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 Projekts New Storytelling
im Studiengang Master Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuer: Sascha Kluth

Abgegeben am 05. Februar 2015

Inhaltsverzeichnis

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:

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
A. Korrektheitsbeweis	Birger Kamp

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 so 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

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

Der jeweilige Ablauf der Phasen lässt sich mit folgenden Punkten spezifieren:

2.3. Modellierung

Für die Modellierung wurde der Algorithmus mithilfe der Spezifikation auf seine essentiellen Bestandteile herunter gebrochen und dann an die Möglichkeiten von Petri Netzen angepasst. Es musste ebenfalls verarbeitet werden, das die Möglichkeiten von *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

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

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, dass 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.

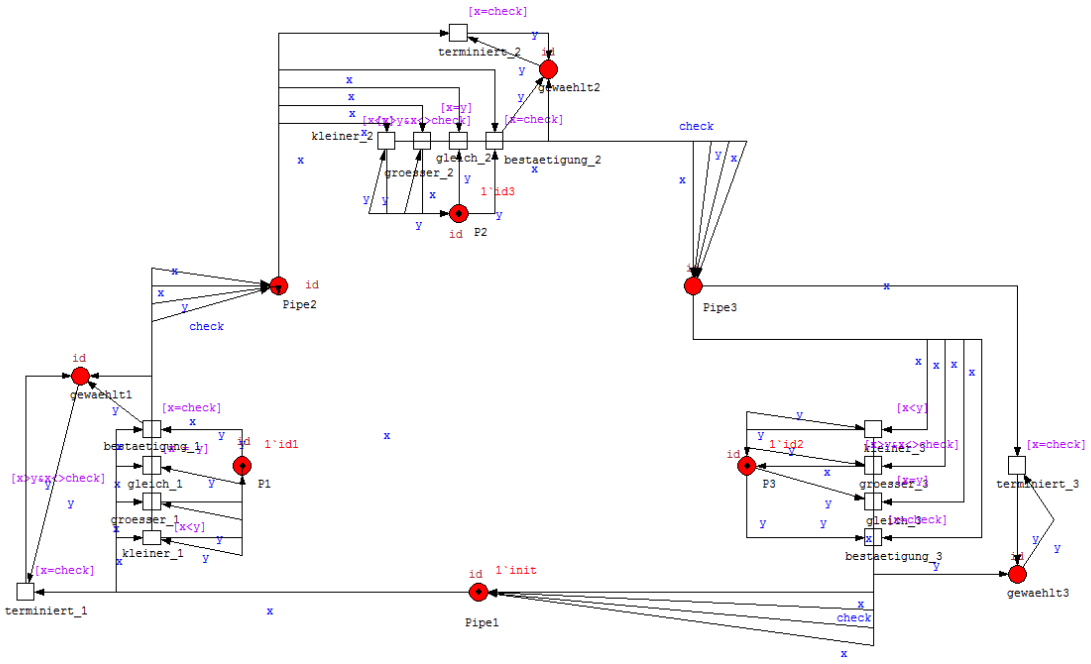

Abbildung 1: Der Token Ring Algorithmus als farbiges Petri Netz

Das Netz besteht im Grunde aus einem Konstrukt das sich dreimal wiederholt. Die Nachrichtenleitung wurde durch drei Stellen umgesetzt und zwar *Pipe_1 - Pipe_3*. 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 *P_n* bekommt Nachrichten immer von der Nachrichtenleitung *P_n* und sendet Nachrichten auf die Leitung *P_n+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 *Pipe_n* kommt mit dem Token der auf *P_n* liegt. Jede Transition kümmert sich um einen Vergleich *kleiner_n* überprüft ob das neue Token kleiner ist als das gespeicherte,*groesser_n* ob das kommende Token größer ist aber nicht so groß wie der 'Gewählt'-Token, *gleich_n* überprüft ob es das Token ist das bereits gespeichert wurde und *bestaetigung_n* ermittelt ob es die 'Gewählt'-Nachricht ist. Je nachdem welche Transition schaltet werden unterschiedliche Aktionen ausgeführt. Ist das ankommende Token kleiner (es schaltet *kleiner_n*) wird es ignoriert und das eigene Token als größeres weitergeleitet, also auf *Pipe_N+1* gelegt und wieder gespei-

chert also auf *Pn* geschoben. Ist das kommende Token größer (es schaltet *groesser_n*) wird das kommende Token auf *Pipe_N+1* weitergeleitet und in *Pn* gespeichert. Wenn die Transition *gleich_n* schaltet sind die beiden Token gleich. Dadurch wird die Phase gewechselt. Also aus der in 2 definierten Phase 2 der Wahl in die Phase drei Bekanntgeben des neuen Masters gewechselt. Es wird das 'Gewählt'-Token initialisiert und auf *Pipe_N+1* gelegt. Das bis eben gespeicherte Token wandert auf die Stelle *gewaehltN*. Die vierte Transition regelt den Fall, dass ein 'Gewählt'-Token kommt. Wenn dies geschiet 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.

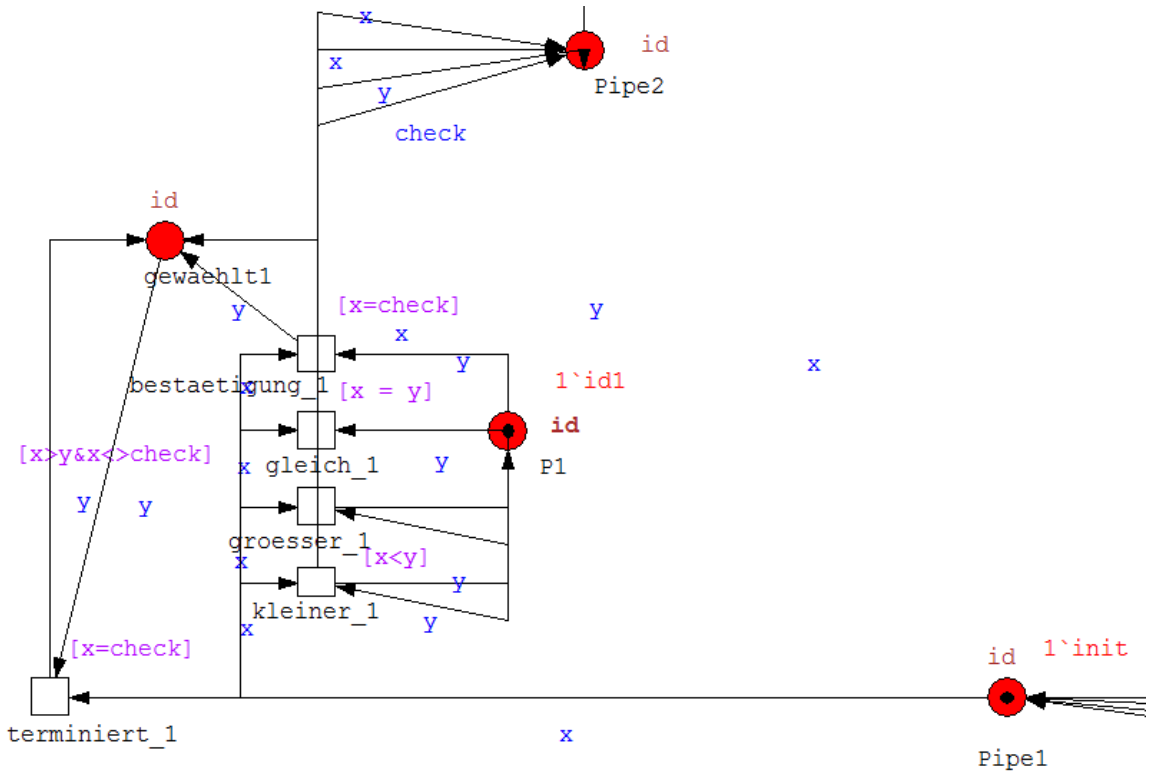

Abbildung 2: Detailausschnitt eines Clienten

2.4. Ablauf

Phase 1 Initialisieren

2.5. 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.

2.5.1. Umwandlung CPN zu PN

Das in Abschnitt **??** 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 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.

Dadurch wird das PN im Vergleich zum CPN sehr groß und unübersichtlich.

2.5.2. Erreichbarkeitsgraph

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

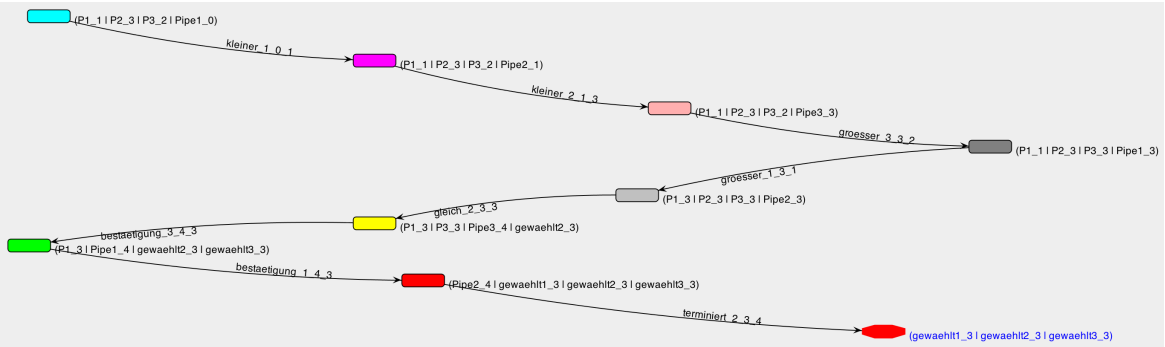

Abbildung 3: Erreichbarkeitsgraph des PN

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 bspw. 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.

2.5.3. Prüfung durch CTL

Auf Grundlage des einfachen PN kann nun die Korrektheit in Bezug auf die Modell-Eigenschaften aus Tabelle X festgestellt werden. Die Übersicht über alle verwendeten Ausdrücke ist in Tabelle A im Anhang zu finden.

Die CTL-Ausdrücke lassen sich fehlerfrei auf das einfache Petri-Netz anwenden. Daher entspricht das erstellt CPN dem Token-Ring-Algorithmus von Chang&Pang.

3. Zusammenfassung und Ausblick

Netz flachklopfen um über IDs ein beliebig großes Netz generieren zu können um zu zeigen, dass auch bei steigender Clienten Zahl der Erreichbarkeitsbaum gradlinig deterministisch bleibt Probleme mit Snoopy bezgl. CTL und CPN erklären

Probleme bei Charlie- Snoopy – erreichbarkeitsgraph , farbige Netze

A. Korrektheitsbeweis