The REG103_IPC Table

Welcome to this notebook where we go through table **REG103_IPC** of PATSTAT Register. Each row of this table has a string containing 0, 1 or more IPC symbols of an application as published in the Bulletin. The individual IPC symbols are separated by semicolons.

In [2]:
```
from epo.tipdata.patstat import PatstatClient
from epo.tipdata.patstat.database.models import REG103_IPC
from sqlalchemy import func
import pandas as pd

# Initialise the PATSTAT client
patstat = PatstatClient(env='PROD')

# Access ORM
db = patstat.orm()
```

ID (Primary Key)

Technical identifier for an application, without business meaning. Its values will not change from one PATSTAT edition to the next.

In [3]:
```
i = db.query(
    REG103_IPC.id
).limit(1000)

df = patstat.df(i)
df
```

Out[3]:

	id
0	125629
1	15908428
2	98944576
3	15192849
4	3003843
...	...
995	14793117
996	12706042
997	19913524
998	22210572
999	21775854

1000 rows × 1 columns

IPC_TEXT

IPC classification of the patent. The text is a string and it can contain up to 500 characters. The default value is an empty string.

The data is not updated on a regular basis. Only due to a request for publication in the EP Bulletin, the IPC data is updated. Up-to-date classification information, like reclassified IPCs, can be retrieved by linking the PATSTAT EP Register data with PATSTAT Global table TLS209_APPLN_IPC via the `appln_id` attribute.

In [14]:
```
ipc_text = db.query(
    REG103_IPC.ipc_text,
    REG103_IPC.id
).filter(
    REG103_IPC.ipc_text != ''
).limit(1000)

ipc_text_df = patstat.df(ipc_text)
ipc_text_df
```

Out[14]:

	ipc_text	id
0	F02D19/06, F02D41/00, F02D41/30, F02D41/40, F0...	22214572
1	H02P1/26, H02P25/08	10747081
2	H03J1/04, H04B1/08	86103756
3	B25J15/04, B25J15/02	18820418
4	C03B19/14, C03B20/00	20214491
...	...	...
995	H04N5/00	2256340
996	G05D1/02, B62D15/02, G01S7/52, G08G1/14	18173828
997	A47K10/16	14746895
998	A61K9/10, A61K47/02	99942429
999	B66F3/02, B60S9/02	8794518

1000 rows × 2 columns

We can see which applications are associated with the highest number of symbols.

In [37]:
```
# Create an empty list to add the number of symbols for each appl
ication id
num_labels = []

# Iterate over the list of symbols (over the rows under "ipc_tex
t")
for label in ipc_text_df['ipc_text']:
    # Convert label in a list: a new element is defined every tim
e a comma is encountered
    sequence = label.split(",")
    # Append the length of list just created
    num_labels.append(len(sequence))

# Create a new column of the dataframes containing the length of
the symbols sequence
ipc_text_df['number_of_labels'] = num_labels
# Sort the dataframe by the number_of_labels attribute in descend
ing order
ipc_text_df = ipc_text_df.sort_values(by='number_of_labels', asce
nding=False)
# Save the top 10 applications
most_labels = ipc_text_df.head(10)
most_labels
```

Out[37]:

	ipc_text	id	number_of_labels
282	C07C381/10, A01N41/10, A01N43/10, A01N43/18, A...	3780722	17
22	B64F5/40, B05B14/30, B05D5/00, B22F3/00, C23C2...	20171967	17
744	C07D413/12, C07D413/06, C07D413/14, C07D401/06...	22832208	15
959	C04B41/53, C04B41/00, C04B41/50, B28B11/00, B2...	19871709	15
128	C09J7/38, B32B7/023, B32B9/00, B32B9/04, B32B2...	23843085	15
488	A61L24/04, C08G18/10, C08G18/32, C08G18/34, C0...	21836582	13
334	A23L33/10, A61K31/045, A61K31/4164, A23L2/52, ...	13803671	13
935	C22C38/02, C22C38/04, C22C38/06, C22C38/42, C2...	16876051	13
480	A61K47/14, A61K39/12, A61K39/13, A61K39/205, A...	13736246	13
162	C07D333/76, C07D409/04, C07D409/06, C07D409/12...	18883494	13

As of now, we have the number of labels associated to each application. In the following, we retrieve the number of occurrences of a certain label in the database.

In [40]:
```
from collections import Counter
import matplotlib.pyplot as plt

# Take the ipc_text column and convert it to a list object
list_of_labels = list(ipc_text_df['ipc_text'].values)

# Count the occurrences of each element
element_counts = Counter(list_of_labels)

# Filter elements with a count greater than or equal to 7
filtered_counts = {k: v for k, v in element_counts.items() if v >
= 2}

# Get the elements and their counts sorted by frequency
elements, counts = zip(*sorted(filtered_counts.items(), key=lambd
a x: x[1], reverse=True))
for element, count in zip(elements, counts):
    print(element, count)
```

```
G06Q10/00 3
H04N7/26, H04N7/50 2
H04W72/04 2
G06F1/00 2
G06Q10/08 2
H05B41/29 2
H05B33/08 2
B41J2/175 2
H04W52/02 2
C12Q1/68 2
G07C9/00 2
G03F1/14 2
B25J9/16 2
```

Finally, we can count the number of empty texts.

In [15]:
```
no_text = db.query(
    func.count(REG103_IPC.id).label('total')
).filter(
    REG103_IPC.ipc_text == ''
)

no_text_df = patstat.df(no_text)
no_text_df['total'].values.item()
```

Out[15]: 18234

CHANGE_DATE

It is the date of when the record was saved in the database.

In [34]:
```
change_date = db.query(
    REG103_IPC.change_date,
    REG103_IPC.id
).limit(100)

change_date_df = patstat.df(change_date)
change_date_df
```

Out[34]:

	change_date	id
0	2001-05-18	125629
1	2018-10-27	15908428
2	2000-11-30	98944576
3	2017-04-07	15192849
4	2003-08-29	3003843
...	...	...
95	1988-04-01	84100882
96	2024-02-02	20425020
97	1997-10-31	96905820
98	2022-01-07	19920950
99	1989-04-08	88310574

100 rows × 2 columns

We can retrieve the applications for which a record was saved in a particular year, let's say 2020.

In [3]:
```
cd = db.query(
    REG103_IPC.change_date,
    REG103_IPC.id
).filter(
    REG103_IPC.change_date > '2019-12-31',
    REG103_IPC.change_date < '2021-01-01'
)

cd_df = patstat.df(cd)
cd_df
```

Out[3]:

	change_date	id
0	2020-02-03	16910142
1	2020-02-24	18703653
2	2020-04-01	17864573
3	2020-03-28	18790751
4	2020-02-05	18708097
...	...	...
372509	2020-12-25	19772287
372510	2020-12-25	19743281
372511	2020-12-25	19740694
372512	2020-12-25	20185617
372513	2020-12-25	19187802

372514 rows × 2 columns

BULLETIN_YEAR

For actions that have been published in the EPO Bulletin, it is the year of the publication in the bulletin. The default value is 0, used for applications that are not published or for which the year is not known. The format is YYYY otherwise.

In [31]:
```
years = db.query(
    REG103_IPC.bulletin_year,
    REG103_IPC.id
).limit(1000)

years_df = patstat.df(years)
years_df
```

Out[31]:

	bulletin_year	id
0	2001	125629
1	0	15908428
2	0	98944576
3	2017	15192849
4	2003	3003843
...	...	...
995	2017	14793117
996	2015	12706042
997	2022	19913524
998	2024	22210572
999	0	21775854

1000 rows × 2 columns

BULLETIN_NR

This is the issue number of the EPO Bulletin for actions that have been published in it. The Bulletin number indicates the calendar week the Bulletin has been published. The default value 0 is used when the attribute `bulletin_year` is 0.

In [33]:
```
bulletin_nr = db.query(
    REG103_IPC.id,
    REG103_IPC.bulletin_nr,
    REG103_IPC.bulletin_year
).limit(100)

bulletin_nr_df = patstat.df(bulletin_nr)
bulletin_nr_df
```

Out[33]:

	id	bulletin_nr	bulletin_year
0	125629	27	2001
1	15908428	0	0
2	98944576	0	0
3	15192849	19	2017
4	3003843	42	2003
...	...	...	...
95	84100882	20	1988
96	20425020	10	2024
97	96905820	51	1997
98	19920950	6	2022
99	88310574	21	1989

100 rows × 3 columns

In [ ]:
```

```