FM26 artifact

Table of content

1. Introduction
2. Note on this document
3. Set Up
4. Interaction language
    - Representation of interactions
    - Gates
5. Artifact structure
6. Smoke tests
    - Composition smoke test
    - Reduced benchmark smoke test
7. Others Composition Examples
8. 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

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 developped in Rust.

Note on this document

We recommend reading this README file as the joint README.pdf file outside of the docker container, because of the images illustrations.

Set Up

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 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 generalizer:latest
```

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).

with the signature (sig.hsf): ~~~ @message{ bwin;cwin;close;blose;busy;msg;sig;free }

@lifeline{ l0;l1;l2;l3 } ~~~

and the interaction (i.hif): ~~~ loopS(seq(par(alt(l0 – cwin ->|, l0 – bwin ->|), busy -> l3), msg -> l0, sig -> l0, l0 – free -> l3)) ~~~

For instance `l0 -- cwin ->|` is an emission of the message `cwin` from lifeline `l0` to environment; and `busy -> l3` is reception of the message `busy` from environment to lifeline `l3`. The term `l0 -- free -> l3` represents the transmission of the message `free` from lifeline `l0` to lifeline `l3`, and is called a `value passing` in the paper.

The above interaction can be visualized as:

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(
                l0 -- cwin ->| [3],
                l0 -- bwin ->| [1]
            ),
            busy -> l3 [5]
        ),
        msg -> l0 [6],
        sig -> l0 [7],
        l0 -- free -> l3
    )
```

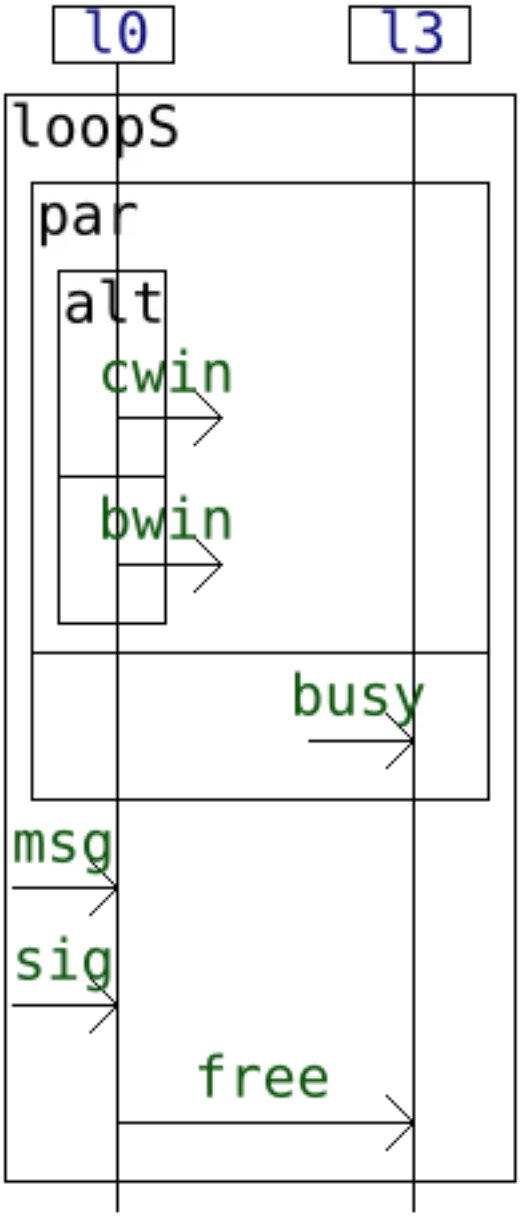


Figure 1: i0

)

which can be visually represented as:

Artifact structure

The `generalizer` folder contains the following subfolders:

```
generalizer
    LICENSE
    README.pdf
    README.md
    Dockerfile
    Executable
    Benchmark
    smoke_tests
    Interactions_examples
    generalizer_sources.tar.gz
    readme
```

The Docker image provides a built executable under the folder `generalizer/Executable`. The smoke tests are located in the `generalizer/smoke_tests` directory, the benchmark scripts in `generalizer/Benchmark` directory; some examples of interactions are provided in `generalizer/Interactions_examples` directory, together with scripts to compose them.

Smoke tests

By running the container, `Docker` will open a shell in a container 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`. 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`.

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

If successful, the success message will be printed in the terminal. 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

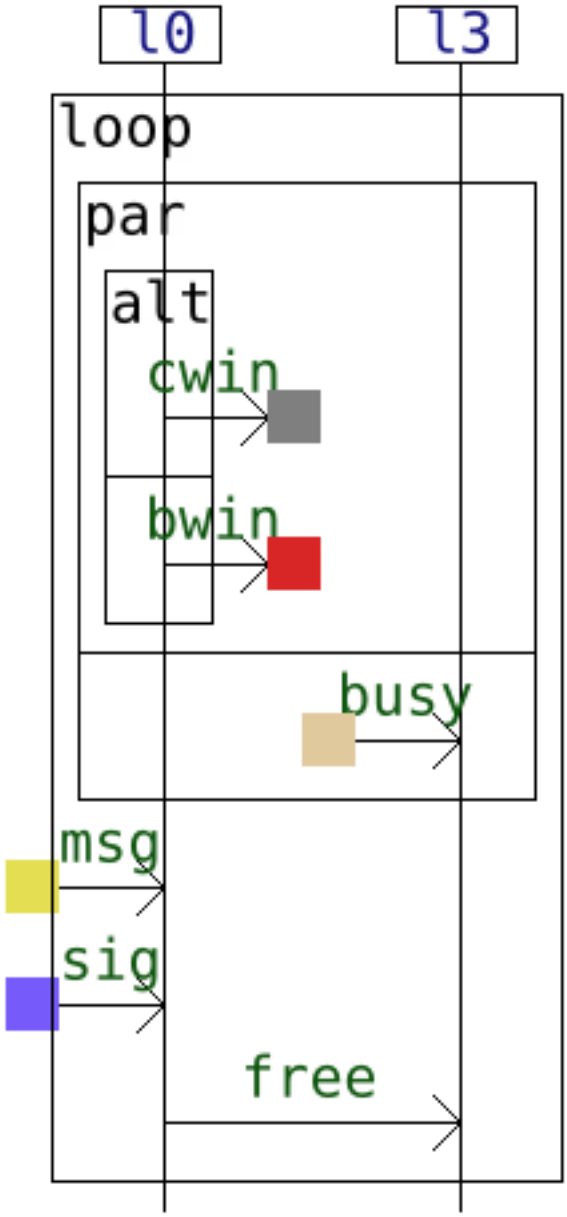


Figure 2: i0

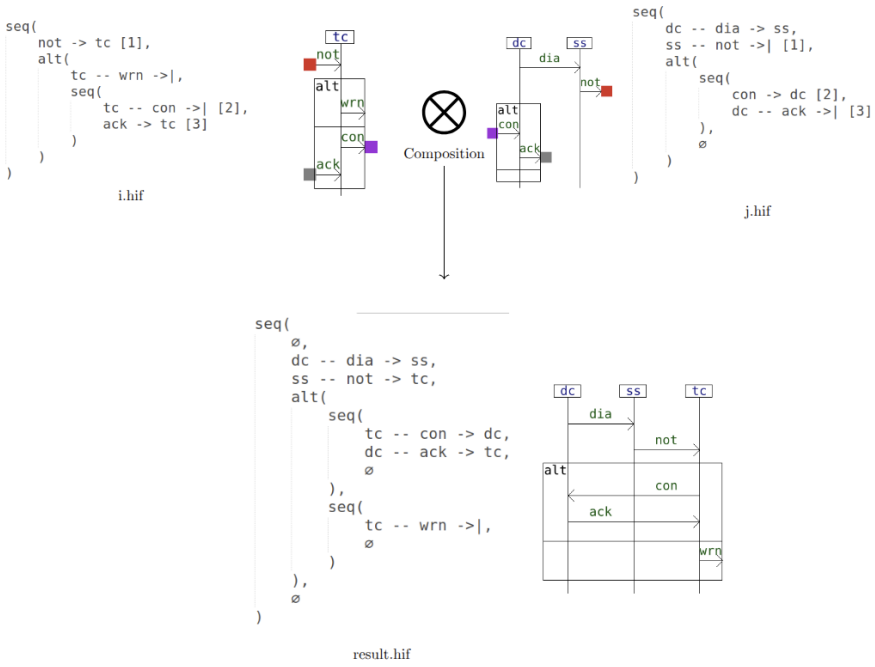


Figure 3: figure

the result). The folder `input` also contains pictures `i.png` and `j.png` of the interactions.

Reduced benchmark smoke test

To quickly check wheher 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
```

The result will be put in the folder `Benchmark_Output`. It containts a csv file `result_one_pass.csv` containing a table akin the exprerimental section of the paper.

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

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

The following table 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	3.803(0k)	18.791(0k)	5.418(0k)	28.168(0k)
FilterCo	11	5	1.144(0k)	1.273(0k)	1.676(0k)	1.889(0k)
TPM	17	7	2.528(0k)	4.174(0k)	3.768(0k)	6.218(0k)

This smoke test executes in one pass the three steps of the benchmark described in details the Section Benchmark below.

Others 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.

```
$ cd Benchmark
```

We benchmark is divided into three steps each performed by the scripts:

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 $\lfloor L/2 \rfloor$.

For each partition (L_1, L_2) of a set of lifelines of a global interaction r:

- project r onto L_1 and L_2 to obtain local interactions i and j;
- we normalize i and j using HIBOU to obtain i_{norm} and j_{norm} respectively.
- we apply mutation operations to i and i, with consists of successively applying N_m times one of the following rewrite operation selected uniformly at random: $\mathsf{alt}(x, y) \to \mathsf{alt}(y, x)$ and $\mathsf{par}(x, y) \to \mathsf{par}(y, x)$. We obtain the interactions i_{mut} and j_{mut} from s and t respectively. The mutations are done with Maude.

```
$ ./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.hsf
         Game.hif
    partition0
        original_locals
        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 folders `with_normalized_locals` and `with_mutated_locals` have the same structure, as well as the partitions folders.

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

Step 2: composition

We compose the pairs $(i_{\text{norm}}, j_{\text{norm}})$ and $(i_{\text{mut}}, j_{\text{mut}})$. The result of the composition is normalized with HIBOU an compared to the normal form of the starting interaction k.

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`.

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

```
Game
    input_global_interaction
         Game.hsf
         Game.hif
    partition0
        original_locals
        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` con-

tain 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/result_one_pass.csv | less -S
```

Or with column shrinked down:

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

TODO: add a picture of table with the results obtained in the docker.

Each interaction corresponds to a row in the table. The second column indicates the size of each interaction, while the third column shows the range of the number of gates in local interactions with respect to partitions of lifelines. The last four columns represent the average composition duration across partitions, with and without the rule **Fail**. In particular, the fourth and fifth columns report the average duration for the composition of normalized local interactions, and the last two columns report the average duration for the mutated local interactions.

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).

Summary of the workflow for the interaction Game

The following figure illustrates our protocol with the Game global interaction, with only the mutation scenario.

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 4: benchmark_table

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.
- Timout 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 timout threshold is of 60s. the flag -m means that in the

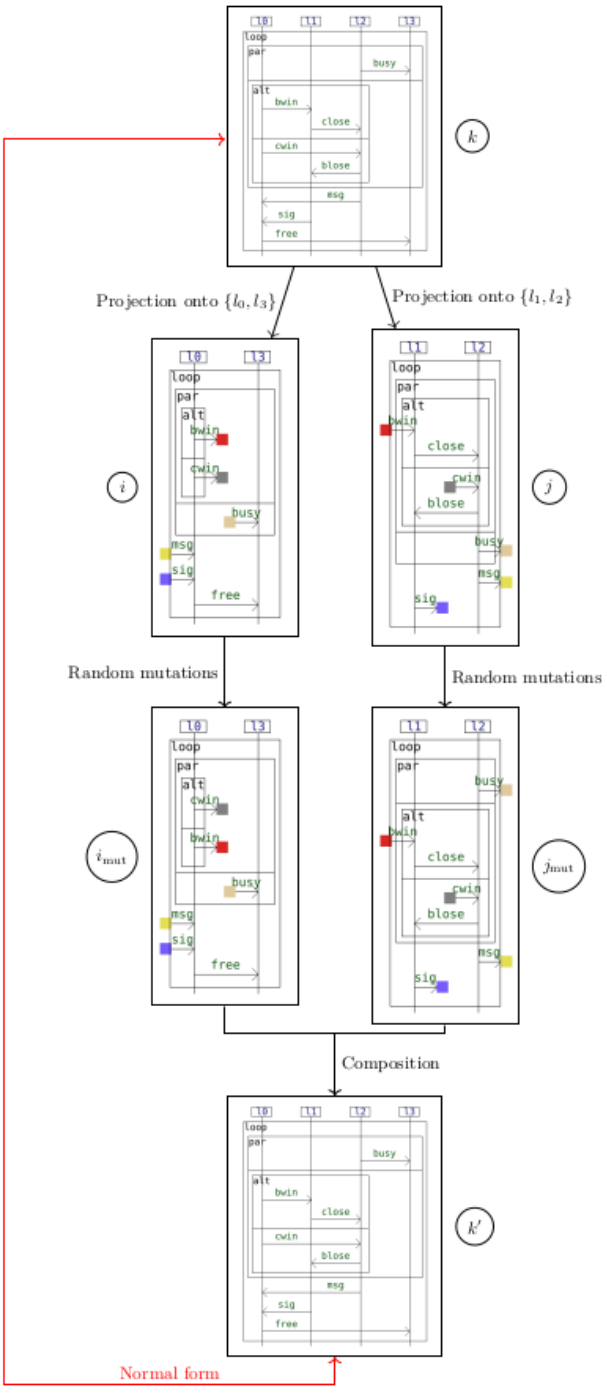


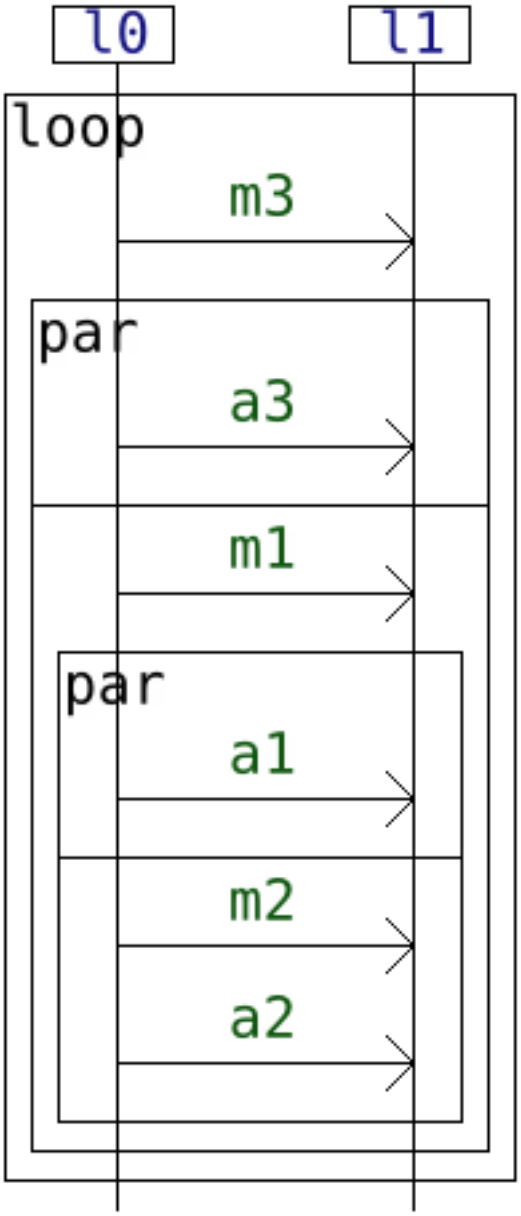
Figure 5: workflow

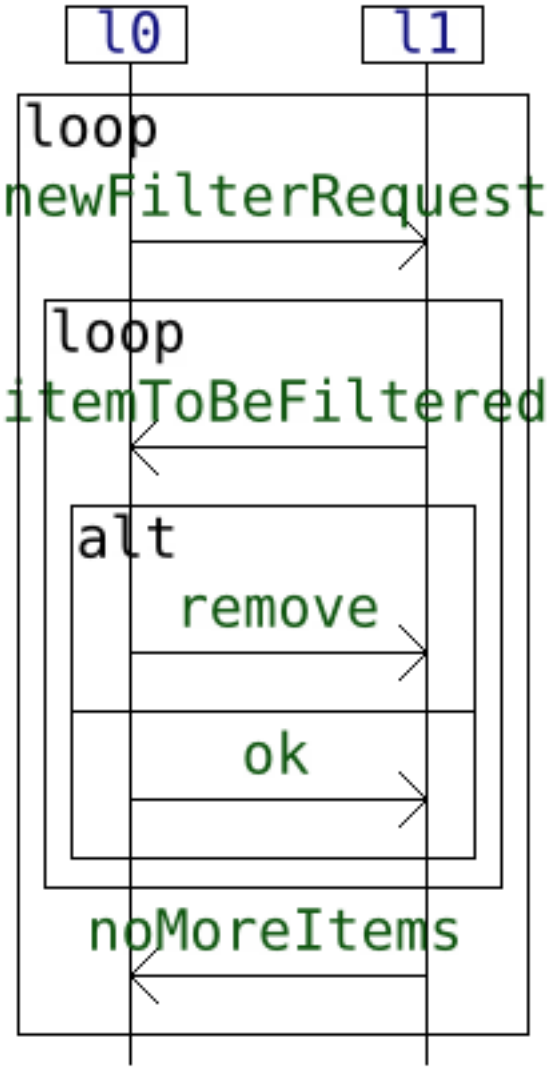
output csv file, the duration will be given in milliseconds. The theory for the composition is ACU (all the rules are used).

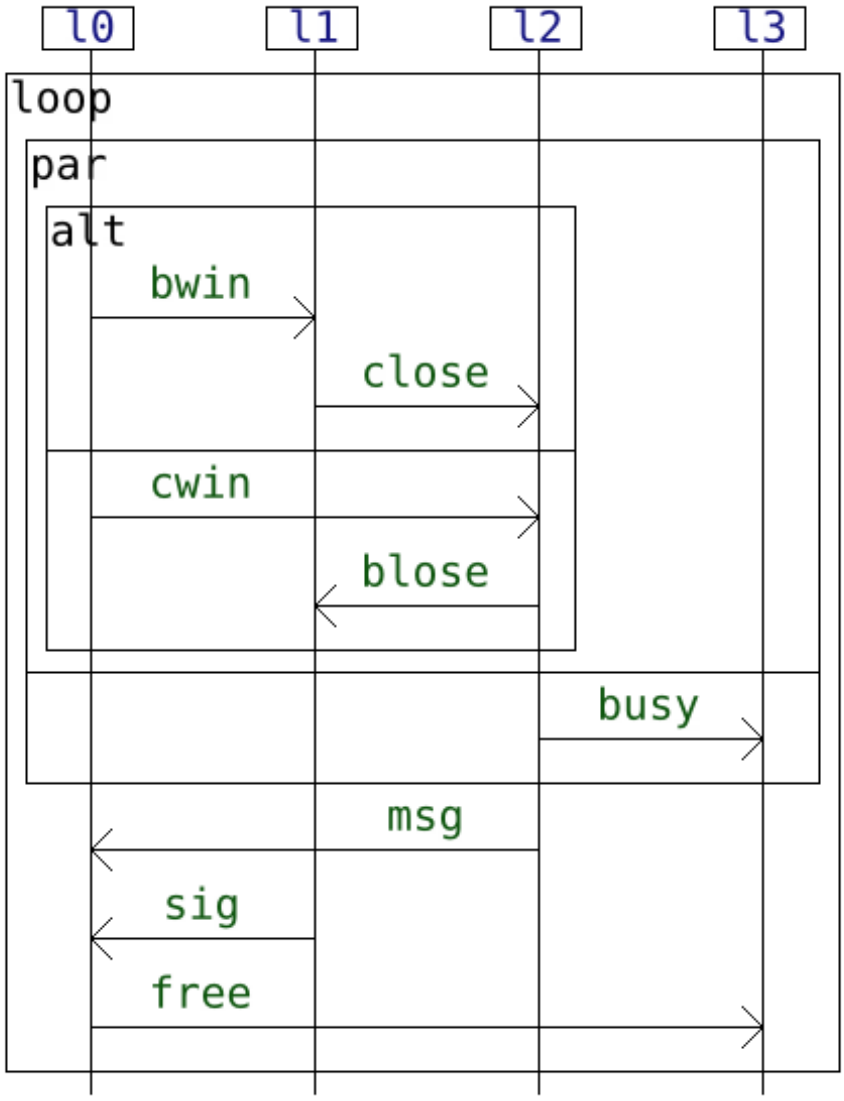
To draw the interactions involved in the process, we can use the flag `-d`.

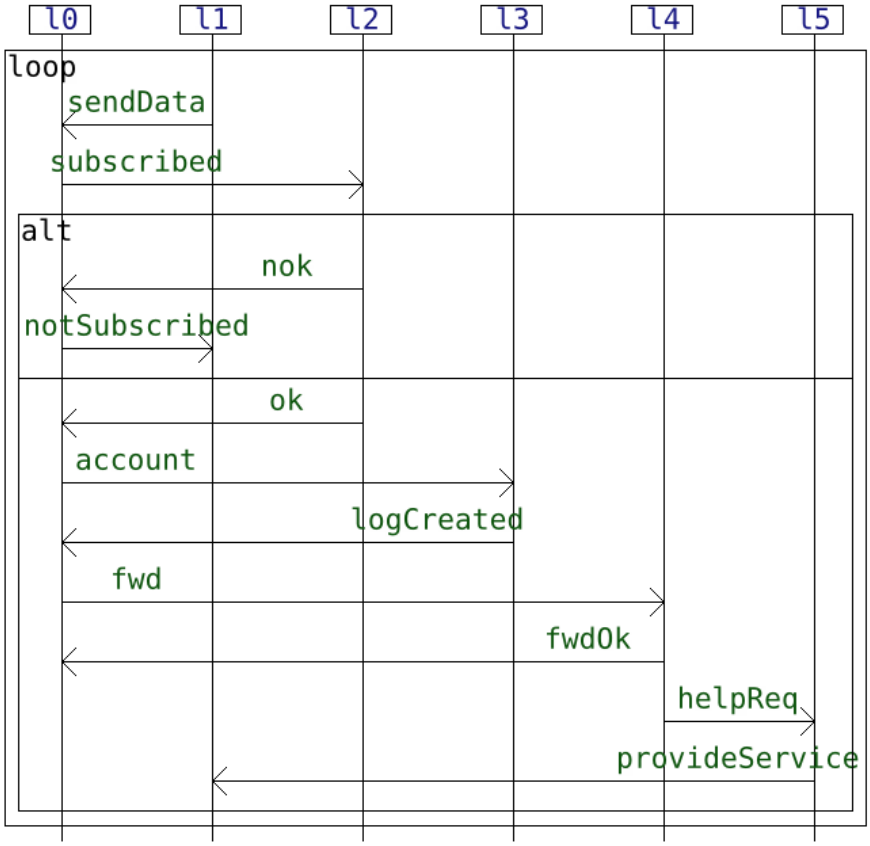
Interactions of the benchmark

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.

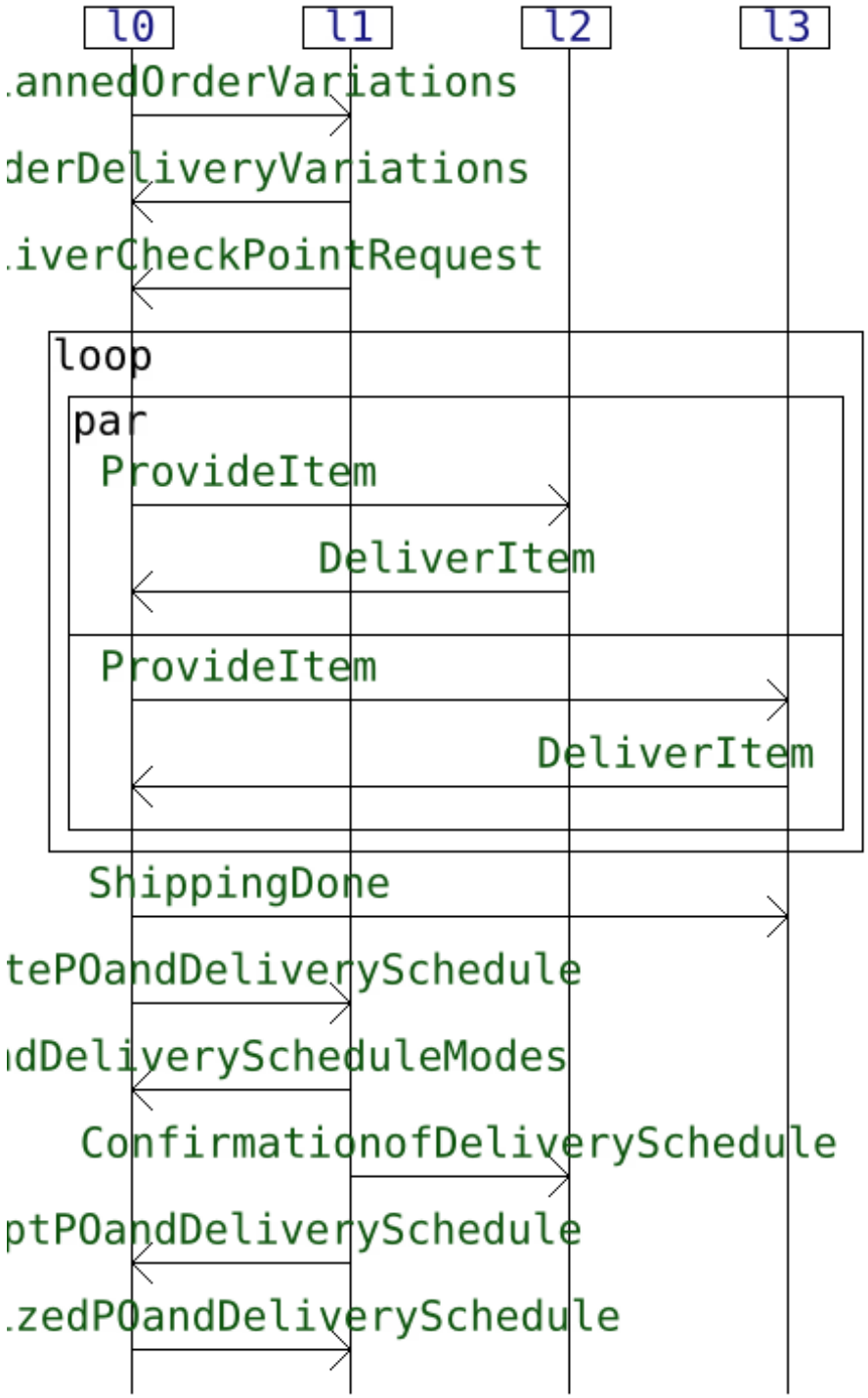
Name	Interaction graphical representation
Alternating3Bit Protocol	

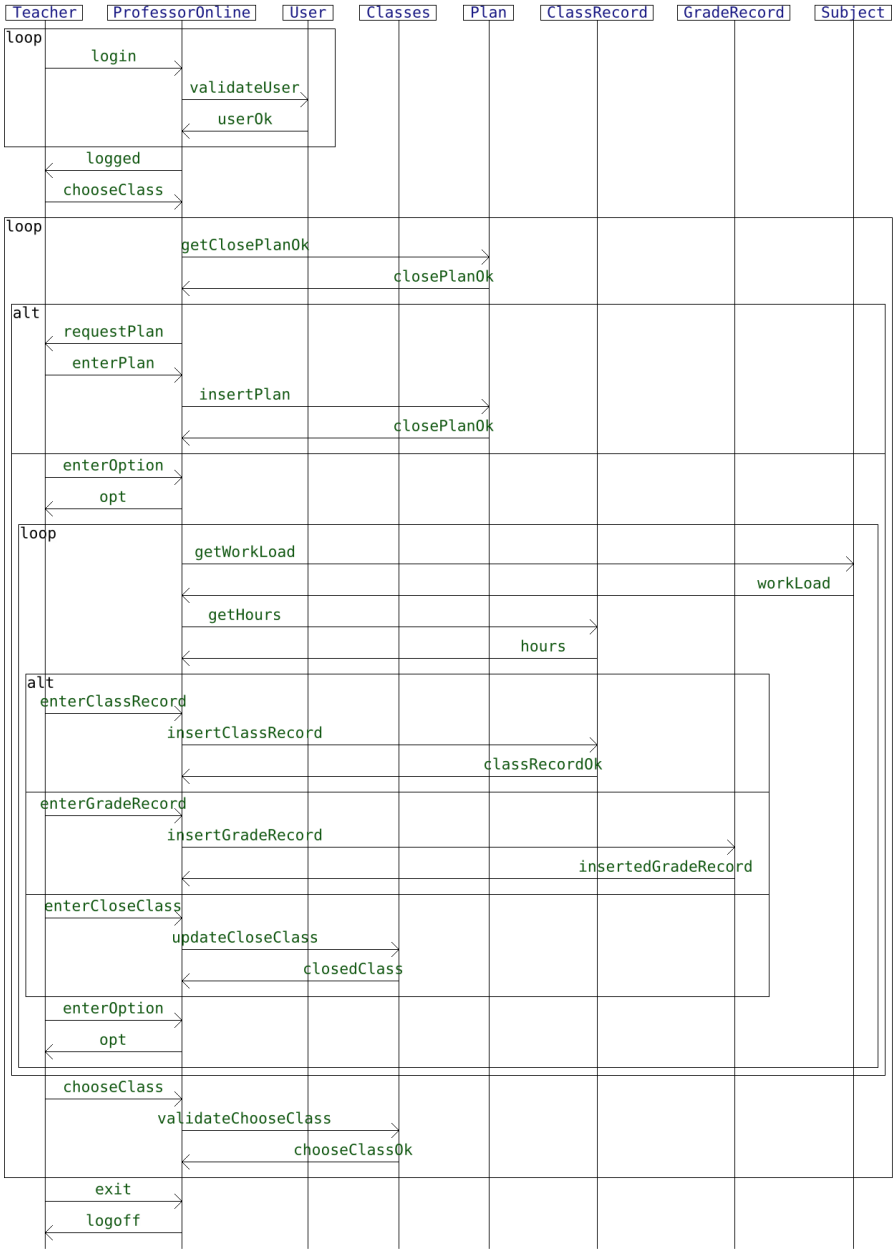
Name	Interaction graphical representation
Filter collaboration	

Name	Interaction graphical representation
Game	

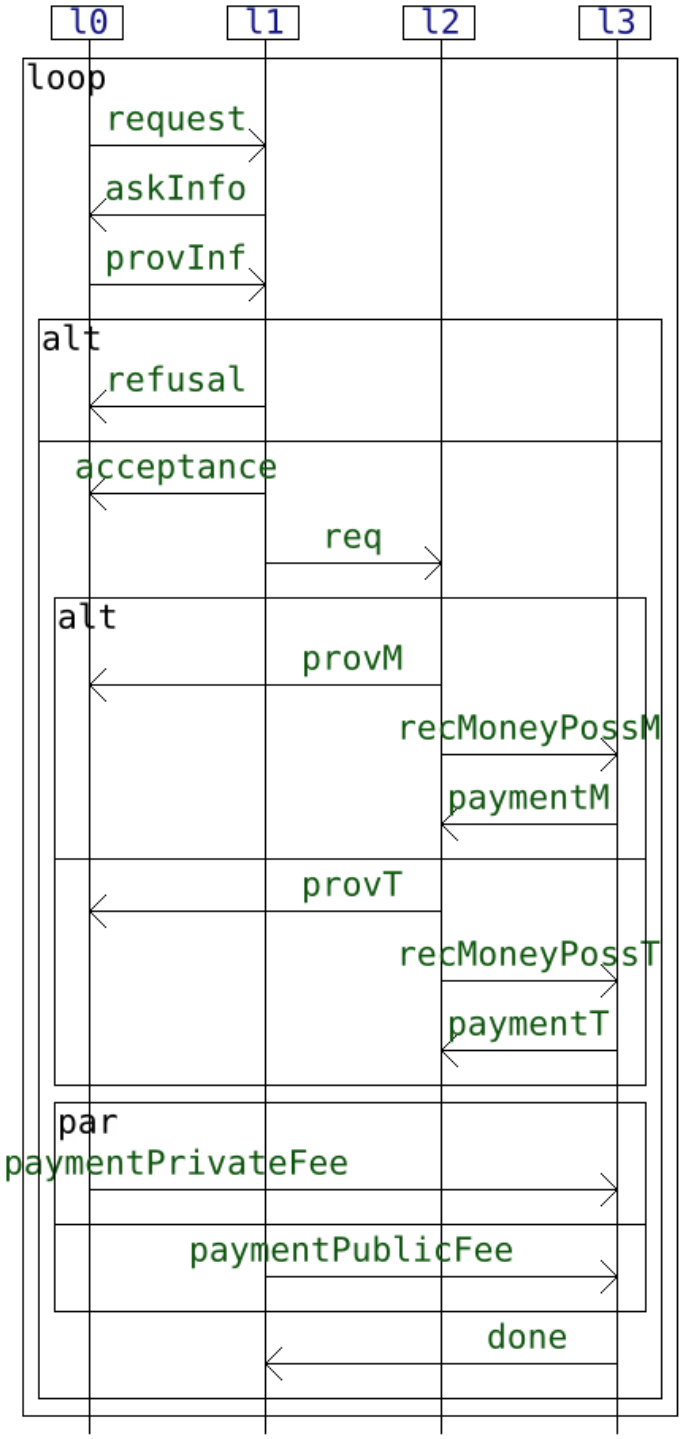
Name	Interaction graphical representation
Health System	

Name	Interaction graphical representation

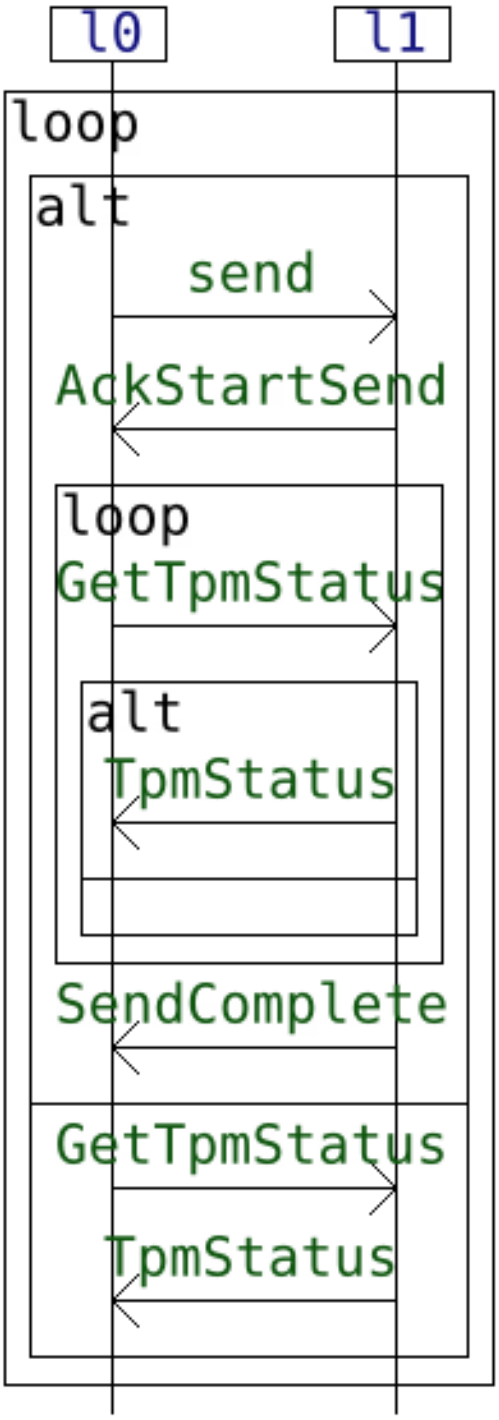


Name	Interaction graphical representation
Professor Online	

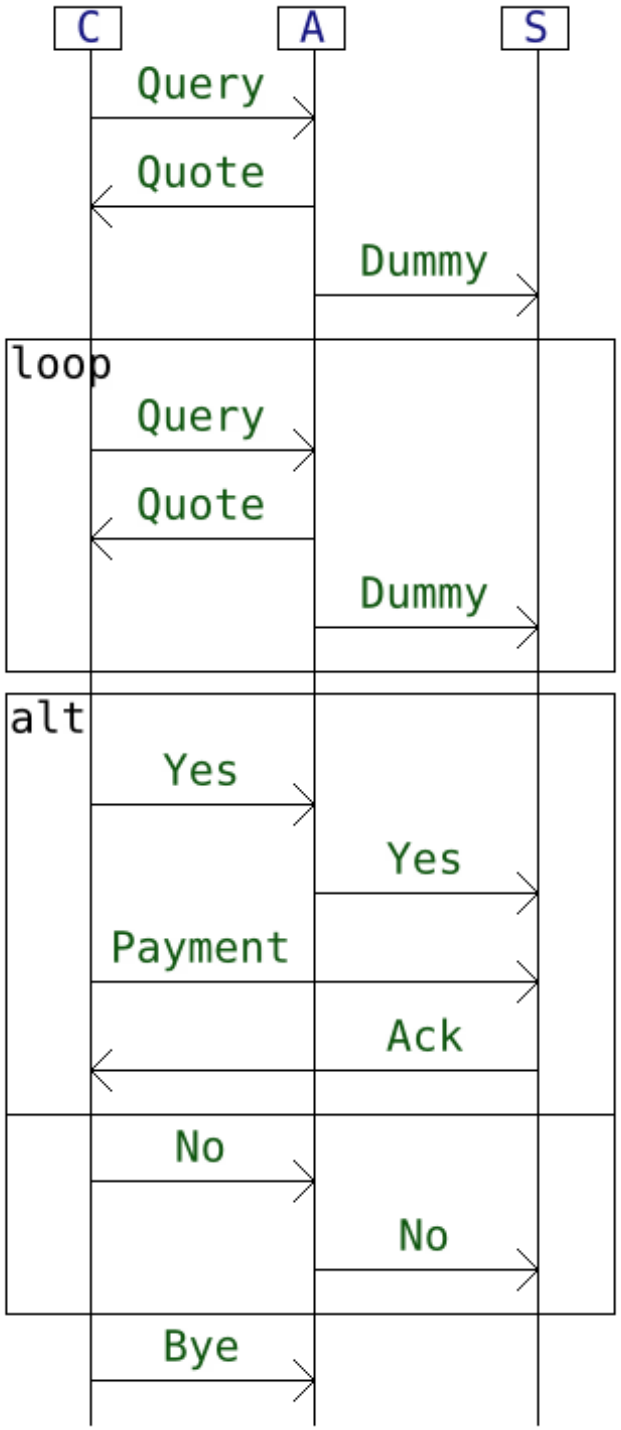
Name	Interaction graphical representation	



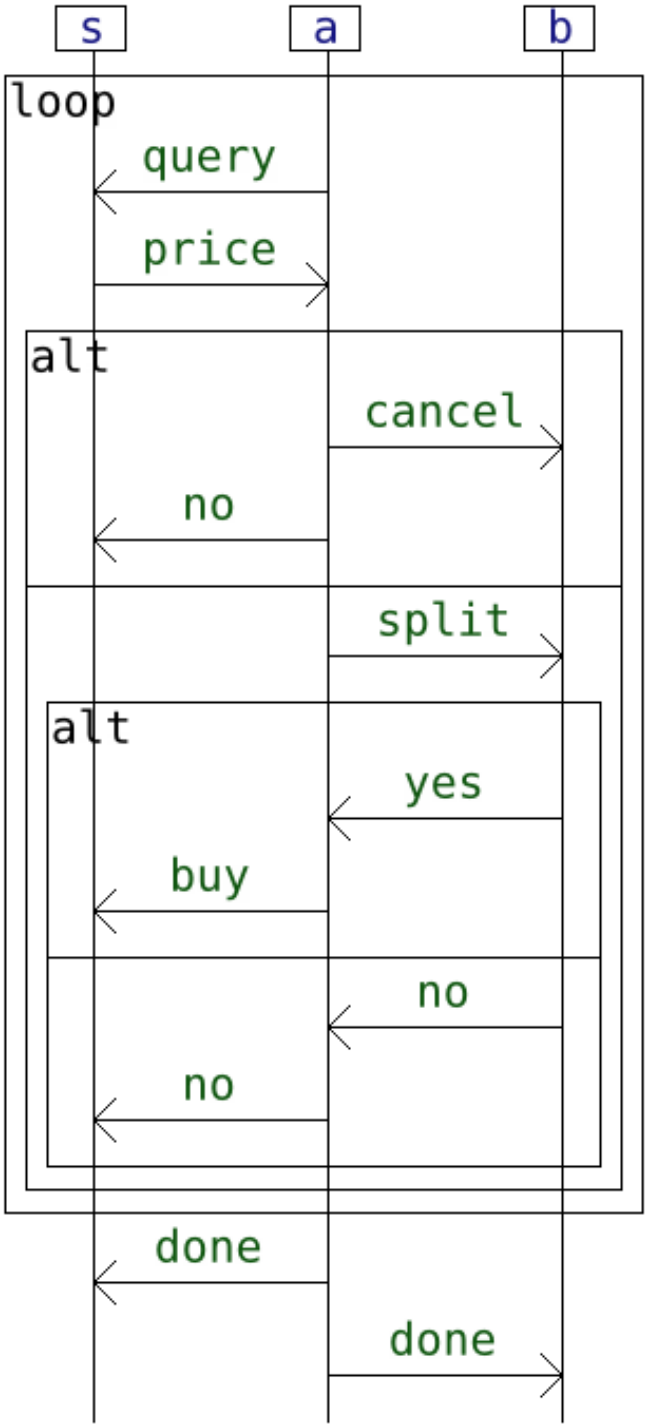
Name	Interaction graphical representation	



Name	Interaction graphical representation	

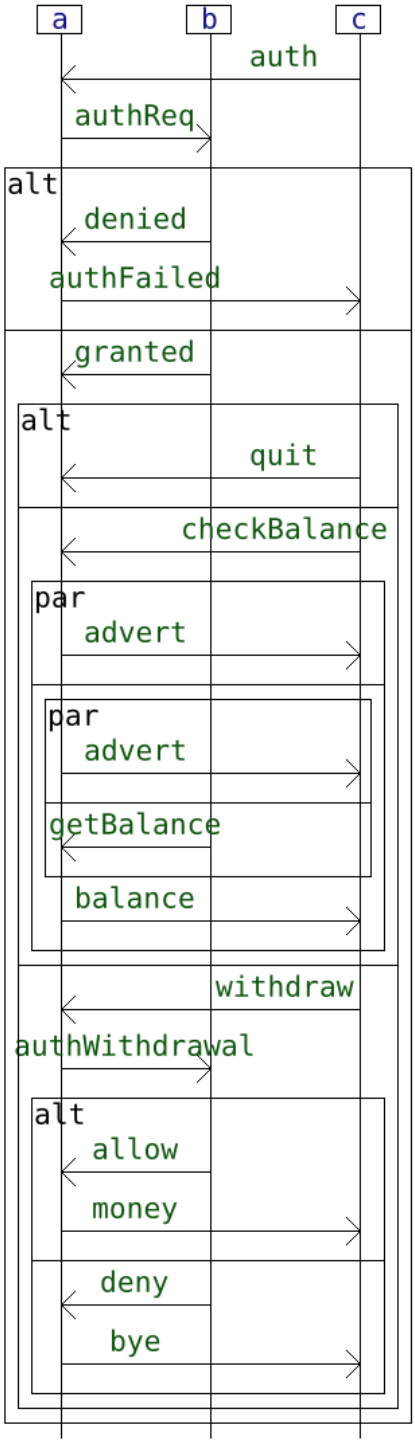


Name	Interaction graphical representation	

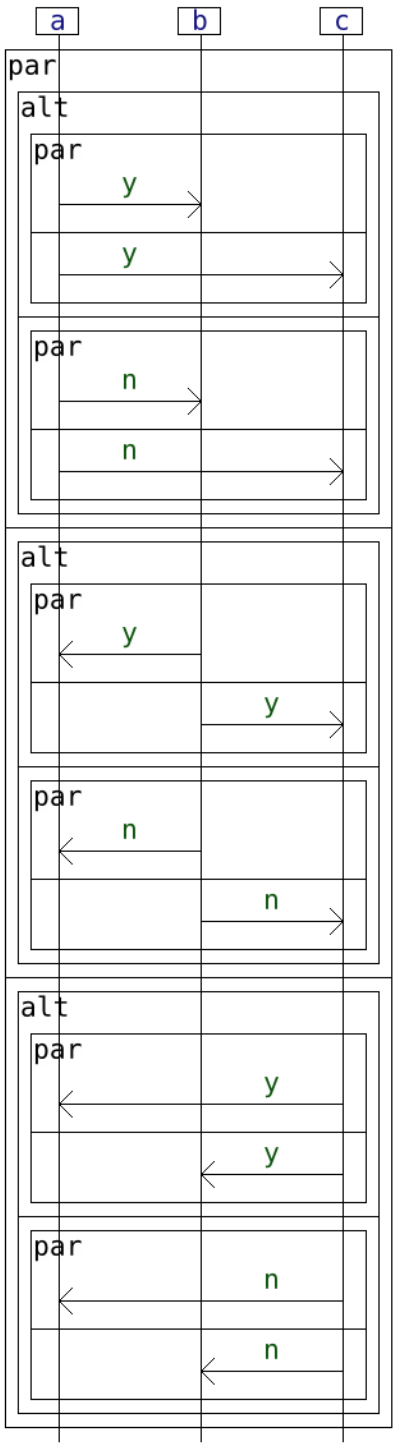


Two Buyers protocol

Name	Interaction graphical representation	



ATM


a
b
c
par
alt
par
y
y
par
n
n
alt
par
y
y
par
n
n
alt
par
y
y
par
n
n

Name	Interaction graphical representation	