MINERful tutorial: real event log

Cecilia Iacometta, c.iacometta@students.uu.nl
Claudio Di Ciccio, c.diciccio@uu.nl
Utrecht University, Netherlands

June 26, 2025

Welcome to the tutorial on MINERful. We assume you have already downloaded the tool from `https://github.com/cdc08x/MINERful/`
To execute the JAR, make sure to have JRE 22+ installed. Any version beyond 11 can work but, in that case, you should recompile the sources to obtain a compatible JAR.

1 What is MINERful?

MINERful is a process mining, simulation, and analysis tool for declarative process specifications.
The main functionalities of MINERful are:

1. Discovery of declarative specifications of process control flows out of event logs. Event logs can be stored as XES, MXML, or text files (a collection of strings, in which every character is considered as an event, every line as a trace);
2. Simulation of declarative specifications of process control flows and creation of synthetic event logs. The input process specification can be given as a JSON file. The event logs can be exported as XES, MXML, or text files;
3. Simplification and inconsistency removal of declarative process specifications;
4. Discovery of declarative specifications of process control flows from windows over event logs.
5. Fitness checker of passed declarative process specifications with event logs.

For more information about MINERful, please visit the Wiki at this address:
`https://github.com/cdc08x/MINERful/wiki`

2 Introduction

The easiest way to launch MINERful if you use a Unix/Linux or a Mac machine is to run the `.sh` files.

1. To launch the miner: `run-MINERful.sh`
2. To create synthetic logs: `run-MINERfulEventLogMaker.sh`
3. To process and simplify specifications: `run-MINERfulSimplifier.sh`
4. To launch the miner on sublogs as windows over an original event log: `run-MINERfulSlider.sh`
5. To check the fitness of a passed declarative process specification with an event log: `run-MINERfulFitnessChecker.sh`

If you use a Windows machine, please run the following command from within a terminal (prompt) locating the current directory in the folder where the MINERful jar lies.

COMMAND LINE: `java -cp ".\lib\*;MINERful.jar" <MAIN_CLASS>`

where `<MAIN_CLASS>` is either of the following, depending on the above functionalities you wish to employ:

1. `minerful.MinerFulMinerStarter`
2. `minerful.MinerFulLogMakerStarter`
3. `minerful.MinerFulSimplificationStarter`
4. `minerful.MinerFulMinerSlider`
5. `minerful.MinerFulFitnessCheckerStarter`

Each of those launchers can be invoked with the `-h` parameter, to get guidance on how to use them. An explanation of all possible parameters that can be passed will be displayed (beware, they are quite an amount so take your time).

For example, MINERful allows the setting of thresholds for both event and trace-based quality measures (confidence, support, and coverage) while discovering constraints from event logs or simplifying existing specifications. The obtained results can be saved in the form of CSV or JSON files. Furthermore, with MINERful we can create a finite state automaton in DOT of the discovered process specification. To open GraphViz DOT files, please check out the GraphViz toolkit (an online viewer is also available at `https://dreampuf.github.io/GraphvizOnline/`).

3 MINERful functionalities starting from a real-world event log: Sepsis

3.1 Discovery

Starting from a public, real-world event log containing events of sepsis cases treated in a hospital (download the Sepsis event log), we can proceed with the discovery. Results are in line with the ones illustrated in 'Analyzing the Trajectories of Patients with Sepsis using Process Mining'. We have used SepsisCases-SORTed.xes which is available in GitHub on `https://github.com/cdc08x/MINERful/` in `logs` folder.
To speed up the post-processing, other techniques than `hierarchyconflictredundancydouble` (say, `hierarchyconflictredundancy`, or just `hierarchy`, the default) can be used. However, this implies less accuracy in the detection (and removal) of the redundant constraints. Generally, the mining per se is very fast. The post-processing takes most of the computation time.

COMMAND LINE (Linux/Mac):

```
./run-MINERful.sh -iLF logs/SepsisCases-SORTed.xes -prune 'hierarchyconflictredundancydouble'
--support 0.04 --confidence 0.85 --coverage 0.04 --trace-support 0.125 --trace-confidence 0.85 --trace-coverage 0.125
-oCSV specifications/sepsis.csv -oJSON specifications/sepsis.json -autoDOT automata/sepsis.dot
```

COMMAND LINE (Windows):

```
java -cp ".\lib\*;MINERful.jar" minerful.MinerFulMinerStarter -iLF logs/SepsisCases-SORTed.xes -prune 'hierarchyconflictredundancydouble'
--support 0.04 --confidence 0.85 --coverage 0.04 --trace-support 0.125 --trace-confidence 0.85 --trace-coverage 0.125
-oCSV specifications/sepsis.csv -oJSON specifications/sepsis.json -autoDOT automata/sepsis.dot
```

Next, please find the table of declarative constraints we obtain, and the GraphViz plot stemming from the DOT file.

Constraint	Template	Activation	Target	Confidence	Coverage	Support	Trace confidence	Trace coverage	Trace support
NotSuccession(Admission IC, ER Registration)	NotSuccession	Admission IC	ER Registration	1.000	0.077	0.077	0.895	1.000	0.895
AtMost2(Admission NC)	AtMost2		Admission NC	0.955	0.069	0.066	0.955	1.000	0.955
NotResponse(Admission NC, LacticAcid)	NotResponse	Admission NC	LacticAcid	0.889	0.078	0.069	0.876	0.762	0.668
NotChainSuccession(Admission NC, LacticAcid)	NotChainSuccession	Admission NC	LacticAcid	0.980	0.174	0.171	0.888	0.894	0.794
Precedence(IV Antibiotics, Admission NC)	Precedence	Admission NC	IV Antibiotics	0.874	0.078	0.068	0.859	0.762	0.654
NotChainSuccession(CRP, ER Sepsis Triage)	NotChainSuccession	CRP	ER Sepsis Triage	0.984	0.283	0.279	0.929	1.000	0.929
NotChainResponse(CRP, Return ER)	NotChainResponse	CRP	Return ER	1.000	0.214	0.214	1.000	0.959	0.959
Init(ER Registration)	Init		ER Registration	0.948	0.069	0.065	0.948	1.000	0.948
AlternateResponse(ER Registration, ER Sepsis Triage)	AlternateResponse	ER Registration	ER Sepsis Triage	0.992	0.069	0.068	0.992	1.000	0.992
ChainResponse(ER Registration, ER Triage)	ChainResponse	ER Registration	ER Triage	0.925	0.069	0.064	0.925	1.000	0.925
AlternateResponse(ER Registration, Leucocytes)	AlternateResponse	ER Registration	Leucocytes	0.960	0.069	0.066	0.960	1.000	0.960
NotChainResponse(ER Sepsis Triage, LacticAcid)	NotChainResponse	ER Sepsis Triage	LacticAcid	0.859	0.069	0.059	0.859	0.999	0.858
ChainPrecedence(ER Triage, ER Sepsis Triage)	ChainPrecedence	ER Sepsis Triage	ER Triage	0.863	0.069	0.059	0.863	0.999	0.862
AtMost1(ER Triage)	AtMost1		ER Triage	0.997	0.069	0.069	0.997	1.000	0.997
AtMost1(IV Antibiotics)	AtMost1		IV Antibiotics	1.000	0.069	0.069	1.000	1.000	1.000
AlternatePrecedence(CRP, IV Antibiotics)	AlternatePrecedence	IV Antibiotics	CRP	0.855	0.054	0.046	0.855	0.784	0.670
NotChainPrecedence(CRP, IV Antibiotics)	NotChainPrecedence	IV Antibiotics	CRP	0.902	0.054	0.049	0.902	0.784	0.707
CoExistence(IV Antibiotics, IV Liquid)	CoExistence	IV Antibiotics	IV Liquid	0.956	0.104	0.099	0.915	0.784	0.717
AlternatePrecedence(Leucocytes, IV Antibiotics)	AlternatePrecedence	IV Antibiotics	Leucocytes	0.855	0.054	0.046	0.855	0.784	0.670
NotChainPrecedence(Leucocytes, IV Antibiotics)	NotChainPrecedence	IV Antibiotics	Leucocytes	0.911	0.054	0.049	0.911	0.784	0.714
AtMost1(IV Liquid)	AtMost1		IV Liquid	1.000	0.069	0.069	1.000	1.000	1.000
AtMost2(LacticAcid)	AtMost2		LacticAcid	0.904	0.069	0.062	0.904	1.000	0.904
NotChainResponse(Leucocytes, Admission NC)	NotChainResponse	Leucocytes	Admission NC	0.955	0.222	0.212	0.857	0.964	0.826
NotChainSuccession(Leucocytes, ER Sepsis Triage)	NotChainSuccession	Leucocytes	ER Sepsis Triage	0.984	0.291	0.287	0.900	1.000	0.900
AlternatePrecedence(Admission NC, Release A)	AlternatePrecedence	Release A	Admission NC	0.999	0.044	0.044	0.999	0.639	0.638
NotChainSuccession(Release A, Admission NC)	NotChainSuccession	Release A	Admission NC	1.000	0.122	0.122	0.914	0.763	0.697
NotChainSuccession(Release A, CRP)	NotChainSuccession	Release A	CRP	0.999	0.259	0.258	0.994	0.965	0.959
NotChainSuccession(Release A, Leucocytes)	NotChainSuccession	Release A	Leucocytes	1.000	0.266	0.266	0.997	0.967	0.964
NotCoExistence(Release B, ER Registration)	NotCoExistence	Release B	ER Registration	0.899	0.073	0.065	0.947	1.000	0.947
NotCoExistence(Release C, ER Registration)	NotCoExistence	Release C	ER Registration	0.953	0.071	0.067	0.976	1.000	0.976
NotCoExistence(Release D, ER Registration)	NotCoExistence	Release D	ER Registration	0.955	0.071	0.067	0.977	1.000	0.977
AtMost1(Release E)	AtMost1		Release E	1.000	0.069	0.069	1.000	1.000	1.000
NotCoExistence(Release E, ER Registration)	NotCoExistence	Release E	ER Registration	0.989	0.069	0.069	0.994	1.000	0.994
NotChainSuccession(Return ER, Admission NC)	NotChainSuccession	Return ER	Admission NC	1.000	0.097	0.097	1.000	0.762	0.762
NotChainSuccession(Return ER, ER Sepsis Triage)	NotChainSuccession	Return ER	ER Sepsis Triage	1.000	0.088	0.088	0.999	1.000	0.999
NotChainSuccession(Return ER, IV Antibiotics)	NotChainSuccession	Return ER	IV Antibiotics	1.000	0.073	0.073	0.964	0.813	0.784
NotChainSuccession(Return ER, IV Liquid)	NotChainSuccession	Return ER	IV Liquid	1.000	0.069	0.069	0.937	0.766	0.717
NotChainSuccession(Return ER, LacticAcid)	NotChainSuccession	Return ER	LacticAcid	1.000	0.116	0.116	0.978	0.837	0.819
NotChainSuccession(Return ER, Leucocytes)	NotChainSuccession	Return ER	Leucocytes	1.000	0.242	0.242	0.999	0.965	0.964

Figure 1: Discovered process specification table with quality measures

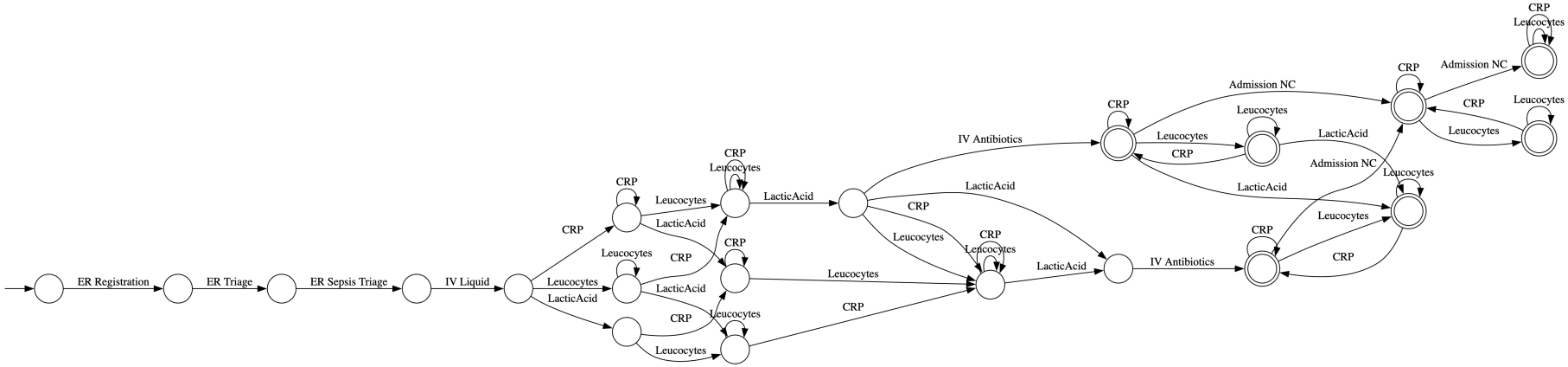

Figure 2: Automaton Sepsis process specification

3.2 Simulation

Starting from Sepsis discovered specification stored as JSON, MINERful simulation functionality permits to create synthetic event logs. The event logs can be exported as XES, MXML files or text files (a collection of strings, in which every character is considered as an event, every line as a trace).

By running MINERful as an event log maker, we can create synthetic event logs partially satisfying a given process specification. We can create a synthetic event log with 50 traces constituted by a number of events between 2 and 24. In this log, we let *Init(ER Registration)* be violated in 5 traces. Next, you can see the constraints to be violated in a JSON format below. Please save the lines below it as `specifications/constraints.json` should look as follows:

```
{
"name": "Sepsis process constraints to be violated",
    "constraints": [
        {"template": "init", "parameters":[["ER Registration"]]}
    ]
}
```

COMMAND LINE (Linux/Mac):

```
./run-MINERfulEventLogMaker.sh --input-specification-file specifications/sepsis.json --size 50 --minlen 2 --maxlen 24
--out-log-encoding xes --out-log-file logs/simulated_Sepsis.xes --sizeviol 5 --viol-const-file specifications/constraints.json
```

COMMAND LINE (Windows):

```
java -cp ".\lib\*;MINERful.jar" minerful.MinerFulLogMakerStarter --input-specification-file specifications/sepsis.json
--size 50 --minlen 2 --maxlen 24 --out-log-encoding xes --out-log-file logs/simulated_Sepsis.xes
--sizeviol 50 --viol-const-file specifications/constraints.json
```

3.3 Fitness Checking

Starting form Sepsis discovered specification and the synthetic event log we can check the fitness.

COMMAND LINE (Linux/Mac):

```
./run-MINERfulFitnessChecker.sh -iLF logs/simulated_Sepsis.xes -iLF xes -iSF specifications/sepsis.json -iSE json
-prune 'hierarchyconflictredundancydouble' --support 0.04 --confidence 0.85 --coverage 0.04
--trace-support 0.125 --trace-confidence 0.85 --trace-coverage 0.125 -oCSV specifications/fitness_Sepsis.csv
```

COMMAND LINE (Windows):

```
java -cp ".\lib\*;MINERful.jar" minerful.MinerFulFitnessCheckStarter -iLF xes -iSF specifications/sepsis.json -iSE json
-prune 'hierarchyconflictredundancydouble' --support 0.04 --confidence 0.85 --coverage 0.04
--trace-support 0.125 --trace-confidence 0.85 --trace-coverage 0.125 -oCSV specifications/fitness_Sepsis.csv
```

Note, we expect *Init(ER Registration)* fitness value to be 0.9 since in 5 out of 50 traces of the synthetic event log *Init(ER Registration)* is violated.