Caso di studio in Artificial Intelligence
Sperimentazione con il dataset Eclipse

1 Studenti

Nome Cognome, matricola, mail
Federico Canistro, 775605, f.canistro@studenti.uniba.it
Ivan De Cosmis, 787066, i.decosmis@studenti.uniba.it

2 Introduzione

In seguito alle sperimentazioni effettuate durante il progetto per l'esame di Formal Methods abbiamo riscontrato diversi problemi durante le operazioni di datamining per creare una **rete di Petri** partendo dai dataset forniti. I dataset forniti contenevano diverse operazioni effettuate da diversi utenti all'interno del programma Eclipse. L'obiettivo era creare un modello con il quale riuscire a identificare se un set di operazioni puo' considerarsi valido.

3 Contesto del lavoro svolto

Una rete di Petri, chiamata anche rete di Petri bipartita o semplicemente Petri net, è un modello matematico utilizzato per descrivere il comportamento di sistemi distribuiti e concorrenti. È stato introdotto da Carl Adam Petri nel 1962 come strumento per analizzare i processi di produzione, ma ha trovato applicazioni in diversi campi, come l'informatica, l'ingegneria del software e la biologia dei sistemi. Una rete di Petri è costituita da due elementi principali: **place** e **transitions**. I place sono rappresentati da cerchi e rappresentano le condizioni o gli stati del sistema, mentre le transition sono rappresentate da rettangoli e rappresentano gli eventi o le azioni che possono verificarsi nel sistema. Le transition sono collegate ai place tramite archi diretti. All'interno della rete di Petri, ci sono dei marcatori che rappresentano l'entità o la risorsa presente in un determinato posto. I marcatori vengono solitamente rappresentati da punti o da un numero all'interno del posto. Nel nostro caso specifico abbiamo escluso i marcatori non avendone la necessità. Il comportamento di una rete di Petri è descritto tramite regole di transizione. Una transition può scattare solo se tutti i suoi place di input contengono un numero sufficiente di marcatori. Quando una transition scatta, i marcatori vengono rimossi dai place di input e vengono aggiunti ai place di output della transition, riflettendo il cambiamento di stato del sistema. Abbiamo deciso di usare lo standard PNML per rappresentare la rete di Petri risultante data l'enorme mole di place e transition. I dataset utilizzati sono stati converiti attraverso l'algoritmo descritto all'intero del file **data_process.py**, dividendoli per utente, per riuscire a lavorare con i dati utilizzando le librerie standard csv in python. La prima riga dei file contiene i nomi delle colonne che sono:

1. *timestamp*
2. *activity*
3. *id_model*
4. *id_exec*
5. *action*
6. *n_action*

Le righe seguenti sono state convertite rispettando lo standard csv per la lettura dei dati.

4 Approccio utilizzato

Per riuscire a leggere i dataset e generare un modello basato su una rete di Petri abbiamo deciso di creare un algoritmo di datamining costruito sul formato di questi

dataset. L'obiettivo era riuscire a leggere riga per riga andando a salvare i dati letti all'interno di oggetti che rappresentano queste righe in place e transition. In seguito è stata creata la rete di Petri basata sui place e sulle transition ricavate dai dataset.

5 Descrizione dell'algoritmo creato

Abbiamo creato diversi oggetti per riuscire a leggere i dati contenuti nei dataset per poterli rappresentare attraverso una rete di petri:

1. **Processes:** rappresenta un processo formato dal place di partenza, dalla transizione e dal place di arrivo (riga 33, main.py).

```
class Processes:
    def __init__(self, input_places,
                 input_transitions):
        self.places = copy.deepcopy(input_places)
        self.transitions =\
        copy.deepcopy(input_transitions)

    def print_places(self):
        for place in self.places:
            place.print_places()

    def get_total_places(self):
        return len(self.places)

    def get_total_transitions(self):
        return len(self.transitions)
```

2. **Place:** Rappresenta un singolo place che contiene due oggetti **activity**, che rappresentano l'inizio con le relative informazioni e la fine con le relative informazioni (riga 58, main.py).

```
class Place:
    def __init__(self, activity_one,
                 activity_two):
        self.activity_one = activity_one
        self.activity_two = activity_two
        self.name = self.activity_one.action

    def print_places(self):
        print("Place: ")
        self.activity_one.print_activity()
        self.activity_two.print_activity()
```

3. **Activity:** Rappresenta una singola activity (una riga del file csv) e contiene *timestamp*, *activity*, *id_model*, *id_exec*, *action*, *n_action* (riga 70, main.py).

```
class Activity:
    def __init__(self, timestamp, activity,
                 id_model, id_exec, action,
                 n_action):
        self.timestamp = timestamp
        self.activity = activity
        self.id_model = id_model
        self.id_exec = id_exec
        self.action = action
        self.n_action = n_action
```

4. **Transition:** Rappresenta la transizione da un place ad uno successivo. Contiene *index_process* (indice del processo corrente), *index_places_from* (indice del place di partenza) e *index_places_two* (indice del place di arrivo) (riga 88,main.py).

```
class Transitions:
    def __init__(self, index_process,
                 index_places_from, index_places_to):
        self.index_process = index_process
        self.index_places_from = index_places_from
        self.index_places_to = index_places_to
```

Questi oggetti vengono salvati in liste globali per la gestione dei dati (riga 100,main.py):

```
processes = []
places = []
activitys = []
transitions = []
```

Per caricare i dataset è stata utilizzata la funzione **get_file_names_in_folder(folder_path)** (riga 14, main.py):

```
def get_file_names_in_folder(folder_path):
    if not os.path.isdir(folder_path):
        print(f"Percorso non valido")
        return []

    file_names = []
    for _, _, files in os.walk(folder_path):
        file_names.extend(files)

    return file_names
```

Questa funzione legge il percorso ricevuto in input e restituisce una lista di nomi di tutti i file contenuti al suo interno.

La funzione **read_data_csv()** permette di leggere i dati dai diversi file e salvarli come oggetti nelle liste corrispondenti (riga 107): Per ogni processo letto vengono salvati le relative activity convertendole in place e transition. Per ogni "begin_of_activity" e "end_of_activity" viene generata un place salvando tutti i dati dell'activity grazie all'oggetto "Place". Ogni place viene collegato al successivo attraveso una transition che viene nominata con il nome dell'azione dell'activity.

Esempio:

```
2020-01-15 10:13:34.655,begin_of_activity,
idmodel_703453finale,idexec_7034532020011510133 4636,
act_ShellBoundsCommand,1
2020-01-15 10:13:34.655,end_of_activity,
idmodel_703453finale,idexec_7034532020011510133 4636,
act_ShellBoundsCommand,1
```

Possiamo identificarlo come place P1, mentre

```
2020-01-15 10:18:32.455,begin_of_activity,
idmodel_703453finale,idexec_7034532020011510133 4636,
act_ShellBoundsCommand,2
2020-01-15 10:18:32.455,end_of_activity,
idmodel_703453finale,idexec_7034532020011510133 4636,
act_ShellBoundsCommand,2
```

Possiamo identificarlo come place P2.

P1 è collegato a P2 attraverso la transizione "act_ShellBoundsCommand" che è l'azione eseguita dalla prima activity per passare alla seconda

Queste operazioni vengono effettuate nella funzione "convert_data_to_petri()"

6 Descrizione approfondita della funzione read_data_csv()

La funzione recupera tutti i file che deve leggere attraverso "get_file_names_in_folder", descritta in precendenza (riga 113, main.py).

```
folder_path = 'dataset/'
file_names = get_file_names_in_folder(folder_path)
```

Ogni file viene aperto attraverso un ciclo for sulla lista "file_name" (riga 116, main.py):

```
for file in file_names:
    print("faccio file:" + file)
    with open(folder_path + file) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        line_count = 0
        index_process = 0
```

In seguito inizia l'analisi di ogni riga contenuta nel file (la prima riga viene saltata perché contiene i nomi delle diverse colonne) (riga 122, main.py):

```
for index, row in enumerate(csv_reader):
    if line_count == 0:
        line_count += 1
```

In seguito vengono analizzati diversi casi:

1. **Caso in cui un processo finisce**

   Nel caso in cui ci troviamo avanti alla fine di un processo, tutti i place e le transition letti vengono utilizzate per creare un nuovo oggetto "Processes" che li conterrá, in seguito resetto le liste utilizzate:

```
if row[1] == "end_of_process":
    processes.append(Processes(copy.deepcopy(places),
                               copy.deepcopy(transitions)))
    index_process += 1
    places = []
    transitions = []
```

2. **Caso in cui un'activity finisce ed é stata salvata un'altra activity in precedenza**

   In questo caso andiamo a creare la transizione tra le due activity (che rappresentano due place), in seguito l'activity viene salvata nella lista dei place. La lista activitys viene resettata (riga 132, main.py).

```
if row[1] == "end_of_activity" and len(places) > 1:
    transitions.append(Transitions(index_process,
                                   len(places)-2, len(places)-1))
    activity = Activity(row[0], row[1], row[2],
                        row[3], row[4], row[5])
    activitys.append(copy.deepcopy(activity))
    places.append(Place(activitys[0], activitys[1]))
    activitys = []
```

3. **Caso in cui un'activity finisce**

   In questo caso andiamo a salvare l'activity come un place all'interno della lista e resettiamo la lista activitys (riga 138, main.py).

```
elif row[1] == "end_of_activity":
    activity = Activity(row[0], row[1],
                        row[2], row[3], row[4], row[5])
    activitys.append(copy.deepcopy(activity))
    places.append(Place(activitys[0], activitys[1]))
    activitys = []
```

4. **Caso in cui un'activity inizia**

   In questo caso andiamo a salvare l'inizio di una activity. In questo dataset viene trovato un solo caso in cui due activity vengono lanciate in parallelo, quindi dobbiamo curare anche questo caso. Dato che un place viene salvato ogni due oggetti activity (che rappresentano la riga di inizio e la riga di fine), dobbiamo verificare quando leggiamo "begin_of_activity" che non sia salvato nessun activity, perché in quel caso significa che questo caso abbiamo trovato due "begin_of_activity" di fila.

```
elif row[1] == "begin_of_activity":
        if len(activitys) > 0:
        print("faccio parallelo")
        activity_two = Activity(row[0], row[1], row[2],
                                row[3], row[4], row[5])
        activitys_two.append(copy.deepcopy(activity_two))
        next_item = next(csv_reader)
```

```
        activity_two = Activity(next_item[0], next_item[1],
                                next_item[2], next_item[3],
                                next_item[4], next_item[5])
        index_process += 1
        activitys_two.append(copy.deepcopy(activity_two))
        places.append(Place(activitys_two[0],
                            activitys_two[1]))
        transitions.append(Transitions(index_process,
                           len(places) - 2,
                           len(places) - 1))
        activitys_two = []
        next_item = next(csv_reader)
        activity = Activity(next_item[0], next_item[1],
                            next_item[2], next_item[3],
                            next_item[4], next_item[5])
        activitys.append(copy.deepcopy(activity))
        index_process += 1
        places.append(Place(activitys[0], activitys[1]))
        transitions.append(Transitions(index_process,
                           len(places) - 3,
                           len(places) - 1))
        activitys = []
    else:
        activity = Activity(row[0], row[1], row[2],
                            row[3], row[4], row[5])
        activitys.append(copy.deepcopy(activity))
```

In particolare andiamo a creare un secondo oggetto activity che verrá salvato in parallelo al precedente. Grazie all'iterator andiamo ad analizzare la successiva riga che sará un "end_of_activity" che sará usata per finire l'oggetto place della seconda activity, e la relativa transition Successivamente utilizzeremo di nuovo l'iteratore per leggere la riga successiva, ovvero il secondo "end_of_activity" che verrá usato per creare l'altro place e la relativa transition. Nel caso in cui non troviamo parallelelismo andiamo a creare la nuova activity. Alla fine di questo procedimento avremo le liste "process" e "transitions" che conterranno tutti i place necessari alla creazione della rete di Petri.

7 Descrizione approfondita della funzione convert_data_to_petri

Questa funzione viene utilizzata per ottenere la rete di Petri in formato pnml dai dati estratti. Come prima cosa creo l'oggetto che verrá salvato come file pnml

```
pnml = ET.Element("pnml")
net = ET.SubElement(pnml, "net")
z = 0
```

In seguito per prima cosa creo due liste che conterranno i place e le transition. Ogni place nella posizione x della lista dei place deve essere collegato alla transizione con il nome nella posizione x della lista delle transizioni, e nel caso in cui non é il place di partenza dobbiamo collegare il place anche alla transizione con il nome nella posizione x-1 (riga 189).

```
for proc in processes:
    print("ripetizione" + str(z))
    p_net_places = []
    p_net_transitions = []
    for i in range(0, len(proc.places)):
        if i == len(proc.places):
            break
        else:
            p_net_places.append(proc.places[i].
                                .activity_one.id_model
                                + "_" + proc.places[i].
                                .activity_one.id_exec
                                + "_" +
                                proc.places[i].name
                                + "_" +
                                proc.places[i].
                                .activity_one.n_action)
            p_net_transitions.append(proc.places[i].name)
            print(p_net_transitions[-1])
```

Infine andiamo a creare i veri e propri place e transition nell'oggetto del file, collegandoli con archi:

```
for i in range(0, len(p_net_places)):
    place1 = ET.SubElement(net, "place", id=p_net_places[i])
    if not does_transition_exist(net, p_net_transitions[i]):
        t_1 = ET.SubElement(net, "transition",
                            id=p_net_transitions[i])
    arc1 = ET.SubElement(net, "arc", id=p_net_transitions[i]+
                                       p_net_transitions[i],
                         source=p_net_places[i],
                         target=p_net_transitions[i])
    if i > 0:
        if p_net_places[i] == "idmodel_699333finale_" \
                              "idexec_69933320200411151148239" \
                              "_act_ecfilesave_36":
            arc1 = ET.SubElement(net, "arc",
                                 id=p_net_transitions[i - 1] +
                                 p_net_places[i],
                                 source=p_net_transitions[i - 2],
                                 target=p_net_places[i])
        else:
            if p_net_places[i] == "idmodel_699333finale_" \
                                  "idexec_" \
                                  "69933320200411151148239" \
                                  "_act_MoveCaretCommand_224":
                arc1 = ET.SubElement(net, "arc",
                                     id=p_net_transitions[i - 1]
                                     + p_net_places[i],
                                     source=
                                     p_net_transitions[i - 1],
                                     target=p_net_places[i])
                arc1 = ET.SubElement(net, "arc",
                                     id=p_net_transitions[i - 1]
                                     + p_net_places[i],
                                     source=
                                     p_net_transitions[i - 2],
                                     target=p_net_places[i])
            else:
                arc1 = ET.SubElement(net, "arc",
                                     id=p_net_transitions[i - 1]
                                     + p_net_places[i],
                                     source=
                                     p_net_transitions[i - 1],
                                     target=p_net_places[i])
```

Troviamo delle condizioni specifiche utilizzate per rilevare l'activity nel dataset che viene eseguita in modo paralello. In questo modo riusciamo a riconoscerla e creare le giuste transizioni tra i place in questione.

Come ultima cosa creiamo il place finale e lo colleghiamo con un nuovo arco. Andiamo a creare il fi-

le, prima in formato xml per poterlo formattare e in seguito viene cambiata l'estensione in pnml (riga 223, main.py).

```
place1 = ET.SubElement(net, "place", id="end")
arc1 = ET.SubElement(net, "arc", id=p_net_transitions[i - 1]
                     + p_net_places[i],
                     source=p_net_transitions[-1],
                     target="end")
tree = ET.ElementTree(pnml)
ET.indent(tree)
tree.write("petri_net.xml ", encoding="utf-8",
           xml_declaration=True)
for filename in glob.iglob(os.path.join("", '*.xml')):
    os.rename(filename, filename[:-4] + '.pnml')
print("finito")
```

8 Conclusioni

La rete ricavata dal dataset grazie a questo algoritmo é in formato pnml.
Il file é stato formattato per consentire una piú facile lettura del risultato.
Il file risultante si trova nella stessa directory del file "main.py" e si chiama "petri_net.pnml"
Prima di lanciare il file "main.py" ricordarsi di cancellare il file "petri_net.pnml" É possibile scaricare questo documento insieme a tutti i file del progetto dalla repository GitHub:

`https://github.com/i-decosmis/AIProject`

9 Considerazioni e sviluppi futuri

Nonostante questo algoritmo riesca a leggere i dati dal dataset fornito in input, riesce a leggere e convertire solamente il dataset in analisi. É possibile pensare a una nuova versione dell'algoritmo che preso in input un dataset WoMan riesca a riconoscere qualsiasi parallelismo e convertirlo nel modo corretto.