

Grundpraktikum Programmierung - Petrinetzanalyse

Markus Eberl

2. Juli 2019

Inhaltsverzeichnis

1	Einleitung	2
1.1	Software-Allgemein:	2
1.1.1	MVC-Model:	2
1.1.2	Petrinetz-Datenstruktur:	2
1.1.3	Erreichbarkeitsbaum-Datenstruktur:	3
1.1.4	Erreichbarkeitsanalyse:	3
1.1.5	Fenster:	4
1.1.6	Ausnahmebehandlung:	4
2	Bedienungsanleitung	5
2.1	Knoten-Informationsfenster	5
2.2	Kameraposition-Reset	5
2.3	Externes Auswertefenster	6
3	Beschreibung der Programmstruktur	6
3.1	Beschreibung der Datenstruktur - Petrinetz	7
3.2	Beschreibung der Datenstruktur - Erreichbarkeitsgraph	8
4	Beschreibung des Beschränktheits-Algorithmus	8
4.1	Algorithmus im Pseudocode	8
4.2	Erklärung des Algorithmus	8

1 Einleitung

1.1 Software-Allgemein:

1.1.1 MVC-Model:

Dieses Modell teilt eine Software in drei Teile die miteinander verbunden werden. Der Teil View (Präsentation) ist für die Gestaltung und Darstellung der Informationen zuständig. Die Informationen kommen vom Teil des Controllers (Steuerung), außerdem verwaltet die Steuerung die Datenmodelle und Koordinierung der Objekte. Zum letzten Teil des Models (Modell) zählen nun noch alle Klassen, die nur für Datenstrukturen verwendet werden und somit auch alle Daten. Im Projekt für das Grundpraktikum Programmieren sind die Klassen in folgende Bereiche eingeteilt:

- **Model - Modell:** DecisionEdge, DecisionNode, DecisionGraph, DecisionTree, DirectedEdge, Element, Place, Transition, NarrownessAnalysis, NarrownessAnalysisResult, PetriNetGraph, PNML, PNMLNet, TransitionContainer
- **View - Präsentation:** PetriNetFrame, NodeDialog, NarrownessAnalysisDialog
- **Controller - Steuerung** PetriNetController, DecisionClickListener, PetriNetClickListener

1.1.2 Petrinetz-Datenstruktur:

- » **Klasse PNML:** Containerklasse für Petrinetze (PNMLNet).
- » **Klasse PNMLNet:** Containerklasse für Objekte der Typen Place, Transition und DirectedEdge.
- » **Klasse PetriNetGraph:**
- » **Klasse Element:** Allgemeine Klasse für Funktionen und Attribute, die für Stellen und Transitionen verwendet werden können. Diese Klasse ist eine Mutterklasse für die Klassen Place (Stelle) und Transition (Transition).
- » **Klasse Place:** Diese Klasse erbt alle Attribute und Methoden von der Mutterklasse *Element*. Zusätzlich wurde noch eine Verwaltung für die Markierung implementiert.

- » **Klasse Transition:** Diese Klasse erbt alle Attribute und Methoden von der Mutterklasse *Element*. Zusätzlich wurde noch die Funktionalität zum Schalten der Transitionen implementiert.
- » **Klasse DirectedEdge:** Containerklasse für die Verbindung zwischen Place's und Transition's.

1.1.3 Erreichbarkeitsbaum-Datenstruktur:

- » **Klasse DecisionTree:** Containerklasse für die Knoten und Kanten, die den Entscheidungsgraphen aufbauen.
- » **Klasse DecisionNode:** Dieses Objekt speichert den bei der Erstellung aktuellen Zustand des Petrinetzes bezüglich der Markierungen.
- » **Klasse DecisionEdge:** Containerklasse für die Transition, die beim Schalten des Knoten benutzt wurde, auf die diese Kante zielt.
- » **Klasse DecisionGraph:** Kindklasse von der externen Graphstream-Bibliothek mit der Mutterklasse Multigraph. Beim erzeugen dieses Objektes wird ein Multigraph erzeugt und im Anschluss kann ein übergebenes Objekt des Typens *DecisionTree* benutzt werden, um den Entscheidungsgraphen zu zeichnen.

1.1.4 Erreichbarkeitsanalyse:

- » **Klasse NarrownessAnalysis:** Diese Objekt beinhaltet die Funktionalität zum Prüfen eines Petrinetzes auf Unbeschränktheit. Falls das Petrinetz unbeschränkt ist bricht der rekursive Algorithmus ab und erzeugt des partiellen Erreichbarkeitsgraphen zur Darstellung.
- » **Klasse NarrownessAnalysisResult:** Containerklasse für NarrownessAnalysis-Objekte und Attribute zur späteren Darstellung der Ergebnisse des Beschränktheitsalgorithmuses.
- » **Klasse TransitionContainer:** Containerklasse für Transitionen, zum Zwischenspeichern bei dem rekursiven Durchlauf des Petrinetzes um den Erreichbarkeitsgraphen zu erzeugen.

1.1.5 Fenster:

- » **Klasse NodeDialog:** Dialogfenster, das Informationen eines Kontens anzeigt.
- » **Klasse PetriNetFrame:** Hauptfenster, das die gesamte Funktionalität der geforderten Aufgabe darstellt. Sämtliche Steuerungselemente werden aus dieser Klasse gesteuert.
- » **Klasse NarrownessAnalysisDialog:** Fenster, das optional aktiviert werden kann um das Ergebnis der Beschränktheitsanalyse mehrerer Dateien übersichtlich in einer Tabelle darzustellen.

1.1.6 Ausnahmebehandlung:

- » **Klasse NarrownesAnalysisException:** Fehler, die bei der Beschränktheitsanalyse entstehen können werden in dieser Containerklasse gespeichert und wiedergegeben.
- » **Klasse PNMLParserException:** Fehler, die bei dem Parsing-Vorgang entstehen können werden in dieser Containerklasse gespeichert und wiedergegeben.
- » **TransitionContainerException:** Fehler, die beim schalten der Transitionen auftreten können, werden in dieser Containerklasse gespeichert und wiedergegeben.

2 Bedienungsanleitung

2.1 Knoten-Informationsfenster

Es wurde ein Dialogfenster eingefügt, das dem Benutzer genauere Informationen über einen angewählten Knoten im Entscheidungsgraph liefert. Die Bedienung ist wie folgt. Der Anwender der Software muss den Entscheidungsgraph im Fokus setzen, sprich einfach in den EG mit der Maus klicken. Anschließend muss die STRG-Taste gehalten werden und auf den gewünschten Knoten im Graphen klicken. Nun öffnet sich ein Fenster, in dem die Software dem Benutzer genauere Informationen des angewählten Objektes des Typs DecisionNode gibt. Diese Informationen umfassen, die Nachbarnknoten, die ausgehenden Kanten (geschaltene Transitionen) und die Markierungen der Stellen zum Zeitpunkt vor des Schaltens der Transition.

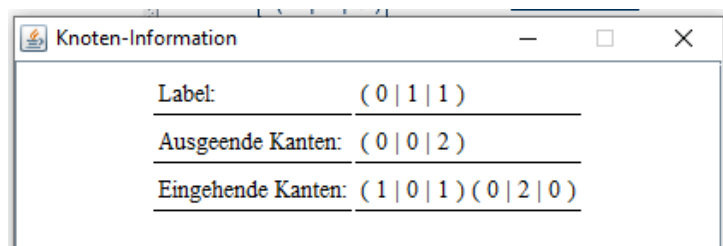


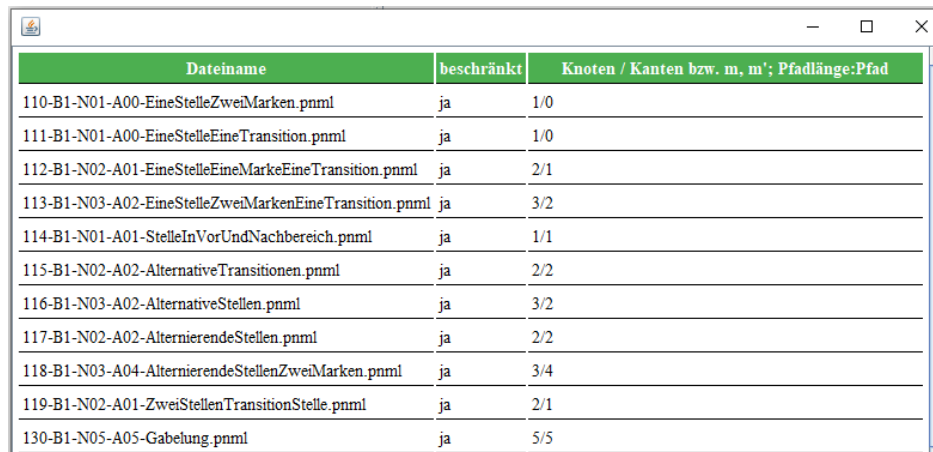
Abbildung 1: NodeDialog

2.2 Kameraposition-Reset

Im Menü "GraphStream" existiert die Auswahlmöglichkeit "Kameraposition reseten", mit dem man wieder zur normalen Darstellung gelangt. Falls der Benutzer die Kameraposition so verstellt hat, kann man mit diesem Button alles wieder rückgängig machen.

2.3 Externes Auswertefenster

Um die Beschränktheitsanalyse von mehreren Dateien schöner darzustellen, wurde eine Einstellung implementiert, die es erlaubt die Auswertung extern in einem eigenem Fenster darzustellen. Hierzu muss im Menü "Visualisierung" ein Hacke in der Checkbox mit dem Namen "Beschränktheitsanalyse Visu" gesetzt werden. Somit öffnet sich nach der Analyse ein Fenster um das Ergebnis in einer Tabelle übersichtlicher darzustellen.



Dateiname	beschränkt	Knoten / Kanten bzw. m, m'; Pfadlänge:Pfad
110-B1-N01-A00-EineStelleZweiMarken.pnml	ja	1/0
111-B1-N01-A00-EineStelleEineTransition.pnml	ja	1/0
112-B1-N02-A01-EineStelleEineMarkeEineTransition.pnml	ja	2/1
113-B1-N03-A02-EineStelleZweiMarkenEineTransition.pnml	ja	3/2
114-B1-N01-A01-StelleInVorUndNachbereich.pnml	ja	1/1
115-B1-N02-A02-AlternativeTransitionen.pnml	ja	2/2
116-B1-N03-A02-AlternativeStellen.pnml	ja	3/2
117-B1-N02-A02-AlternierendeStellen.pnml	ja	2/2
118-B1-N03-A04-AlternierendeStellenZweiMarken.pnml	ja	3/4
119-B1-N02-A01-ZweiStellenTransitionStelle.pnml	ja	2/1
130-B1-N05-A05-Gabelung.pnml	ja	5/5

Abbildung 2: Externes Auswertefenster

3 Beschreibung der Programmstruktur

Das Programm ist in drei Teile aufgeteilt. Alle Klasse, die zur Darstellung oder zur Bedienung benötigt werden sind im Package View ausgelagert. Das Package Model enthält alle Klasse, die benötigt werden um die Datenstrukturen des Petrinetzes und des Erreichbarkeitsbaumes zu erstellen. Des weiteren inkludiert diese Package auch Klassen zur Erstellung graphischen Datenstrukturen mittels GraphStream-Framework. Das letzte Package namens Controller beinhaltet Klassen, die zur Implementierung der Funktionalitäten benötigt werden.

Ziel dieses Programms ist es ein Petrinetz aus einer PNML-Datei mittels eines Parser zu erzeugen, dieses Petrinetz kann im Anschluss entweder manuell weiterverwendet werden um einen partiellen Erreichbarkeitsgraphen zu generieren oder es wird zur automatischen Erzeugung der Beschränktheitsanalyse benutzt um im Anschluss das Ergebnis mittels vollständigem Erreichbarkeitsgraphen darzustellen.

Die Beschränktheitsanalyse kann auch auf mehrere Dateien gleichzeitig angewendet werden. Hierfür gibt es extra ein Menüpunkt, die die Analyse von vorher ausgewählten Dateien startet und das Ergebnis am Ende darstellt.

3.1 Beschreibung der Datenstruktur - Petrinetz

Die Datenstruktur des Petrinetzes besteht aus zwei verschiedene Knoten, zum einen die Transition und zum anderen die Stelle. Die Knoten werden mittels Kanten verbunden. Diese Verbindungen stellen eine Abhängigkeit in einer vorgegebenen Richtung dar und sind essentiell für das Schalten der Transitionen. Die Transitionen können jedoch nur geschalten werden, wenn alle Stellen die mittels einer Kante auf diese Transition zeigen mindestens eine Markierung aufweisen. Nach dem erfolgreichem Schalten werden alle Markierungen der Stellen von denen aus dieser Transition Kanten zeigen inkrementiert. Transitionen werden graphisch durch ein rechteckiges Element und die Stellen werden graphisch durch ein Kreis-Element dargestellt.

3.2 Beschreibung der Datenstruktur - Erreichbarkeitsgraph

Die Datenstruktur des Erreichbarkeitsgraphen besteht aus Knoten, die die Zustände der Markierungen der Stellen im Petrinetz darstellen und aus Kanten, die die Transitionen darstellen um auf die Markierungen der folgenden Knoten zu kommen. Ist das Unbeschränktheitskriterium für zwei Knoten auf einem Pfad erfüllt, endet der Erreichbarkeitsgraph unvollständig und man bezeichnet diesen dann als partiellen Erreichbarkeitsgraphen.

4 Beschreibung des Beschränktheits-Algorithmus

4.1 Algorithmus im Pseudocode

Data: Petrinetz, Entscheidungsbaum
Result: Partieller oder vollständiger Erreichbarkeitsgraph
 transitionen = speichereAlleSchaltbarenTransitionen();
 speichereMarkierung();
while *transitionen.nochSchaltbareTransitionVerfügbar()* **do**
 transition = transitionen.pop();
 transition.schalteTransition();
 erzeugeKnotenUndKanteImEntscheidungsbaum();
 if *prüfeEntscheidungsbaumAufUnbeschränktheit()* **then**
 return true;
 else
 if *not letzterErzeugteKnotenSchonVorhanden* **then**
 result = StarteBeschränktheitsAlgorithmus();
 end
 setzeUrsprünglicheMarkierung();
 if *return* **then**
 return true;
 end
 end
end
return false;

Algorithm 1: StarteBeschränktheitsAlgorithmus

4.2 Erklärung des Algorithmus

Der Algorithmus erzeugt den Erreichbarkeitsgraphen rekursiv. Im ersten Schritt werden alle schaltbaren Transitionen im Petrinetz in einer Liste gespeichert. Der zweite Schritt wirkt sich im Speichern der zu diesem Zeitpunkt aktuellen Markierung aus. Im Anschluss wird mit einer Schleife durch alle vorher gespeicherten noch schaltbaren Transitionen iteriert. Diese Iteration beinhaltet das Schalten einer Transition von der Liste der noch zu schaltenden Transitionen. Des weiteren wird nun eine neue Kanten und ein Knoten, der die neue Markierung des Petrinetzes repräsentiert im Entscheidungsbaum erzeugt. Während des folgenden Schritts wird der neuen Entscheidungsbaum auf Unbeschränktheitskriterien geprüft. Falls der diese erfüllt werden bricht der Algorithmus ab. Im Falle der Beschränktheit wird eine erneuter Aufruf des Algorithmus folgen um diesen wieder mit den neuen Bedingungen von

vorne beginnen zu lassen. Im Anschluss wird die ursprüngliche Markierung dieses Petrinetzes wiederhergestellt um eine Art Tiefensuche zu implementieren. Nun wird das Ergebnis der rekursiven vorhergegangenen Aufrufe des Algorithmus bewertet, falls diese nicht die Unbeschränktheit aufweist, wird eine neue Transition geladen, um den Iterator neu zu starten. Sollte keine Transition mehr gefunden werden wird der Aufruf beendet.