Model-Checking mit Hilfe regulärer Sprachen

Eike Möhlmann

Universität Oldenburg, FK II - Department für Informatik
Ausarbeitung für das Seminar Beyond First Order Logic
bei Dr. Andreas Schäfer

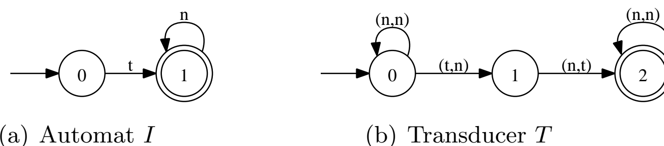
Zusammenfassung: Diese Ausarbeitung beschäftigt sich mit Verfahren zum Modellieren komplexer Systeme mit unendlichen Zustandsmengen, bei denen die Konfigurationen durch Wörter über einem endlichen Alphabet dargestellt werden. Es werdem reguläre Sprachen verwendet, um Mengen von Konfigurationen zubeschreiben. Weiterhin werden Möglichkeiten zum Model-Checking vorgestellt.

1 Einleitung

Die Informatik hat es geschafft, Maschinen viele Aufgaben algorithmisch lösen zu lassen. Doch wie beweist man, dass eine Maschine, welche einen Algorithmus ausführt, die Aufgabe korrekt löst. Dazu werden oft Abläufe eines Systems nur ausprobiert oder man hofft, dass mögliche Fehler nie eintreten. Es gibt aber auch Ansätze, bestimmt Eigenschaften mathematisch zu beweisen, zum Beispiel das sogenannte Model-Checking. Diese Ausarbeitung beschäftigt sich mit solchen Verfahren, welche für Systeme mit einer unendlichen Anzahl von Zuständen (engl. infinite-state systems) eingesetzt werden können. In Abschnitt 2 wird ein bekanntes Beispiel eingeführt, das *token passing protocol*. Die Logic LTL(MSO) ist eine Kombination der monadischen Logik zweiter Stufe (MSO) und linear-time-logic (LTL) und wird in Abschnitt 3 zur Beschreibung von Systemen und Anforderungen eingeführt. Zu LTL(MSO) kann man Automaten konstruieren, die in Abschnitt 4 vorgestellt und in Abschnitt 5 zum Model-Checking verwendet werden.

2 Beispiel

2.1 Token-passing-protocol



(a) Automat I (b) Transducer T

Abbildung 1: Das Token-passing-protocol [AJNS04, S.3]

Das Token-passing-protocol stellt ein System mit endlich vielen Prozessen dar, wobei zu Beginn der Prozess ganz links den Token t hat und alle anderen Prozesse nicht n, der

Zustand lässt sich durch $tnnn\ldots$ beschreiben. Hier wird die Menge der initialen Zustände durch den Automaten in Abbildung 1(a), und die Übergänge, also das Weiterreichen des Tokens, werden durch den Transducer in Abbildung 1(b), dargestellt. Die Menge der initialen Zustände lässt sich durch den regulären Ausdruck tn^* und alle weiteren Zustandsmengen durch n^*tn^* beschreiben.

2.2 Formales Model

Das Token-Passing-Beispiel lässt sich nun auch in LTL(MSO) wie folgt beschreiben:

$$\begin{aligned}
\textbf{initial} &= \forall i(t[i] \leftrightarrow i = 0) \\
\textbf{idle}(i) &= t[i] \leftrightarrow t'[i] \\
\textbf{pass}(i) &= (t[i] \wedge \neg t'[i]) \wedge (\neg t[i+1] \wedge t'[i+1]) \\
&\quad \wedge \forall j((j \neq i+1 \wedge j \neq i) \rightarrow \textbf{idle}(j)) \\
\textbf{system} &= \textbf{initial} \wedge \Box(\exists i\ \textbf{pass}(i) \vee \forall i\ \textbf{idle}(i))
\end{aligned}$$

Die genaue Erklärung der Logik erfolgt in Abschnitt 3.3. An dieser Stelle sei schon beschrieben, dass $t[i]$ bedeutet, dass der Prozess i zu einem Zeitpunkt $\mathfrak{t}$ das Token t besitzt und $t'[i]$ zum Zeitpunkt $\mathfrak{t}+1$. Sowie das System lässt sich in LTL(MSO) auch eine Sicherheitseigenschaft, wie auch Fariness- und Terminierungs- Eigenschaften beschreiben. Hier zum Beispiel die Sicherheitseigenschaft:

$$\textbf{safety} = \Box\neg\exists i, j(i \neq j \wedge t[i] \wedge t[j])$$

Diese Formel bedeutet, dass immer gilt, dass es keine Konfiguration gibt, so dass zwei verschiedene Prozesse (i, j) gleichzeitig den Token haben.

3 LTL, MSO und LTL(MSO)

3.1 LTL

Die Logik linearer Zeit (engl. Linear-time Logic - kurz LTL) hat folgende Syntax:

$$\begin{aligned}
\varphi \quad ::= \quad & \mathit{true} \mid \mathit{false} \mid \\
& \Box\varphi \mid \Diamond\varphi \mid \bigcirc\varphi \mid \varphi_1 \mathbb{U} \varphi_2 \mid \\
& \varphi_1 * \varphi_1,\ \text{wobei } * \in \{\wedge, \vee, \rightarrow, \leftrightarrow\} \mid \neg\varphi
\end{aligned}$$

$\Box\varphi$ bedeutet, dass φ immer (engl. always) gilt. $\Diamond\varphi$ bedeutet, dass φ irgendwann (engl. eventually) gilt. $\varphi_1 \mathbb{U} \varphi_2$ bedeutet, dass φ_1 gilt bis einmal (engl. until) φ_2 gilt.

$\Box$ und $\Diamond$ sind nur Abkürzungen und können durch $\mathbb{U}$ dargestellt werden, den $\mathbb{U}$-Operator werden wir aber nicht weiter benötigen und er ist, sowie $\bigcirc$, nur zu vollständigkeit aufgeführt. [KPoR98, S.4]

3.2 MSO

Monadische Logik Zweiter Stufe (engl. Monadic Second Order Logic - kurz MSO) ist eine Erweiterung der Logik erster Stufe um Mengen. Die Syntax von MSO lässt sich induktiv so beschreiben:

$$\begin{aligned} Form &::= \top \mid \bot \mid \neg F \mid (F) \mid F * G \mid p(t_1, \ldots, t_n) \\ Term &::= v \mid V \mid f(t_1, \ldots, t_n) \\ \varphi &::= \exists x\varphi \mid \forall x\varphi \mid F \end{aligned}$$

Hier sind $F, G \in Form, t_1, \ldots, t_n \in Term, * \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$, $v \in Var_{FO}$, $V \in Var_{SO}$, $x \in Var_{FO} \cup Var_{SO}$, $p \in Präd$ und $f \in Func$. Außerdem sind Var_{FO} und Var_{SO} abzählbar unendliche Menge von Variablen erster bzw. zweiter Ordnung und $Präd$ und $Func$ Mengen von Prädikaten bzw. Funktionen. Hier sei zu beachten, dass die Mengen $Präd, Func, Var_{FO}, Var_{SO}$ immer paarweise disjunkt sind. Die Erweiterung lässt sich nun wie folgt beschreiben: Es gibt kleine Buchstaben Var_{FO} die Variablen erster Ordnung (engl. first order), also Elemente die mit Werten des Grundbereichs belegt werden, und große Buchstaben Var_{SO} die Variablen zweiter Ordnung (engl. second Order), also Mengen von Elementen die mit Werten des Grundbereichs belegt werden, beschreiben. Um nun auf die Elemente einer solchen Menge zugreifen zu können, wird die Syntax erweitert, um $X(x)$ mit der Bedeutung $X(x) == \mathit{true}$, wenn $x \in X$ und sonst $X(x) == \mathit{false}$. [RS97, S. 395f]

3.3 Verknüpfung LTL(MSO)

Die Verknüpfung von LTL und MSO wird nun so vorgenommen, dass man temporale Aussagen statt über Formeln der Aussagenlogik über Formeln der monadischen Logik zweiter Stufe macht. Hier werden nun kleine Buchstaben $i, j, k, \ldots$ als Variablen erster Ordnung (meist als Indizes für Konfigurationen) und $x, y, y, \ldots$ als Konfigurationsvariablen genutzt. Weiterhin werden, wie bei MSO auch, Großbuchstaben als Variablen zweiter Ordnung verwendet. Wir beschreiben die Syntax induktiv (siehe Abbildung 2).

$$\begin{aligned} \varphi \quad ::= \quad & i \in I \mid I \subseteq J \mid i = j + 1 \mid i = j \mid x[i] \mid x'[i] \mid \\ & \mathit{true} \mid \mathit{false} \mid \Box\varphi \mid \Diamond\varphi \mid \varphi_1 * \varphi_2 \mid \neg\varphi \ , \end{aligned}$$

Abbildung 2: Induktive Definition der Syntax LTL(MSO) [AJN+04, S. 4]

wobei $* \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$. Formeln in LTL(MSO) werden über Matrizen interpretiert. Diese Matrizen haben die Dimension $\infty \times n$ mit $n \in \mathbb{Z}_+$. Dabei beschreibt die eine Dimension die Zeit (engl. Time) und die andere Dimension den Raum (engl. Space) und gibt somit die Position in der Konfiguration an.

Hier werden $\mathsf{t} \in Time$ als Zeit und $\mathfrak{I}$ als Belegung von Variablen erster und zweiter Ordnung benutzt. So beschriebt nun eine Matrix M den Verlauf einer Variable i. [AJN+04, S.4]

$$
\begin{aligned}
i \in I &\quad \text{gdw.} \quad \mathfrak{I}(i) \in \mathfrak{I}(I) \\
I \subseteq J &\quad \text{gdw.} \quad \mathfrak{I}(I) \subseteq \mathfrak{I}(J) \\
i = j + 1 &\quad \text{gdw.} \quad \mathfrak{I}(i) = \mathfrak{I}(j) + 1 \\
x[i] &\quad \text{gdw.} \quad x \in M(\mathsf{t}, \mathfrak{I}(i)) \\
x'[i] &\quad \text{gdw.} \quad x \in M(\mathsf{t}+1, \mathfrak{I}(i))
\end{aligned}
$$

4 Büchi-Transducer

4.1 Büchi-Automaten

Büchi-Automaten sind ω-Automaten, die ω-Worte mit Büchi-Bedingung akzeptieren. Ein ω-Automat wird beschrieben durch

$$\mathcal{A} = (Q, \Sigma, q_0, \Delta, Acc)$$

- Q ist die Menge der Zustände
- Σ ist das Eingabealphabet
- q_0 der Initialzustand
- Δ die Menge der Transitionen der Form $Q \times \Sigma \times Q$
- Acc eine Akzeptanzbedingung

Bei einem Büchi-Automaten besteht die Akeptanzbedingung Acc aus einer Menge von Endzuständen $F \subseteq Q$. Wörter werden genau dann akzeptiert, wenn der Automat dabei **mindestens einen** Endzustand unendlich oft durchläuft.

Büchi-Automaten sind abgeschlossen gegen Vereinigung, Schnitt und Komplement. Außerdem ist das Leerheitsproblem entscheidbar. Wir beschränken uns hier auf deterministische Büchi-Automaten, um die Komplexität gering zu halten. Wir verwenden sie, um z.B. die Menge der Zustände (Worte) eines Systems zu beschreiben, wobei die Menge die von dem Automaten akzeptierte Sprache ist.

4.2 Transducer

Transducer sind spezielle Büchi-Automaten (siehe Abschnitt 4.1), nur besitzen sie noch ein weiteres Alphabet, das Ausgabealphabet Γ.

$$\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta, Acc)$$

Da Zustände des Systems druch Sprachen beschrieben werden, können Transducer nun durch Transformation auf Sprachen den Übergang eines Systems beschreiben.

5 Verifikation

5.1 Problemstellung

Nun können wir auch Zustände definieren, welche die Safety-Eigenschaft nicht erfüllen. Die Menge dieser Zustände nennen wir B. Bezeichnen wir weiterhin die Menge der initialen Zustände mit I und den Transducer mit T (Abbildung 1(b)), so können wir durch Prüfen der Aussage $(I \circ T^*) \cap B = \emptyset$ feststellen, ob die Sicherheitseigenschaft verletzt ist, das heißt ob wir ein Zustand aus der Menge B von einem Zustand aus I über die Iteration von T^* erreichbar ist. Wir können dieses Problem lösen, indem wir die Menge $Inv = I \circ T^*$ berechnen und prüfen, ob es einen Schnitt mit der Menge B gibt.

Zum Lösen des Problems, benötigen wir nun algorithmische Verfahren um T^* bestimmen zu können. Hier werden zwei Verfahren vorgestellt. Grundlegend ist dabei die Quotientenbildung, also das Zusammenfassen von äquivalenten Zuständen. Dabei wird je nach verwendeter Äquivalenzrelation die Sprache vergrößert. Das erste Verfahren ist aufgrund einer Produktkonstruktion und einem Verfahren zum Finden der Äquivalenzrelation, zur Vermeidung der Sprachenvergrößerung, sehr aufwendig. Ein weiteres Problem ist, dass beim Finden der Äquivalenzrelation die Abbruchbedingung nicht immer erfüllt ist. Das zweite Verfahren ist durch direkte Anwendung der Quotientenbildung weniger aufwendig und kann trotzdem bei leerem Schnitt sofort sagen, dass die Safty-Eigenschaft nicht verletzt ist.

5.2 Lösung - durch Quotientenbildung

Im ersten Schritt wird eine Produktkonstruktion verwendet, um T^n für $n = 1, 2, 3, \ldots$ zu bilden, die Zustandsname ergeben sich dabei aus den durchlaufenen Zuständen in den verschiedenen Iterationen des Transdurcers (Abbildung 1(b)). Der History-Transducer ist nun die Vereinigung dieser Produktautomaten (siehe Abbildung 3) und beschreibt T^+. Da dieser History-Transducers unendlich viele Zustände hat, wird im zweiten Schritt,

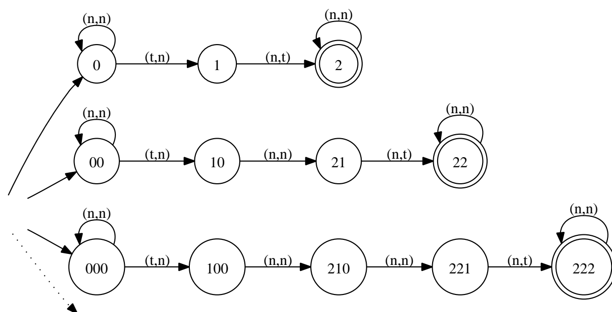


Abbildung 3: Produkt-Konstruktion des History-Transducers für T^+ [AJNS04, S.8]

der eigentlichen Quotientenbildung, eine Äquivalenzrelation $\simeq$ auf der Zustandsmenge definiert und der Restklassenautomat betrachtet. Das Verfahren und die Verbesserun-

gen zum Bestimmen der Äquivalenzrelation wird in [DLS01], [AJNd02] und [AJNd03] beschrieben. Die Idee dabei ist, Vorwärts- und Rückwärts-Bisimulationen zu kombinieren, um äquivalente Zustände zufinden. In diesem Beispiel ergeben sich dadurch die Äquivalenzklassen $[0^+]$, $[10^+]$, $[2^+10^+]$, $[1]$ $[2^+1]$ und $[2^+]$, [AJNS04, S.7f] die der Äquivalenzklassenautomat (siehe Abbildung 4) als Zustandsmenge hat.

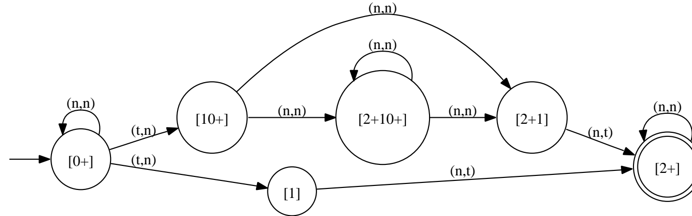
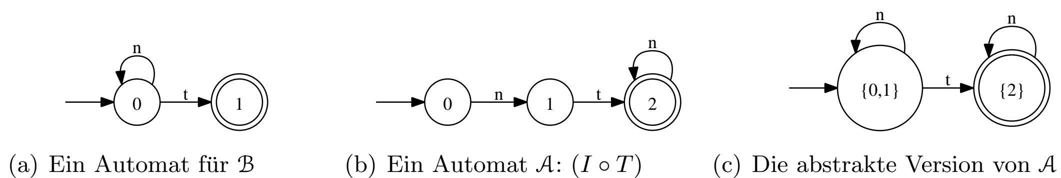


Abbildung 4: Äquivalenzklassen-Konstruktion des History-Transducers [AJNS04, S.8]

5.3 Lösung - durch Abstraktion

Bei der Abstraktion werden die Äquivalenzrelationen anders gebildet. Um eine geeignete Äquivalenzrelation $\simeq$ zu finden, werden nun Post-Sprachen verwendet. Eine Post-Sprache $\mathcal{L}(\mathcal{A}, q)$ zu einem Automat $\mathcal{A}$ (siehe Abbildgun 5(b)) und dem Zustand q von $\mathcal{A}$, beinhaltet alle Wörter, die ab dem Zustand q wie sonst bei Büchi-Automaten akzeptiert werden. So das nun gilt $q \simeq q'$, wenn für alle Zustände r des Automaten $\mathcal{B}$ (siehe Abbildung 5(a)) gilt, dass $\mathcal{L}(\mathcal{A}, q) \cap \mathcal{L}(\mathcal{B}, r) = \emptyset$ genau dann wenn $\mathcal{L}(\mathcal{A}, q') \cap \mathcal{L}(\mathcal{B}, r) = \emptyset$. Aufbauend auf dieser Äquivalenzrelation lässt sich die Sprache $\mathcal{L}(T^{lim})$ (der Automat dazu siehe Abbildung 5(c)) durch eine Sequenz $(((I \circ T)/\simeq) \circ T)/\simeq ...$ bilden. Dabei wird die akzeptierte Sprache vergrößert. Also einsteht dadurch eine Sprache $\mathcal{L}(T^{lim})$ mit der Eigenschaft $\mathcal{L}(I \circ T^*) \subseteq \mathcal{L}(T^{lim})$. Nun kann getestet werden, ob $T^{lim} \cap B = \emptyset$. Ist dies der Fall, dann gilt durch die Eigenschaft von $\mathcal{L}(T^{lim})$ auch, dass $(I \circ T^*) \cap B = \emptyset$. Andernfalls wird geprüft, ob das Gegenbeispiel, also der Schnitt, auch im original Automaten nachvollzogen werden kann, sonst muss eine feinere Äquivalenzrelation gefunden werden und das Verfahren wiederholt werden.



(a) Ein Automat für $\mathcal{B}$ (b) Ein Automat $\mathcal{A}$: $(I \circ T)$ (c) Die abstrakte Version von $\mathcal{A}$

Abbildung 5: Beispiel einer Abstraktion [AJNS04, S.9]

6 Fazit

Wir haben gesehen, dass es durch grundlegende Mittel wie reguläre Ausdrücke, Automaten und Sprachen möglich ist, Systeme mit einer unendlichen Anzahl von Zuständen beschreiben zu können und Anforderungen zu definieren. Auch können wir durch kombinieren und einsetzen, teilweise schon bekannter Techniken, die Verifikation der Eigenschaften computerunterstützt ablaufen lassen. Es ist ein weiterer Schritt in Richtung korrekter Systeme, bei denen das Verhalten bekannt und gewollt ist.

Literatur

[AJN+04] Parosh Abdulla, Bengt Jonsson, Marcus Nilsson, Julien d'Orso und Mayank Saksena. Regular Model Checking for LTL(MSO). In: Rajeev Alur und Doron A. Peled, Herausgeber, *Computer Aided Verification*, Band 3114 von *Lecture Notes in Computer Science*, S. 348–360. Springer-Verlag, Berlin, 2004.

[AJNd02] Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson und Julien d'Orso. Regular Model Checking Made Simple and Efficient. In: *Proc. CONCUR 2000-12th Int. Conf. on Concurrency Theory*, Band 2421 von *Lecture Notes in Computer Science*, S. 116–130, 2002.

[AJNd03] Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson und Julien d'Orso. Algorithmic Improvements in Regular Model Checking. In: *Proc. CAV '03-15th Int. Conf. on Computer Aided Verification*, Band 2725 von *Lecture Notes in Computer Science*, S. 236–248, 2003.

[AJNS04] Parosh Abdulla, Bengt Jonsson, Marcus Nilsson und Mayank Saksena. A Survey of Regular Model Checking. In: Philippa Gardner, Herausgeber, *CONCUR 2004 - Concurrency Theory: 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings.*, Band 3170 von *Lecture Notes in Computer Science*, S. 35–48. Springer, 2004.

[DLS01] Dennis Dams, Yassine Lakhnech und Martin Steffen. Iterating Transducers. In: *CAV '01: Proceedings of the 13th International Conference on Computer Aided Verification*, S. 286–297, London, UK, 2001. Springer-Verlag.

[KPoR98] Yonit Kesten, Amir Pnueli und Li on Raviv. Algorithmic Verification of Linear Temporal Logic Specifications. In: *ICALP*, S. 1–16, 1998.

[RS97] Grzegorz Rozenberg und Arto Salomaa, Herausgeber. *Handbook of Formal Languages, vol. 3: Beyond Words.* Springer-Verlag New York, Inc., New York, NY, USA, 1997.