Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Modellierung und Verifikation des Token Ring Algorithmus

Birger Kamp und Maria Lüdemann
Formale Simulation und Verifikation verteilter Algorithmen
Sommersemester 2016

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Birger Kamp und Maria Lüdemann

Formale Simulation und Verifikation verteilter Algorithmen
Sommersemester 2016

Modellierung und Verifikation des Token Ring Algorithmus eingereicht
im Rahmen des Moduls Formale Simulation und Verifikation verteilter Algorithmen
im Studiengang Master Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuerin: Prof. Dr. Julia Padberg

Abgegeben am 20. Juni 2016

Inhaltsverzeichnis

1. Einleitung **4**

2. Hauptteil **4**
2.1. Der Token Ring Algorithmus 4
2.2. Spezifikation 6
2.3. Modellierung 7
2.3.1. Das Netz 8
2.3.2. Ablauf 11
2.3.3. Abbildung der Spezifikation in der Modellierung 11
2.4. Korrektheit 13
2.4.1. Umwandlung CPN zu PN 13
2.4.2. Erreichbarkeitsgraph 13
2.4.3. Prüfung durch CTL 14

3. Zusammenfassung und Ausblick **16**
3.1. Fazit 16
3.2. Ausblick 17

A. Snoopy Do's and Don'ts **19**

1. Einleitung

Dieses Dokument beschreibt die Ergebnisse des SVA Praktikums. Es sollte ein verteilter Algorithmus gewählt werden um ihn dann in mehreren Schritten in einem Petri Netz zu modellieren, zu spezifizieren und seine Korrektheit zu zeigen. Dafür sollte das Programm Snoopy zur Modellierung verwendet werden und in Charlie die Korrektheit nachgewiesen werden. Diese Arbeit beschreibt welcher Algorithmus gewählt wurde, die Umsetzung der Modellierung und die Beschreibung und Ergebnisse der Spezifikation und Korrektheit. Dabei wird sich damit auseinander gesetzt welche Probleme und Erkenntnisse daraus entstanden und beschreibt einen kleinen Ausblick auf weitere Möglichkeiten.

Die Arbeit an diesem Dokument teilt sich wie folgt auf:

Kapitel	**Geschrieben von**
1 Einleitung	Maria Lüdemann
2.1 Der Token Ring Algorithmus	Maria Lüdemann
2.2 Spezifikation	Birger Kamp
2.3 Modellierung	Maria Lüdemann
2.4 Korrektheit	Birger Kamp
3 Zusammenfassung und Ausblick	Maria Lüdemann

Tabelle 1: Arbeitsteilung

2. Hauptteil

2.1. Der Token Ring Algorithmus

Der Token Ring Algorithmus ist ein Wahlalgorithmus der von Chang und Roberts 1979 entworfen wurde. Er kann verteilt auf mehreren Clienten verwendet werden die in einer Ring-Topologie miteinander verbunden sind. Das Ziel des Algorithmus ist, bei Ausfall des Master-Clienten im Netz einen neuen zu wählen.

Voraussetzungen

Damit der Algorithmus auf eine Ring-Topologie angewandt werden kann, müssen folgende Voraussetzungen im Netz gegeben sein:

- Jeder Client kennt seinen Nachfolger

- Jeder Client ist mit seinem Nachfolger verbunden, sodass er mit ihm kommunizieren kann
- Jeder Client hat eine eindeutige ID
- Jeder Client kennt die gesamte Ring-Topologie

Ablauf

Der Algorithmus startet wenn der Master-Client ausfällt. Der Vorgänger des ausgefallenen Master-Clients baut eine Verbindung zum Nachfolger des ausgefallenen Master-Clients auf, sodass die Ring-Topologie wieder vollständig ist.

Der Client, der den Ausfall bemerkt, startet die Wahl in dem er seinem Nachfolger eine Nachricht mit seiner ID und der Info dass es sich um eine Wahl handelt schickt. Dieser nimmt die Nachricht und überprüft ob seine eigene ID darin vor kommt. Falls nicht, hängt er seine eigene ID hinten an und schickt die vervollständigte Nachricht an seinen Nachfolger.

Wenn ein Client feststellt, dass seine eigene ID bereits in der Nachricht vorhanden ist, nimmt er die höchste ID aus der Liste der gesammelten IDs in der Nachricht. Anschließend sendet er eine "GewähltNachricht mit der höchsten ID an seinen Nachfolger. Der Empfänger der "GewähltNachricht merkt sich, dass der gewählte Client nun der neue Master ist und sendet seinem Nachfolger die gleiche "GewähltNachricht. Jeder wird somit benachrichtigt was die höchste ID ist. Kommt die "Gewählt"Nachricht wieder am Initiator der "GewähltNachricht an, wird die Wahl erfolgreich beendet und der Algorithmus ist terminiert.

Eigenschaften

Die Laufzeit dieses Algorithmus befindet sich in einem Bereich von $0 = 2N$ wobei N die Anzahl der teilnehmenden Clienten ist. Diese Einschätzung ergibt sich daraus, dass es zwei Nachrichten gibt die jeweils einmal jeden Clienten erreichen müssen. Dabei durchläuft die erste Nachricht, die Wahl N Clienten und die Nachricht, dass ein neuer Master gewählt wurde durchläuft ebenfalls noch einmal N Clienten.

Die Eigenschaften des Algorithmus lassen sich wie folgt beschreiben

- Laufzeit von 2N
- 2 Arten von Nachrichten "WahlNachricht mit ID Liste und "GewähltNachricht mit der neuen Master ID

- Der Algorithmus findet in einer endlichen Anzahl von Clienten immer den neuen Master (höchste ID)
- Alle Teilnehmenden Clienten setzten irgendwann ihren Master auf den gewählten Master
- Der Algorithmus terminiert sicher

2.2. Spezifikation

Damit ein Modell erstellt werden kann, das den Algorithmus abbildet, müssen zunächst die charakteristischen Eigenschaften des Algorithmus bestimmt werden.

Der Algorithmus (s. Abschnitt 2.1) lässt sich in folgende drei Phasen einteilen:

Phase 1: Ein Client bemerkt den Ausfall des bisherigen Masters

Phase 2: Sammeln aller beteiligten Client-IDs

Phase 3: Bekanntgeben des neuen Masters

In Tabelle 2 ist aufgeführt, mit welchen Eigenschaften die einzelnen Phasen beschrieben werden.

Phase	Eigenschaft	Nr.
Phase 1: Master-Ausfall bemerkt	Sendet Nachricht zum Wählen und hängt seine eigene ID daran	1
Phase 2: Wahl	Client, der Wahl-Nachricht erhält, hängt seine eigene ID an die Nachricht	2
	Client sendet die erweiterte Nachricht an seinen Nachfolger	3
	Sobald ein Client eine Wahl-Nachricht erhält, in der seine eigene ID bereits enthalten ist, geht der Algorithmus in Phase 3 über	4
Phase 3: Neuen Master mitteilen	Der Client, der feststellt, dass Phase 2 vorbei ist, sendet eine Nachricht mit dem neuen Master an seinen Nachfolger	5
	Ein Client, der die Nachricht über einen Master erhält, merkt sich den neuen Master	6
	Ein Client, der die Nachricht über einen Master erhält, teilt seinem Nachfolger diese Nachricht mit	7
	Sobald der Client, der Phase 3 eingeleitet hat, die Nachricht über den neuen Master erhalten hat, terminiert der Algorithmus	8

Tabelle 2: Spezifikation der Phasen des Algorithmus

2.3. Modellierung

Für die Modellierung wurde der Algorithmus mithilfe der Spezifikation auf seine essentiellen Bestandteile untersucht und an die Möglichkeiten von Petri Netzen angepasst. Es musste ebenfalls verarbeitet werden, dass die Möglichkeiten des zu verwendenden Programms *Snoopy* hinsichtlich des Umfangs und der Mächtigkeit der Petri Netze eingeschränkt ist. So konnten zum Beispiel die ID Listen in den Nachrichten nicht umgesetzt werden.

Einige Eigenschaften des Algorithmus bereiten Probleme wenn sie in einem normalen Petri Netz abgebildet werden sollen. Dazu gehört:

1. Clienten haben eine eigene ID
2. Nachrichten übermitteln unterschiedliche Datenstrukturen (Nachrichten Art, ID Listen)

3. Bei unterschiedlichen Nachrichten reagiert der Client anders

Einige dieser Probleme können dadurch gelöst werden indem ein farbiges Petri Netz genutzt wird (CPN). Dieses ermöglicht mithilfe von Farben und Konstanten jedem Clienten eine eigene ID zu geben. Durch Guards an den Transitionen und Variablen kann ein unterschiedliches Verhalten erzeugt werden. Ein Problem das sich damit allerdings nicht lösen lies war die Datenstrukturen, ein Token kann nur einen einzigen Wert haben.

Um dieses Problem zu beheben wurde der Algorithmus ein wenig verändert. Für die Modellierung werden keine ID Listen weiter gereicht sondern nur noch eine einzige ID und zwar die lokal größte. Das bedeutet, der Initiator schickt seine ID zu seinem Nachbarn. Dieser vergleicht die ID mit der eigenen und schickt die Größere weiter. Dies stellt keine Grundlegende Änderung in der Funktionsweise des Algorithmus dar sondern verschiebt bloß den Schritt des Vergleichs. Nun überprüft nicht mehr nur der Initiator welches die höchste ID ist, sondern jeder Client überprüft die eigene ID mit der empfangenen. Wie der Algorithmus nun im Detail abläuft wird im Folgenden beschrieben.

Um den Algorithmus abzubilden wurde ein farbiges Petri Netz gewählt um den Token eine Gewichtung geben zu können die die ID der jeweiligen Clienten widerspiegelt. Des Weiteren war es notwendig an den Transitionen entscheiden zu können, wann diese schalten. Auch dies bietet das farbige Petri Netz (CPN) durch Guards.

2.3.1. Das Netz

In der untenstehenden Abbildung ist das gesamte Netz zu sehen, das das Ergebnis der Modellierung war. Um eine gewisse Übersichtlichkeit zu wahren wurden nur drei Clienten modelliert. Dies reicht um die Funktionalität des Algorithmus zu modellieren hält aber den Komplexitätslevel und vor allem die Unübersichtlichkeit in Grenzen.

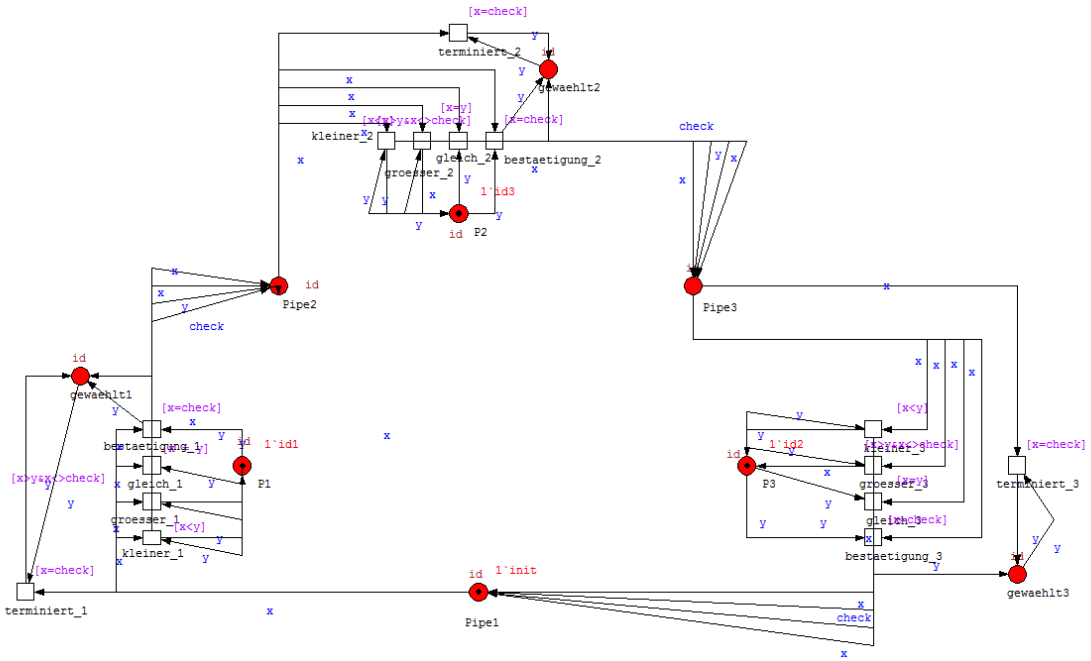

Abbildung 1: Der Token Ring Algorithmus als farbiges Petri Netz

Das Netz besteht im Grunde aus einem Konstrukt das den Clienten beschreibt und sich N mal wiederholt, in unserem Fall ist $N = 3$, es gibt also drei Clienten. Die Nachrichtenleitung wurde durch drei Stellen umgesetzt und zwar *Pipe1* - *Pipe3* die die Clienten miteinander verbinden. Zwischen diesen Stellen befindet sich jeweils ein aus fünf Transitionen und zwei Stellen bestehendes Konstrukt das jeweils einen Clienten darstellt. Die untenstehende Abbildung zeigt einen Clienten zur Veranschaulichung.

Ein Client *Pn* bekommt Nachrichten immer von der Nachrichtenleitung *Pn* und sendet Nachrichten auf die Leitung *Pn+1*. Die ankommende Nachricht am Client hat die Möglichkeit fünf Transitionen zu schalten. Die Guards an den Transitionen sind allerdings so gewählt das niemals alle geschaltet werden können sondern höchstens eine. Den Algorithmus bilden davon vier der Transitionen ab, die fünfte *terminiert_ n* ist dafür da, dass die Token aufhören zu wandern wenn die Wahl getroffen ist und geben die Leitung damit frei. Dies ist eine reine Formsache. Die anderen vier Transitionen vergleichen den Token der von *Pipen* kommt mit dem Token der auf *Pn* liegt. Jede Transition kümmert sich um einen Vergleich *kleiner_ n* überprüft ob der neue Token kleiner ist als der gespeicherte, *groesser_ n* ob der kommende Token größer ist aber nicht so groß wie der 'Gewählt'-Token, *gleich_ n* überprüft ob es der Token ist der bereits gespeichert wurde und *bestaetigung_ n* ermittelt ob es die 'Gewählt'-Nachricht ist. Je nachdem welche Transition schaltet werden unterschiedliche Aktionen ausgeführt. Ist der ankommende Token kleiner (es schaltet *kleiner_ n*) wird er ignoriert

und der eigene Token als größerer weitergeleitet, also auf *Pipe_N+1* gelegt und wieder gespeichert also auf *Pn* geschoben. Ist der kommende Token größer (es schaltet *groesser_n*) wird der kommende Token auf *Pipe_N+1* weitergeleitet und in *Pn* gespeichert. Wenn die Transition *gleich_n* schaltet sind beide Token gleich. Dadurch wird die Phase gewechselt. Also aus der in Tabelle 2 definierten Phase 2 der Wahl in die Phase 3 - Bekanntgeben des neuen Masters, gewechselt. Es wird der 'Gewählt'-Token initialisiert und auf *Pipe_N+1* gelegt. Der bis eben gespeicherte Token wandert auf die Stelle *gewaehltN*. Die vierte Transition regelt den Fall, dass ein 'Gewählt'-Token kommt. Wenn dies geschieht schaltet die Transition nur in dem Fall, dass der Client die Nachricht nicht selbst initialisiert hat. Nur dann kann die *bestaetigung_n* Transition schalten. Hat der Client selbst die 'Gewählt'-Nachricht initialisiert liegt auf *Pn* kein Token mehr. In diesem Fall schaltet die *terminiert_N* Transition und zieht den 'Gewählt'-Token von der Pipe. Diese Transition kann nur dann schalten weil dann bereits der gespeicherte Token auf der *gewaehltN* Stelle liegt. An dieser Stelle terminiert der Algorithmus und alle Clienten haben die höchste ID auf der *gewaehltN* Stelle liegen.

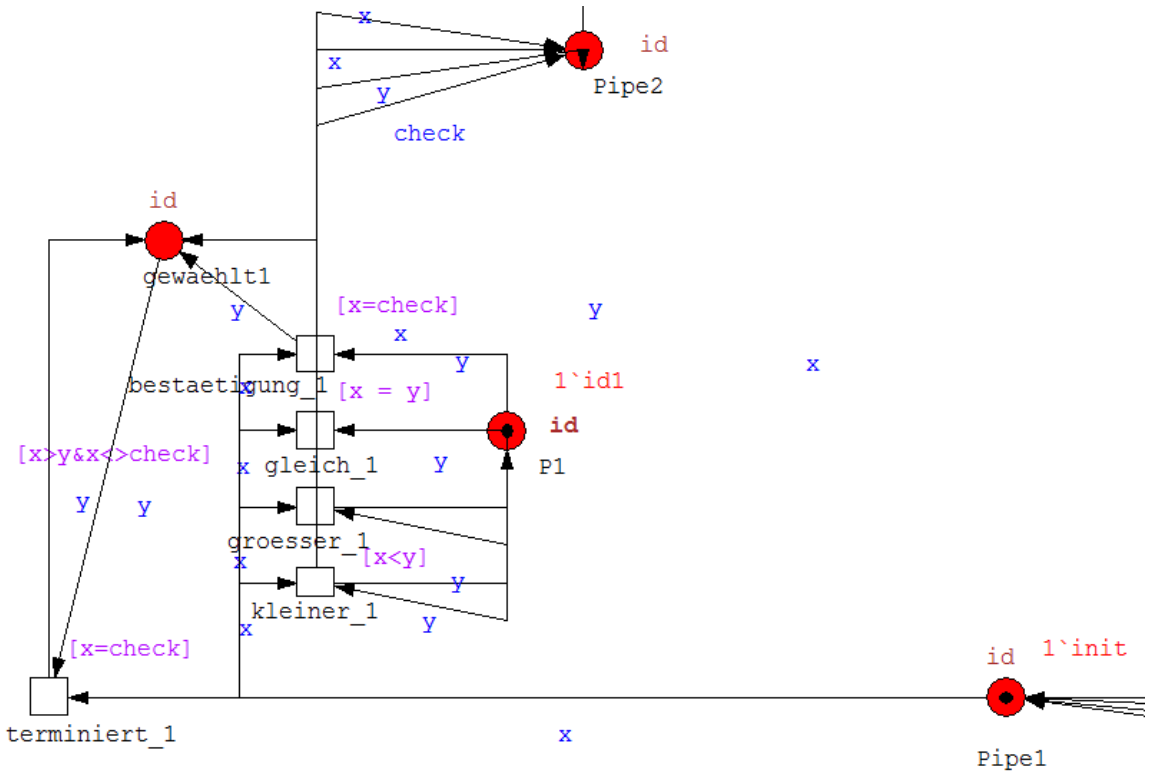

Abbildung 2: Detailausschnitt eines Clienten

2.3.2. Ablauf

Phase 1 - Master-Ausfall bemerkt
Das Netz ist so konfiguriert, dass alle Clienten mit ihrer eigenen einzigartigen ID als Token auf ihrem internen Speicher *Pn* starten. Auf der Initiator-Stelle *Pipe_1* liegt ein Token mit einer sehr geringen Wertung. Diese Stelle ist in der Regel eine normale Leitung, durch den Start mit einem Token der Wertigkeit 1 (alle Ids sind größer als 1) initiiert es am Anfang die Wahl in dem der Token zum ersten Clienten wandert der ihn mit seinem eigenen vergleicht.

Phase 2 - Wahl
Nach der Initialisierung läuft ein Token mit der aktuell höchsten ID herum. Dabei überprüft jeder Client bei dem der Token ankommt, ob die eigene bzw gespeicherte ID größer ist. Je nachdem wird entweder die eigene ID weitergeschickt oder die neue und dann auch gespeichert. Stellt ein Client fest, dass die verglichenen Token gleich sind, ist der Token einmal herum gelaufen und die größte ID steht fest. Also schreibt der Client die ID weg und löst die 'Gewählt'-Nachricht aus und startet dadurch Phase 3.

Phase 3 - Neuen Master mitteilen
Die dritte Phase ist dafür da, allen Clienten mitzuteilen dass die ID, die sie gerade gespeichert haben, die endgültige ist. Dies passiert in dem die 'Gewählt'-Nachricht herum geschickt wird. Jeder Client, der diese Nachricht erhält, schreibt die ID von *Pn* auf die *gewaehltN* Stelle und schickt die 'Gewählt'-Nachricht weiter. Kommt diese Nachricht wieder bei dem Clienten an, der sie losgesendet hat, schaltet die *terminiert_n* Transition und nimmt die Nachricht aus der Pipe. Das Netz kann nun nicht mehr schalten und hat an allen Clienten die höchste ID gewählt.

2.3.3. Abbildung der Spezifikation in der Modellierung

Die folgende Abbildung verdeutlicht wie die Spezifikation durch die Modellierung abgedeckt wurde.

Phase und Spezifikiationseigenschaft	Modell Eigenschaft	Nr.
Phase 1: Sendet Nachricht zum Wählen und hängt seine eigene ID daran	Auf *Pipe1* liegt der Initiator-Token und startet den ersten Vergleich bei Client 1	1
Phase 2: Client, der Wahl-Nachricht erhält, hängt seine eigene ID an die Nachricht	Jeder Client vergleicht und sendet nur die höchste ID	2
Phase 2:Client sendet die erweiterte Nachricht an seinen Nachfolger	Die höhere ID wird auf die *PipeN+1* gelegt die den Token an den nächsten Clienten liefert	3
Phase 2:Sobald ein Client eine Wahl-Nachricht erhält, in der seine eigene ID bereits enthalten ist, geht der Algorithmus in Phase 3 über	Erhält ein Client eine ID die gleich der gespeicherten ist schaltet die *gleich_N* Transition die veranlasst das die ID weggeschrieben wird und die 'Gewählt'-Nachricht auf *PipeN+1* legt. Dadurch wird nun die Spezielle Nachricht weitergeleitet und nicht mehr eine ID	4
Phase 3: Der Client, der feststellt, dass Phase 2 vorbei ist, sendet eine Nachricht mit dem neuen Master an seinen Nachfolger	Die 'Gewählt'-Nachricht signalisiert jedem Clienten das die ID die er gerade gespeichert hat die höchste ist.	5
Phase 3: Ein Client, der die Nachricht über einen Master erhält, merkt sich den neuen Master	Erhält ein Client die 'Gewählt'-Nachricht schaltet die *bestaetigung_n* Transition und die gespeicherte ID wird weggespeichert. Die an diesem Punkt gespeicherte ID ist die höchste im Ring	6
Phase 3: Ein Client, der die Nachricht über einen Master erhält, teilt seinem Nachfolger diese Nachricht mit	Schaltete die *bestaetigung_n* Transition wird der Wert weggespeichert und die 'Gewählt'-Nachricht weitergeleitet in dem sie auf *PipeN+1* gelegt wird.	7
Phase 3: Sobald der Client, der Phase 3 eingeleitet hat, die Nachricht über den neuen Master erhalten hat, terminiert der Algorithmus	Kommt die 'Gewählt'-Nachricht bei ihrem Initiator an kann dort die *bestaetigung_n* Transition nicht schalten da auf *Pn* kein Token mehr liegt, somit schaltet *terminiert_n* und zieht den Token von der Pipe. Das Netz kann nun nicht mehr schalten und der Algorithmus terminiert.	8

Tabelle 3: Abbildung der Spezifikation auf das Modell

2.4. Korrektheit

Um zu zeigen, dass das erstellte Modell dem Algorithmus entspricht, wird im Folgenden die Korrektheit bewiesen. Dazu wird die Existenz der spezifizierten Eigenschaften aus Tabelle 2 im erstellten Petri-Netz durch CTL-Ausdrücke geprüft.

2.4.1. Umwandlung CPN zu PN

Das in Abschnitt 3 erstellte Modell ist ein farbiges Petri-Netz (engl: Colored Petri Net, kurz: CPN), daher lässt sich darauf nicht die CTL anwenden. Um die CTL verwenden zu können, muss vorher das erstellte Netz in ein einfaches Petri-Netz (PN) umgewandelt werden.

Das Modell-CPN verwendet einige Transition-Guards und einen Datentyp mit einem begrenzten Wertraum. Diese Logik ist nicht in einfachen Petri-Netzen erlaubt, daher muss sie umgewandelt werden.

Wenn im CPN auf der Stelle *P1* ein Token mit dem Wert *3* liegt, wird dies im PN dargestellt indem auf der Stelle *P1_3* ein Token liegt. Es gibt daher für *P1* für jeden Wert des Datentyps eine Stelle. Die Transition-Guards des CPN werden im PN durch komplexe Stellen-Transitions-Schaltungen abgebildet.

Durch diese Umwandlung wird das PN im Vergleich zum CPN sehr groß und unübersichtlich.

2.4.2. Erreichbarkeitsgraph

Der Erreichbarkeitsgraph wird auf dem einfachen Petri-Netz gebildet, da die CTL-Ausdrücke auch auf dem einfachen Petri-Netz geprüft werden.

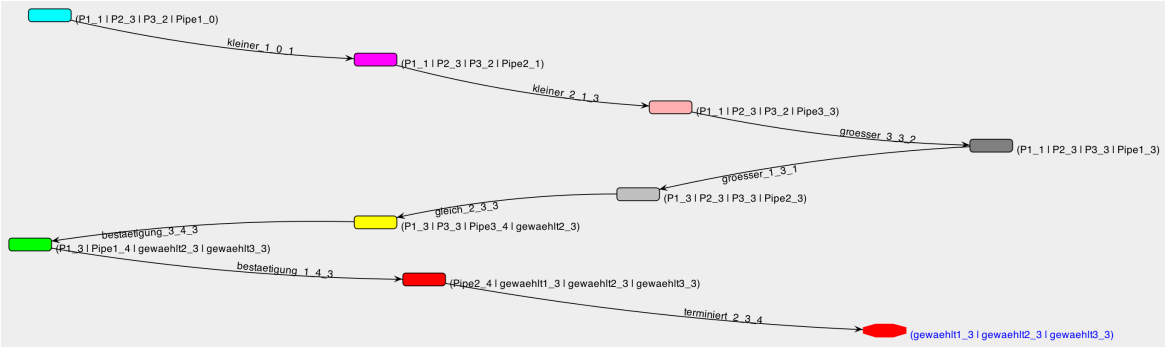

Abbildung 3: Erreichbarkeitsgraph des PN

In Abbildung 3 ist der generierte Erreichbarkeitsgraph zu sehen. Es fällt auf, dass der Graph keine Abzweigungen hat, sondern nur einen Pfad enthält. Das bedeutet, dass der Algorithmus bei gleichen Bedingungen sich deterministisch verhält.

Würde man die Bedingungen verändern, sodass beispielsweise ein anderer Client den Master-Ausfall bemerkt oder ein anderer Client im Netzwerk hat die höchste ID, dann würde sich die Reihenfolge der Kanten im Erreichbarkeitsgraph verändern und die Knoten würden andere Stellen beinhalten. Der resultierende Erreichbarkeitsgraph wäre allerdings nach wie vor geradlinig.

2.4.3. Prüfung durch CTL

Auf Grundlage des einfachen PN kann nun die Korrektheit in Bezug auf die Modell-Eigenschaften aus Tabelle 3 festgestellt werden. Um im einfachen PN eine Aussage prüfen zu können, wie beispielsweise ob auf der Stelle P1 nur ein Token mit dem Wert *3* liegt, ist folgender Ausdruck nötig: $P1\_1 == 0$ & $P1\_2 == 0$ & $P1\_3 == 1$ & $P1\_4 == 0$

Wie zu sehen ist, ist ein langer CTL-Ausdruck nötig um eine einfache Aussagen zu prüfen. Die Ausdrücke, die nötig sind um das Modell auf die Spezifikation zu prüfen, sind deutlich komplexer. Wir haben daher eine Pseudo-CPN-CTL verwendet, die die Ausdrücke deutlich verkürzt. Möchte man feststellen ob auf der Stelle P1 ein Token mit dem Wert *3* liegt, dann würde man das wie folgt ausdrücken: $P\_1 == 1'3$

Diese Ausdrucksweise ist deutlich kürzer und lesbarer. Die Aussagen zum Prüfen des Modells werden daher in der Tabelle 4 in der Pseudo-CPN-CTL verfasst. In der Tabelle sind die Ausdrücke zu den jeweiligen Modell-Eigenschaften aufgeführt.

Modell Eigenschaft Nr.	Pseudo-CPN-CTL
1	EF(Pipe1==0 & Pipe2==0 & Pipe3==0 & P1==1'1 & P2==1'2 & P3==1'3)
2	EF(Pipe2!=0 & P2!=0 & Pipe2>P2 & Pipe2==Pipe3)
3	EF(Pipe2!=0 & P2!=0 & Pipe2>P2 & Pipe2==Pipe3)
4	EF(Pipe2!=0 & P2!=0 & Pipe2==P2 & AX(Pipe3==1'4))
5	EF(Pipe2!=0 & P2!=0 & Pipe2==P2 & AX(Pipe3==1'4))
6	EF(Pipe1==1'4 & P1!=0) & UX(Gewaehlt1 != 0 & P1 ==0)
7	EF(Pipe3==1'4 & P3!=0 & P3!=1'4 & AX(Pipe1==1'4))
8	EF(Pipe2==1'4 & AX(P2==0 & Gewaehlt2==1'3 & Pipe1==0 & Pipe2==0 & Pipe3==0 & Gewaehlt1==1'3 & Gewaehlt2==1'3))

Tabelle 4: Prüfen des PetriNetz-Modells mittels Pseudo-CPN-CTL

Zum Vergleich dazu sind in Tabelle 5 zwei der Pseudo-CTL-Ausdrücke beispielhaft als einfache CTL-Ausdrücke dargestellt. Vergleicht man diese, sieht man den Unterschied zwischen den Schreibweisen hinsichtlich der Lesbarkeit noch deutlicher.

Modell Eigenschaft Nr.	Pseudo-CPN-CTL.
1	EF(P1_1==1 & P2_3==1 & P3_2 == 1 & (Pipe1_0==1 & Pipe1_1==0 & Pipe1_2==0 & Pipe1_3==0 & Pipe1_4==0)& (Pipe2_0==0 & Pipe2_1==0 & Pipe2_2==0 & Pipe2_3==0 & Pipe2_4==0) & (Pipe3_0==0 & Pipe3_1==0 & Pipe3_2==0 & Pipe3_3==0 & Pipe3_4==0));
2	EF((Pipe2_4==0 & (Pipe2_1==1 | Pipe2_2==1 | Pipe2_3==1)) & (P2_1==1 | P2_2==1 | P2_3==1) & ((P2_2==1 & Pipe2_1==1) | (P2_3==1 & (Pipe2_1==1 | Pipe2_2==1))) & AX((Pipe3_1==1 & P2_1==1) | (Pipe3_2==1 & P2_2==1) | (Pipe3_3==1 & P2_3==1)));

Tabelle 5: Prüfen des PetriNetz-Modells mittels CTL

Aus Gründen der Lesbarkeit werden nicht alle Pseudo-CPN-CTL-Ausdrücke aus Tabelle 4 in CTL-Ausdrücke in Tabelle 5 überführt.

Übersetzt man alle Ausdrücke und führt diese auf das einfache PN aus, erhält man ein positives Ergebnis. Das bedeutet dass alle spezifizierten Eigenschaften im Modell zu finden sind und dass das Modell dem modellierten Algorithmus entspricht.

3. Zusammenfassung und Ausblick

Im diesem Teil werden die Erfahrungen dieser Arbeit zusammengefasst und es wird ein kurzer Ausblick gegeben.

3.1. Fazit

Im Laufe dieser Arbeit ergaben sich unterschiedliche Probleme. Zum einen war es anfänglich schwer abzuschätzen, welche Anforderungen an den Algorithmus gestellt sind, um den Umfang der Aufgabe gerecht zu werden und dabei interessante Ergebnisse zu erzielen. Unsere Wahl des Token Ring Algorithmus wurde dem Umfang gerecht, ließ

sich aber nicht durch ein normales Petri Netz in Snoopy abbilden und ist vor allem deterministisch, was einen trivialen Erreichbarkeitsgraphen nach sich zieht. Neben den Schwierigkeiten der generellen Aufgabenstellung gab es immer wieder Probleme mit dem Programm, das verwendet werden sollte. Die Verwendung eines farbigen Petri Netzes führte zu den Problemen, dass eine Dokumentation des Programms kaum vorhanden und nicht wirklich hilfreich einsetzbar ist. Ebenfalls nicht abschätzbar waren dadurch die Schwierigkeiten im weiteren Verlauf der Arbeit, da das Erzeugen eines Erreichbarkeitsgraphen nur über Umwege zu realisieren war, die auch auf Anhieb nicht zu finden waren. Im Anschluss daran traten Probleme beim Korrektheitsbeweis mit einer CTL in Snoopy auf, aufgrund der Tatsache dass wir ein farbiges Netz benutzten das keinen normalen Erreichbarkeitsgraphen hatte konnte Snoopy das Netz nicht sinnvoll mit einer CTL prüfen. Um dieses Problem zu umgehen nutzen wir in erster Linie eine Pseudo CTL, also eine CTL die das beschreiben des Netzes ermöglicht aber nicht mit Snoopy testbar ist. Im Nachhinein haben wir die Pseudo CTL Ausdrücke in sehr umständliche und unübersichtliche CTL Ausdrücke übersetzt, die zwar testbar, jedoch zu umständlich sind um wirklich handhabbar zu sein.

Somit gab es in der Aufgabe einige Probleme, die hin und wieder dafür sorgten, dass Ergebnisse nicht wie erwartet waren oder Arbeitsschritte nicht wie gewünscht möglich waren. Snoopy erwies sich als ein Tool mit einigen schönen Vorteilen, aber auch einigen Problemen, die das Arbeiten erschweren und extra Lösungen notwendig macht.

3.2. Ausblick

Der Ausblick, der Arbeit befasst sich hauptsächlich damit, was an dem Netz geändert werden könnte um zufriedenstellendere Ergebnisse zu bekommen, hinsichtlich der Flexibilität oder der Testbarkeit.

Da immer wieder Probleme in Hinsicht auf Snoopys Fähigkeiten mit farbigen Netzen auftraten, wäre es das beste, wenn das farbige Netz in ein nicht farbiges Umgewandelt werden könnte. Diese Umwandlung ist allerdings derart aufwändig, dass sie im Rahmen dieser Arbeit nicht ausreichend betrachtet werden konnte.

In den letzten Zügen der Arbeit kam das Thema auf, wie das Netz etwas flexibler gestaltet werden könnte. Der Erreichbarkeitsgraph wird in allen Variationen immer deterministisch bleiben, da der Algorithmus auf den natürlichen Zahlen (IDs) arbeitet und somit eindeutig ist wie der Algorithmus ablaufen muss. Eine Änderung, die jedoch vorstellbar wäre, ist die variable Anzahl an Clienten zu ermöglichen. Im aktuellen Netz gibt es drei Clienten, die explizit dargestellt sind. Dies ist ein Umstand, der dadurch umgangen werden könnte, wenn das Netz auf einen Regelkreis minimiert wird, in dem ein Konstrukt alle Clienten darstellt und die Anzahl der Clienten einzig von der Anzahl

der IDs, die genutzt werden, bestimmt wird. Auch diese Änderung blieb aus Zeitgründen bloß eine Idee und wurde nicht weiter verfolgt. Durch die Variable Anzahl der Clienten könnte der Determinismus des Algorithmus noch schöner heraus gearbeitet werden, da der Errichbarkeitsgraph auch bei steigender Clienten Zahl zwangsläufig gradlinig deterministisch bleiben muss.

A. Snoopy Do's and Don'ts

Dinge die Snoopy gut kann, die gut klappen:

- man kann Unternetze definieren
- es lassen sich Petri Netze modellieren
- unterstützt viele PetriNetz-Arten

Dinge die Snoopy nicht gut kann, die nicht klappen:

- jegliche Form von Usability
- Es ist absturzgefährdet, also oft speichern!
- Sobald Modellierungen größer werden wird es sehr schnell unübersichtlich und man kann nicht nach rechts erweitern nur nach links, also Achtung wo man anfängt zu modellieren
- das Verschieben von Stellen und Transitionen ist in älteren Versionen nur bedingt möglich
- Objekte (Stellen, Transitionen, Kanten, Beschriftungen), die dicht beieinander liegen, lassen sich nicht dediziert auswählen. Man muss erst die anderen Objekte zur Seite schieben, bis man das gewünschte Objekt auswählen kann.
- es gibt keine *Undo*-Funktionalität (Strg + Z) !