FernUniversität in Hagen

Fakultät für **Mathematik und Informatik**

B. Sc. Informatik

Grundpraktikum Programmierung, WiSe 2020/21

Lehrgebiet Softwaretechnik und Theorie der Programmierung

Petrinetze: Dokumentation

Vorgelegt von: **Hannes Wilms**

Matr.-Nr. **9739335**

Osnabrück, im Dezember 2020

Inhaltsverzeichnis

1 Einleitung

Ein Petrinetz ist ein mathematisches Modellierungsmodell zur Beschreibung von verteilten Systemen. Ein Petrinetz ist ein gerichteter bipartiter Graph, in dem es zwei Typen von Elementen, Stellen und Transitionen, gibt. Diese können miteinander verbunden sein. Dabei gilt dieVoraussetzung, dass auf eine Stelle nur Transitionen folgen dürfen und auf eine Transition nur Stellen folgen dürfen. Eine Transition kann eine, mehrere oder keine Stellen im Vorbereich und im Nachbereich haben. Die Stellen des Netzes können Marken tragen. Die Zuordnung der Marken zu allen im Netz befindlichen Stellen wird als Markierung bezeichnet. Zur Identifizierung eines Petrinetzes muss auch immer die Anfangsmarkierung mit angegeben werden.

Tragen alle Stellen im Vorbereich einer Transition mindestens eine Marke, so wird diese Transition als aktiviert bezeichnet und kann schalten. Transitionen ohne Vorbereich sind immer aktiviert und können beliebig oft schalten. Beim Schalten wird jeder Stelle im Vorbereich eine Marke entfernt und jeder Stelle im Nachbereich eine Stelle hinzugefügt. Durch die neue Belegung der Stellen entsteht eine neue Markierung. Durch das Schalten kann die Gesamtanzahl der Marken im Netz reduziert oder erhöht werden.

Die von der Anfangsmarkierung aus durch Schalten der Transitionen erreichbaren Markierungen werden in einem gerichteten Graphen zusammengefasst – dem Erreichbarkeitsgraphen.

Je nach Anordnung von Stellen und Transitionen sowie der Belegung der Stellen kann ein Petrinetz beschränkt oder unbeschränkt sein. Bei einem beschränkten Petrinetz hat der Erreichbarkeitsgraph endlich viele Knoten (Markierungen) und Kanten (Transitionen, die den Übergang von einer Markierung zur Folgemarkierung bewirken). In einem unbeschränkten Petrinetz ist die Anzahl der Knoten und Kanten unbegrenzt, sodass dieses immer weiter wachsen kann. Von einem beschränkten Petrinetz kann der vollständige Erreichbarkeitsgraph ermittelt und gezeichnet werden, während bei einem unbeschränkten Petrinetz nur ein partieller Erreichbarkeitsgraph ermittelt werden kann.

Ein Petrinetz gilt als unbeschränkt, wenn es eine Markierung m gibt, von der durch Schalten der Transitionen eine Folgemarkierung m' erreichbar ist, welche folgende Eigenschaft besitzt: die Markierung m' weist an jeder Stelle mindestens genau so viele Marken auf wie die Markierung m und an mindestens einer Stelle weist die Markierung m' mehr Marken auf.

Im Zuge dieses Praktikums soll ein Programm entwickelt werden, welches Petrinetze im WoPeD-Format aus XML-Dateien einliest und analysiert. Dabei soll es möglich sein, das Petrinetz händisch durch Schalten der aktivierten Transitionen zu untersuchen. Darüber hinaus sollen Marken zu Stellen hinzugefügt oder von Stellen entfernt werden können, woraus sich jeweils eine neue Anfangsmarkierung ergibt. Des Weiteren soll das eingelesene Petrinetz automatisch untersucht werden können sowie eine Stapelverarbeitung mehrerer Petrinetze ohne grafische Darstellung sollen implementiert werden. Wichtige Informationen werden in einem Textfeld im unteren Teil der grafischen Oberfläche ausgegeben.

2 Bedienungsanleitung

Neben den in der Aufgabenstellung velangten Optionen enthält das Programm ein Toggle-Button zur gleichzeitigen Auswahl mehrerer Stellen, ein weiteres Menü „Export“ und ein weiteres Menü „Hilfe“. Das Menü Export hat folgende Einträge:

1. Speichern
2. Petrinetz-Darstellung speichern
3. Erreichbarkeitsgraph-Darstellung speichern

Durch Klicken des Eintrags „Speichern“ wird ein Speichern-Dialog geöffnet, der die Erzeugung einer PNML-Datei im XML-Format ermöglicht. So kann das Petrinetz mit der aktuellen Markierung als neue Anfangsmarkierung zu einem späteren Zeitpunkt geladen werden. Die beiden anderen Optionen ermöglichen es, die grafische Darstellung der beiden Graphen einzeln als JPG-Datei auf der Festplatte zu speichern.

Der einzige Eintrag „Über“ des Hilfe-Menüs öffnet ein Meldungsfenster, in dem die grundlegenden Daten zum Projekt und Programmierer vermerkt sind.

Durch das Klicken des Toggle-Buttons „Auswahl mehrerer Stellen“ wird dem Programm mitgeteilt, ob mehrere Stellen gleichzeitig bearbeitet werden dürfen oder nicht. Ist der Toggle-Button aktiv, können mehrere Stellen nacheinander angeklickt und zur Auswahl hinzugefügt werden. Durch Klicken auf „Marke hinzufügen“ bzw. „Marke entfernen“ wird die Anzahl aller ausgewählten Stellen verändert. Wird der Toggle-Button deaktiviert, werden alle Stellen – mit Ausnahme der zuletzt ausgewählten Stelle – wieder abgewählt, sodass lediglich die Anzahl der Marken an einer Stelle verändert wird.

In der grafischen Darstellung des Petrinetzes werden ausgewählte Stellen grün eingefärbt, die restlichen Stellen sind grau. Aktivierte Transitionen sind gelb eingefäbt, nicht aktivierte Transitionen sind weiß. Im Erreichbarkeitsgraphen ist die Anfangsmarkierung grau eingefärbt. Die durch die Analyse evtl. festgestellten Knoten m und m’ sind in gelb (m) und rot (m’) eingefärbt. Der Pfad m $\rightarrow$ m’ ist blau hinterlegt. Die Kante der zuletzt ausgeführten Transition wird im Erreichbarkeitsgraph rot eingefärbt. Wurde die Anfangsmarkierung bei einer vorhergehenden Analyse als Knoten m oder m’ identifiziert wird er ausschließlich in dieser Farbe angezeigt und nicht mehrin grau eingefärbt. Die aktuelle Markierung hat eine fette Umrandung und die Schrift innerhalb des Knotens ist fett.

3 Programm- und Datenstruktur

3.1 Programmarchitektur

Für die Programmarchitektur wurde das Prinzip von Modell-View-Controller (MVC) gewählt. Dabei liegt ein eigenständiges Modell des Petrinetzes sowie des Erreichbarkeitsgraphen vor, welches auch in einem anderen Umfeld – ohne den hier verwendeten Controller und die grafische Oberfläche – verwendet werden kann. Der Controller und die grafische Nutzoberfläche sind zum Teil miteinander verzahnt und können nicht ohne Weiteres getrennt werden.

Der Controller nimmt Befehle von der grafischen Oberfläche entgegen und ruft die Methoden des Petrinetzes auf, um somit die gewünschten Aktionen zu bewirken. Anschließend fordert er die grafische Oberfläche dazu auf, die Ergebnisse für den Nutzer darzustellen. Dabei werden die Ergebnisse im Textfeld oder im Graphen dargestellt.

Die grafische Oberfläche nimmt selbst keine Berechnungen und Aktionen vor, sondern benachrichtigt den Controller darüber, welche Aktionen der Benutzer durch anklicken der jeweiligen Elemente aufrufen möchte.

Insgesamt besteht das Programm aus vier Packages: main, model, view und controller. Im Package main liegt ausschließlich die Klasse Petrinets_9739335_Wilms_Hannes, deren Methode main zum Starten der Anwendung aufgerufen wird. Die drei Packages model, view, controll beinhalten folgende Klassen:

Tabelle 3.1 Aufteilung der Klassen in die Packages model, view und controll

model	view	controll
• Arc • BoundednessTestResult • Marking • MarkingConnector	• MainFrame • PetrinetGraph • ReachabilityGraph	• Controller • PetrinetClickListener • ReachabilityClickListener

Für die Darstellung der Graphen wird die Graphstream-Bibliothek verwendet.

3.2 Modellierung des Petrinetzes

Für die Modellierung des Petrinetzes wurde die Klasse **Petrinet** als Hauptobjekt angelegt. Darüber hinaus werden die Knoten des Petrinetzes in der abstrakten Klasse **Node** und die Kanten des Petrinetzes in der Klasse **Arc** behandelt.

Die Klasse **Node** die privaten Attribute `id`, `name`, `posX` und `posY`, welche mittels Vererbung an die Subklassen **Place** und **Transition** weitergegeben werden. Stellen haben darüber hinaus die Attribute `tokens` und `isSelected`, sodass die Anzahl der Marken der Stelle sowie die Information, ob die Stelle zur Zeit ausgewählt ist lokal gespeichert werden. Transitionen hingegen besitzen zwei Listen `input` und `output`, in denen Verweise auf die Stellen im Vor- bzw. Nachbereich gespeichert sind.

Objekte der Klasse **Petrinet** besitzen drei Listen für Place, Transition und Arc, sodass dort alle Elemente des Petrinetzes „zusammengehalten“ werden. Darüber hinaus ist im Petrinetz der zugehörige (partielle) Erreichbarkeitsgraph der Klasse **ReachabiltyNet** hinterlegt sowie der Container mit den Ergebnissen der Beschränktheitsanalyse der Klasse **BoundednessTestResult**.

Die Markierungen des Netzes (die Knoten des Erreichbarkeitsgraphen) werden durch die Klasse **Marking** dargestellt. Diese beinhaltet neben der `id` (welche sich aus der Belegung der Marken zusammensetzt) auch noch die Belegung, die Markierungen, die auf dem Pfad zu dieser Markierung besucht wurden, die Kanten auf dem Pfad zu dieser Markierung sowie die Informationen, ob es sich bei dieser Markierung um m oder m’ handelt.

Die Belegung der Stellen bei dieser Markierung werden in einer TreeMap<String,Integer> gespeichert, sodass auf die Markierung in alphabetischer Reihenfolge nach der id der Stellen zugegriffen werden kann. Die Knoten und Kanten auf dem Pfad zu dieser Markierung werden je in einer LinkedList gespeichert.

Die Transformationen von einer Markierung zur Folgemarkierung mittels einer Transition (die Kanten des Erreichbarkeitsgraphen) werden durch die Klasse **MarkingConnector** repräsentiert. Diese Klasse enthält die `id` der Kante (welche sich aus den ids der vorhergehenden und der folgenden Markierung sowie der Transition zusammensetzt), die ids der vorhergehenden und nachfolgenden Markierungen sowie der Transition und die Informationen, ob sich die Kante auf dem (eventuell vorhandenen) Pfad m $\rightarrow$ m’ befindet und ob es sich um, die zuletzt beschrittene Kante handelt.

Im internen Modell des (partiellen) Erreichbarkeitsgraphen (**ReachabilityNet**) sind alle bisher erreichten Markierungen in der Liste `markings` und alle bisher beschrittenen Kanten von einer Markierung mittels einer Transition zu einer anderen Markierung in der Liste `markingConnectors` gespeichert sowie je ein Verweis auf die Anfangsmarkierung und die aktuelle Markierung – die beiden letztgenannten Zeiger können auf dasselbe Objekt zeigen.

Der Algorithmus zur Berechnung der Beschränktheit eines Petrinetzes ist innerhalb der Klasse **Petrinet** realisiert.

4 Beschränktheitsalgorithmus

Die Analyse der Beschränktheit des Petrinetzes besteht aus zwei Teilen. Im ersten Teil wird zunächst geprüft, ob im Petrinetz stellen vorhanden sind. Ist dies der Fall wird ein Objekt der Klasse **BoundednessTestResult** als Container für die Berechnungsergebnisse erzeugt und in diesem wird der Wert für Unbeschränktheit auf falsch gesetzt. Danach wird ein neuer Zeiger auf die Anfangsmarkierung gesetzt und diesem gemeinsam mit dem Container für die Berechnungsergebnisse an den eigentlichen rekursiven Berechnungsalgorithmus übergeben. Die Initialisierung des Beschränktheitsalgorithmus ist in Listing 1 dargestellt.

```
procedure Initialisierung();
  { Initialisiert den Container fuer die Berechnungsergebnisse,
  Startet den Beschraenktheitsalgorithmus,
  Speichert Informationen in Elemente des Petrinetzes
  }

type
BoundednessTestResult = (Name           : String,
                         Start          : Markierung,
                         End            : Markierung,
                         AnzahlKnoten   : int,
                         AnzahlKanten   : int,
                         Pfad           : array[maxInt] of Kante,
                         beschraenkt    : bool);

begin
  if Petrinetz hat Stellen then
    neues BoundesnessTestResult erzeugen;
    Unbeschraenktheit im Ergebnis auf falsch setzen;
    Name des BoundednessTestResult setzen (Dateiname);
    Erreichbarkeitsgraph auf Anfangsmakerung setzen;
    Zeiger(aktuelleMarkierung) auf Anfangsmarkierung setzen;
    Aufruf analysisMechanism(Zeiger, BoundednessTestResult);

    if Petrinetz ist beschraenkt then
      Start- und Endmarkierung festlegen;
      Kante.isInPath fuer Pfad m -> m' wahr setzen;
    endif; { Petrinetz ist beschraenkt }

    Petrinetz auf aktuelle Markierung setzen;
    Anzahl Knoten und Kanten des EGs speichern;
  endif; { Petrinetz hat Stellen }
end.
```

Listing 1 Initialisierung des Beschränktheitsalgorithmus

Im zweiten Teil der Analyse wird geprüft, ob das Netz Transitionen besitzt. Anschließend werden nacheinander die Transitionen durchlaufen. Zunächst wird die lokale Variable mit der Information, ob die zuletzt erzeugte Kante bereits im Erreichbarkeitsgraphen vorhanden ist auf *falsch* gesetzt und das Petrinetz wird auf die übergebene Markierung zurückgesetzt. Wenn die

ausgewählte Transition aktiviert ist, wird sie geschaltet und eine neue Markierung und eine Kante von der vorherigen zur neuen Markierung werden erzeugt.

Es wird geprüft, ob die neue Markierung und die neue Kante bereits im Erreichbarkeitsgraph vorhanden sind. Ist dies nicht der Fall, werden die Markierung und die Kante zum Erreichbarkeitsgraphen hinzugefügt. Sind die Elemente bereits im Erreichbarkeitsgraphen vorhanden werden die entsprechenden Elemente aus dem Erreichbarkeitsgraphen geladen und die Zeiger werden auf diese Elemente gesetzt.

Die Liste der Kanten in der neuen Markierung wird dahingehend geändert, dass die Liste der vorherigen Markierung gesetzt wird und die Kante von der vorherigen Markierung zu dieser Markierung hinzugefügt wird. Ebenso wird die Liste der Markierungen auf dem Pfad von der vorherigen Liste übernommen und die vorherige Markierung wird hinten angefügt. Bei diesen Aktionen werden bei bereits vorhandenen Markierungen die Listen überschrieben, sodass immer nur der zuletzt beschrittene Pfad gespeichert ist.

Anschließend wird für alle Markierungen auf dem Pfad überprüft, ob die neue Markierung an allen Stellen gleich viele Marken besitzt und an wenigstens einer Stelle mehr Marken trägt als diese Markierungen. Ist dies der Fall, wird im Ergebnis der Wert für Unbeschränktheit auf *wahr* gesetzt. Zusätzlich dazu werden die Markierungen m (Markierung auf dem Pfad) und m' (neue Markierung) als Start- und Endmarkierung gesetzt, sowie der Pfad von der Anfangsmarkierung zur neuen Markierung.

Für den Erreichbarkeitsgraph wird die neue Markierung als aktuelle Markierung festgelegt.

Wenn im Container des Berechnungsergebnisses der Wert für Unbeschränktheit auf *falsch* gesetzt ist und die zuletzt ermittelte Kante noch nicht im Erreichbarkeitsgraphen vorhanden ist, wird die Methode rekursiv mit der aktuellen Markierung aufgerufen.

Listing 2 zeigt den Ablauf des rekursiven Beschränktheitsalgorithmus.

```
procedure analysisMechanism(aktuelleMarkierung : Markierung,
    Ergebnis : BoundednessTestResult);
  { Schaltet nacheinander die aktivierten Transitionen fuer die
  jeweilige Markierung. Prueft, ob die entstehende Markierung oder
  die Kante bereits im Erreichbarkeitsgraph sind.
  Fuegt neue Kanten und Markierungen in den Erreichbarkeitsgraph ein.
  Prueft, ob das Petrinetz unbeschraenkt ist.
  Gibt ein Objekt der Klasse BoundednessTestResult zurueck.
  }

begin
  if Petrinetz hat Transitionen then
    for alle Transitionen im Petrinetz do
      if Transition i ist aktiviert then
        Transition schalten;
        neue Markierung erzeugen und aktuelle Werte eintragen;
        pruefen, ob Markierung bereits im Erreichbarkeitsgraph;
        neue Kante von alter Markierung zu neuer Markierung erzeugen;
        pruefen, ob Kante bereits im Erreichbarkeitsgraph;
        Pfad der Markierung anpassen;
        for alle Markierungen auf dem Pfad zur neuen Markierung do
          m = alte Markierung auf dem Pfad;
          m' = neue Markierung;
          pruefeUnbeschraenkt(m, m');
          if pruefeUnbeschraenkt = true then
            Ergebnis.unbeschraenkt = true;
            Ergebnis.Start = m;
            Ergebnis.End = m';
            Ergebnis.Pfad = Pfad von Anfangsmarkierung bis m';
          endif; {pruefeUnbeschraenkt = true}
        end; { for }
        if Ergebnis.unbeschraenkt = false and Kante ist nicht im EG then
          analysisMechanism(neueMarkierung, Ergebnis);
        endif; { unbeschraenkt = false and Kante nicht im EG}
      endif; {Transition ist aktiviert}
    end; { for }
  endif; { Petrinetz hat Transitionen }
  return Ergebnis;
end.
```

Listing 2 Beschränktheitsalgorithmus

Die Untersuchung nach der Eigenschaft m -> m' setzt sich aus zwei Teilstufen zusammen. In der ersten Stufe wird überprüft, ob die beiden Markierungen identisch sind. Ist dies nicht der Fall, wird für jede Stelle des Netzes untersucht, ob diese in der neuen Markierung gleich viele oder mehr Marken hat als in der alten Markierung. Hat eine Stelle weniger Marken, ist die Eigenschaft m -> m' nicht vorhanden. Ansonsten ist die Eigenschaft m -> m' vorhanden, da die Gleichheit der Markierungen vorher ausgeschlossen wurde.

Listing 3 zeigt das Vorgehen bei der Untersuchung von zwei Markierungen auf die Eigenschaft m -> m'.

```
procedure pruefeUnbeschraenkt (m : Markierung, mStrich : Markierung);
  { Ueberprueft, ob alle Stellen der Markierung mStrich
  genauso viele Marken haben wie bei der Markierung m und
  mindestens eine Stelle bei der Markierung mStrich mehr
  Marken aufweist als bei der Markierung m.
  }

begin
  if m.id ist identisch mit mStrich.id then
    return false;
  else
    for alle Stellen im Petrinetz do
      if tokens an Stelle i in m groesser als in mStrich then
        return false;
      endif;
    end; { for }
  endif; { m.id identisch mStrich.id }
  return true;
end.
```

Listing 3 Beschränktheitsalgorithmus