project

January 18, 2025

[3]:

```
import pm4py.objects.conversion.log.converter as log_converter
import pm4py.algo.discovery.alpha.algorithm as alpha_miner
import pm4py.visualization.petri_net.visualizer  as pn_visualizer
import pm4py.algo.analysis.workflow_net.algorithm as wf_net
import pm4py.objects.petri_net.utils as petri_utils
import pm4py.objects.log.importer.xes.importer as xes_importer
import pm4py
import pandas as pd

#Path to the dataset file
event_log_path = "data/BPI_Challenge_2013_incidents.xes"


#Step 1: Import the event log
def import_event_log(file_path):
    print("\n--->Step 1: Importing the event log")
    event_log = xes_importer.apply(file_path)

    return event_log

#Step 2: Discover a Petri net from the log
def discover_petri_net_inductive(event_log):
    print("Discovering Petri net by inductive...")
    net, initial_marking, final_marking = pm4py.
 ↪discover_petri_net_inductive(event_log)
    return net, initial_marking, final_marking

def discover_petri_net_Heuristic(event_log):
    print("Discovering Petri net by Heuristic...")
    net, initial_marking, final_marking = pm4py.discovery.
 ↪discover_petri_net_heuristics(event_log)
    return net, initial_marking, final_marking

def discover_petri_net_alpha(event_log):
    print("Discovering Petri net by Heuristic...")
    net, initial_marking, final_marking = pm4py.discovery.
 ↪discover_petri_net_alpha(event_log)
```

```python
    return net, initial_marking, final_marking

#Step 3: Visualize the Petri net
def visualize_petri_net(net, initial_marking, final_marking):
    print("\n--->Step 3: Visualize the Petri net")
    gviz = pn_visualizer.apply(net, initial_marking, final_marking)
    pn_visualizer.view(gviz)
    print("Petri net visualization complete.")

# Step 4: Split the net in train and test and check the properties of the Petri␣
↪net
def checking_petri_net_properties(net, initial_marking, final_marking,␣
↪event_log_test):
    print("\n--->Step 4: Checking the properties of the Petri net")

    # Check the properties of the Petri net
    print("Number of places:", len(net.places))
    print("Number of transitions:", len(net.transitions))
    print("Number of arcs:", len(net.arcs))
    print("Initial marking:", initial_marking)
    print("Final marking:", final_marking)
    print("The petri net is a workflow net? ", wf_net.apply(net))
    print("Soundness: ",pm4py.analysis.check_soundness(net, initial_marking,␣
↪final_marking)[0])
    #petri_net_invisible_transition = pm4py.analysis.
↪reduce_petri_net_invisibles(net)
    #visualize_petri_net(petri_net_invisible_transition, initial_marking,␣
↪final_marking)
    # print("Maximal decomposition: ",pm4py.analysis.maximal_decomposition(net,␣
↪initial_marking, final_marking))
    print("Precision: ",pm4py.algo.evaluation.precision.algorithm.
↪apply(event_log_test, net, initial_marking, final_marking))
    print("Simplicity: ",pm4py.algo.evaluation.simplicity.algorithm.apply(net))
    print("Replay fitness: ",pm4py.algo.evaluation.replay_fitness.algorithm.
↪apply(event_log_test, net, initial_marking, final_marking))
    print("Generalization: ",pm4py.algo.evaluation.generalization.algorithm.
↪apply(event_log, net, initial_marking, final_marking))
    #df_diagnostics = pm4py.
↪conformance_diagnostics_token_based_replay(event_log, net, initial_marking,␣
↪final_marking, return_diagnostics_dataframe=True)
    #print("Conformance dignostics token based reply: ",df_diagnostics)
    #df_diagnostics.to_csv("data/conformance_diagnostics.csv")
```

1 PROCESS MINING

[ ]:
```python
print("Process Mining with PM4Py: Discovering a Petri net from an event log")

# Import the event log
event_log = import_event_log(event_log_path)
(event_log_train, event_log_test) = pm4py.ml.split_train_test(event_log)

# Discover the Petri net with INDUCTIVE MINER
print("Discovery the Petri net with INDUCTIVE MINER")
net, initial_marking, final_marking =␣
 ↪discover_petri_net_inductive(event_log_train)
# Check the properties of the Petri net
checking_petri_net_properties(net, initial_marking, final_marking,␣
 ↪event_log_test)
# Visualize the Petri net
visualize_petri_net(net, initial_marking, final_marking)

# Discovery the Petri net with HEURISTIC MINER
print("Discovery the Petri net with HEURISTIC MINER")
net, initial_marking, final_marking =␣
 ↪discover_petri_net_Heuristic(event_log_train)
# Check the properties of the Petri net
checking_petri_net_properties(net, initial_marking, final_marking,␣
 ↪event_log_test)
# Visualize the Petri net
visualize_petri_net(net, initial_marking, final_marking)

# Discovery the Petri net with ALPHA MINER
print("Discovery the Petri net with ALPHA MINER")
net, initial_marking, final_marking = discover_petri_net_alpha(event_log_train)
# Check the properties of the Petri net
checking_petri_net_properties(net, initial_marking, final_marking,␣
 ↪event_log_test)
# Visualize the Petri net
visualize_petri_net(net, initial_marking, final_marking)
```

```
Process Mining with PM4Py: Discovering a Petri net from an event log

--->Step 1: Importing the event log
```

```
c:\Users\nikba\Desktop\roba\uni\fm\esame\ProcessMiningPetriNet\.venv\Lib\site-
packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
parsing log, completed traces :: 100%|      | 7554/7554 [00:01<00:00,
4012.79it/s]
```

Discovering Petri net by inductive…

--->Step 4: Checking the properties of the Petri net
Number of places: 17
Number of transitions: 23
Number of arcs: 50
Initial marking: ['source:1']
Final marking: ['sink:1']
The petri net is a workflow net? True
Soundness: True

computing precision with alignments, completed variants :: 100%| | 2798/2798 [00:39<00:00, 71.67it/s]

Precision: 0.5854597319370092
Simplicity: 0.6666666666666666

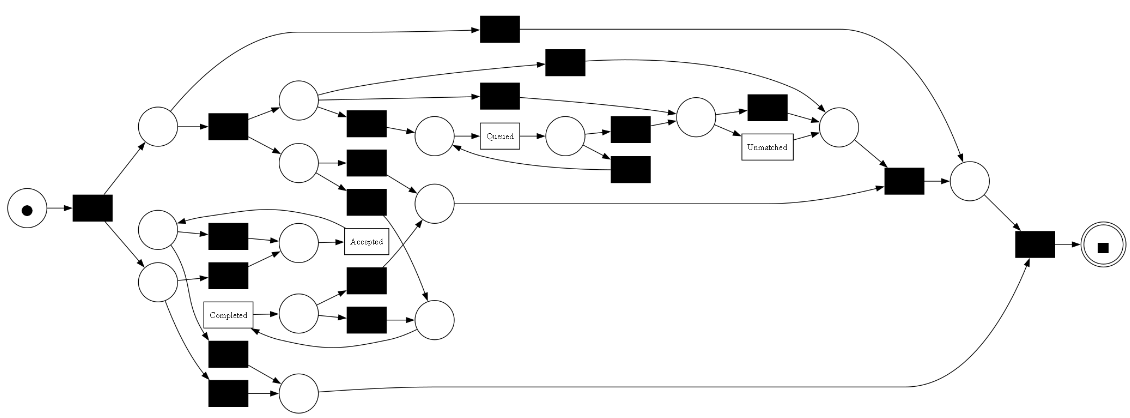
aligning log, completed variants :: 100%| | 414/414 [00:07<00:00, 58.33it/s]

Replay fitness: {'percFitTraces': 100.0, 'averageFitness': 1.0, 'percentage_of_fitting_traces': 100.0, 'average_trace_fitness': 1.0, 'log_fitness': 0.9998104488934022}

replaying log with TBR, completed traces :: 100%| | 1511/1511 [00:01<00:00, 1447.90it/s]

Generalization: 0.871132170779936

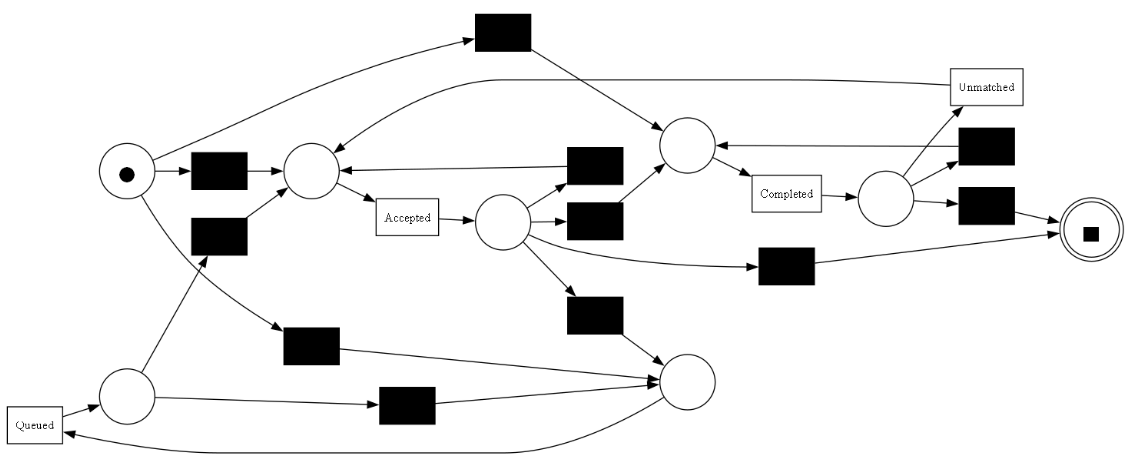
--->Step 3: Visualize the Petri net



Petri net visualization complete.
Discovering Petri net by Heuristic…

--->Step 4: Checking the properties of the Petri net
Number of places: 8
Number of transitions: 15

```
Number of arcs: 30
Initial marking: ['source0:1']
Final marking: ['sink0:1']
The petri net is a workflow net?  True
Soundness:  True

computing precision with alignments, completed variants :: 100%|      |
2798/2798 [00:04<00:00, 624.86it/s]

Precision:  0.8526033344387832
Simplicity:  0.6216216216216216

aligning log, completed variants :: 100%|      | 414/414 [00:01<00:00,
299.80it/s]

Replay fitness:  {'percFitTraces': 92.71523178807946, 'averageFitness':
0.9935108585911195, 'percentage_of_fitting_traces': 92.71523178807946,
'average_trace_fitness': 0.9935108585911195, 'log_fitness': 0.9915833746991571}

replaying log with TBR, completed traces :: 100%|      | 1511/1511
[00:00<00:00, 1812.75it/s]

Generalization:  0.9024161645591694

--->Step 3: Visualize the Petri net
```




```
Petri net visualization complete.
Discovering Petri net by Heuristic…

--->Step 4: Checking the properties of the Petri net
Number of places: 2
Number of transitions: 4
Number of arcs: 5
Initial marking: ['start:1']
Final marking: ['end:1']
```

```
The petri net is a workflow net?  False
Soundness:  False
```

```
computing precision with alignments, completed variants :: 100%|      |
2798/2798 [00:02<00:00, 1264.87it/s]
```

```
Precision:  0.6
Simplicity:  1.0
```

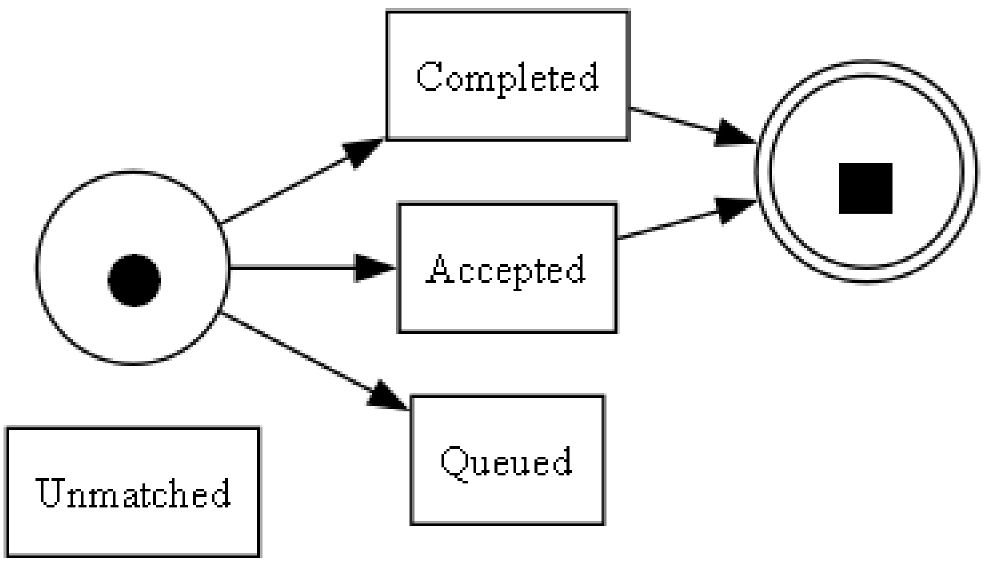
```
aligning log, completed variants :: 100%|      | 414/414 [00:00<00:00,
877.80it/s]
```

```
Replay fitness:  {'percFitTraces': 0.0, 'averageFitness': 0.29569602581011556,
'percentage_of_fitting_traces': 0.0, 'average_trace_fitness':
0.29569602581011556, 'log_fitness': 0.2111553784860558}
```

```
replaying log with TBR, completed traces :: 100%|      | 1511/1511
[00:00<00:00, 2922.11it/s]
```

```
Generalization:  0.8824986163548602
```

```
--->Step 3: Visualize the Petri net
```




```
Petri net visualization complete.
```

2 PRE-PROCESSING

We first print the first 5 events in the dataset

[6]:
```
df = pm4py.convert_to_dataframe(event_log)
print(df.head())
```

```
  org:group resource country organization country org:resource  \
0       V30           France                    fr      Frederic
1       V30           France                    fr      Frederic
2    V5 3rd           France                    fr      Frederic
3    V5 3rd           France                    fr   Anne Claire
4       V30           France                    fr   Anne Claire

  organization involved org:role concept:name  impact  product  \
0           Org line A2     A2_4     Accepted  Medium  PROD582
1           Org line A2     A2_4     Accepted  Medium  PROD582
2           Org line A2     A2_5       Queued  Medium  PROD582
3           Org line A2     A2_5     Accepted  Medium  PROD582
4           Org line A2     A2_4       Queued  Medium  PROD582

  lifecycle:transition            time:timestamp case:concept:name
0          In Progress 2010-03-31 16:59:42+00:00       1-364285768
1          In Progress 2010-03-31 17:00:56+00:00       1-364285768
2  Awaiting Assignment 2010-03-31 17:45:48+00:00       1-364285768
3          In Progress 2010-04-06 16:44:07+00:00       1-364285768
4  Awaiting Assignment 2010-04-06 16:44:38+00:00       1-364285768
```

Then we print some statistical data about the cases, the events, the lifecycle transitions and the resources

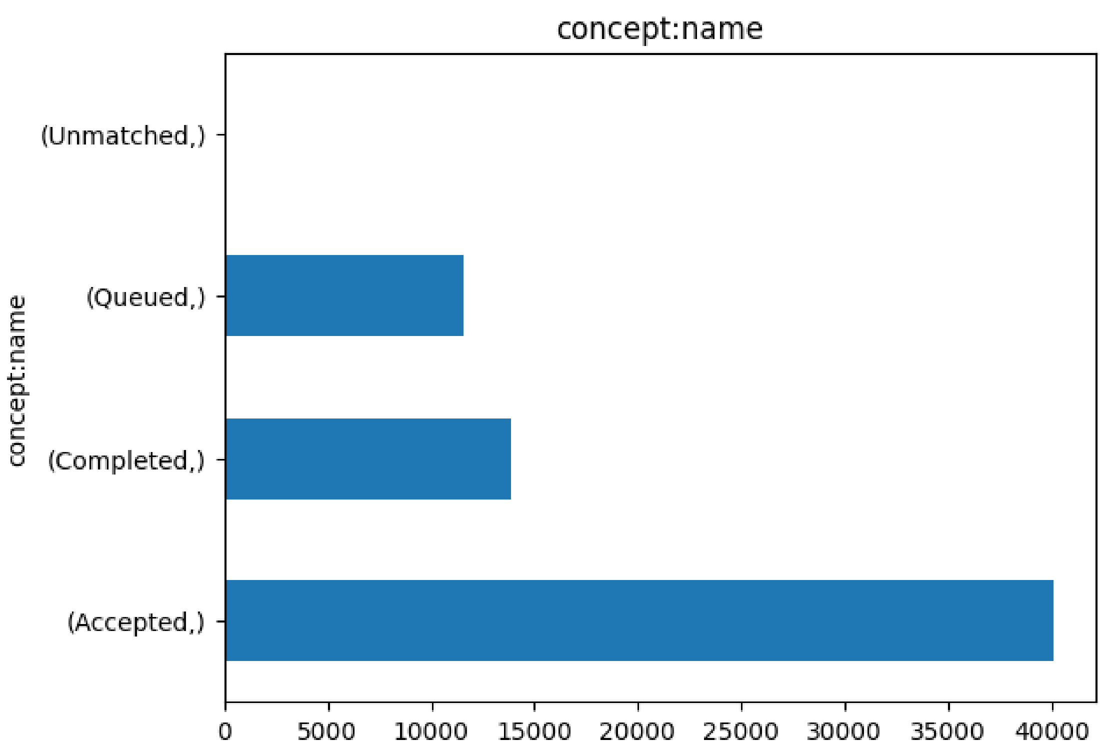
[7]:
```python
df = pm4py.convert_to_dataframe(event_log)
print("Concept:name (event)\n",df["concept:name"].describe())
print("\n\ncase:concept:name (case)\n",df["case:concept:name"].describe())
print("\n\nlifecycle:transition (step of the event)\n",df["lifecycle:
 ↪transition"].describe())
print("\n\nResource\n",df["org:resource"].describe())
```

```
Concept:name (event)
 count        65533
unique           4
top       Accepted
freq         40117
Name: concept:name, dtype: object


case:concept:name (case)
 count           65533
unique           7554
top       1-687082195
freq              123
Name: case:concept:name, dtype: object


lifecycle:transition (step of the event)
```

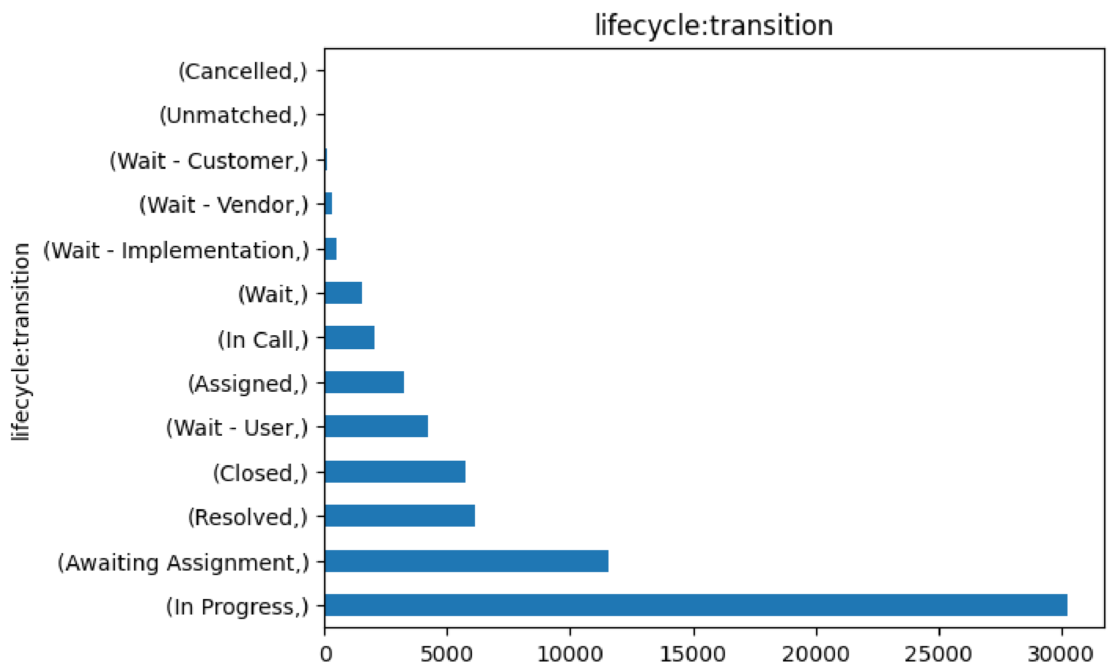
```
 count           65533
unique              13
top       In Progress
freq             30239
Name: lifecycle:transition, dtype: object


Resource
 count      65533
unique      1440
top       Siebel
freq        6162
Name: org:resource, dtype: object
```

Now we plot some data in order to see the event and lifecycle transitions distribution

[8]:
```python
import matplotlib.pyplot as plt

# Value counts: concept:name
df.value_counts(subset=["concept:name"]).plot(y='concept:name', kind='barh',␣
 ↪title='concept:name')
plt.show()
```



[9]:
```python
# Value counts: concept:name
df.value_counts(subset=["lifecycle:transition"]).plot(y='lifecycle:transition',␣
↪kind='barh', title='lifecycle:transition')
plt.show()
```



Now we convert each timestamp (which is absolute) to a relative timestamp.

The obtained timestamps would be relative to the minimum timestamp represented in the dataset

Each timestamp is then converted in seconds, in order to being able to process this data

[10]:
```python
#Transform from absolute time to relative time (timestamps become relative to␣
↪the minimum one present in the dataset)
#This is done to facilitate dicovery process

df['time:timestamp'] = pd.to_datetime(df['time:timestamp'])

df['time:timestamp'] = df['time:timestamp'] - df['time:timestamp'].min()

df["time:timestamp"] = df["time:timestamp"].dt.total_seconds()

print(df["time:timestamp"])
```

```
0               0.0
1              74.0
2            2766.0
3          517465.0
```

```
4          517496.0
             …
65528    66136081.0
65529    66146338.0
65530    66146577.0
65531    66147801.0
65532    66174885.0
Name: time:timestamp, Length: 65533, dtype: float64
```

Then we check the quantity of null/NaN values for each feature of the dataset, in order to remove cases for which there is a null/NaN value among all its columns

[11]:
```python
#check null values
print("Null values:\n",df.isna().sum())
```

```
Null values:
 org:group                    0
resource country            0
organization country        0
org:resource                0
organization involved       0
org:role                 6950
concept:name                0
impact                      0
product                     0
lifecycle:transition        0
time:timestamp              0
case:concept:name           0
dtype: int64
```

Now we delete the cases for which events have org:role equals to null

[12]:
```python
#Delete tuples with null values for org:role
#df = df.dropna(subset=['org:role'])

#get all the cases where org:role is null
cases_to_remove = df[df["org:role"].isna()]["case:concept:name"].unique()

#remove cases where org:role is null
df = df[~df["case:concept:name"].isin(cases_to_remove)]

print("Number of cases after deletion of null values:",df["case:concept:name"].
 ↪unique().size)
```

```
Number of cases after deletion of null values: 6168
```

Now we remove all duplicated events accoring to case:concept:name (case), concept:name (event) and timestamp

[13]:
```python
#Delete duplicates of samples with the same case name, activity and timestamp
df = df.drop_duplicates(subset=['case:concept:name', 'concept:name', 'time:
↪timestamp'])
```

Now we calculate, for each case, its duration as difference between the maximum timestamp of its events and the minimum timestamp

This is done to conduct a successive outliers detection analysis

Then, statistical data about case durations are showed

[14]:
```python
#New we percorm the outliers detection on the activity duration
#for each case (case:concept:name) we calculate how long that case is by␣
↪looking at timestamps of samples with the same case:concept:name
#So we calculate the case duration and put it into a new column "case_duration"
df = df.sort_values(by=['case:concept:name', 'time:timestamp'])

#df['case_duration'] = df.groupby('case:concept:name')['time:timestamp'].
↪diff(-1)

# Calculate the start and end timestamp for each case
case_durations = df.groupby('case:concept:name').agg(
    start_time=('time:timestamp', 'min'),  # First event (min timestamp)
    end_time=('time:timestamp', 'max')     # Last event (max timestamp)
)

# Calculate the duration for each case (end_time - start_time)
case_durations['case_duration'] = case_durations['end_time'] -␣
↪case_durations['start_time']

# Merge the case durations back into the original DataFrame
df = df.merge(case_durations[['case_duration']], on='case:concept:name',␣
↪how='left')

print("Case duration:\n",df['case_duration'].describe())
```

```
Case duration:
 count    4.945000e+04
mean     1.277508e+06
std      2.483723e+06
min      0.000000e+00
25%      4.666180e+05
50%      7.582610e+05
75%      1.496120e+06
max      6.664479e+07
Name: case_duration, dtype: float64
```

Now we conduct outliers detections to detect all the cases with an anomalous duration

To do so we calculate the first and third quartile $Q1, Q3$ among all the case durations. Then we calculate the IQR parameter and we obtain two bounds: - lower bound: $Q1 - 1.5 * IQR$ - upper bound: $Q3 + 1.5 * IQR$

Then all cases of which duration is lower than the lower bound or higher than the upper bound are considered as outliers and so are removed from the dataset

[15]:
```
#then we calculate the percentile Q1, Q3 for the case_duration column, and
 ↪calculate the IQR
Q1 = df['case_duration'].quantile(0.25)
Q3 = df['case_duration'].quantile(0.75)
IQR = Q3 - Q1

print("Quratile 25%:",Q1)
print("Quartile 75%:",Q3)

# Define the extremises 'outlier_lower_bound' and 'Outlier_upper_bound' such
 ↪that an outliers has case_duration lower than 'outlier_lower_bound' or
 ↪higher than 'outlier_upper_bound'
outlier_lower_bound = Q1 - 1.5 * IQR
outlier_upper_bound = Q3 + 1.5 * IQR
print("Lower bound for case duration:",outlier_lower_bound)
print("Upper bound for case duration:",outlier_upper_bound)

#remove whole cases that are identified as outliers accoring to their duration
df_cleaned = df[(df['case_duration'] <= outlier_upper_bound) &
 ↪(df['case_duration'] >= outlier_lower_bound)]

print("New number of samples after outliers elimination:",df_cleaned.shape[0])
```

```
Quratile 25%: 466618.0
Quartile 75%: 1496120.0
Lower bound for case duration: -1077635.0
Upper bound for case duration: 3040373.0
New number of samples after outliers elimination: 46051
```

After the outliers removal, the number of cases remained in the dataset is the following one

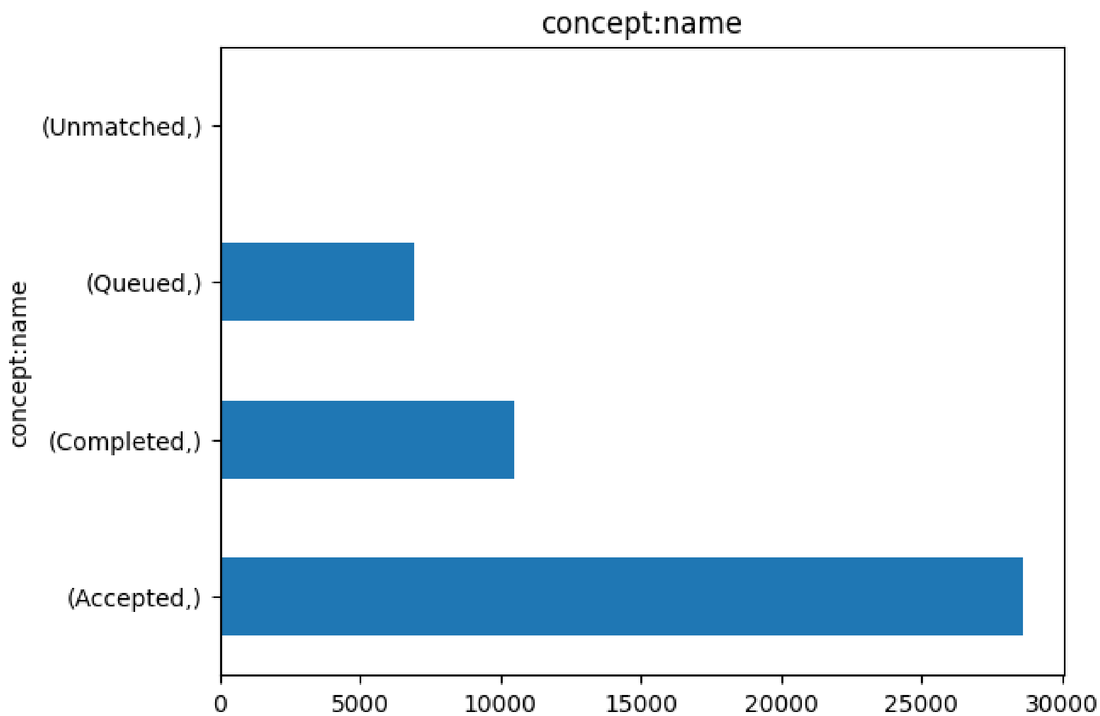
[16]:
```
print("Remained cases:",df_cleaned["case:concept:name"].unique().size)
print("Removed outlier cases:",df["case:concept:name"].unique().
 ↪size-df_cleaned["case:concept:name"].unique().size)
```

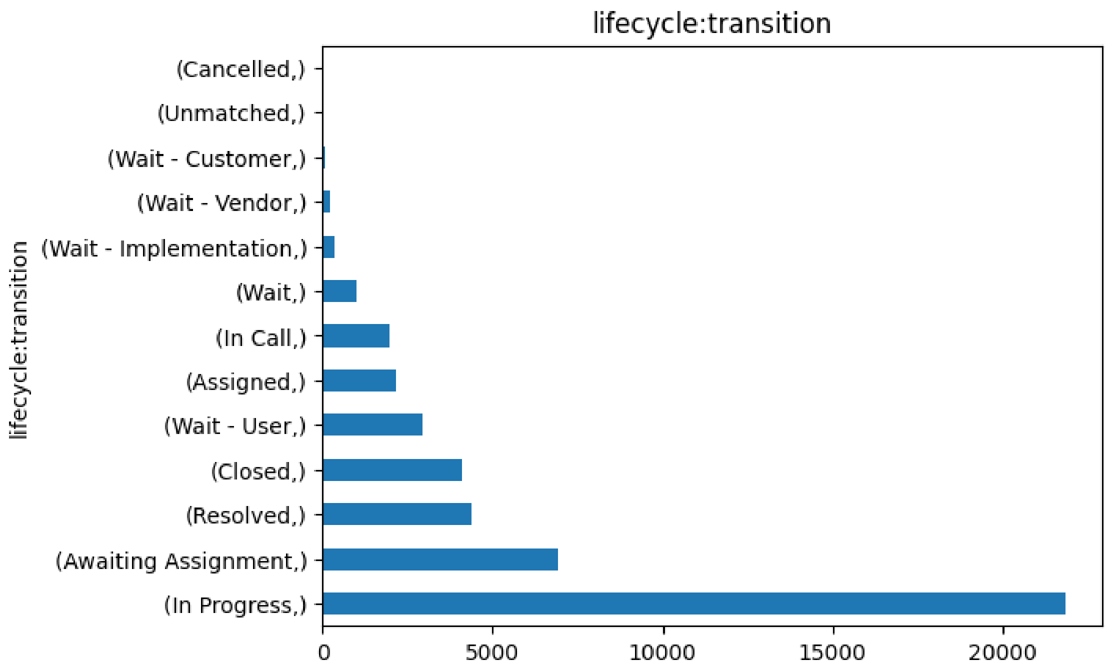
```
Remained cases: 5963
Removed outlier cases: 205
```

Now we print again some value distribution after the whole prepocessing phase

[17]:
```
# Value counts: concept:name
df_cleaned.value_counts(subset=["concept:name"]).plot(y='concept:name',
 ↪kind='barh', title='concept:name')
plt.show()
```

```
# Value counts: concept:name
df_cleaned.value_counts(subset=["lifecycle:transition"]).plot(y='lifecycle:
↪transition', kind='barh', title='lifecycle:transition')
plt.show()
```





Lastly we show statistical data about the dataset after being preprocessed

[18]:
```
print("Attrice descriptions after the preprocessing:")

print("Concept:name (event)\n",df_cleaned["concept:name"].describe())
print("\n\ncase:concept:name (case)\n",df_cleaned["case:concept:name"].
  ↪describe())
print("\n\nlifecycle:transition (step of the event)\n",df_cleaned["lifecycle:
  ↪transition"].describe())
print("\n\nResource\n",df_cleaned["org:resource"].describe())
```

```
Attrice descriptions after the preprocessing:
Concept:name (event)
 count        46051
unique           4
top       Accepted
freq         28627
Name: concept:name, dtype: object


case:concept:name (case)
 count            46051
unique            5963
top       1-722362086
freq                75
Name: case:concept:name, dtype: object
```

```
lifecycle:transition (step of the event)
 count           46051
unique              13
top       In Progress
freq            21840
Name: lifecycle:transition, dtype: object


Resource
 count      46051
unique      1086
top       Siebel
freq        4450
Name: org:resource, dtype: object
```

3 PROCESS MINING AFTER PREPROCESSING

In this phase we conduct again the process mining discovery on the preprocessed data, in order to compare the petri net and its properties with the ones obtained by using non-preprocessed data (original dataset)

[19]:
```
from pm4py.convert import convert_to_event_log

df_cleaned["time:timestamp"] = pd.to_datetime(df_cleaned['time:timestamp'],␣
 ↪unit='s')
preprocessed_event_log = convert_to_event_log(df_cleaned)
```

```
C:\Users\nikba\AppData\Local\Temp\ipykernel_22488\1339938909.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned["time:timestamp"] = pd.to_datetime(df_cleaned['time:timestamp'],
unit='s')
```

[20]:
```
print("Process Mining with PM4Py: Discovering a Petri net from an event log␣
 ↪AFTER PREPROCESSING")

# Import the event log
event_log = preprocessed_event_log
(event_log_train, event_log_test) = pm4py.ml.split_train_test(event_log)

# Discover the Petri net with INDUCTIVE MINER
```

```
print("Discovery the Petri net with INDUCTIVE MINER")
net, initial_marking, final_marking =␣
 ↪discover_petri_net_inductive(event_log_train)
# Check the properties of the Petri net
checking_petri_net_properties(net, initial_marking, final_marking,␣
 ↪event_log_test)
# Visualize the Petri net
visualize_petri_net(net, initial_marking, final_marking)

# Discovery the Petri net with HEURISTIC MINER
print("Discovery the Petri net with HEURISTIC MINER")
net, initial_marking, final_marking =␣
 ↪discover_petri_net_Heuristic(event_log_train)
# Check the properties of the Petri net
checking_petri_net_properties(net, initial_marking, final_marking,␣
 ↪event_log_test)
# Visualize the Petri net
visualize_petri_net(net, initial_marking, final_marking)

# Discovery the Petri net with ALPHA MINER
print("Discovery the Petri net with ALPHA MINER")
net, initial_marking, final_marking = discover_petri_net_alpha(event_log_train)
# Check the properties of the Petri net
checking_petri_net_properties(net, initial_marking, final_marking,␣
 ↪event_log_test)
# Visualize the Petri net
visualize_petri_net(net, initial_marking, final_marking)
```

```
Process Mining with PM4Py: Discovering a Petri net from an event log AFTER
PREPROCESSING
Discovery the Petri net with INDUCTIVE MINER
Discovering Petri net by inductive…

--->Step 4: Checking the properties of the Petri net
Number of places: 17
Number of transitions: 23
Number of arcs: 50
Initial marking: ['source:1']
Final marking: ['sink:1']
The petri net is a workflow net?  True
Soundness:  True
```

```
computing precision with alignments, completed variants :: 100%|      |
1705/1705 [00:20<00:00, 81.92it/s]
```

```
Precision:  0.5688125247272933
Simplicity:  0.6666666666666666
```

```
aligning log, completed variants :: 100%|      | 270/270 [00:04<00:00,
```

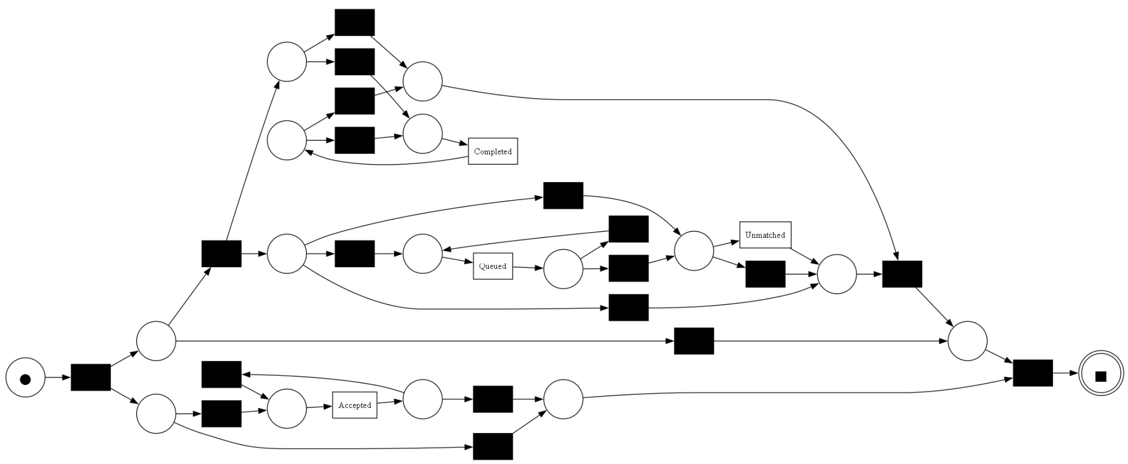
67.40it/s]

Replay fitness: {'percFitTraces': 100.0, 'averageFitness': 1.0, 'percentage_of_fitting_traces': 100.0, 'average_trace_fitness': 1.0, 'log_fitness': 0.9997986320086187}

replaying log with TBR, completed traces :: 100%| | 1018/1018 [00:00<00:00, 1645.19it/s]

Generalization: 0.8512953463140139

--->Step 3: Visualize the Petri net



Petri net visualization complete.
Discovery the Petri net with HEURISTIC MINER
Discovering Petri net by Heuristic…

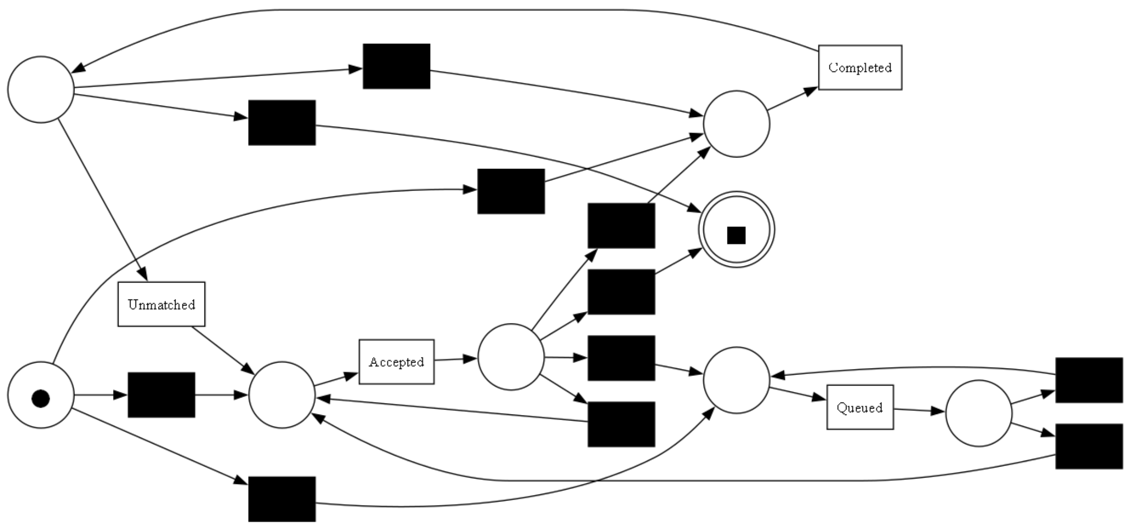
--->Step 4: Checking the properties of the Petri net
Number of places: 8
Number of transitions: 15
Number of arcs: 30
Initial marking: ['source0:1']
Final marking: ['sink0:1']
The petri net is a workflow net? True
Soundness: True

computing precision with alignments, completed variants :: 100%| | 1705/1705 [00:02<00:00, 572.56it/s]

Precision: 0.8193670940724431
Simplicity: 0.6216216216216216

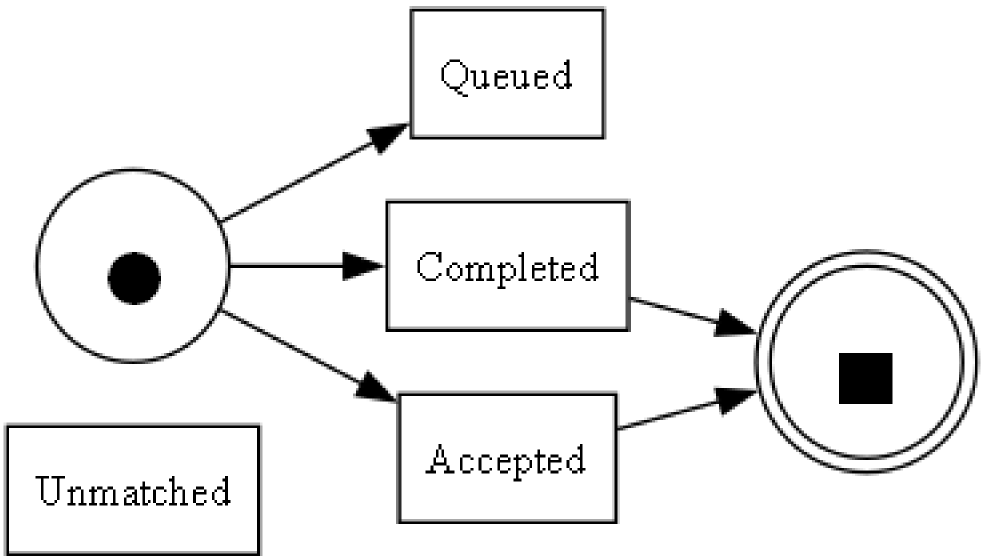
aligning log, completed variants :: 100%| | 270/270 [00:00<00:00, 301.06it/s]

```
Replay fitness:  {'percFitTraces': 95.30201342281879, 'averageFitness':
0.9957313554452376, 'percentage_of_fitting_traces': 95.30201342281879,
'average_trace_fitness': 0.9957313554452376, 'log_fitness': 0.9930283627528207}

replaying log with TBR, completed traces :: 100%|     | 1018/1018
[00:00<00:00, 2001.50it/s]

Generalization:  0.892654462246839

--->Step 3: Visualize the Petri net
```




```
Petri net visualization complete.
Discovery the Petri net with ALPHA MINER
Discovering Petri net by Heuristic…

--->Step 4: Checking the properties of the Petri net
Number of places: 2
Number of transitions: 4
Number of arcs: 5
Initial marking: ['start:1']
Final marking: ['end:1']
The petri net is a workflow net?  False
Soundness:  False

computing precision with alignments, completed variants :: 100%|     |
1705/1705 [00:01<00:00, 1433.06it/s]

Precision:  0.4
Simplicity:  1.0

aligning log, completed variants :: 100%|     | 270/270 [00:00<00:00,
959.21it/s]
```

```
Replay fitness:  {'percFitTraces': 0.0, 'averageFitness': 0.32252999995551146,
'percentage_of_fitting_traces': 0.0, 'average_trace_fitness':
0.32252999995551146, 'log_fitness': 0.23754980079681276}

replaying log with TBR, completed traces :: 100%|      | 1018/1018
[00:00<00:00, 3185.79it/s]

Generalization:  0.8812739580003811

--->Step 3: Visualize the Petri net
```




```
Petri net visualization complete.
```

4 DATA AUGMENTATION WITH GPT

We asked Chat GPT to augment the dataset by using the following prompt:"Augment the dataset given in the provided file, by creating new consistent data, basing on the data that already exists in the dataset"

The response was the following code:

```
[21]: from datetime import datetime, timedelta
import random

# Provided dataset structure
base_data = {
    "org:group": ["V30", "V5 3rd", "V13 2nd 3rd", "S42", "N52 2nd", "O3 3rd",␣
 ↪"G140 2nd", "M25"],
```

```
    "resource": ["Frederic", "Anne Claire", "Eric", "Sarah", "Loic", "Adam",␣
↪"Denny", "Paul", "Åse", "Joseph", "Björn T", "Suliman", "Jon", "Gustav",␣
↪"Magnus"],
    "country": ["France", "Sweden"],
    "organization country": ["fr", "se"],
    "org:resource": ["Frederic", "Anne Claire", "Eric", "Adam", "Denny",␣
↪"Paul", "Åse", "Joseph", "Björn T", "Suliman", "Jon", "Gustav", "Magnus"],
    "organization involved": ["Org line A2", "Org line C", "Org line B3", "Org␣
↪line D"],
    "org:role": ["A2_4", "A2_5", "B3_1", "V3_2", "E_10", "C_6"],
    "concept:name": ["Accepted", "Queued", "Completed", "Unmatched"],
    "impact": ["Medium"],
    "product": ["PROD582", "PROD453", "PROD613"],
    "lifecycle:transition": ["In Progress", "Awaiting Assignment", "Assigned",␣
↪"Resolved", "Closed", "Wait - User"],
    "case:concept:name": ["1-364285768", "2-467153946", "3-582936478",␣
↪"4-731926543"],
}

# Generate synthetic data
def generate_synthetic_data(base_data, num_entries=50):
    new_data = []
    start_time = datetime(2010, 3, 31, 16, 0)  # Example start timestamp

    for i in range(num_entries):
        # Generate random case ID to introduce new cases
        case_id = f"{random.randint(5, 999)}-{random.randint(100000000,␣
↪999999999)}"
        case_duration = random.randint(20000, 100000000)  # Random case duration

        entry = {
            "org:group": random.choice(base_data["org:group"]),
            "resource": random.choice(base_data["resource"]),
            "country": random.choice(base_data["country"]),
            "organization country": "fr" if random.choice(base_data["country"])␣
↪== "France" else "se",
            "org:resource": random.choice(base_data["org:resource"]),
            "organization involved": random.choice(base_data["organization␣
↪involved"]),
            "org:role": random.choice(base_data["org:role"]),
            "concept:name": random.choice(base_data["concept:name"]),
            "impact": base_data["impact"][0],
            "product": random.choice(base_data["product"]),
            "lifecycle:transition": random.choice(base_data["lifecycle:
↪transition"]),
```

```
            "time:timestamp": (start_time + timedelta(seconds=random.randint(0,␣
↪case_duration))).isoformat(),
            "case:concept:name": case_id,
            "case_duration": case_duration,
        }

        new_data.append(entry)

    return pd.DataFrame(new_data)

# Generate 100 synthetic rows
synthetic_data = generate_synthetic_data(base_data, num_entries=100)

synthetic_data['time:timestamp'] = pd.to_datetime(synthetic_data['time:
↪timestamp'])

# Display sample of augmented data
print(synthetic_data.head())
```

```
     org:group     resource country organization country org:resource  \
0       O3 3rd         Adam  Sweden                   se          Jon
1   V13 2nd 3rd      Suliman  Sweden                   se         Eric
2       O3 3rd         Loic  Sweden                   se       Joseph
3     G140 2nd          Åse  Sweden                   se         Adam
4     G140 2nd  Anne Claire  Sweden                   se         Eric

  organization involved org:role concept:name  impact  product  \
0           Org line B3     A2_5       Queued  Medium  PROD582
1           Org line A2     A2_4    Completed  Medium  PROD582
2            Org line C      C_6    Completed  Medium  PROD613
3            Org line D     A2_5     Accepted  Medium  PROD582
4            Org line C     A2_4    Unmatched  Medium  PROD453

  lifecycle:transition      time:timestamp case:concept:name  case_duration
0               Closed 2011-05-22 10:59:00     232-293225442       41294072
1             Resolved 2010-05-17 00:44:12     222-348030515       57817934
2  Awaiting Assignment 2011-02-05 18:29:38     852-743384944       32904769
3          Wait - User 2010-08-28 06:33:13     748-882116532       24299549
4          In Progress 2010-05-02 13:55:05     837-228608952       11046274
```

[22]:
```
df_incremented_GPT = pd.concat([df_cleaned,synthetic_data])
```

[23]:
```
GPT_event_log = convert_to_event_log(df_incremented_GPT)
```

[24]:
```
print("Process Mining with PM4Py: Discovering a Petri net from an event log␣
↪AFTER PREPROCESSING")
```

```
# Import the event log
event_log = GPT_event_log
(event_log_train, event_log_test) = pm4py.ml.split_train_test(event_log)

# Discover the Petri net with INDUCTIVE MINER
print("Discovery the Petri net with INDUCTIVE MINER")
net, initial_marking, final_marking =␣
 ↪discover_petri_net_inductive(event_log_train)
# Check the properties of the Petri net
checking_petri_net_properties(net, initial_marking, final_marking,␣
 ↪event_log_test)
# Visualize the Petri net
visualize_petri_net(net, initial_marking, final_marking)

# Discovery the Petri net with HEURISTIC MINER
print("Discovery the Petri net with HEURISTIC MINER")
net, initial_marking, final_marking =␣
 ↪discover_petri_net_Heuristic(event_log_train)
# Check the properties of the Petri net
checking_petri_net_properties(net, initial_marking, final_marking,␣
 ↪event_log_test)
# Visualize the Petri net
visualize_petri_net(net, initial_marking, final_marking)

# Discovery the Petri net with ALPHA MINER
print("Discovery the Petri net with ALPHA MINER")
net, initial_marking, final_marking = discover_petri_net_alpha(event_log_train)
# Check the properties of the Petri net
checking_petri_net_properties(net, initial_marking, final_marking,␣
 ↪event_log_test)
# Visualize the Petri net
visualize_petri_net(net, initial_marking, final_marking)
```

```
Process Mining with PM4Py: Discovering a Petri net from an event log AFTER
PREPROCESSING
Discovery the Petri net with INDUCTIVE MINER
Discovering Petri net by inductive…

--->Step 4: Checking the properties of the Petri net
Number of places: 17
Number of transitions: 23
Number of arcs: 50
Initial marking: ['source:1']
Final marking: ['sink:1']
The petri net is a workflow net?  True
Soundness:  True
```

computing precision with alignments, completed variants :: 100%| |
1635/1635 [00:17<00:00, 94.79it/s]

Precision: 0.60554777248529
Simplicity: 0.6666666666666666

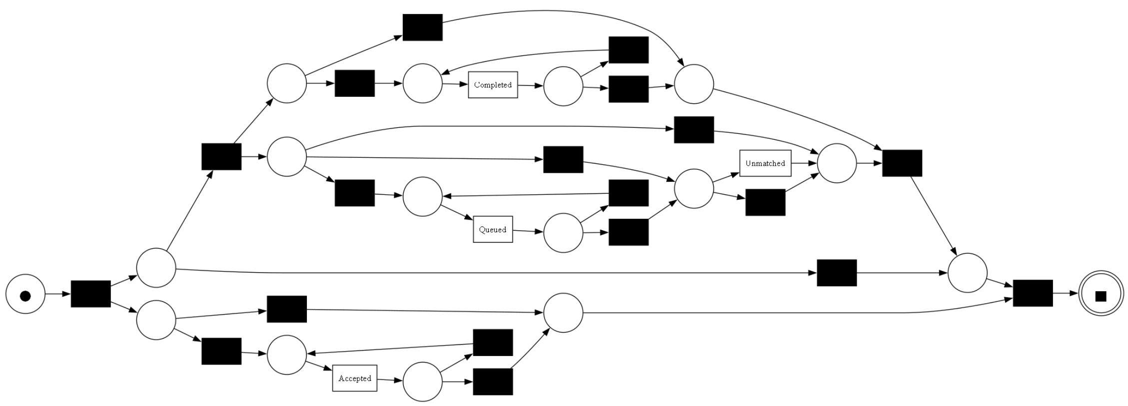
aligning log, completed variants :: 100%| | 283/283 [00:03<00:00,
75.62it/s]

Replay fitness: {'percFitTraces': 100.0, 'averageFitness': 1.0,
'percentage_of_fitting_traces': 100.0, 'average_trace_fitness': 1.0,
'log_fitness': 0.9997982074413613}

replaying log with TBR, completed traces :: 100%| | 1021/1021
[00:00<00:00, 1674.21it/s]

Generalization: 0.9540829825864406

--->Step 3: Visualize the Petri net

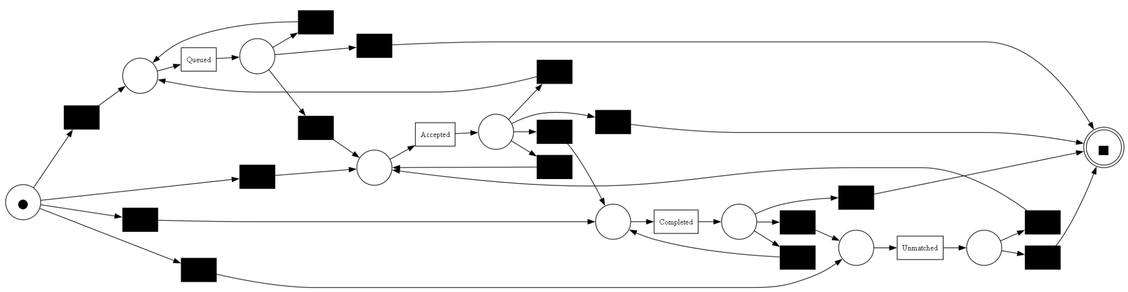


Petri net visualization complete.
Discovery the Petri net with HEURISTIC MINER
Discovering Petri net by Heuristic…

--->Step 4: Checking the properties of the Petri net
Number of places: 10
Number of transitions: 20
Number of arcs: 40
Initial marking: ['source0:1']
Final marking: ['sink0:1']
The petri net is a workflow net? True
Soundness: True

computing precision with alignments, completed variants :: 100%| |
1635/1635 [00:02<00:00, 716.01it/s]

```
Precision:  0.8352765562593295
Simplicity:  0.6000000000000001

aligning log, completed variants :: 100%|     | 283/283 [00:00<00:00,
296.45it/s]

Replay fitness:  {'percFitTraces': 94.55445544554455, 'averageFitness':
0.9948507127287126, 'percentage_of_fitting_traces': 94.55445544554455,
'average_trace_fitness': 0.9948507127287126, 'log_fitness': 0.9926021790345992}

replaying log with TBR, completed traces :: 100%|     | 1021/1021
[00:00<00:00, 1860.96it/s]

Generalization:  0.887822601104677

--->Step 3: Visualize the Petri net
```



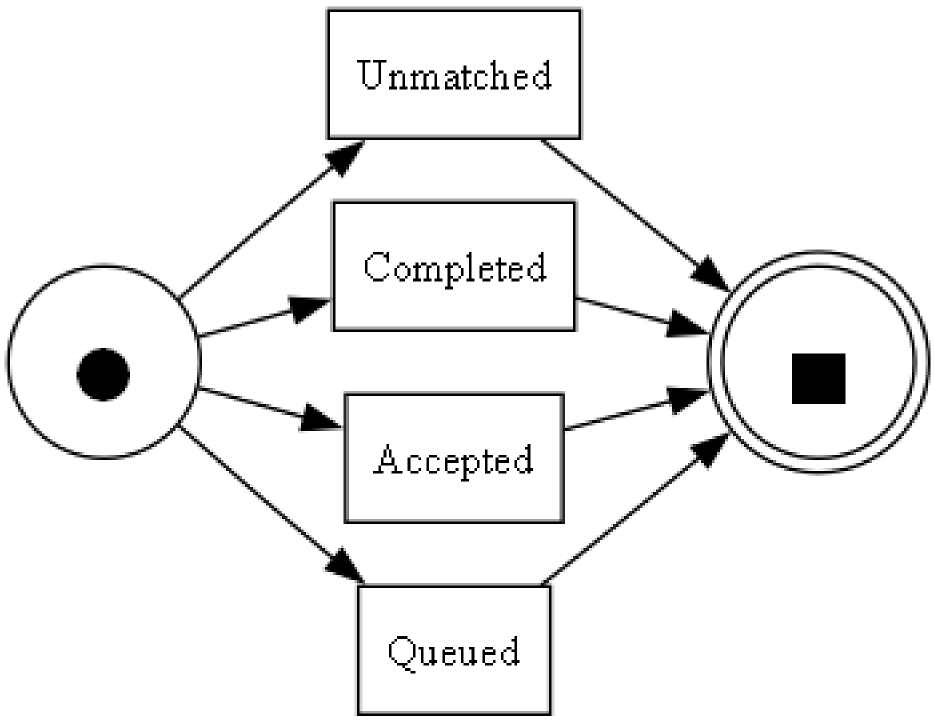

```
Petri net visualization complete.
Discovery the Petri net with ALPHA MINER
Discovering Petri net by Heuristic…

--->Step 4: Checking the properties of the Petri net
Number of places: 2
Number of transitions: 4
Number of arcs: 8
Initial marking: ['start:1']
Final marking: ['end:1']
The petri net is a workflow net?  True
Soundness:  True

computing precision with alignments, completed variants :: 100%|     |
1635/1635 [00:00<00:00, 2676.44it/s]

Precision:  1.0
Simplicity:  0.6000000000000001

aligning log, completed variants :: 100%|     | 283/283 [00:00<00:00,
1073.43it/s]

Replay fitness:  {'percFitTraces': 1.4026402640264026, 'averageFitness':
0.3298421655245669, 'percentage_of_fitting_traces': 1.4026402640264026,
```

'average_trace_fitness': 0.3298421655245669, 'log_fitness': 0.2390768320347174}

replaying log with TBR, completed traces :: 100%| | 1021/1021
[00:00<00:00, 3205.81it/s]

Generalization: 0.948186392941476

--->Step 3: Visualize the Petri net



Petri net visualization complete.