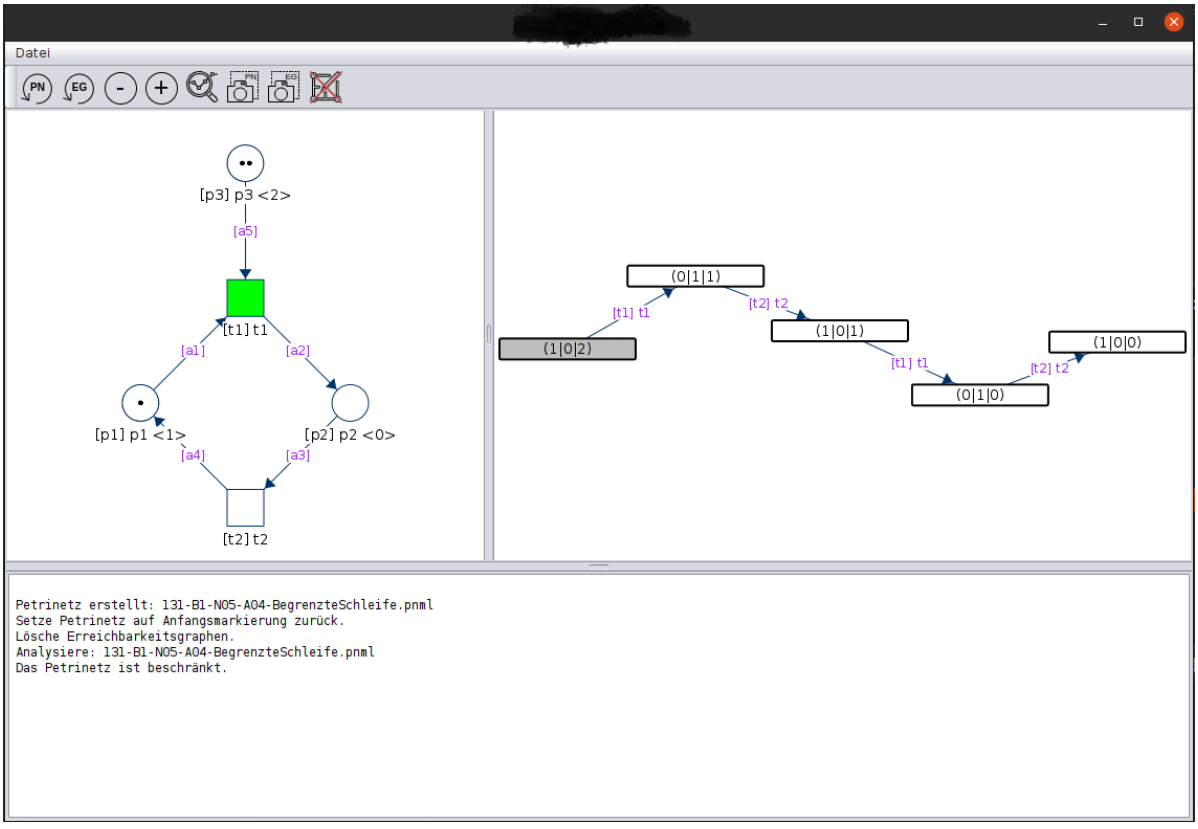
Programmierpraktikum Wintersemester 2020/21 *

Einführung in das Programm

Petrinets



*im Lehrgebiet Softwaretechnik und Theorie der Programmierung

Inhaltsverzeichnis

1 Einleitung

Im Rahmen des Programmierpraktikums habe ich an Hand der umfangreichen Aufgabenstellung ein Programm geschrieben, das über eine grafische Oberfläche Petrinetze und ihre (partiellen) Erreichbarkeitsgraphen aus eingelesenen PNML-Dateien erstellt. Sie lassen sich manipulieren, auf Beschränktheit überprüfen und händisch schalten.

Für die in der Aufgabenstellung geforderten Buttons der Toolbox habe ich folgende Icons[1] verwendet:

Setzt das Petrinetz auf seine Anfangsmarkierung zurück.

Löscht den (partiellen) Erreichbarkeitsgraphen und setzt
dabei auch das Petrinetz wieder auf die Anfangsmarkeirung zurück.

Erhöht die Anzahl der Marken an der ausgewählten Stelle um eins.

Verringert die Anzahl der Marken an der ausgewählten Stelle um eins.

Analysiert das aktuelle Petrinetz auf Beschränktheit.

[1] Das Icon zur Analyse des PN stammt von Flaticon.com

2 Zusätzliche Funktionalitäten

Zusätzlich zu den Anforderungen aus der Aufgabenstellung besitzt mein Programm noch die Möglichkeit, das AutoLayout für die Darstellung des Erreichbarkeitsgraphen bei Bedarf aus- und wieder einzuschalten. Außerdem lassen sich einfach Screenshots des aktuellen Petrinetzes oder des aktuellen (ggf. partiellen) Erreichbarkeitsgraphen erstellen. Diese Funktionalitäten werden über die folgende Buttons[2] in der Toolbox bereitgestellt:

2.1 AutoLayout des EG aus- und einschalten

Das AutoLayout für den Erreichbarkeitsgraphen ist eingeschaltet. Ein Klick auf den Button schaltet es aus. Die Einzelnen Knoten lassen sich dann frei verschieben. Der Button verändert sein Aussehen zu dem Folgenden.

Das AutoLayout für den Erreichbarkeitsgraphen ist ausgeschaltet.
Ein Klick auf den Button schaltet es wieder ein.
Der Button verändert sein Aussehen zu dem Vorherigen.

Das Umschalten des AutoLayouts erfolgt direkt in der Klasse des Hauptfensters. Bei Klick auf den entsprechenden Button wird die Funktion autoLayoutUmschalten(JButton button) aufgerufen. Diese überprüft den Status des AutoLayouts und ändert diesen mittels eines Aufrufs von enable-/disableAutoLayout auf das Viewer-Objekt des Erreichbarkeitsgraphen. Zusätzlich wird auch das Icon des Buttons ensprechend geändert.

2.2 Screenshots von PN und EG speichern

Öffnet einen Speichern-Dialog und speichert das aktuelle Petrinetz in dem ausgewählten Pfad als png-Datei.

Öffnet einen Speichern-Dialog und speichert den aktuellen EG in dem ausgewählten Pfad als png-Datei.

Auch die Umsetzung der Screenshot-Funktion findet komplett in der Hauptfenster-Klasse statt, da es sich hier um eine FUnktionalität handelt, die ausschließlich mit der View zu tun hat. Bei Klicken des jeweiligen Buttons wird die Funktion sreenshot(JPanel panel) aufgerufen und ihr das zu speichernde Panel übergeben. Das Bild wird mit der JPanel-Methode paintAll() erstellt und der Pfad zum Speichern wird mittels eines JFileChoosers ausgewählt. Die Screenshots werden im png-Format gespeichert.

[2] Die Icons stammen von Flaticon.com

2.3 Tastatur-ShortCuts

Alle Funktionen meines Programms lassen sich mittels Tastatur-ShortCuts aufrufen, um Zeit zu sparen und den Workflow reibungsloser zu gestalten. Hier ist eine Übersicht über die Tastenkombinationen und ihren zugeordneten Funktionen:

Strg-O	Datei –> Öffnen...
Strg-N	Datei –> Neu laden
Strg-A	Datei –> Analyse mehrer Dateien...
Strg-Q	Datei –> Beenden(Quit)

Alt-R	Petrinetz auf aktuelle Anfangsmarkierung zurücksetzen
Alt-L	(partiellen) Erreichbarkeitsgraph löschen
Alt-M	Markenzahl an ausgewählter Stelle minus eins
Alt-P	Markenzahl an ausgewählter Stelle plus eins
Alt-A	Beschränktheitsanalyse des aktuellen Petrinetzes
Alt-T	Schaltet das AutoLayout ein/aus (Toggle)

3 Die Programmstruktur

Das Projekt ist nach dem MVC-Prinzip aufgebaut und in die entsprechenden Packages aufgeteilt.

Das Model unterteilt sich in die Klassen, die zur Petrinetz-Datenstruktur gehören und die Klassen, die zur Erreichbarkeitsgraph-Datenstruktur gehören.

Das Petrinetz besteht aus den Klassen Stelle und Transition, welche ihre gemeinsamen Attribute und Methoden von der abstrakten Klasse PNKnoten erben, sowie der Klasse Kante. In der Klasse Petrinetz laufen alle Objekte dieser Klassen zusammen und werden dort gebündelt. Die Petrinetz-Klasse ist als Unterklasse von Observable implementiert, um mögliche Observer aus der View direkt über Änderungen zu informieren. Auch der PNMLParser befindet sich hier, da er vom Petrinetz selbst erzeugt wird und somit weniger Methoden public sein müssen, um über den Parser das Petrinetz mit Daten zu füllen.

Der Erreichbarkeitsgraph besteht aus den Klassen Knoten und Uebergang, welche in der Klasse Erreichbarkeitsgraph zusammenlaufen und gebündelt werden. Da zu jedem Petrinetz ein Erreichbarkeitsgraph gehört, wird dieser direkt beim Erzeugen eines Petrinetz-Objektes ebenfalls erzeugt. Wird im Petrinetz eine Transition geschalten, wird der Erreichbarkeitsgraph bei Bedarf direkt erweitert. Wie das Petrinetz, erbt auch der EG von Observable, um sein View-Äquivalent über Änderungen zu informieren.

In der View befindet sich das Hauptfenster sowie die beiden Graph-Klassen für PN und EG, die jeweils von der Graphstream-Klasse Multigraph erben und die Klasse Observer implementieren. Aktivitäten in dem Hauptfenster werden an den Controller weitergeleitet, über den das PN oder der EG manipuliert werden. Bei Änderungen im PN oder im EG holen sich die Graph-Klassen als Observer ihre Daten selbstständig aus dem Model.

Der Controller wird mit dem Öffnen des Hauptfensters erstellt. Er erzeugt je ein Objekt der View-Klassen PNGraph und EGGraph und stellt diese dem Hauptfenster zur Verfügung. Beim Öffnen einer Datei über das Hauptfenster erstellt der Controller ein neues Petrinetz-Objekt und übergibt diesem die geladene PNML-Datei. Interaktionen mit dem Hauptfenster werden mittels ActionListener bzw ClickListener (für die Graph-Views) an den Controller übergeben, welcher diese wiederum an die entsprechenden Klassen zur Ausführung weiterleitet.

4 Der Beschränktheitsalgorithmus

Die Beschränktheit wird in Form einer Tiefensuche durch Schaltung der nächsten aktivierten Transition und rekursive Aufrufe von **analysiereNetz(pn)** überprüft - wobei pn das Petrinetz unter der aktuellen Markierung ist und die Funktion immer nach dem Schalten aufgerufen wird. Die (noch nicht geschalteten) aktivierten Transitionen sind für jede Rekursionsebene in einem **Stack aktiveTransitionen** gespeichert, von dem aus sie abgearbeitet werden.

Wenn in einer Rekursionsebene alle aktivierten Transitionen einmal geschaltet (und damit analysiert) wurden, ist die Funktion in dieser Ebene beendet und es wird mit der vorhergehenden Ebene weitergemacht.

Bei jedem Aufruf der Funktion wird mittels der Funktion **pfadOK(aktuelleMarkierung)** überprüft, ob auf dem Weg vom Startknoten zur aktuellen Markierung kein Knotenpaar das Unbeschränktheitskriterium erfüllt. Um das zu realisieren, gibt es einen rekursionsbenenübergreifenden **Stack markierungen**, auf den die jeweils aktuelle Markierung zu Beginn der Funktion gepusht wird. Liefert pfadOK(aktuelleMarkierung) false zurück, ist das Petrinetz unbeschränkt.

```
Stack markierungen = new Stack();
```

rekursionsebenenübergreifender Stack

```
analysiereNetz(Petrinetz pn) {
  Stack aktiveTransitionen = new Stack();
  aktuelleMarkierung = pn.getMarkierung();

  beschraenkt = pfadOK(aktuelleMarkierung);
  markierungen.push(aktuelleMarkierung);

  // nur weitermachen, wenn noch beschränkt
  if (beschraenkt) {

    // aktivierte Transitionen auf den Stack packen
    for(transition in pn.getAktivierteTransitionen()
      aktiveTransitionen.push(transition);

    // Transitionen schalten und Analyse rekursiv fortfahren
    while (!aktiveTransitionen.empty()) {
      pn.schalten(aktiveTransitionen.pop());
      analysiereNetz(pn);

      // wenn unbeschränkt, mit false beenden
      if (!beschraenkt)
        return false;

      // sonst Petrinetz und Pfad zurücksetzen auf diese
         Rekursionsebene
      pn.resetTo(aktuelleMarkierung);
      while (!markierungen.peek().equals(aktuelleMarkierung))
        markierungen.pop();
    }
  }
  return true;
}
```

der eigentliche Algorithmus

Die Funktion pfadOK(aktuelleMarkierung) überprüft den aktuellen Pfad an Markierungen auf das Unbeschränktheitskriterium. Dafür erstellt sie sich eine Kopie des markierungen-Stacks, auf welcher sie arbeitet. Dieser **Stack markierungenKopie** wird Markierung für Markierung rückwärts durchlaufen und jedes Element mit der aktuellen Markierung an die Funkion erfuellUnbeschraenktKriterium(aktuelleMarkierung,

markierung) übergeben. Wenn diese Funktion true zurück gibt, sind die beiden Markierungen unsere m und m' auf dem unbeschränkten Pfad.

```
pfadOK(String aktuelleMarkierung) {
  Stack markierungenKopie = new Stack();
  markierungenKopie.addAll(markierungen);

  while (!markierungenKopie.empty()) {
    markierung = markierungenKopie.pop();
    if (erfuelltUnbeschraenktKriterium(aktuelleMarkierung,
       markierung)) {
      // Pfad nicht OK
      return false;
    }
  }
  // sonst Pfad OK
  return true;
}
```

die Pfadüberprüfung für jede Rekursionsbene

Die Funktion erfuelltUnbeschraenktKriterium(m1, m2) erstellt sich aus zwei Strings zwei Char-Arrays und iteriert über deren Inhalt. Wenn m1 an einer Stelle kleiner als m2 ist, gibt sie false zurück - die beiden Markierungen weisen nicht auf Unbeschränktheit hin. Wenn m1 an jeder Stelle größer oder gleich m2 ist, erfüllen die beiden Markierung das Unbeschränktheitskriterium. Eine komplette Gleichheit ist ausgeschlossen, da es bei Kreisen nicht zu einem Aufruf dieser Funktion kommt.

```
private Boolean erfuelltUnbeschraenktKriterium(String m1,
   String m2) {
  char[] m1Chars = m1.toCharArray();
  char[] m2Chars = m2.toCharArray();

  // Gibt false zurück, wenn m1 an einer Stelle kleiner als m2
     ist
  for (int i = 0; i < s1Chars.length; i++)
    if (m1Chars[i] < m2Chars[i])
      return false;
  // gibt true zurück, wenn m1 an jeder Stelle größer oder
     gleich m2 ist
  return true;
}
```

der Vergleich zweier Markierungen um (Un-)Beschränktheit festzustellen