Universität Rostock

Traditio et Innovatio

Literaturarbeit

Verifikation von Petrinetzen

Vorgelegt von:

Tom Meyer

Matrikel-Nr.: 8200839

Eingereicht am:

29. März 2019

Betreuer:

Karsten Wolf

UNIVERSITÄT ROSTOCK | FAKULTÄT FÜR INFORMATIK UND ELEKTROTECHNIK

Lehrstuhl für Theoretische Informatik

Albert-Einstein-Straße 22, 18059 Rostock | Tel: +49 381 498 7641

Inhaltsverzeichnis

1 Einleitung

Eines der ältesten und verbreitetsten Modelle zur Beschreibung von, sowohl sequentiellen, als auch parallelen Rechnern, ist der endliche Automat.

Automaten eigenen sich um Aussagen über bestimmte Eigenschaften der beschriebenen Systeme zu treffen. Eine erschöpfende Analyse wird jedoch bei größeren Modellen immer schwieriger, da die Anzahl der möglichen Zustände die angenommen werden können, gewöhnlich exponentiell mit der Größe des Systems wächst. Dieses Phänomen wird als Zustandsexplosion bezeichnet und ist eines der größten Hindernisse im Gebiet der Softwareverifikation.

Gerade bei verteilten Systemen ist es aber selten nötig Auswirkungen von Aktionen über den gesamten Zustandsraum zu prüfen. Viele Aktionen haben nur einen sehr lokalen Einfluss und können nur bestimmte Teile des Zustands verändern. Die Konstruktion des kompletten Zustandsraums ist also nicht unbedingt notwendig um bestimmte Eigenschaften zu überprüfen.

Das war auch einer der Gedanken, die Carl Adam Petri umtrieb und ihn dazu inspirierte einen anderen Formalismus zur Beschreibung von Computern zu entwickeln. Die nach ihm benannten Petri-Netze.

Petri-Netze haben einige interessante Eigenschaften, die man in der Verifikation gut ausnutzen kann. Bevor wir allerdings zur Verifikation mit Petri Netzen kommen, sehen wir uns im folgenden Kapitel diese Eigenschaften etwas genauer an.

2 Petri-Netze

Petri-Netze sind Bipartite Grafen mit gerichteten Kanten. D.h. sie bestehen aus zwei Disjunkten Knotenmengen – **Stellen** und **Transitionen** – die nur mit der jeweils anderen Menge durch Kanten verbunden sind.

Die Stellen (oder auch Plätze) können mit einer beliebigen Menge an **Marken** belegt sein. Die Menge der Marken auf allen Stellen repräsentiert den aktuellen Zustand des Systems und wird **Markierung** genannt.

Transitionen repräsentieren die möglichen **Aktionen** des Systems. Eine Transition ist **aktiviert**, wenn auf allen Stellen die mit eingehenden Kanten verbunden sind, mindestens eine Marke liegt (bzw. eine Menge entsprechend der Kantengewichte). Eine **aktivierte** Transition kann zu einem beliebigen Zeitpunkt **feuern**.

Feuert eine Transition werden auf allen Stellen, die mit eingehenden Kanten verbunden sind, Marken **konsumiert** und auf allen Stellen, die mit einer Ausgehenden Kante verbunden sind werden Marken **produziert**. Auch hier ist die Menge der konsumierten und produzierten Marken gleich des Kantengewichts der dazugehörigen Kante.

Somit kann ein Petri-Netz als Sechs-Tupel [S, T, F, W, m_0] definiert werden mit:

- S - Menge der Stellen
- T - Menge der Transitionen
- F $\subseteq \{S \times T\} \cup \{T \times S\}$ - Menge der Kanten
- W: $F \to \mathbb{N} \setminus \{0\}$ - Menge der Kantengewichte
- m_0: $S \to \mathbb{N} \cup \{0\}$ - Anfangsmarkierung der Stellen

Auf dieser Definition aufbauend können einige wichtige Eigenschaften hergeleitet werden. Die Details sind außerhalb des Umfangs dieser Arbeit, deswegen beschränken wir uns auf eine Informale Beschreibung der drei Wichtigsten.

2.1 Linearität des Schaltens

Petri-Netze lassen sich als lineare Gleichungssysteme darstellen. Dadurch können die Regeln der linearen Algebra auf das System anwenden.

Mittels der **Zustandsgleichung** kann durch einfache Lösung des Gleichungssystems z.B. bereits eine Aussage darüber getroffen werden, welche Markierung `nicht` von der Anfangsmarkierung aus erreicht werden können.

Außerdem lassen sich durch Stellen- und Transitionsinvarianten Aussagen über die Lebendigkeit und Beschränktheit des Systems treffen. Ob ein System Deadlocks hat, ist z.B. eng mit der Lebendigkeit des Netzes verbunden.

2.2 Monotonie des Schaltens

Im Gegensatz zu einigen anderen Formalismen sind Petri-Netze frei von Seiteneffekten. Aus diesem Grund lassen sich, mit dem Wissen der gemachten Aktionen und dem aktuellen Zustand, die vergangenen Zustände berechnen. In Problemen wo die Zustandsabfolge von Interesse ist, kann der benötigte Speicher dadurch deutlich reduziert werden.

Eine weitere Eigenschaft die damit zusammenhängt, ist die Überdeckung von Markierungen. Wenn von einer Markierung eine echt größere erreicht werden kann – also eine Markierung, die auf jeder Stelle mehr (oder gleich viele) Marken hält – so kann der Weg dahin wiederholt werden, um noch mehr Marken zu generieren. Die neue Markierung überdeckt damit die Alte.

Da in der neuen Markierung mindestens dieselben Transitionen aktiviert sind wie in der vorherigen, ist es möglich eine unendliche Menge an Marken anzusammeln. Man muss lediglich denn selben Weg wieder und wieder schalten.

So lassen sich Aussagen über die Beschränktheit des Systems machen und eine ganze Klasse von Markierungen zusammenfassen.

2.3 Lokalität des Schaltens

Die letzte wichtige Eigenschaft, ist die eingangs erwähnte Lokalität von Petri-Netzen. Das Prinzip auf dem Petri seinen Formalismus aufgebaut hat.

In Petri-Netzen lässt sich formal erkennen, ob Aktionen unabhängig voneinander sind. Bei unabhängigen Aktionen spielt es keine Rolle in welcher Reihenfolge sie ausgeführt werden. Ob sich jemand erst die linke Socke anzieht und dann die Rechte, ändert nicht das Ergebnis im Vergleich zur umgekehrten Vorgehensweise (wohingegen die Reihenfolge von Socke und Schuh unterschiedliche Reaktionen hervorrufen dürfte).

Wenn es keine Rolle spielt in welcher Reihenfolge wir Aktionen ausführen, können wir uns eine aussuchen (und die anderen ignorieren). Auf diese Weise können wir die Menge der möglichen Aktionen reduzieren.

Da es gerade in verteilten Systemen viele parallele (und damit unabhängige) Aktionen gibt, ist dieser Ansatz gerade dort eine sehr effektive Möglichkeit der Zustandsexplosion entgegenzuwirken.

All diese Eigenschaften lassen sich durch Methoden, die wir uns im nächsten Kapitel ansehen, wirksam ausnutzen um andere formale Eigenschaften zu verifizieren.

3 Verifikation

Petri Netze eignen sich gut, um verteilte Systeme zu modellieren. Solche Systeme mit einer hohen Anzahl an parallel ausführbaren Aktionen sind allerdings besonders von dem Problem der Zustandsexplosion betroffen.

Trotzdem ist es in einigen Anwendungsfällen wichtig, dass das System gewisse Eigenschaften erfüllt. Die Softwareverifikation ist ein Bereich der Informatik, der sich mit diesem Problem beschäftigt.

Auch Systeme, die als Petri-Netz modelliert sind lassen sich verifizieren. Damit dies effizient ablaufen kann und z.B. die Auswirkungen der Zustandsexplosion gering gehalten werden können, gibt es einige Methoden, die die Eigenschaften, die im vorherigen Kapitel vorgestellt wurden, ausnutzen.

Einen Überblick über einige wichtige sehen wir uns in diesem Kapitel an.

3.1 Endlichkeit durch den Überdeckungsgraph

Verifikation in Form von explizitem Model Checking arbeitet auf dem Zustandsraum eines Systems. Bei dieser Methode werden solange Zustände aufgedeckt bis eine Eigenschaft verifiziert oder falsifiziert ist... Oder bis der Model Checker aus dem Speicher läuft.

Hier ist es also wichtig die Anzahl aufgedeckter Zustände niedrig zu halten. Eine unangenehme Eigenheit einiger Systeme ist jedoch das der Zustandsraum nicht einfach nur explodiert, sondern von vornherein unendlich ist – ärgerlich. Ein einfaches Beispiel wäre ein Programm, das die natürlichen Zahlen aufzählt.

Glücklicherweise bietet uns die Monotonie des Schaltens hier eine Möglichkeit, eine unendliche Menge an Zuständen, in eine endliche Menge von Speicher abzulegen.

Statt alle erreichbaren Zustände aufzulisten, können wir aufhören, wenn wir feststellen das eine Markierung, eine andere überdeckt (siehe 2.2). So brauchen wir nur noch Speichern, dass wir einen Zustand gefunden haben der unbeschränkt ist.

Die passende Datenstruktur hierfür heißt Überdeckungsgraph und ist immer endlich groß.

3.2 Ein Blick in die Zukunft mit der Zustandsgleichung

Für einige Eigenschaften brauchen wir allerdings überhaupt keine Zustände aufzudecken. Insbesondere für die Erreichbarkeit von Zuständen können wir die Linearität des

Schaltens (2.1) für unsere Zwecke nutzen.

Mit hilfe der linearen Algebra und der Zustandsgleichung können wir nur durch Lösen eines linearen Gleichungssystems erkennen, ob ein Zustand unerreichbar ist. Das erfolgreiche Lösen des Gleichungssystems ist zwar nur eine notwendige Bedingung zur Erreichbarkeit. Aber durch Nutzung von meist sehr effizienten Mathematik-Algorithmen, ist es oft Sinnvoll, zunächst herauszufinden ob überhaupt die Möglichkeit besteht einen Zustand zu erreichen. Danach kann man immer noch den vergleichsweise großen Zustandsraum aufdecken.

3.3 Zustandsamnesie mit Sweepline

Die Linearität des Schaltens kommt uns in einer weiteren Methode zu Gute. Wieder um den Zustandsraum zu reduzieren. Bzw. um die benötigten Ressourcen zum Speichern der Zustände zu reduzieren.

Durch einen Fortschrittswert können wir Zustände untereinander in Beziehung setzen. Oft sind neue Zustände nicht mehr direkt abhängig von älteren zuvor aufgedeckten. Wenn wir zusätzlich feststellen, dass es keine Möglichkeit mehr gibt, diese vergangenen Zustände zu erreichen, können wir diese auch ohne schlechtes Gewissen vergessen.

Die frei gewordenen Ressourcen können so für die weitere Suche recycelt werden.

3.4 Symmetrie – nicht nur eine Frage der Optik

Die Graphentheorie und die Lokalität des Schaltens (2.3) unterstützen uns um ein weiteres Mal den Zustandsraum, den wir aufdecken müssen, zu reduzieren.

Da Petri-Netze selbst als Graph aufgefasst werden können, können auch Graphenalgorithmen und Konzepte auf sie angewendet werden. Besonders nützlich sind hier die Graphautomorphismen bzw. die Symmetrie.

Teile des Petri-Netzes können sich zu anderen Teilen symmetrisch verhalten. D.h. wenn ein Teil des Petri-Netzes bestimmte Aktionen ausführen kann, kann es auch ein anderer Teil der sich zum ersten symmetrisch verhält.

Solche Symmetrien lassen sich besonders in verteilten Systemen beobachten, die das gleiche Programm auf verschiedenen (oder allen) Knoten ausführen. Hier kann offensichtlich jeder Knoten exakt die gleichen Aktionen ausführen wie alle anderen.

Standardmäßig kann ein Model Checker diese Information jedoch nicht nutzen und müsste trotzdem das gesamte System überprüfen. Durch das Erkennen von symmetrischen Strukturen im Petri-Netz können wir unsere Rechenleistung jedoch für weniger repetitive Aufgaben nutzen und neben Zeit wieder Speicherplatz einsparen.

3.5 Effizienz durch Ordnung mit Partial-Order-Reduction

Partial-Order-Reduction ist eine weitere Technik um den Zustandsraum durch die Lokalitätseigenschat zu reduzieren. Und wieder wird die Unabhängigkeit von Aktionen ausgenutzt, die besonders durch Parallele Systeme eingeführt wird.

Mit Hilfe von Stubborn-Sets können Zustandsfolgen gefunden werden, die die gleichen Aktionen (in unterschiedlicher Reihenfolge) nutzen und denselben Endzustand erreichen. Hier ist es im Sinne der Verifikation wieder egal, welche Reihenfolge gewählt wird. Das Schalten eines Pfads ist genug um (passende) Eigenschaften zu untersuchen.

3.6 Das Ganze ist mehr als die Summe seiner Teile

Die meisten der zuvor genannten Techniken, haben das Ziel die Menge der Zustände zu reduzieren und dieses immer wieder erwähnte Problem der Zustandsexplosion in den Griff zu bekommen. Die einzelnen Techniken sind allerdings nicht immer auf jedes Problem anwendbar. Oft erhalten sie nur Teile der Netzeigenschaften.

Angenehmer weise lassen sich die Techniken untereinander Kombinieren um die Menge der zu untersuchenden Zustände auf ein Minimum zu reduzieren.

Viele Probleme der Praxis lassen sich so auch ohne Kühlschrankgroße spezial Rechner lösen. Soweit, dass Systeme oft interaktiv zu überprüfen sind.

4 Aktueller Stand

Die Eigenschaften und Techniken die in dieser Arbeit beschrieben wurden sind nur ein kleiner Ausschnitt des Möglichen. Gleichzeitig ist die unterliegende Theorie immernoch nicht vollständig ausgereizt.

So ist es nicht überraschend das in diesem Gebiet weiterhin geforscht wird. Der zu durchsuchende Zustandsraum kann nie klein genug sein.

Ein Bestreben die benutzten Techniken zu optimieren zeigt sich z.B bei der Partial-Order-Reduction[1] oder der Reduktion durch Symmetrie[2]. Und während die Auswertung von Datenstrukturen in Petri-Netzen eher schwierig ist, gibt es auch hier Ansätze diese Probleme in den Griff zu bekommen[3].

Doch auch die besten Reduktionstechniken nützen nichts, solange sie nicht komfortabel zu nutzen sind. Eines der jüngeren Tools in diesem Bereich ist Charlie[4]. Aber auch ältere Tools werden weiter gepflegt und entwickelt, wie sich an LoLA2 zeigt[5].

Literaturverzeichnis

[1] F. M. Bønneland, P. G. Jensen, K. G. Larsen, M. Muniz, and J. Srba, "Start pruning when time gets urgent: Partial order reduction for timed systems," in International Conference on Computer Aided Verification. Springer, 2018, pp. 527–546.

[2] P.-A. Bourdil, B. Berthomieu, S. Dal Zilio, and F. Vernadat, "Symmetry reduction for time petri net state classes," Science of Computer Programming, vol. 132, pp. 209–225, 2016.

[3] D. Xiang, G. Liu, C. Yan, and C. Jiang, "Detecting data inconsistency based on the unfolding technique of petri nets," IEEE Transactions on Industrial Informatics, vol. 13, no. 6, pp. 2995–3005, 2017.

[4] M. Heiner, M. Schwarick, and J.-T. Wegener, "Charlie–an extensible petri net analysis tool," in International Conference on Applications and Theory of Petri Nets and Concurrency. Springer, 2015, pp. 200–211.

[5] K. Wolf, "Petri net model checking with lola 2," in International Conference on Applications and Theory of Petri Nets and Concurrency. Springer, 2018, pp. 351–362.