Faster Smarter Induction in Isabelle/HOL (Artifact Submission)

Yutaka Nagashima[1 [0000-0001-6693-5325]]

University of Innsbruck, Innsbruck, Austria
Yutaka.Nagashima@student.uibk.ac.at

Abstract. We present `semantic_induct`, an automatic tool to recommend how to apply proof by induction in Isabelle/HOL. Given an inductive problem, `semantic_induct` produces candidate arguments to the `induct` tactic and selects promising ones using heuristics. Our evaluation based on 1,095 inductive problems from 22 theory files shows that `semantic_induct` achieves a 90.0% increase of the coincidence rate for the most promising candidate within 5.0 seconds of timeout compared an existing tool, `smart_induct`, while achieving a 62.0% decrease of the median value of execution time. This abstract is part of the artifact submission of our paper submitted to TACAS2021. The artifact is available through EasyChair; however, it is publicly available at GitHub, as well.

1 What is Included in the Artifact Submission

Our artifact submission includes the following items:

- This PDF file.
- `License.txt`, which allows the Artifact Evaluation Committee to evaluate our artifact.
- `Readme.txt`, which has the instruction of how to replicate the results in plain text.
- `screenshot.png`, which shows how the formatted result should look like.
- Isabelle2020, which is the latest official release of Isabelle/HOL.
- PSL, which contains both `semantic_induct` and `smart_induct`.

Our artifact is available through EasyChair; however, it is publicly available at GitHub, as well.

Prerequisite: Unpack the Artifact.

To use our artifact submission, we have to unpack it first. The submitted ZIP file should contain two directories: Isabelle2020 and PSL. In the following we assume that these directories are stored in Desktop of the virtual machine provided by the Artifact Evalution Committiee as follows.

```
/home/tacas21/Desktop/Yutaka/Isabelle2020
/home/tacas21/Desktop/Yutaka/PSL
```

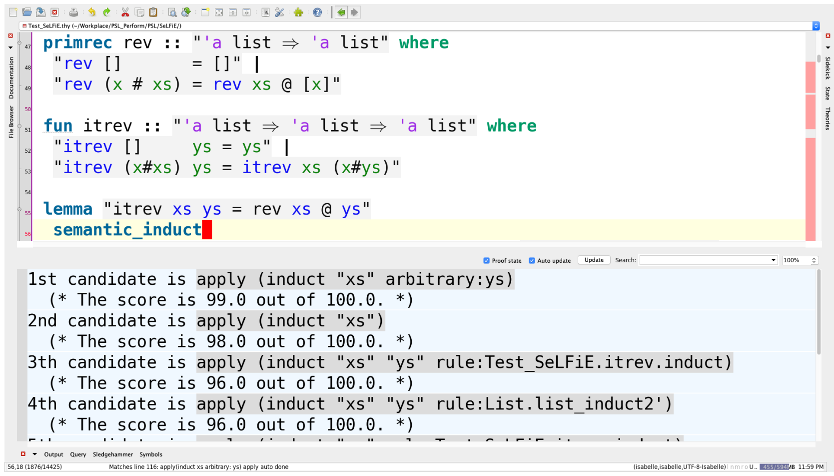

Fig. 1: The user-interface of `semantic_induct`.

2 How to See `semantic_induct` at Work

We can see `semantic_induct` at work for the running example presented in our paper in the interactive mode of Isabelle/HOL. The necessary command is the following.

```
/home/tacas21/Desktop/Yutaka/Isabelle2020/bin/isabelle jedit -d
/home/Desktop/Yutaka/PSL -l Smart_Isabelle /home/tacas21/Desktop/Yutaka/PSL/SeLFiE/Test_-
SeLFiE.thy
```

Then, the Output panel of Isabelle/jEdit should show the table presented in our paper as shown in Fig. 1.

3 How to Replicate the Evaluation Results

This Section explains how to replicate the evaluation results reported in our paper titled "Faster Smarter Induction in Isabelle/HOL" submitted to TACAS.

Important notice: by default VirtualBox assigns about 2 GB of memory and 1 processor to the virtual machine. This is not enough. The limited memory leads to not only poor performance but also the failure of the experiment. When we conducted our evlauation. We used a MacBook Pro (15-inch, 2019) with 2.6 GHz Intel Core i7 6-core memory 32 GB 2400 MHz DDR4.

This experiment consists of two phases:

- (Optional) Phase 1 produces the raw output files. These files are are named `Database.txt`.
- Phase 2 formats the raw output files, so that the results becomes easier for human engineers to interpret.

Phase 1 takes about 10-20 hours depending on the computational resources. Therefore, we also pre-built the results from Phase 1, so that the Artifact Evaluation Committiee (AEC) can skip Phase 1 and proceed to Phase 2 if they wish so.

Phase 1: Optional Construction of the Raw Results.

Step 1. We build the raw output file for `semantic_induct`, our tool presented in the paper. The evaluation suite for `semantic_induct` resides in `/home/tacas/Desktop/Yutaka/PSL/SeLFiE/Evaluation`.

The evaluation target theory files also reside in this directory. Therefore, we move our current directory to this directory by typing the following:

```
cd /home/tacas21/Desktop/PSL/SeLFiE/Evaluation
```

Then, we build the raw evaluation result, `Database.txt`, using the following command:

```
/home/tacas21/Desktop/Yutaka/Isabelle2020/bin/isabelle build -D
. -c -j1 -o threads=10
```

This command should use the ROOTS file stored in this directory to run Isabelle2020 stored in `/home/tacas21/Desktop/Yutaka/Isabelle2020`.

The results should appear in

```
/home/tacas21/Desktop/Yutaka/PSL/SeLFiE/Evaluation/Eval_Base/Database.txt
```

Step 2. We build the raw output file for `smart_induct` to compare the performance of `semantic_induct`. The evaluation suite for `smart_induct` resides in `/home/tacas21/Desktop/Yutaka/PSL/Semantic_Induct/Evaluation`.

The evaluation target theory files also reside in this directory. Therefore, we move our current directory to this directory by typing the following:

```
cd /home/tacas21/Desktop/Yutaka/PSL/Smart_Induct/Evaluation
```

Then, we build the raw evaluation result, `Database.txt`, using the following command:

```
/home/tacas21/Desktop/Yutaka/Isabelle2020/bin/isabelle build -D
. -c -j1 -o threads=10
```

This command should use the ROOTS file stored in this directory to run Isabelle2020 stored in `/home/tacas21/Desktop/Yutaka/Isabelle2020`.

The results should be appear in

```
/home/tacas21/Desktop/Yutaka/PSL/Smart_Induct/Evaluation/Eval_Base/Database.txt
```

This completes Phase 1.

Phase 2: Format the Raw Results.

Step 1. We copy the raw results from Phase 1 to the right locations with the right names, so that theory files for formatting the raw results can handle them. Note that we have already produced these files at the right locations with the right names, so that the AEC can skip Phase 1.

For `semantic_induct`,

```
cp /home/tacas21/Desktop/Yutaka/PSL/SeLFiE/Evaluation/Eval_Base/Database.txt
/home/tacas21/Desktop/Yutaka/PSL/SeLFiE/Evaluation/Format_Reulst/tacas2021_-
timeout5.csv
```

For `smart_induct`,

```
cp /home/tacas21/Desktop/Yutaka/PSL/Smart_Induct/Evaluation/Eval_-
Base/Database.txt /home/tacas21/Desktop/Yutaka/PSL/Smart_Induct/Evaluation/Format_-
Reulst/tacas2021_timeout5.csv
```

Now we open Isabelle/HOL in the interactive mode with `semantic_induct` and `smart_induct` using the ROOT file in `/home/tacas21/Desktop/Yutaka/PSL`. We can do so by typing the following command in one line.

```
/home/tacas21/Desktop/Isabelle2020/bin/isabelle jedit
-d /home/tacas21/Desktop/Yutaka/PSL/ -l Smart_Isabelle
/home/tacas21/Desktop/Yutaka/PSL/Smart_Induct/Evaluation/Format_-
Result/Format_Result_Smart_Induct.thy
```

`Format_Result_Smart_Induct.thy` imports `SeLFiE/Evaluation/Format_-Result/Format_Result_Semantic_Induct.thy`, which formats the raw file for `semantic_induct`.

And Line 298 of `Format_Result_Smart_Induct.thy` produces a table presented in our paper. We can observe this formatted result by moving the cursor of the virtual machine on top of Line 298, which states

```
val _ = map tracing both_results
```

Then, the Output panel of Isabelle/jEdit should show the table presented in our paper as shown in Fig. 2.

4 Evaluation dataset

We evaluated `semantic_induct` against `smart_induct` [11]. Our focus is to measure the accuracy of recommendations and execution time necessary to produce recommendations. All evaluations are conducted on a MacBook Pro (15-inch, 2019) with 2.6 GHz Intel Core i7 6-core memory 32 GB 2400 MHz DDR4.

Unfortunately, it is, in general, not possible to mechanically decide whether a given application of the `induct` tactic is right for a given problem. In particular, even if we can finish a proof search after applying the `induct` tactic, this does not

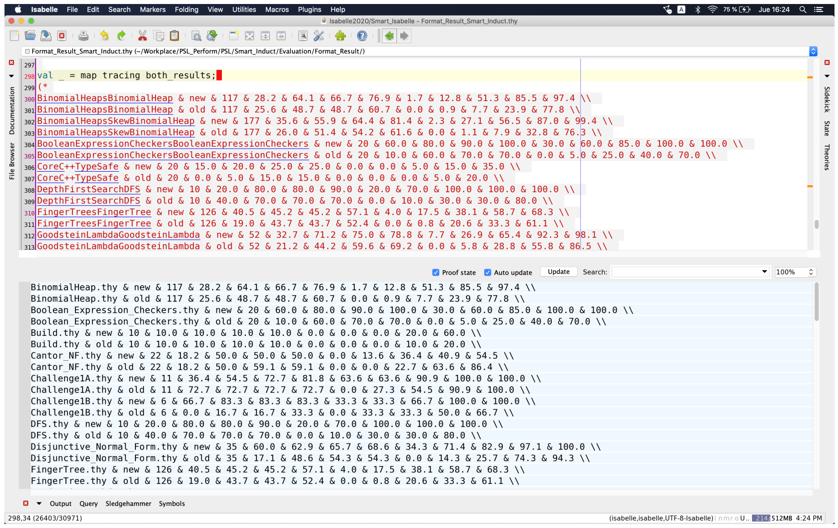

Fig. 2: The user-interface of `semantic_induct`.

guarantee that the arguments passed to the `induct` tactic are the right combination. For example, it is possible to prove our motivating example by applying `(induct ys)`; however, the necessary proof script following this application of the `induct` tactic becomes unnecessarily lengthy.

For this reason, we adopt *coincidence rates* as our indicator to approximate the accuracy of `semantic_induct`'s recommendations: we measure how often recommendations of `semantic_induct` *coincide* with the choice of human engineers. Since there are often multiple equally valid combinations of induction arguments for a given inductive problem, we should regard coincidence rates as conservative estimates of true success rates. For example, if `semantic_induct` recommends `(induct xs ys rule: itrev.induct)` this produces a negative data point that is not counted in when computing the corresponding coincidence rates since this is not the choice made by Nipkow *et al.*, even though `auto` can discharge all the sub-goals emerging from this `induct` tactic.

As our evaluation target, we use 22 Isabelle theory files with 1,095 applications of the `induct` tactic from the Archive of Formal Proofs (AFP) [6]. The AFP is an online repository of formal proofs in Isabelle/HOL. Each entry in the AFP is peer-reviewed by Isabelle experts prior to acceptance, which ensures the quality of our target theory files. Therefore, if `semantic_induct` achieves higher coincidence rates for our target theory files, it is safe to consider that `semantic_induct` tends to produce accurate recommendations. To the best of our knowledge, this is the most diverse dataset used to measure automation tools for proof by induction. For example, when Nagashima evaluated `smart_induct` they used 109 invocations of the `induct` tactic from 5 theory files, all of which are included in our dataset.

In the rest of the paper, we use the following abbreviations to represent the 22 target theory files.

- *BHeap* and *SHeap* represent `BinomialHeap.thy` and `SkewBinomialHeap.thy`, respectively [9].
- *Build*, *KDTree*, and *Nearest* stand for `Build.thy`, `KD_Tree.thy`, `Nearest_Neighbors.thy`, respectively, from the formalisation of multi-dimensional binary search trees [15].
- *Cantor* stands for `Cantor_NF.thy`, which is a part of a formalisation of ZFC set theory [14].
- *C1A* and *C1B* stand for `Challenge1A.thy` and `Challenge1B.thy`, respectively. They are parts of the solution for VerifyThis2019, a program verification competition associated with ETAPS2019. [8].
- *DFS* stands for `DFS.thy`, which is a formalisation of depth-first search [12].
- *DNF* stands for `Disjunctive_Normal_Form.thy`, which is a part of a formalisation of linear temporal logic [17].
- *Ftree* stands for `FingerTree.thy`, which implements 2-3 finger trees [13].
- *Goodstein* is for `Goodstein_Lambda.thy`, which is an implementation of the Goodstein function in lambda-calculus [2].
- *HL* refers to `Hybrid_Logic.thy`. which is a formalisation of a Seligman-style tableau system for Hybrid Logic [3].
- *Kripke* refers to `Kripke.thy`. which is a part of a general scheme for compiling knowledge-based programs to executable automata [4].
- *NBE* stands for `NBE.thy`, which formalises normalisation by evaluation as implemented in Isabelle [1].
- *OpSem* stands for `OpSem.thy`, which is a part of a formalisation of logical relations for PCF [5].
- *PST* stands for `PST_RBT.thy`, which is from a formalisation of priority search tree [7].
- *RFG* stands for `Rep_Fin_Groups.thy`, which is a formal framework for the theory of representations of finite groups [18].
- *SStep* stands for `SmallStep.thy`, which is a the theory of a sequential imperative programming language, Simpl [16].
- *TSafe* stands for `TypeSafe.thy`, which is a part of an operational semantics and type safety proof for multiple inheritance in C++ [19].
- *Graphs* stands for `Graphs.thy`, which is a part of a a formalization of probabilistic timed automata [20].

5 Coincidence Rates within 5.0 Seconds of Timeout

Table 1 shows the evaluation results of both `semantic_induct` and `smart_induct`. In each row of this table, the left most column shows the name of the target theory file. And the second column shows the tool used to measure coincidence rates: “new” stands for `semantic_induct`, while “old” stands for `smart_induct`. The third column shows how many invocations of the `induct` tactic appear in each theory file.

The columns in the middle of Table. 1 show the coincidence rates for each target theory file within 5 seconds of timeout. The numbers in the second row in the columns for coincidence rates show how many recommendations are considered to count coincidence rates.

For example, the coincidence rate of "new" for BHeap is 64.1 for 3. This means that the combination of induction arguments used by human researchers appear among the 3 most promising combinations recommended by `semantic_induct` for 64.1% of the uses of the `induct` tactic in BHeap. On the other hand, the coincidence rate of "old" for BHeap is 60.7 for 10. This means that even if we check for the 10 most promising candidates recommended by `smart_induct`, `smart_induct`'s recommendations coincide with the choice of human researchers only for 60.7% of the uses of the `induct` tactic in BHeap.

A careful observation reveals that the gaps between the coincidence rates for these tools are particularly large for Nearest, in which 81.8% of applications of the `induct` tactic involves generalisation. In fact, when Nagashima evaluated `smart_induct` in a similar setting but without a timeout they reported `smart_induct`'s low coincidence rates for induction involving generalisation [11] and concluded "recommendation of variable generalisation remains as a challenging task". Their tool, `smart_induct`, was based on `LiFtEr` [10], which is not expressive enough to encode generalisation heuristics that take the definitions of relevant constants into consideration.

6 Return Rates for 5 Timeouts

`semantic_induct` achieves higher coincidence rates than `smart_induct` does mainly because `semantic_induct` uses the `SeLFiE` interpreter to examine the definitions of constants relevant to the inductive problem at hand. Inevitably, this requires larger computational resources: the `SeLFiE` interpreter has to examine not only the syntax tree representing proof goals but also the syntax trees representing the definitions of relevant constants. However, thanks to the fast `SeLFiE` interpreter, and the smart construction of candidate inductions and pruning of less promising candidates, `semantic_induct` provides recommendations faster than `smart_induct` does.

This performance improvement is presented in the columns on the right-hand side of Table 1, which show how often `semantic_induct` and `smart_induct` return recommendations within certain timeouts specified in the second row.

For example, the return rate of "new" for BHeap is 85.5 for 2.0. This means that `semantic_induct` returns recommendations for 85.5% of proofs by induction in BHeap within 2.0 seconds. On the other hand, the return rate of "old" for BHeap is 77.8 for 5.0. This means that even if we give 5.0 seconds of timeout to `smart_induct`, `smart_induct` returns recommendations for only 77.8% of inductive problems in BHeap.

A quick look at Table 1 reveals that for all theory files `semantic_induct` produces more recommendations than `smart_induct` does for all specified timeouts (0.2 seconds, 0.5 seconds, 1.0 second, 2.0 seconds, and 5.0 seconds), proving the superiority of `semantic_induct` over `smart_induct` in terms of the execution time necessary to produce recommendations.

In fact, the median values of execution time for these 1,095 problems are 1.06 seconds for `semantic_induct` and 2.79 seconds for `smart_induct`. That is to say, `semantic_induct` achieved 62% of reduction in the median value of execution time.

Acknowledgement

This work was supported by the European Regional Development Fund under the project AI & Reasoning (reg.no. CZ.02.1.01/0.0/0.0/15_003/0000466) and by NII under NII-Internship Program 2019-2nd call.

References

1. Aehlig, K., Nipkow, T.: Normalization by evaluation. Archive of Formal Proofs (Feb 2008), http://isa-afp.org/entries/NormByEval.html, Formal proof development
2. Felgenhauer, B.: Implementing the goodstein function in lambda-calculus. Archive of Formal Proofs (Feb 2020), http://isa-afp.org/entries/Goodstein_Lambda.html, Formal proof development
3. From, A.H.: Formalizing a seligman-style tableau system for hybrid logic. Archive of Formal Proofs (Dec 2019), http://isa-afp.org/entries/Hybrid_Logic.html, Formal proof development
4. Gammie, P.: Knowledge-based programs. Archive of Formal Proofs (May 2011), http://isa-afp.org/entries/KBPs.html, Formal proof development
5. Gammie, P.: Logical relations for pcf. Archive of Formal Proofs (Jul 2012), http://isa-afp.org/entries/PCF.html, Formal proof development
6. Klein, G., Nipkow, T., Paulson, L., Thiemann, R.: The Archive of Formal Proofs (2004), https://www.isa-afp.org/
7. Lammich, P., Nipkow, T.: Priority search trees. Archive of Formal Proofs (Jun 2019), http://isa-afp.org/entries/Priority_Search_Trees.html, Formal proof development
8. Lammich, P., Wimmer, S.: Verifythis 2019 – polished isabelle solutions. Archive of Formal Proofs (Oct 2019), http://isa-afp.org/entries/VerifyThis2019.html, Formal proof development
9. Meis, R., Nielsen, F., Lammich, P.: Binomial heaps and skew binomial heaps. Archive of Formal Proofs (Oct 2010), http://isa-afp.org/entries/Binomial-Heaps.html, Formal proof development
10. Nagashima, Y.: Lifter: Language to encode induction heuristics for isabelle/hol. In: Programming Languages and Systems - 17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1-4, 2019, Proceedings. pp. 266–287 (2019). https://doi.org/10.1007/978-3-030-34175-6_14, https://doi.org/10.1007/978-3-030-34175-6_14
11. Nagashima, Y.: Smart induction for Isabelle/HOL (tool paper). In: Proceedings of the 20th Conference on Formal Methods in Computer-Aided Design – FMCAD 2020 (2020). https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_32, http://dx.doi.org/10.34727/2020/isbn.978-3-85448-042-6_32

Table 1: Coincidence rates and return rates within timeouts. Coincidence rates are based on 5.0 seconds of timeout. The unit of each rate is %.

			coincidence rates				return rates				
theory name	tool	goal	1	3	5	10	0.2	0.5	1.0	2.0	5.0
BHeap	new	117	28.2	64.1	66.7	76.9	1.7	12.8	51.3	85.5	97.4
	old	117	25.6	48.7	48.7	60.7	0.0	0.9	7.7	23.9	77.8
Boolean	new	20	60.0	80.0	90.0	100.0	30.0	60.0	85.0	100.0	100.0
	old	20	10.0	60.0	70.0	70.0	0.0	5.0	25.0	40.0	70.0
Build	new	10	10.0	10.0	10.0	10.0	0.0	0.0	0.0	20.0	60.0
	old	10	10.0	10.0	10.0	10.0	0.0	0.0	0.0	10.0	20.0
Cantor	new	22	18.2	50.0	50.0	50.0	0.0	13.6	36.4	40.9	54.5
	old	22	18.2	50.0	59.1	59.1	0.0	0.0	22.7	63.6	86.4
C1A	new	11	36.4	54.5	72.7	81.8	63.6	63.6	90.9	100.0	100.0
	old	11	72.7	72.7	72.7	72.7	0.0	27.3	54.5	90.9	100.0
C1B	new	6	66.7	83.3	83.3	83.3	33.3	33.3	66.7	100.0	100.0
	old	6	0.0	16.7	16.7	33.3	0.0	33.3	33.3	50.0	66.7
DFS	new	10	20.0	80.0	80.0	90.0	20.0	70.0	100.0	100.0	100.0
	old	10	40.0	70.0	70.0	70.0	0.0	10.0	30.0	30.0	80.0
DNF	new	35	60.0	62.9	65.7	68.6	34.3	71.4	82.9	97.1	100.0
	old	35	17.1	48.6	54.3	54.3	0.0	14.3	25.7	74.3	94.3
FTree	new	126	40.5	45.2	45.2	57.1	4.0	17.5	38.1	58.7	68.3
	old	126	19.0	43.7	43.7	52.4	0.0	0.8	20.6	33.3	61.1
Goodstein	new	52	32.7	71.2	75.0	78.8	7.7	26.9	65.4	92.3	98.1
	old	52	21.2	44.2	59.6	69.2	0.0	5.8	28.8	55.8	86.5
HL	new	89	47.2	58.4	62.9	65.2	13.5	28.1	44.9	64.0	79.8
	old	89	16.9	39.3	53.9	64.0	0.0	5.6	24.7	52.8	74.2
KDTree	new	9	77.8	77.8	100.0	100.0	11.1	33.3	88.9	100.0	100.0
	old	9	77.8	77.8	77.8	77.8	0.0	0.0	33.3	100.0	100.0
Kripke	new	13	53.8	69.2	69.2	76.9	0.0	15.4	38.5	53.8	100.0
	old	13	0.0	15.4	30.8	30.8	0.0	0.0	7.7	15.4	30.8
NBE	new	104	30.8	49.0	54.8	71.2	5.8	23.1	48.1	70.2	88.5
	old	104	15.4	38.5	46.2	56.7	0.0	3.8	21.2	41.3	70.2
Nearest	new	11	54.5	63.6	72.7	72.7	0.0	0.0	0.0	9.1	72.7
	old	11	0.0	0.0	0.0	9.1	0.0	0.0	0.0	0.0	9.1
OpSem	new	33	45.5	66.7	78.8	81.8	9.1	18.2	36.4	54.5	84.8
	old	33	12.1	30.3	42.4	45.5	0.0	9.1	15.2	21.2	45.5
PST	new	24	41.7	95.8	100.0	100.0	0.0	0.0	20.8	58.3	100.0
	old	24	45.8	45.8	45.8	45.8	0.0	0.0	4.2	16.7	45.8
RFG	new	99	41.4	58.6	67.7	68.7	5.1	17.2	29.3	47.5	76.8
	old	99	9.1	38.4	42.4	45.5	0.0	1.0	7.1	29.3	69.7
SHeap	new	177	35.6	55.9	64.4	81.4	2.3	27.1	56.5	87.0	99.4
	old	177	26.0	51.4	54.2	61.6	0.0	1.1	7.9	32.8	76.3
SStep	new	66	45.5	75.8	77.3	77.3	15.2	21.2	33.3	47.0	83.3
	old	66	21.2	37.9	47.0	50.0	0.0	1.5	19.7	48.5	63.6
TSafe	new	20	15.0	20.0	25.0	25.0	0.0	0.0	5.0	15.0	35.0
	old	20	0.0	5.0	15.0	15.0	0.0	0.0	0.0	5.0	20.0
Graphs	new	41	31.7	70.7	78.0	87.8	36.6	61.0	75.6	87.8	100.0
	old	41	19.5	41.5	51.2	61.0	0.0	12.2	41.5	56.1	87.8
overall	new	1095	38.2	59.3	64.5	72.7	8.8	24.7	47.8	69.8	86.8
	old	1095	20.1	42.8	48.5	55.3	0.0	3.5	16.9	38.3	70.2

12. Nishihara, T., Minamide, Y.: Depth first search. Archive of Formal Proofs (Jun 2004), http://isa-afp.org/entries/Depth-First-Search.html, Formal proof development
13. Nordhoff, B., Körner, S., Lammich, P.: Finger trees. Archive of Formal Proofs (Oct 2010), http://isa-afp.org/entries/Finger-Trees.html, Formal proof development
14. Paulson, L.C.: Zermelo fraenkel set theory in higher-order logic. Archive of Formal Proofs (Oct 2019), http://isa-afp.org/entries/ZFC_in_HOL.html, Formal proof development
15. Rau, M.: Multidimensional binary search trees. Archive of Formal Proofs (May 2019), http://isa-afp.org/entries/KD_Tree.html, Formal proof development
16. Schirmer, N.: A sequential imperative programming language syntax, semantics, hoare logics and verification environment. Archive of Formal Proofs (Feb 2008), http://isa-afp.org/entries/Simpl.html, Formal proof development
17. Sickert, S.: Linear temporal logic. Archive of Formal Proofs (Mar 2016), http://isa-afp.org/entries/LTL.html, Formal proof development
18. Sylvestre, J.: Representations of finite groups. Archive of Formal Proofs (Aug 2015), http://isa-afp.org/entries/Rep_Fin_Groups.html, Formal proof development
19. Wasserrab, D.: Corec++. Archive of Formal Proofs (May 2006), http://isa-afp.org/entries/CoreC++.html, Formal proof development
20. Wimmer, S., Hölzl, J.: Probabilistic timed automata. Archive of Formal Proofs (May 2018), http://isa-afp.org/entries/Probabilistic_Timed_Automata.html, Formal proof development