Main.native -single -domain polyhedra tests/seq-single.c` | [0.015], "i don't know"=1

`$ ./Main.native -tree -domain octagons tests/seq-tree.c` | [0.084], "mixed"=1

`$ ./Main.native -tree -domain polyhedra tests/seq-tree.c` | [0.045], "mixed"=1

`sum01*.c`.

`$ ./Main.native -single -domain polyhedra tests/sum01_bug02-single.c` | [0.008], "i don't know"=1

`$ ./Main.native -tree -domain octagons tests/sum01_bug02-tree.c` | [0.009], "mixed"=1

`$ ./Main.native -tree -domain polyhedra tests/sum01_bug02-tree.c` | [0.041], "mixed"=1

`$ ./Main.native -tree -domain boxes tests/sum01_bug02-tree.c` | optionally results for domain="boxes" | [0.009], "mixed"=1

`count_up_d*.c`.

`$ ./Main.native -single -domain polyhedra tests/count_up_down-single.c` | [0.002], "i don't know"=1

`$ ./Main.native -tree -domain octagons tests/count_up_down-tree.c` | [0.008], "mixed"=1

`$ ./Main.native -tree -domain polyhedra tests/count_up_down-tree.c` | [0.011], "mixed"=1

`hhk2008.c`.

`$ ./Main.native -single -domain polyhedra tests/hhk2008-single.c` | [0.003], "i don't know"=1

`$ ./Main.native -tree -domain octagons tests/hhk2008-tree.c` | [0.073], "mixed"=1

`$ ./Main.native -tree -domain polyhedra tests/hhk2008-tree.c` | [0.032], "mixed"=1

`gsv2008.c`.

`$ ./Main.native -single -domain polyhedra tests/gsv2008-single.c` | [0.002], "i don't know"=1

`$ ./Main.native -tree -domain octagons tests/gsv2008-tree.c` | [0.007], "mixed"=1

`$ ./Main.native -tree -domain polyhedra tests/gsv2008-tree.c` | [0.015], "mixed"=1

`Mysore.c`.

`$ ./Main.native -single -domain polyhedra tests/Mysore-single.c` | [0.0008], "i don't know"=1

`$ ./Main.native -tree -domain octagons tests/Mysore-tree.c` | [0.002], "mixed"=1

`$ ./Main.native -tree -domain polyhedra tests/Mysore-tree.c` | [0.004], "mixed"=1

**Table 2** Performance results for single analysis $\mathcal{A}(I)$ vs. lifted analysis $\overline{\mathcal{A}}_{\mathbb{T}}(I)$ with one and two features on selected e-mail variant simulators. All times are in seconds.

| Benchmark | LOC | $\mathcal{A}(I)$, 0 feature | | | $\overline{\mathcal{A}}_{\mathbb{T}}(I)$, 1 feature | | | $\overline{\mathcal{A}}_{\mathbb{T}}(I)$, 2 features | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Time | Unrea. | Rea. | Time | Unrea. | Mix | Time | Unrea. | Mix |
| e-mail_spec0 | 2645 | 16.2 | 80 | 48 | 29.3 | 80 | 48(1:1) | 50.7 | 80 | 48(3:1) |
| e-mail_spec6 | 2660 | 18.8 | 6 | 26 | 23.6 | 16 | 16(1:1) | 24.2 | 16 | 16(3:1) |
| e-mail_spec8 | 2665 | 14.6 | 12 | 20 | 19.1 | 12 | 20(1:1) | 27.7 | 12 | 20(2:2) |
| e-mail_spec11 | 2660 | 15.2 | 160 | 96 | 24.7 | 160 | 96(1:1) | 32.1 | 160 | 96(3:1) |
| e-mail_spec27 | 2630 | 14.5 | 384 | 128 | 28.4 | 384 | 128(1:1) | 38.4 | 384 | 128(3:1) |

**Copenhagen.c.**

`$ ./Main.native -single -domain polyhedra tests/Copenhagen-single.c` | [0.002], "i don't know"=1
`$ ./Main.native -tree -domain octagons tests/Copenhagen-tree.c` | [0.012], "mixed"=1
`$ ./Main.native -tree -domain polyhedra tests/Copenhagen-tree.c` | [0.021], "mixed"=1

## A.3 Benchmarks from "E-mail system" paragraph

We now present results shown in Table 1 on pp. 19 in the paper [1] (here reproduced in Table 2). Since the output is huge, we use "-minimal" option to print out only the analysis result regarding assertions. If you want to see the complete output, including the invariants in all program locations, just remove "-minimal" option. You can also send the output in a textual file for more easier inspection. For each row, we present results for 1st column ($\mathcal{A}(I)$ single-program analysis with intervals/boxes), 2nd column ($\overline{\mathcal{A}}_{\mathbb{T}}(I)$ decision-tree lifted analysis with boxes and 1 feature), and 3rd column ($\overline{\mathcal{A}}_{\mathbb{T}}(I)$ decision-tree lifted analysis with boxes and 2 features).

**e-mail_spec0.**

`$ ./Main.native -single -domain boxes -minimal spl-tests/email_spec0-single.c` | 1st column of Table 2
`$ ./Main.native -tree -domain boxes -minimal spl-tests/email_spec0-feat1.c` | 2nd column of Table 2
`$ ./Main.native -tree -domain boxes -minimal spl-tests/email_spec0-feat2.c` | 3rd column of Table 2

**e-mail_spec6.**

```
$ ./Main.native -single -domain boxes -minimal spl-tests/email_spec6-single.c
$ ./Main.native -tree -domain boxes -minimal spl-tests/email_spec6-feat1.c
$ ./Main.native -tree -domain boxes -minimal spl-tests/email_spec6-feat2.c
```

**e-mail_spec8.**

```
$ ./Main.native -single -domain boxes -minimal spl-tests/email_spec8-single.c
$ ./Main.native -tree -domain boxes -minimal spl-tests/email_spec8-feat1.c
$ ./Main.native -tree -domain boxes -minimal spl-tests/email_spec8-feat2.c
```

`e-mail_spec11.`

```
$ ./Main.native -single -domain boxes -minimal spl-tests/email_spec11-single.c
$ ./Main.native -tree -domain boxes -minimal spl-tests/email_spec11-feat1.c
$ ./Main.native -tree -domain boxes -minimal spl-tests/email_spec11-feat2.c
```

`e-mail_spec27.`

```
$ ./Main.native -single -domain boxes -minimal spl-tests/email_spec27-single.c
$ ./Main.native -tree -domain boxes -minimal spl-tests/email_spec27-feat1.c
$ ./Main.native -tree -domain boxes -minimal spl-tests/email_spec27-feat2.c
```

## References


1 Aleksandar S. Dimovski and Sven Apel. Lifted static analysis of dynamic program families by abstract interpretation. In *35th European Conference on Object-Oriented Programming, ECOOP 2021*, volume 194 of *LIPIcs*, pages 14:1–14:28. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2021. doi:10.4230/LIPIcs.ECOOP.2021.14.

2 Aleksandar S. Dimovski and Sven Apel. Tool artifact for "lifted static analysis of dynamic program families by abstract interpretation". *Zenodo*, 2021. doi:10.5281/zenodo.4718697.

3 Aleksandar S. Dimovski, Sven Apel, and Axel Legay. A decision tree lifted domain for analyzing program families with numerical features. In *Fundamental Approaches to Software Engineering - 24th International Conference, FASE 2021, Proceedings*, volume 12649 of *LNCS*, pages 67–86. Springer, 2021. doi:10.1007/978-3-030-71500-7_4.

4 Bertrand Jeannet and Antoine Miné. Apron: A library of numerical abstract domains for static analysis. In *Computer Aided Verification, 21st International Conference, CAV 2009. Proceedings*, volume 5643 of *LNCS*, pages 661–667. Springer, 2009. doi:10.1007/978-3-642-02658-4_52.