

# FM26 artifact

---

## Table of content

1. [Artifact structure](#)
2. [Docker instructions](#)
3. [Smoke tests](#)
4. [Introduction](#)
5. [Interaction language](#)
  - [Representation of interactions](#)
  - [Gates](#)
  - [Composition smoke test](#)
  - [Reduced benchmark smoke test](#)
6. [Composition Examples](#)
7. [Benchmark](#)
  - [Step 1: projection, normalization and mutation](#)
  - [Step 2: composition](#)
  - [Step 3: Normal Form Checking](#)
  - [Summary of the workflow for the interaction Game](#)
  - [To Execute all three steps in one pass](#)
  - [Interactions of the benchmark](#)

## Artifact structure

The [generalizer](#) folder contains the following subfolders:

```
generalizer
  LICENSE.txt
  README.pdf
  README.md
  Dockerfile
  Executable
  Benchmark
  smoke_tests
  Interactions_examples
  generalizer_sources.zip
  readme
  Benchmark_with_results.zip
  smoke_tests_with_results.zip
  Interaction_examples_with_results.zip
```

The Docker image includes a pre-built executable located in [generalizer/Executable](#).

Additional resources are organized as follows:

- **Smoke tests:** `generalizer/smoke_tests`
- **Benchmark scripts and files:** `generalizer/Benchmark`
- **Interaction examples** (including composition scripts):  
`generalizer/Interactions_examples`

The provided archives contain :

- `generalizer_sources.zip` — Source code of the program
- `Benchmark_with_results.zip` — Benchmark results
- `smoke_tests_with_results.zip` — Smoke test results
- `Interaction_examples_with_results.zip` — Interaction examples composition results

## Docker instructions

The artefact is wrapped in a docker image available on Zenodo(todo: link). After downloading the image, it is loaded with the following command:

```
$ docker load -i generalizer.tar.gz
```

Alternatively, the image can be built from the root of the repository `generalizer` with the following command:

```
$ docker build -t generalizer .
```

After loading or building the image, running the container is done with the following command:

```
$ docker run -it --rm --name custom_container generalizer:latest
```

To avoid conflicts, use a different container name for each run (in this example `custom_container`).

Our experiments generate images that cannot be easily visualized inside the docker image directly. We recommend copying output images to the host machine.

While the container is running (its name is `custom_container` in the example above), you can copy a file from the container to the host machine with the following command:

```
$ docker cp custom_container:/home/fm/generalizer/README.pdf  
target_folder_on_host
```

where `target_folder_on_host` is the folder where you want to copy the file in the host machine. The above command will copy the file `generalizer/README.pdf` from the container to the folder `target_folder_on_host`.

## Smoke tests

By running the container, Docker will open as shell inside a directory named `generalizer`. The smoke tests are located in the `generalizer/smoke_tests` directory. There are two smoke tests: a composition smoke test and a reduced benchmark smoke test.

### Composition smoke test

To check whether the composition of two interactions works, we check that with the example in the introduction of the paper. It is located in `generalizer/smoke_tests/composition_smoke_test`. The folder contains:

- `signature.hsf`: the signature file of the interactions containing the declaration of lifelines and messages.
- `i.hif`: the first interaction.
- `j.hif`: the second interaction.
- `composition_smoke_test.sh`: the script to run the composition of the interaction models `i` and `j`.

The `.hsf` and `.hif` can be visualized with the `cat` command.

```
$ cat signature.hsf  
$ cat i.hif  
$ cat j.hif
```

```
$ cd smoke_tests/composition_smoke_test  
$ ./composition_smoke_test.sh
```

The *Figure 1* illustrates the composition of the two interactions.

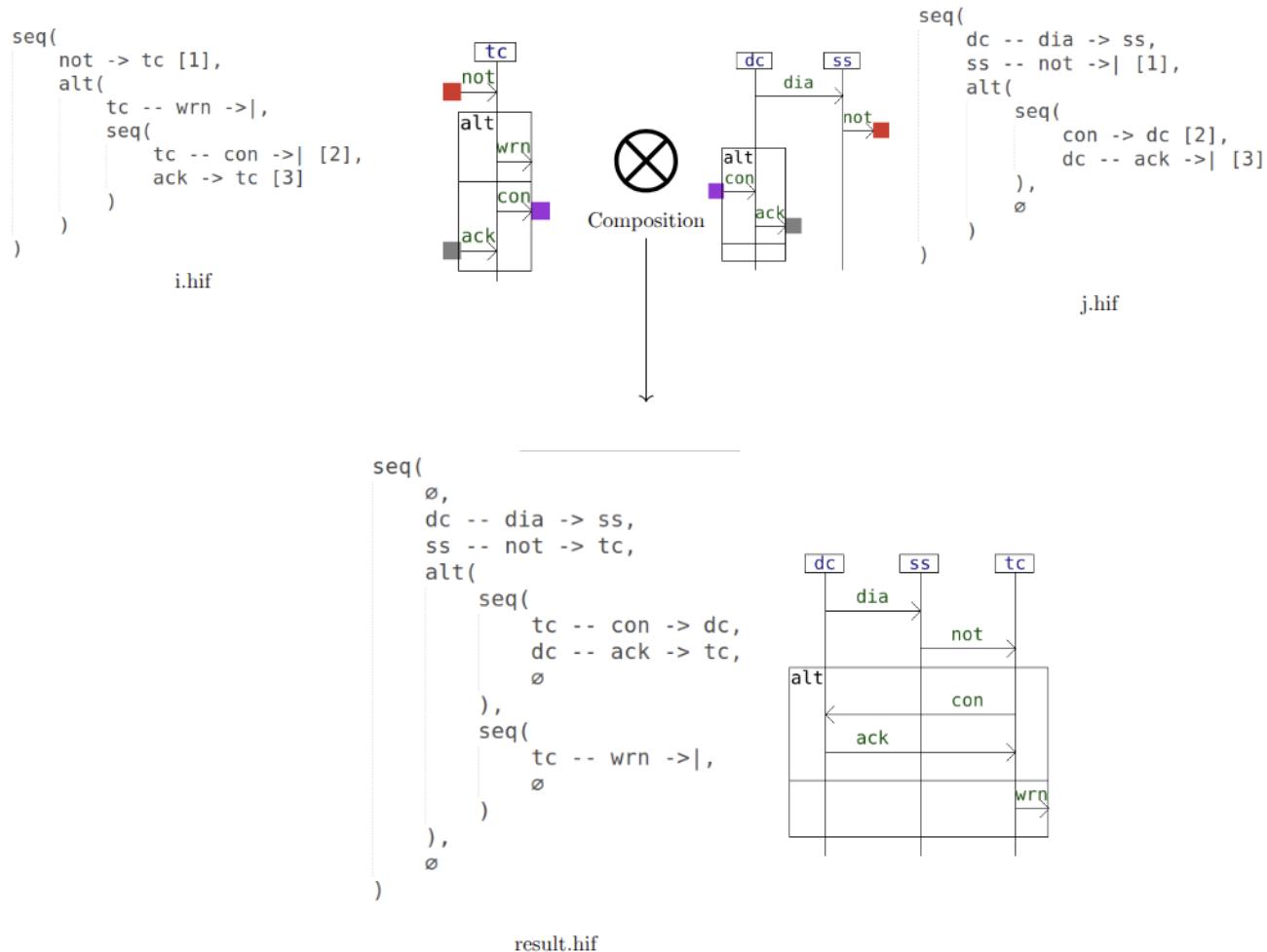


Figure 1: Illustration of the composition smoke test.

If successful, the success message will be printed in the terminal as shown in Figure 2.

```

-----
Composition smoke test with a simple example
This script uses the executable at the location: generalizer/target/release/generalizer
Starting composition: ven. 13 févr. 2026 12:05:35 CET
Results will be written to: ./Composition Output
-----
```

# GENERALIZER

Using Anti-unification modulo ACU

```

Composition successful
Duration: 0.000993334 s
-----
Execution time: 0 seconds
Composition smoke test: ven. 13 févr. 2026 12:05:35 CET
Check results inside: ./Composition Output

```

Figure 2: Success message printed in the terminal for the composition smoke test.

The command runs in less than 1 seconds. The result will be put in the folder `Composition_Output` which contains a folder `result` containing the files `result.hif`(interaction file) and `result.png`(visual representation of the result). The folder `input` also contains pictures `i.png` and `j.png` of the interactions. All the images can be visualized by copying them to the host machine according to the

instruction in [Docker instructions](#). The file `result.hif` can be visualized with the `cat` command to obtain the model represented in *Figure 1*.

```
$ cat Composition_Output/result/result.hif
```

## Reduced benchmark smoke test

To quickly check whether the benchmark runs successfully, we provide a reduced version of the benchmark. It is located in `generalizer/smoke_tests/reduced_benchmark_smoke_test`. The folder contains the script `reduced_benchmark_smoke_test.sh` to run the small benchmark.

```
$ cd smoke_tests/reduced_benchmark_smoke_test
$ ./reduced_benchmark_smoke_test.sh
```

If successful, a success message will be printed in the terminal as shown in *Figure 3*.

```
-----  
Reduced benchmark smoke test  
Starting reduced benchmark: ven. 13 févr. 2026 12:09:01 CET  
Results will be written to: ./Benchmark output  
-----  
GENERALIZER  
Max number of partitions 5  
Number of mutations 7  
Composition timeout 60s  
Using Anti-unification modulo ACU  
Composition of local interactions of Alt3bit completed for each partition of lifelines  
The duration of composition for each partitions are recorded in Benchmark_Output/Alt3bit/Alt3bit_composition_durations.csv  
-----  
Composition of local interactions of FilterCo completed for each partition of lifelines  
The duration of composition for each partitions are recorded in Benchmark_Output/FilterCo/FilterCo_composition_durations.csv  
-----  
Composition of local interactions of TPM completed for each partition of lifelines  
The duration of composition for each partitions are recorded in Benchmark_Output/TPM/TPM_composition_durations.csv  
-----  
Benchmark finished successfully  
-----  
Execution time: 3 seconds  
Reduced benchmark finished: ven. 13 févr. 2026 12:09:04 CET  
Check results inside: ./Benchmark output  
Check the file results.csv for the computation durations
```

*Figure 3: Success message printed in the terminal for the reduced benchmark smoke test.*

The execution takes approximatively 3 seconds. The result will be put in the folder `Benchmark_Output`. It contains a csv file `result_one_pass.csv` containing a table akin the experimental section of the paper.

To visualize the results inside the docker container, the following command can be used:

```
$ csvlook -d '&' Benchmark_Output/results_one_pass.csv | less -S
```

To shrink the size of columns, the following command can be used:

```
$ csvlook -d '&' --max-column-width 10
Benchmark_Output/results_one_pass.csv | less -S
```

The table of *Figure 4* should be printed (up to some small differences in numbers, which are durations):

Global ...	Size of...	Gates r...	(Normal...)	(Normal...)	(Mutate...)	(Mutate...)
Alt3bit	12	6	4.143(0k)	20.925(0k)	5.833(0k)	30.848(0k)
FilterCo	11	5	1.54(0k)	1.728(0k)	2.165(0k)	2.433(0k)
TPM	17	7	2.818(0k)	4.733(0k)	4.288(0k)	7.083(0k)

*Figure 4: Table of results for the reduced benchmark smoke test*

This smoke test executes in one pass the three steps of the benchmark described in details the Section [Benchmark](#) below.

## Introduction

This README file describes the artifact related to the paper ["Specializing anti-unification for interaction models composition via gate connections"] accepted to the [FM26](#) conference.

The paper proposes an approach to the composition of interaction models using anti-unification. The program, named [generalizer](#) is developed in Rust.

## Interaction language

### Representation of interactions

Our implementation of Interactions models is based on the work of [Mahe et al.](#) and the tool [HIBOU](#).

We follow the notation of HIBOU for signature files (.hsf) and interaction files (.hif).

Let us consider the signature (sig.hsf):

```
@message{
    bwin;cwin;close;blose;busy;msg;sig;free
}

@lifeline{
    10;l1;l12;l13
}
```

and the interaction (i.hif):

```
loopS(
    seq(
        par(
            alt(
```

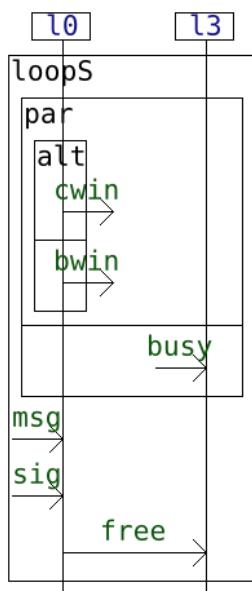
```

    10 -- cwin ->|,
    10 -- bwin ->|
),
busy -> 13
),
msg -> 10,
sig -> 10,
10 -- free -> 13
)
)
)

```

For instance `10 -- cwin ->|` is an emission of the message `cwin` from lifeline `10` to environment; and `busy -> 13` is reception of the message `busy` from environment to lifeline `13`. The term `10 -- free -> 13` represents the transmission of the message `free` from lifeline `10` to lifeline `13`, and is called a `value passing` in the paper.

The above interaction can be visualized as depicted in *Figure 5*.



*Figure 5: Sequence diagram of a local interaction*

## Gates

We introduce *gates* in our implementation to mark complementary communications for the composition as described in the paper.

Gates are assigned by adding number under brackets next to the relevant action.

For example, the previous interaction decorated with gates is:

```

loopS(
    seq(
        par(
            alt(
                10 -- cwin ->| [3],

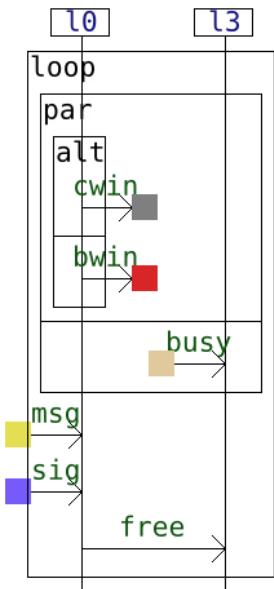
```

```

        10 -- bwin ->| [1]
),
busy -> 13 [5]
),
msg -> 10 [6],
sig -> 10 [7],
10 -- free -> 13
)
)
)

```

which can be visually represented as depicted in *Figure 6*.



*Figure 6: Sequence diagram of a local interaction with gates*

## Composition Examples

The folder `Interactions_examples` contains several examples of interactions composition described in the appendix of the paper, and the example of the introduction. Each folder contains a signature file `signature.hsf`, and interaction files `i.hif` and `j.hif`. In addition there is a script `example_run.sh` to run the composition of the two interactions, in exactly the same way as in the smoke test of the composition.

## Benchmark

To execute the benchmark, move into the `Benchmark` folder from the root of `generalizer` folder. The produced images may be visualized by copying them to the host machine according to the instruction in [Docker instructions](#).

```
$ cd Benchmark
```

The paper's experiments were run on an Intel Core i7-13850HX (20-core, 2.1 GHz) with 32 GB RAM. The benchmark is divided into three steps each performed by the scripts described in the following table,

Script	Description	est. time
benchmark_step_1_projection.sh	Projection,mutation,normalization	~21 seconds
benchmark_step_2_composition.sh	Composition of local interactions	~33 minutes
benchmark_step_3_nf_checking.sh	Normal form Checking	~1 seconds
benchmark_one_pass.sh	Run all three steps at once	~ 33 minutes

## Step 1: projection, normalization and mutation

We use the interactions in the folder **Benchmark** as our starting global models. For each global interaction **k**, we extract at most **N\_p** partitions of its set of lifelines **L** into a pair of subsets each of size at least **L/2**.

For each partition (**L1**, **L2**) of a set of lifelines of a global interaction **k**:

- we project **k** onto **L1** and **L2** to obtain local interactions **i1** and **i2**;
- we normalize **i1** and **i2** using HIBOU to obtain **i1\_norm** and **i2\_norm** respectively.
- we apply mutation operations to **i1** and **i2**, which consists of successively applying **N\_m** times one of the following rewrite operation selected uniformly at random: **alt(x,y) -> alt(y,x)** and **par(x,y) -> par(y,x)**. We obtain the interactions **i1\_mut** and **i2\_mut** from **i1** and **i2** respectively. The mutations are done with **Maude**. Those mutation operations are achieved by the scripts under the folder **maude\_mutation**.

To start the first step, we execute the following command:

```
$ ./benchmark_step_1_projection.sh
```

The program will create a folder **Benchmark\_Output** containing a folder for each starting global interaction.

In the case of the interaction **Game**, we have the following structure:

```

Game
  input_global_interaction
    Game.png
    Game.hif
    Game_tree.png
  Partition0
    original_locals
      i1.hif
      i1.png
      i1_tree.png
      i2.hif
      i2.png
      i2_tree.png
    with_mutated_locals
      mutated_local_interactions
        i1.hif

```

```

    i1.png
    i1_tree.png
    i2.hif
    i2.png
    i2_tree.png
    results_with_rule_fail
    results_without_rule_fail
    with_normalized_locals
    ...
    Partition1
    ...
    Partition2
    ...
    Partition3
    ...
    Partition4
    ...

```

The folder `original_locals` contains the local interactions `i1` and `i2` obtained after the projection of the global interaction.

The partition folders have the same structure. Each of them contains the folders `with_normalized_locals` and `with_mutated_locals` have the same structure. They contain the models `i1_norm`, `i2_norm` and `i1_mut`, `i2_mut` respectively (`.hif` files and `.png` pictures).

The folders `results_with_rule_fail` and `results_without_rule_fail` are empty at this stage, are meant to contain the results of the composition with and without the rule **Fail**, in the next step.

## Step 2: composition

This step consists of the composition of the pairs (`i1_norm`, `i2_norm`) and (`i1_mut`, `i2_mut`).

In the case of the interaction `Game`, this step will compose the interaction `i1.hif` and `i2.hif` in the folders of each of the folders

`partition{i}/with_normalized_locals/normalized_local_interactions` and `partition{i}/with_mutated_locals/mutated_local_interactions`, for each of the partitions `i`.

This step is performed by the script `benchmark_step_2_composition.sh`.

```
$ ./benchmark_step_2_composition.sh
```

```

Game
Game_composition_durations.csv
input_global_interaction
    Game.png
    Game.hif
    Game_tree.png
Partition0
    original_locals

```

```
i1.hif  
i1.png  
i1_tree.png  
i2.hif  
i2.png  
i2_tree.png  
with_mutated_locals  
    mutated_local_interactions  
        i1.hif  
        i1.png  
        i1_tree.png  
        i2.hif  
        i2.png  
        i2_tree.png  
    results_with_rule_fail  
        result.hif  
        result.png  
        result_tree.png  
        time.txt  
    results_without_rule_fail  
        result.hif  
        result.png  
        result_tree.png  
        time.txt  
with_normalized_locals  
...  
Partition1  
...  
Partition2  
...  
Partition3  
...  
Partition4  
...
```

The folders `results_with_rule_fail` and `results_without_rule_fail` contain the results of the composition with and without the rule **Fail**. The duration of the compositions are in the file `time.txt` in each folder.

This step produces a csv file `results_step_2.csv` in the folder `Benchmark_Output`.

You can visualize it with the following command:

```
$ csvlook -d '&' Benchmark_Output/results_step_2.csv | less -S
```

Or with column shrinked down:

```
$ csvlook -d '&' --max-column-width 10 Benchmark_Output/results_step_2.csv  
| less -S
```

We obtain the table of *Figure 7* (with shrunked down columns).

Global ...	Size of...	Gates r...	(Normalized locals) Av.Composition duration with Fail(ms)	(Normalized locals) Av.Composition duration without Fail(ms)	(Mutate...)	(Mutate...)
ATM	33	[7, 17]	13,872	timeout	25,621	timeout
Alt3bit	12	6	3,654	19.916	5,700	29.604
DistVoting	23	8	7,192	timeout	10,144	timeout
FilterCo	11	5	1,288	1.392	1,876	2.108
Game	16	[5, 6]	2,146	3.099	2,856	4.2
HealthSys	22	[5, 9]	4,493	18.363	6,057	47.957
Logistic	26	[6, 11]	11,358	timeout	15,577	timeout
ProfOnline	68	[23, 25]	57,644	timeout	75,822	timeout
Sanitary	30	[6, 13]	8,527	28.438	11,487	41.636
TPM	17	7	2,695	4.528	4,069	6.785
Travel	26	[6, 11]	6,892	timeout	9,119	timeout
TwoBuyers	22	[5, 11]	3,052	5.665	4,198	7.564

*Figure 7: Table of results for the step 2 of the benchmark benchmark.*

Each interaction corresponds to a row in the table. The second column reports the size of the interaction, while the third column indicates the range of gate counts in the local interactions obtained after projection onto the lifeline partitions. The last four columns present the average composition time across partitions, both with and without the optimization rule **Fail**. This rule is designed to compute compositions more efficiently. Specifically, the fourth and fifth columns show the average duration for composing normalized local interactions, whereas the last two columns report the average duration for mutated local interactions.

A truncated version of the table is shown in *Figure 8*.

Global interaction	Size of the global interaction	Gates range	(Normalized locals) Av.Composition duration with Fail(ms)	(Normalized locals) Av.Composition duration without Fail(ms)
ATM	33	[7, 17]	13,872	timeout
Alt3bit	12	6	3,654	19.916
DistVoting	23	8	7,192	timeout
FilterCo	11	5	1,288	1.392
Game	16	[5, 6]	2,146	3.099
HealthSys	22	[5, 9]	4,493	18.363
Logistic	26	[6, 11]	11,358	timeout
ProfOnline	68	[23, 25]	57,644	timeout
Sanitary	30	[6, 13]	8,527	28.438
TPM	17	7	2,695	4.528
Travel	26	[6, 11]	6,892	timeout
TwoBuyers	22	[5, 11]	3,052	5.665

*Figure 8: Truncated table of results for the step 2 of the benchmark benchmark.*

Consider the interaction **ATM**, highlighted in yellow. Its size is **33**. After projection, the number of gates in its local interactions ranges from **7** to **17**. With the optimization rule **Fail**, the average composition time across partitions for normalized local interactions is **13.872 ms**. Without the **Fail** rule, the composition process times out (after 60 seconds) for at least one partition.

Now consider the interaction **Game**, highlighted in red. Its size is **16**, and the projected local interactions contain either **5** or **6** gates. With the **Fail** optimization, the average composition time across partitions is **2.146 ms**. Without this optimization, the average duration across partitions increases to **3.099 ms**. These results illustrate that the **Fail** rule not only reduces the average composition time but can also prevent timeouts (set to 60 seconds).

In addition, in each folder corresponding to a global interaction, there is a **.csv** file showing the composition duration for each partitions non-averaged. For example, for the interaction **Game**, such a file is **Benchmark\_Output/Game/Game\_composition\_durations.csv**. It contains a table similar to the one in *Figure 9*.

Partition	(Normal...)	(Normal...)	(Mutate...)	(Mutate...)
Partiti...	2,355	3,503	3,174	5,237
Partiti...	1,977	3,377	3,328	5,154
Partiti...	2,070	3,412	4,291	4,458
Partiti...	2,180	2,104	2,844	4,404

Figure 9: Composition durations for each partitions for the interaction Game

### Step 3: Normal Form Checking

In this step, we check whether the normal form of the results of compositions in the previous step is the same as the normal form of the original interactions.

It is accomplished by applying the normal form checking algorithm of [HIBOU](#) to the interactions obtained in the previous step.

We execute the following command:

```
$ ./benchmark_step_3_nf_checking.sh
```

It produces a csv file `results_step_3.csv` in the folder `Benchmark_Output`. The new csv file is basically `results_step_2.csv` with a verdict (Ok) besides durations to confirm that the normal form of the result of each composition across partitions matches with the normal form of the original interaction before projections.

The final table should be similar to the one in the experiment section of the paper (up to some small differences in numbers, due to the randomness of the mutation operations and different execution environments).

An execution gives the table of *Figure 10*.

Global ...	Size of...	Gates r...	(Normal...)	(Normal...)	(Mutate...)	(Mutate...)
ATM	33	[7, 17]	13.872(0k)	timeout	25.621(0k)	timeout
Alt3bit	12	6	3.654(0k)	19.916(0k)	5.7(0k)	29.604(0k)
DistVoting	23	8	7.192(0k)	timeout	10.144(0k)	timeout
FilterCo	11	5	1.288(0k)	1.392(0k)	1.876(0k)	2.108(0k)
Game	16	[5, 6]	2.146(0k)	3.099(0k)	2.856(0k)	4.2(0k)
HealthSys	22	[5, 9]	4.493(0k)	18.363(0k)	6.057(0k)	47.957(0k)
Logistic	26	[6, 11]	11.358(0k)	timeout	15.577(0k)	timeout
ProfOnline	68	[23, 25]	57.644(0k)	timeout	75.822(0k)	timeout
Sanitary	30	[6, 13]	8.527(0k)	28.438(0k)	11.487(0k)	41.636(0k)
TPM	17	7	2.695(0k)	4.528(0k)	4.069(0k)	6.785(0k)
Travel	26	[6, 11]	6.892(0k)	timeout	9.119(0k)	timeout
TwoBuyers	22	[5, 11]	3.052(0k)	5.665(0k)	4.198(0k)	7.564(0k)

Figure 10: Table of results for the step 3 of the benchmark benchmark.

The table of results from the paper is shown in *Figure 11*.

Interaction $k$	size( $k$ )	Nb. Gates	Avg. composition Time (ms)			
			$(i_{\text{norm}}, j_{\text{norm}})$		$(i_{\text{mut}}, j_{\text{mut}})$	
			with F	without F	with F	without F
ATM	33	[7, 17]	15.016(✓)	timeout	17.757(✓)	timeout
Alt3bit	12	6	4.132(✓)	21.804(✓)	6.01(✓)	32.932(✓)
DistVoting	23	8	8.243(✓)	timeout	10.955(✓)	timeout
FilterCo	11	5	1.396(✓)	1.565(✓)	2.069(✓)	2.339(✓)
Game	16	[5, 6]	2.525(✓)	3.179(✓)	3.163(✓)	4.283(✓)
HealthSys	22	[4, 8]	5.978(✓)	18.496(✓)	7.495(✓)	22.786(✓)
Logistic	26	[6, 11]	9.22(✓)	timeout	12.457(✓)	timeout
ProfOnline	68	[14, 27]	59.567(✓)	timeout	75.474(✓)	timeout
Sanitary	30	[6, 13]	11.491(✓)	40.242(✓)	14.795(✓)	51.265(✓)
TPM	17	7	3.204(✓)	4.911(✓)	4.915(✓)	7.437(✓)
Travel	26	[6, 11]	6.411(✓)	timeout	8.203(✓)	timeout
TwoBuyers	22	[5, 11]	3.78(✓)	5.558(✓)	4.852(✓)	7.162(✓)

Figure 11: Table of results from the paper.

The **Ok** in the csv files are represented by green checkmarks in the table of the paper.

### Summary of the workflow for the interaction Game

The *Figure 12* illustrates the protocol of the benchmark with the Game global interaction, with only the mutation scenario.

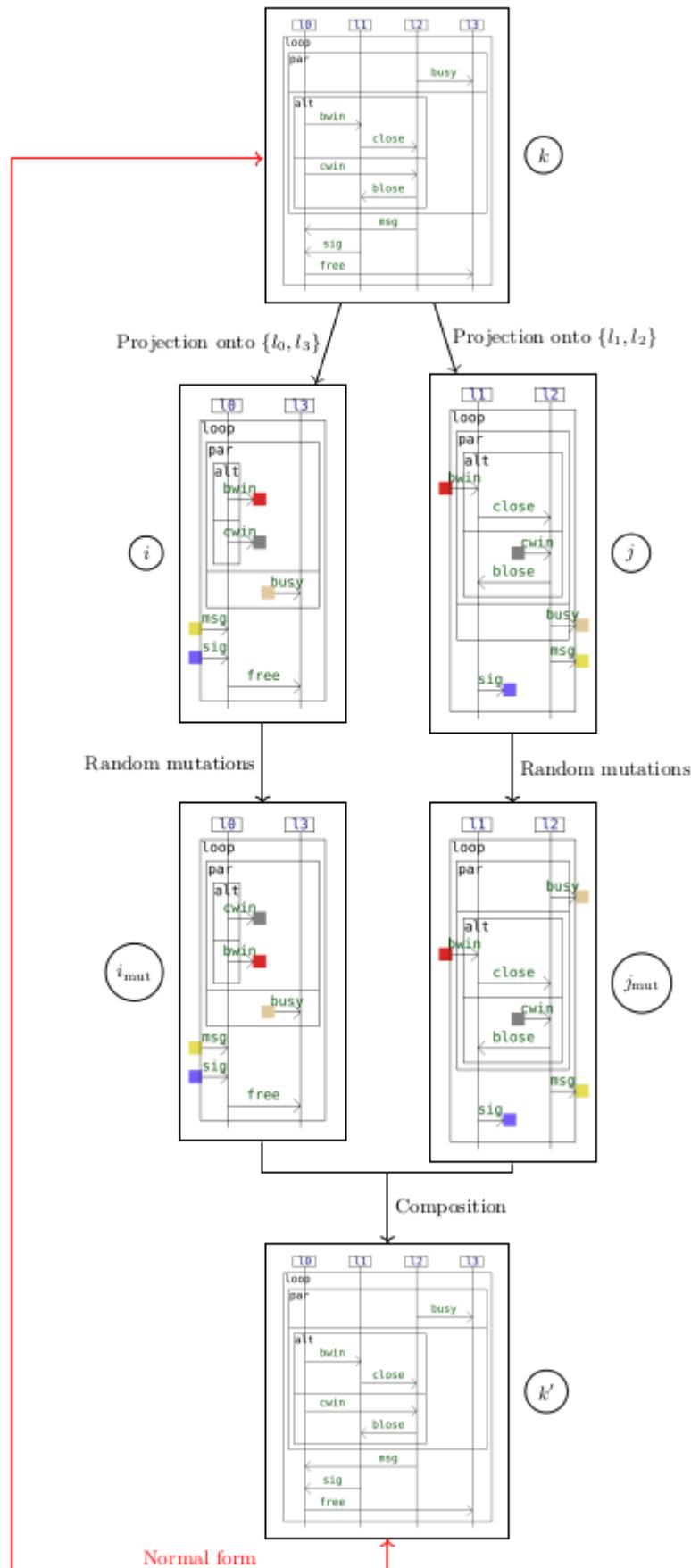


Figure 12: Protocol of the benchmark for the interaction Game.

To Execute all three steps in one pass

To execute all three steps in one pass, we can use the script `benchmark_one_pass.sh`.

```
$ ./benchmark_one_pass.sh
```

It directly produces a csv file `result_one_pass.csv` in the folder `Benchmark_Output` which is the same as the one produced at the end of step 3.

To take a closer look at the command running the benchmark in one pass, The subcommand to run the benchmark is `benchmark`. It takes as arguments:

- the name of the subfolder containing the interactions. In the downloadable folder, it is `Benchmark`.
- the number of mutation per partition
- the maximal number of random partitions extracted by global interaction.
- Timeout in seconds

We can add flags, `-m` to have the duration in milliseconds, `-d` to draw the models for visualization.

The command to execute to have the result in the table above is:

```
$ generaliser benchmark Benchmark 7 5 60 -m
```

It means:

For each global interaction, at most 5 partitions of its lifelines will be extracted; after projection onto the partitions, 7 random mutations are operated in the local interactions. The timeout threshold is of 60s. the flag `-m` means that in the output csv file, the duration will be given in milliseconds. The theory for the composition is ACU (all the rules are used).

To draw the interactions involved in the process, we can use the flag `-d`.

## Interactions of the benchmark

We present in the following table sequence diagram representation of the interactions of the benchmark, which files are in the folder `Benchmark`. Those interaction were adapted from examples and experiments from the literature.

Name	Interaction graphical representation	Reference
------	--------------------------------------	-----------

Name	Interaction graphical representation	Reference
Alternating3Bit Protocol	<pre> sequenceDiagram     participant L0     participant L1     L0-&gt;&gt;L1: m3     activate L1     L1-&gt;&gt;L0: a3     deactivate L1     L0-&gt;&gt;L1: m1     activate L1     L1-&gt;&gt;L0: a1     deactivate L1     L0-&gt;&gt;L1: m2     activate L1     L1-&gt;&gt;L0: a2     deactivate L1   </pre> <p>This sequence diagram illustrates the Alternating3Bit protocol. It shows two participants, L0 and L1, connected by vertical lifelines. The protocol consists of three alternating phases: m (message), a (acknowledgment), and m (message). The first phase starts with L0 sending m3 to L1. L1 then sends a3 back to L0. The second phase starts with L0 sending m1 to L1. L1 then sends a1 back to L0. The third phase starts with L0 sending m2 to L1. L1 then sends a2 back to L0.</p>	Lange et al.
Filter collaboration	<pre> sequenceDiagram     participant L0     participant L1     L0-&gt;&gt;L1: newFilterRequest     activate L1     L1-&gt;&gt;L0: itemToBeFiltered     activate L0     L0-&gt;&gt;L1: alt     activate L1     L1-&gt;&gt;L0: remove     deactivate L1     L0-&gt;&gt;L1: ok     deactivate L1     L1-&gt;&gt;L0: noMoreItems     deactivate L0   </pre> <p>This sequence diagram illustrates a filter collaboration. It shows two participants, L0 and L1, connected by vertical lifelines. The process begins with L0 sending a newFilterRequest to L1. L1 then sends an itemToBeFiltered back to L0. L0 then sends an alt (alternative) message to L1. L1 responds with a remove message. L0 then sends an ok message to L1. Finally, L1 sends a noMoreItems message back to L0.</p>	Lange et al.

Name	Interaction graphical representation	Reference
Game	<p>The diagram illustrates a sequence of interactions between several components. At the top, four lifelines are labeled <code>l0</code>, <code>l1</code>, <code>l2</code>, and <code>l3</code>. Below them, a vertical stack of components is shown: <code>loop</code>, <code>par</code>, <code>alt</code>, <code>bwin</code>, <code>close</code>, <code>cwin</code>, <code>blose</code>, <code>busy</code>, <code>msg</code>, <code>sig</code>, and <code>free</code>. The <code>alt</code> component contains the <code>bwin</code> and <code>cwin</code> components. The <code>close</code> component contains the <code>blose</code> component. The <code>busy</code> component contains the <code>msg</code> component. The <code>sig</code> component contains the <code>free</code> component. Interactions are represented by arrows: <code>bwin</code> sends to <code>close</code>; <code>cwin</code> sends to <code>blose</code>; <code>close</code> sends to <code>busy</code>; <code>msg</code> sends to <code>sig</code>; and <code>sig</code> sends to <code>free</code>.</p>	Lange et al.

Name	Interaction graphical representation						Reference
	l0	l1	l2	l3	l4	l5	
Health System	<pre> sequenceDiagram     participant HS as Health System     participant L as Logistic     loop l0         HS-&gt;&gt;L: sendData         activate L         L--&gt;&gt;HS: subscribed         deactivate L         alt alt             L--&gt;&gt;HS: nok         else             L--&gt;&gt;HS: notSubscribed             HS-&gt;&gt;L: account             L--&gt;&gt;HS: ok             HS-&gt;&gt;L: logCreated             L--&gt;&gt;HS: fwd             HS-&gt;&gt;L: fwd0k             L--&gt;&gt;HS: helpReq             HS-&gt;&gt;L: provideService             L--&gt;&gt;HS:          end     end     HS-&gt;&gt;L:  </pre>						Lange et al.
Logistic							Lange et al.

Name	Interaction graphical representation	Reference
	<pre>sequenceDiagram     participant l0     participant l1     participant l2     participant loop     l0-&gt;&gt;l1: plannedOrderVariations     activate l1     l1-&gt;&gt;l2: deliveryVariations     activate l2     l2-&gt;&gt;l1: driverCheckPointRequest     deactivate l2     l1-&gt;&gt;loop: ProvideItem     activate loop     loop-&gt;&gt;l1: DeliverItem     activate loop     loop-&gt;&gt;l1: DeliverItem     deactivate loop     l1--&gt;&gt;l1: ShippingDone     activate l1     l1-&gt;&gt;l2: updateP0andDeliverySchedule     activate l2     l2-&gt;&gt;loop: idDeliveryScheduleModes     activate loop     loop-&gt;&gt;l1: ConfirmationofDeliverySchedule     deactivate loop     l1-&gt;&gt;loop: updateP0andDeliverySchedule     activate loop     loop-&gt;&gt;l1: .zedP0andDeliverySchedule</pre>	

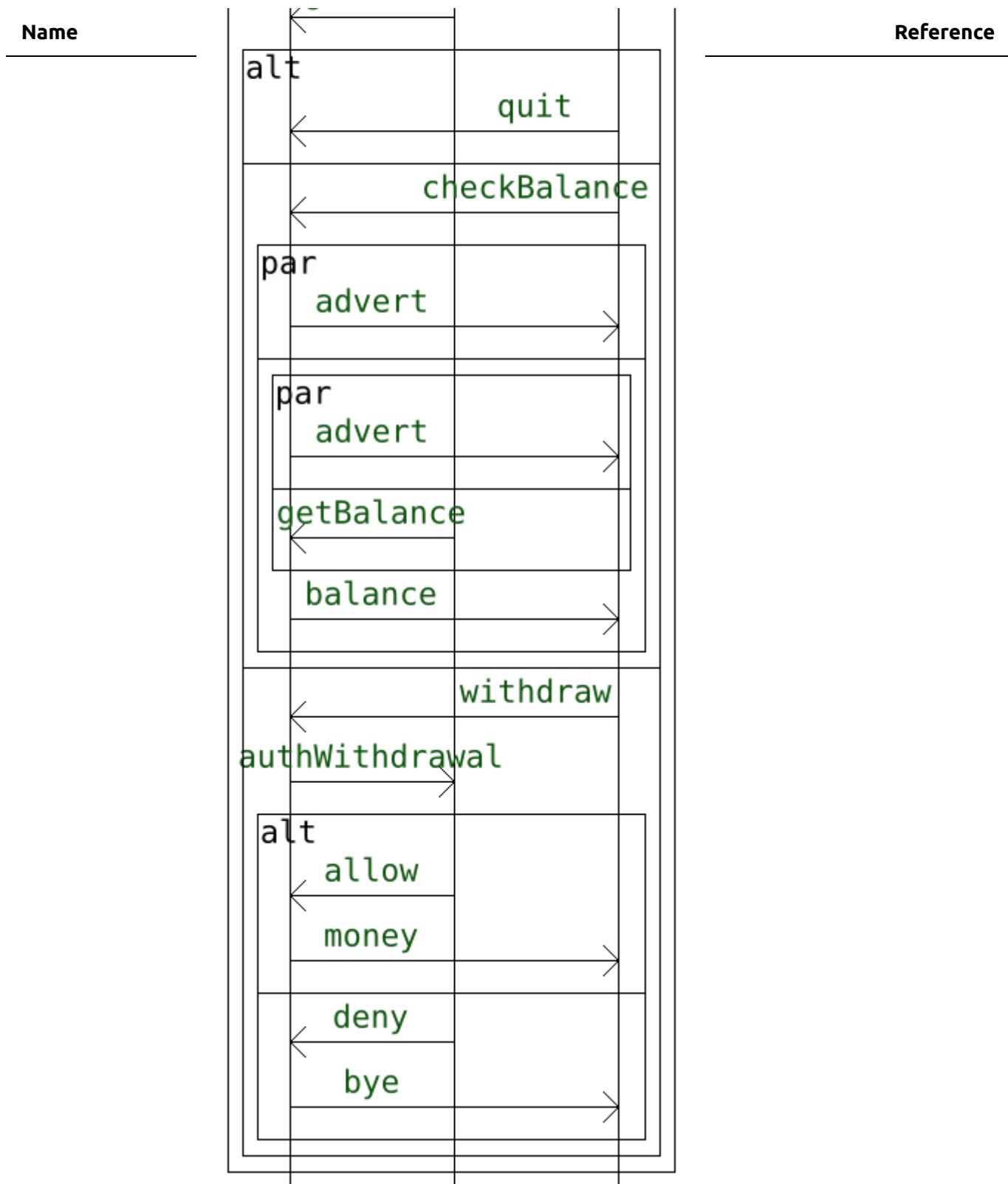
Name	Interaction graphical representation	Reference
Professor Online	<pre> sequenceDiagram     participant Teacher     participant ProfessorOnline     participant User     participant Classes     participant Plan     participant ClassRecord     participant GradeRecord     participant Subject      Note left of Teacher: loop     Note left of ProfessorOnline: loop     Note left of User: alt     Note left of Classes: loop     Note left of Plan: alt     Note left of ClassRecord: loop     Note left of GradeRecord: alt     Note left of Subject: loop      Teacher-&gt;&gt;User: login     User-&gt;&gt;Teacher: validateUser     Teacher--&gt;&gt;User: userOk     User--&gt;&gt;Teacher: logged     User-&gt;&gt;ProfessorOnline: chooseClass     ProfessorOnline--&gt;&gt;User: getClassPlanOk     User--&gt;&gt;ProfessorOnline: closePlanOk     alt User         ProfessorOnline-&gt;&gt;User: requestPlan         User--&gt;&gt;ProfessorOnline: enterPlan         ProfessorOnline-&gt;&gt;User: insertPlan         User--&gt;&gt;ProfessorOnline: closePlanOk     end      User-&gt;&gt;ProfessorOnline: enterOption     User--&gt;&gt;User: opt     ProfessorOnline--&gt;&gt;User: getWorkLoad     User--&gt;&gt;ProfessorOnline: getHours     ProfessorOnline--&gt;&gt;User: hours     User--&gt;&gt;workLoad: workLoad      alt ProfessorOnline         User-&gt;&gt;ProfessorOnline: enterClassRecord         ProfessorOnline-&gt;&gt;User: insertClassRecord         User--&gt;&gt;ProfessorOnline: classRecordOk     end      alt ProfessorOnline         User-&gt;&gt;ProfessorOnline: enterGradeRecord         ProfessorOnline-&gt;&gt;User: insertGradeRecord         User--&gt;&gt;ProfessorOnline: insertedGradeRecord     end      alt ProfessorOnline         User-&gt;&gt;ProfessorOnline: enterCloseClass         ProfessorOnline-&gt;&gt;User: updateCloseClass         User--&gt;&gt;ProfessorOnline: closedClass     end      User-&gt;&gt;ProfessorOnline: enterOption     User--&gt;&gt;User: opt     ProfessorOnline--&gt;&gt;User: chooseClass     User-&gt;&gt;ProfessorOnline: validateChooseClass     ProfessorOnline--&gt;&gt;User: chooseClassOk     User--&gt;&gt;exit: exit     User--&gt;&gt;logoff: logoff   </pre>	Rocha et al.
Sanitary Agency		Lange et al.

Name	Interaction graphical representation	Reference
	<pre> sequenceDiagram     participant L0     participant L1     participant L2     participant L3     L0-&gt;&gt;L1: request     activate L1     L1-&gt;&gt;L2: askInfo     activate L2     L2-&gt;&gt;L0: provInf     deactivate L2     deactivate L1     alt         L0-&gt;&gt;L1: refusal     end     alt         L1-&gt;&gt;L0: provM         L0-&gt;&gt;L1: recMoneyPossM         L1-&gt;&gt;L0: paymentM         L0-&gt;&gt;L1: provT     end     par         L0-&gt;&gt;L1: paymentPrivateFee         L1-&gt;&gt;L0: paymentPublicFee         L0-&gt;&gt;L1: done     end </pre>	

Name	Interaction graphical representation	Reference
TPM Contract v2	<p>The sequence diagram illustrates the interaction between two parties, <b>l0</b> and <b>l1</b>. The process begins with <b>l0</b> sending a message labeled <b>send</b> to <b>l1</b>. <b>l1</b> responds with <b>AckStartSend</b>. Subsequently, <b>l1</b> sends a message labeled <b>GetTpmStatus</b> to <b>l0</b>. <b>l0</b> responds with <b>TpmStatus</b>. This exchange repeats, with <b>l1</b> sending <b>SendComplete</b> to <b>l0</b> and <b>l0</b> responding with <b>GetTpmStatus</b>, which <b>l1</b> then sends back to <b>l0</b> as <b>TpmStatus</b>.</p>	Lange et al.
Travel		Bouma et al.

Name	Interaction graphical representation	Reference
	<pre>sequenceDiagram     participant C     participant A     participant S     C-&gt;&gt;A: Query     activate A     A--&gt;&gt;C: Quote     S-&gt;&gt;A: Dummy     deactivate A     activate loop     C-&gt;&gt;A: Query     activate A     A--&gt;&gt;C: Quote     S-&gt;&gt;A: Dummy     deactivate A     activate alt     C-&gt;&gt;A: Yes     activate A     A--&gt;&gt;C: Yes     S-&gt;&gt;A: Payment     deactivate A     activate alt     C-&gt;&gt;A: No     activate A     A--&gt;&gt;C: No     S-&gt;&gt;A: Bye     deactivate A</pre>	

Name	Interaction graphical representation	Reference
Two Buyers protocol	<pre> sequenceDiagram     participant S     participant a     participant b     S-&gt;&gt;a: loop     activate a     a--&gt;&gt;S: query     S-&gt;&gt;b: price     deactivate a     b--&gt;&gt;S: cancel     S-&gt;&gt;a: no     activate a     a--&gt;&gt;S: split     S-&gt;&gt;b: yes     deactivate a     b--&gt;&gt;S: no     S-&gt;&gt;a: no     activate a     a--&gt;&gt;S: buy     S-&gt;&gt;b: no     deactivate a     b--&gt;&gt;S: no     S-&gt;&gt;a: done     deactivate a     a--&gt;&gt;S: done   </pre>	Honda et al.
ATM	<pre> sequenceDiagram     participant a     participant b     participant c     b-&gt;&gt;c: auth     activate c     c--&gt;&gt;b: authReq     deactivate c     a--&gt;&gt;b: denied     activate a     a--&gt;&gt;b: authFailed     deactivate a     b--&gt;&gt;a: granted   </pre>	Edixhoven et al.



Distributed Voting

Edixhoven et al.

