Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Synthese energieeffizienter Steuerungen für moderne Produktionsanlagen aus szenariobasierten Spezifikationen

Masterarbeit

im Studiengang Informatik

von

Florian Werner

Prüfer: Prof. Dr. Joel Greenyer
Zweitprüfer: Prof. Dr.-Ing. Bernardo Wagner
Betreuer: Prof. Dr. Joel Greenyer

Hannover, 3. Juni 2015

Zusammenfassung:

Durch stetig steigende Strompreise wird die Energieeffizienz von Produktionsanlagen zu einem immer wichtigerem Thema. Die Energieeffizienz einer Produktionsanlage kann unter anderem dadurch verbessert werden, indem die Energie von Bremsvorgängen in elektrische Energie umgewandelt und anschließend wiederverwendet wird. Damit die Energie jedoch effektiv genutzt werden kann, muss zeitgleich ein Vorgang aktiv sein, der die Energie nutzen kann. Somit könnte durch die Umstrukturierung der Ablaufsteuerung die Energieeffizienz einer Produktionsanlage gesteigert werden.
Mit Hilfe des Ansatzes der szenariobasierten Spezifikation kann eine Produktionsanlage beschrieben werden, ohne dabei eine feste Ablauffolge vorzugeben. Auf der Basis so einer Spezifikation könnte eine Ablaufsteuerung synthetisiert werden, die so viel Bremsenergie, wie möglich wiederverwendet.
In dieser Arbeit werden zwei Lösungsvorschläge vorgestellt, mit denen es möglich ist, eine Ablaufsteuerung aus einer szenariobasierten Spezifikation zu synthetisieren mit der so viel Bremsenergie, wie möglich wiederverwendet werden kann.

Abstract

Due to constantly rising electricity prices, energy efficiency is becoming increasingly important in production systems. The energy efficiency of a production system can be improved by converting energy of braking processes to electrical energy and reusing it. To reuse the energy effectively, there must be an additional process at the same time. The production system could become more energy efficient by reordering the control logic. Through scenario-based specification, a production system can be described without a fixed control logic. Thus, an energy efficient control logic could be synthesized from the specification. In this thesis, there are two suggested solutions presented. With the proposed solutions, it is possible to synthesize an energy efficient control logic from a scenario-based specification.

Inhaltsverzeichnis

1 Einführung

Durch stetig steigende Strompreise wird die Energieeffizienz von Produktionsanlagen zu einem immer wichtigerem Thema. Eine Möglichkeit, um eine moderne Produktionsanlage energieeffizienter zu gestalten besteht darin, die Bremsenergie, die zum Beispiel beim Abbremsen von einem Roboterarm entsteht, wieder in elektrische Energie umzuwandeln und die zurückgewonnene Energie zu verwenden, um damit zum Beispiel einen anderen Roboterarm zu beschleunigen. Auf diese Weise kann der Energiebedarf verringert werden, den die Produktionsanlage aus einer externen Energiequelle beziehen muss.

1.1 Problemstellung

Das Problem ist jedoch, dass die generierte Energie von Bremsvorgängen direkt wieder verbraucht werden muss, damit die Energie auch effizient genutzt werden kann. Das bedeutet, dass die Bremsenergie eines Bremsvorgangs nur in den Situationen vollständig genutzt werden kann, in denen ein zweiter Vorgang aktiv ist, der die Energie direkt wiederverwenden kann.

Für die meisten Produktionsanlagen gibt es einen gewissen Spielraum, in welcher Reihenfolge bestimmte Vorgänge ausgeführt werden können. Zum Beispiel gibt es oft Vorgänge, dessen Durchführung nicht sofort erforderlich sind, sondern auch zu einem späteren Zeitpunkt ausgeführt werden können. Solche Vorgänge könnten so angeordnet werden, dass in der Folge öfters Situationen entstehen, in denen sowohl ein Vorgang, der Energie verbraucht, als auch ein Vorgang, der Energie generiert, gleichzeitig aktiv sind. Bei der richtigen Anordnung aller Vorgänge könnte die Energie, die wiederverwendet werden kann, somit maximiert und der Energiebedarf der Produktionsanlage auf ein Minimum reduziert werden. Bei komplexeren Systemen ist es jedoch extrem schwierig zu erkennen, welche Möglichkeiten der Anordnung es gibt und welche davon den geringsten Energiebedarf besitzen. Vor allem muss immer darauf geachtet werden, dass die so gewählte Anordnung gegen keine der geforderten Anforderungen verstößt und die Produktionsanlage somit immer noch ihre gewünschte Funktionsweise beibehält.

1.2 Lösungsansatz

Da an jede Produktionsanlage gewisse Anforderungen gestellt werden, ist es vor allem bei komplexeren System extrem aufwändig eine Ablaufsteuerung manuell zu erstellen, die alle Vorgaben einhält. Deswegen wird seit einer gewissen Zeit an dem Konzept der szenariobasierten Spezifikation geforscht. So eine szenariobasierte Spezifikation kann mit Hilfe von Modal Sequence Diagrams(MSDs) [5] erstellt werden. So eine Spezifikation besteht dabei aus mehreren Szenarien, die jeweils beschreiben, was das System in einer bestimmten Situation tun soll, tun muss oder nicht tun darf. Auf diese Weise kann das geforderte Verhalten einer Produktionsanlage beschrieben werden, ohne explizit festzulegen in welcher Reihenfolge dies geschehen soll. Ein Algorithmus kann dann auf Basis der szenariobasierten Spezifikation eine Ablaufsteuerung finden, die alle Anforderungen einhält und bei der es möglich ist, so viel Energie wie möglich wiederzuverwenden. Für MSD Spezifikationen gibt es bereits Algorithmen, die eine gültige Ablaufsteuerung synthetisieren können, jedoch wird bei der Synthese nur darauf geachtet, dass alle Anforderungen eingehalten werden, nicht jedoch darauf, dass energieerzeugende und energieverbrauchende Vorgänge zeitlich aufeinander abgestimmt werden.

Im Rahmen dieser Masterarbeit soll deswegen ein Algorithmus entwickelt werden, der eine Ablaufsteuerung aus einer szenariobasierten Spezifikation synthetisiert, die das größte Potential für die Wiederverwendung von Bremsenergie besitzt. Das heißt, es wird versucht eine Ablaufsteuerung zu finden, bei denen energieverbrauchende und energieerzeugende Vorgänge zeitlich so aufeinander abgestimmt werden, dass somit die maximal mögliche Energie innerhalb des Systems direkt wiederverwendet werden kann, ohne dabei die Anforderungen der Spezifikationen zu verletzen. Auf diese Weise kann der Energiebedarf, der von einer externen Energiequelle bezogen werden muss, reduziert werden.

Da eine Produktionsanlage auch immer ein Ziel verfolgt, wie zum Beispiel die Fertigstellung eines Produkts, wird im Rahmen dieser Masterarbeit die Energieeffizienz einer Ablaufsteuerung anhand des durchschnittlichen Energiebedarfs von einer externen Energiequellen pro Zielerreichung gemessen.

1.3 Gliederung der Arbeit

In Kapitel 2 werden alle Grundlagen erklärt, die als Basiswissen für die weiteren Kapitel verwendet werden.

In Kapitel 3 werden verwandte Arbeiten vorgestellt, die sich mit Wiederverwendung von Bremsenergie und der Synthese einer Ablaufsteuerung befassen.

In Kapitel 4 wird beschrieben wie eine MSD Spezifikation modelliert werden muss, damit die entwickelten Algorithmen eine energieeffiziente Ablaufsteuerung synthetisieren können.

In Kapitel 5 und 6 wird jeweils ein Konzept für einen möglichen Algorithmus beschrieben, der eine energieeffiziente Ablaufsteuerung aus einer MSD Spezifikation synthetisieren kann.

In Kapitel 7 werden ein paar Details über die Implementierung des Prototyps genannt.

In Kapitel 8 wird die Evaluierung des Prototyps beschrieben.

In Kapitel 9 befindet sich eine Zusammenfassung und ein Fazit zu den entwickelten Algorithmen.

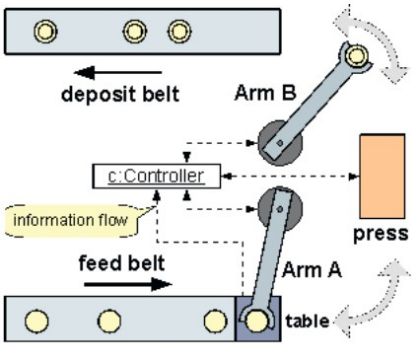
In Kapitel 10 wird ein Ausblick gegeben, welche Problemstellungen noch offen sind und wie die entwickelten Algorithmen noch erweitert werden können.

2 Grundlagen

In diesem Kapitel werden die wesentlichen Konzepte einer MSD Spezifikation vorgestellt. Die Beschreibungen orientieren sich dabei an den Beschreibungen von Greenyer et al. [4].

2.1 Einführendes Beispiel

Zunächst wird an dieser Stelle als einführendes Beispiel eine Produktionsanlage vorgestellt, mit dessen Hilfe die wesentlichen Konzepte einer MSD Spezifikation Schritt für Schritt erklärt werden sollen. Das Beispiel wurde von Greenyer et al. [4] übernommen, da die wesentlichen Konzepte anhand dieses Beispiels leicht zu erklären und somit auch leichter zu verstehen sind.



Arm A:

R1) When a blank plate arrives at the table, Arm A must pick it up, move it to the press, and release it into the press. The arm then has to move back to the table, where it must arrive before the next blank arrives.

A1) If Arm A is ordered to move to the press, it will eventually arrive at the press

A2) If Arm A is ordered to move to the table, it will eventually arrive at the table

A3) If Arm A picks up a blank, moves to the press, and returns to the table, the next blank will not arrive before Arm A has returned to the table.

Press:

R2) When Arm A releases the blank into the press, the press must press. The next blank must not be released before the pressing finished.

A4) The press finishes pressing before Arm A returns to the table.

Arm B:

R4) After the press has finished pressing, Arm B must pick up the pressed plate and move to the deposit belt... [*further specification omitted for brevity*]

Abbildung 1: Produktionsanlage und die entsprechende textuelle Spezifikation (Bild entnommen aus Greenyer et al. [4, S.390])

Die Produktionsanlage, die hier als Beispiel verwendet werden soll, besteht aus zwei Roboterarmen (ArmA und ArmB), zwei Fließbändern, einer Presse, einem Tisch-Sensor und einem Controller (siehe auch die Skizze auf der linken Seite in Abbildung 1). Der Controller der Produktionsanlage ist dafür zuständig, die Reihenfolge der verschiedenen Aktionen, die durchgeführt werden müssen, festzulegen. Dafür kommuniziert der Controller mit den beiden Roboterarmen, der Presse und dem Tisch-Sensor mit Hilfe von Nachrichten.

Die Aufgabe des Tisch-Sensors besteht darin, dem Controller mitzuteilen, wenn ein neuer Rohling den Tisch über das untere Fließband erreicht hat.

Steht ein Rohling zur Abholung bereit, dann muss ArmA den Rohling aufheben, zur Presse befördern und bei der Presse wieder ablegen. Dabei wird davon ausgegangen, dass solange kein neuer Rohling erscheint, bis ArmA wieder zum Tisch zurückgekehrt ist. Die Presse bringt den Rohling dann anschließend in die gewünschte Form. ArmB holt dann das Endprodukt von der Presse ab und befördert es zu einem zweiten Fließband, von wo aus das Endprodukt das System wieder verlässt. Die beiden Roboterarme und die Presse teilen dabei dem Controller immer wieder durch eine Nachricht mit, wenn ein Schritt erledigt wurde. Auf Basis dieser Nachrichten entscheidet der Controller, was als nächstes ausgeführt wird. Auf der rechten Seite von Abbildung 1 befindet sich die entsprechende Spezifikation der Produktionsanlage aufgeschlüsselt in Anforderungen und Annahmen.

2.2 Modal Sequence Diagrams (MSDs)

Wie bereits in der Einleitung erwähnt wurde, kann eine szenariobasierte Spezifikation mit Hilfe von sogenannten Modal Sequence Diagrams (MSDs) erstellt werden, dessen Konzept von Harel und Maoz [5] entwickelt wurde. MSDs basieren wiederum auf dem Konzept von Live Sequence Charts (LSCs) [2].

Mit Hilfe von MSDs ist es möglich, die verschiedenen Szenarien, die in einer Produktionsanlage auftreten können, zu modellieren und somit das Verhalten in den verschieden Situationen zu beschreiben. Ein MSD besteht dabei aus mehreren *Lifelines*, die jeweils ein Objekt der Produktionsanlage repräsentieren. Die *Lifelines* bzw. die Objekte können untereinander mit Hilfe von Nachrichten kommunizieren. Es wird dabei zwischen zwei Arten von Objekten unterschieden. Zum einen gibt es die *System-Objekte*, wie zum Beispiel den Controller und zum anderen die *Umwelt-Objekte*, wie zum Beispiel die Presse, der TischSensor und die beiden Roboterarme.

Es wird davon ausgegangen, dass das Verhalten der Umwelt nicht steuerbar ist und die Nachrichten der *Umwelt-Objekte* somit willkürlich auftreten können. Deswegen werden Nachrichten von *Umwelt-Objekten* auch als *unkontrollierbar* bezeichnet. Nachrichten, die hingegen von einem System-Objekt verschickt werden, bezeichnet man als *kontrollierbar*.

Doch auch wenn die Umwelt generell nicht steuerbar ist, können gewisse Annahmen getroffen werden, wie sich die Umwelt in gewissen Situationen verhält. Zum Beispiel kann für das einführende Beispiel die Annahme getroffen werden, dass wenn ArmA zur Presse geschickt wird, ArmA auch irgendwann bei der Presse ankommt und eine entsprechende Mitteilungs-Nachricht verschickt. Um solche Annahmen zu modellieren gibt es

sogenannte Assumptions MSDs. Für die Modellierung von Anforderungen werden hingegen Requirement MSDs verwendet.

In der folgenden Abbildung befindet sich auf der linken Seite ein Beispiel für ein Requirement MSD und auf der rechten Seite ein Beispiel für ein Assumption MSD. Das dargestellte Assumption MSD modelliert die eben beschriebene Annahme.

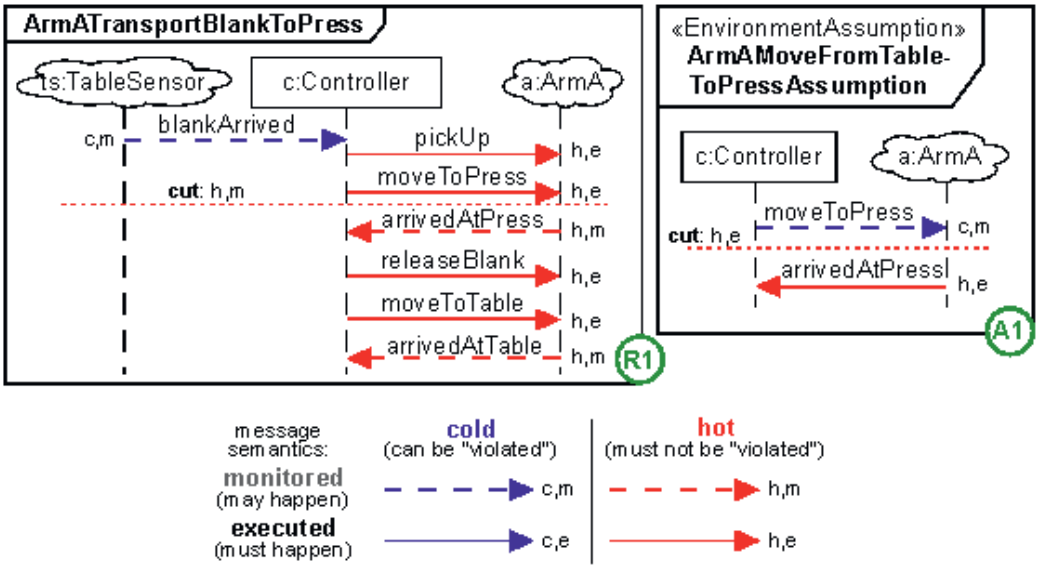


Abbildung 2: Requirement MSD und Assumption MSD (Bild entnommen aus Greenyer et al. [4, S.391])

Das in der Abbildung 2 dargestellte Requirement MSD ***ArmATransport-BlankToPress*** modelliert die beschriebenen Aufgaben, die ArmA ausführen muss. Das MSD besitzt drei Lifelines. Eine Lifeline für den TischSensor, eine für den Controller und eine für ArmA. Die Pfeile zwischen den Lifelines stellen die Nachrichten dar, die zwischen den Objekten verschickt werden. Ein auftretendes Nachrichten-Event kann einer Nachricht aus einem MSD zugeordnet werden, falls das Nachrichten-Event und die Diagramm-Nachricht den gleichen Namen besitzen und das sendende und das empfangende Objekt des Events identisch sind mit dem sendenden und empfangenden Objekt der Nachricht, die durch die entsprechenden Lifelines dargestellt werden. Tritt ein Event auf, das der ersten Nachricht eines MSDs zugeordnet werden kann, wird eine aktive Kopie des entsprechenden MSDs erzeugt. Tritt im Folgenden dann ein Event auf, das der nächsten Nachricht des MSDs entspricht, schreitet das MSD voran. Der aktuelle Zustand des MSD wird dabei durch den sogenannten *Cut* gekennzeichnet. Er stellt eine imaginäre Grenze zwischen den Nachrichten dar, die schon aufgetreten sind und denen, die noch nicht aufgetreten sind. Eine Nachricht, die sich hinter einem *Cut* befindet, wird auch *enabled* genannt. In der Abbildung 2 wird der *Cut* durch eine rot gestri-

chelte Linie dargestellt. Erreicht der Cut das Ende eines MSDs, wird das entsprechende MSD beendet.

Jede Nachricht in einem MSD besitzt eine *temperature*. Eine Nachricht ist rot gefärbt, wenn die temperature der Nachricht *hot* ist und blau gefärbt, wenn die temperature der Nachricht *cold* ist. Zusätzlich besitzt eine Nachricht noch einen *execution kind*, der entweder *monitored* oder *executed* ist. Der Pfeil einer Nachricht wird gestrichelt dargestellt, wenn der *execution kind* der Nachricht *monitored* ist und er wird durchgezogen dargestellt, falls der *execution kind* der Nachricht *executed* ist. In der Abbildung 2 befindet sich unterhalb der MSDs eine Tabelle mit den grafischen Darstellungen der verschiedenen Nachrichtenarten.

Der Cut in einem MSD hat immer die gleiche *temperature* und den gleichen *execution kind* wie die Nachricht, die momentan *enabled* ist.

Ist eine Nachricht *monitored*, dann bedeutet das, dass diese Nachricht irgendwann auftreten kann, aber nicht auftreten muss. Wenn eine Nachricht hingegen *executed* ist, dann muss diese Nachricht irgendwann auftreten. Wenn eine Nachricht *hot* ist und sie momentan *enabled* ist, dann darf keine andere Nachricht aus dem entsprechenden MSD vor dieser Nachricht auftreten. Bei einer Nachricht, die *cold* ist, gilt diese Einschränkung nicht. Eine Nachricht, die *enabled* und *executed* ist, wird auch als *aktive* Nachricht bezeichnet.

Die erste Nachricht vom MSD *ArmATransportBlankToPress* zum Beispiel ist *cold* und *monitored*. Dadurch, dass die Nachricht *monitored* ist, wird angezeigt, dass es möglich ist, dass diese Nachricht irgendwann auftritt, jedoch nicht auftreten muss. Sobald das Event auftritt, schreitet das MSD voran und die Nachricht *pickUp* wird *enabled*. Da die Nachricht *pickUp executed* ist, muss sie irgendwann auftreten. Zusätzlich ist die Nachricht auch noch *hot*, und somit darf keine andere Nachricht die zu dem gleichen MSD gehört vor dieser Nachricht auftreten. Sobald diese Nachricht dann auftritt, schreitet das MSD wieder voran und die Nachricht *moveToPress* wird *enabled*. Die Nachricht *moveToPress* ist ebenfalls *hot* und *executed*. Wenn im Folgendem dann diese Nachricht auftritt, dann schreitet zum einen das MSD *ArmATransportBlankToPress* voran und zum anderen wird auch eine aktive Kopie von dem Assumption MSD *ArmAMoveFromTableToPressAssumption* erzeugt, da *moveToPress* die erste Nachricht des Assumptions ist. Somit wären nun zwei MSDs aktiv und es müsste nun im weiteren Verlauf auf beide geachtet werden.

Nachrichten können des Weiteren auch einen Parameter besitzen, denen ein konkreter Wert zugewiesen werden kann. Es ist jedoch auch möglich einem Parameter keinen Wert zuzuweisen, dann hat der Parameter nur einen symbolischen Zweck.

2.2.1 Negation

MSD Spezifikationen bieten auch die Möglichkeit Annahmen zu modellieren, wie sich die Umwelt in einer Situation nicht verhält. Zum Beispiel wurde in dem einführenden Beispiel angenommen, dass kein neuer Rohling erscheint, bis ArmA wieder zum Tisch zurückgekehrt ist. In der Abbildung 3 ist ein MSD Assumption zu sehen, die diese Annahme modelliert. In dem dargestellten MSD befindet sich die Nachricht, die nicht auftreten darf, in dem Rechteck mit der Bezeichnung ***neg*** in der oberen linken Ecke. Für eine MSD Spezifikation bedeutet dies, dass solange eine Kopie von diesem MSD aktiv ist, die Nachricht *blankArrived* nicht auftreten darf.

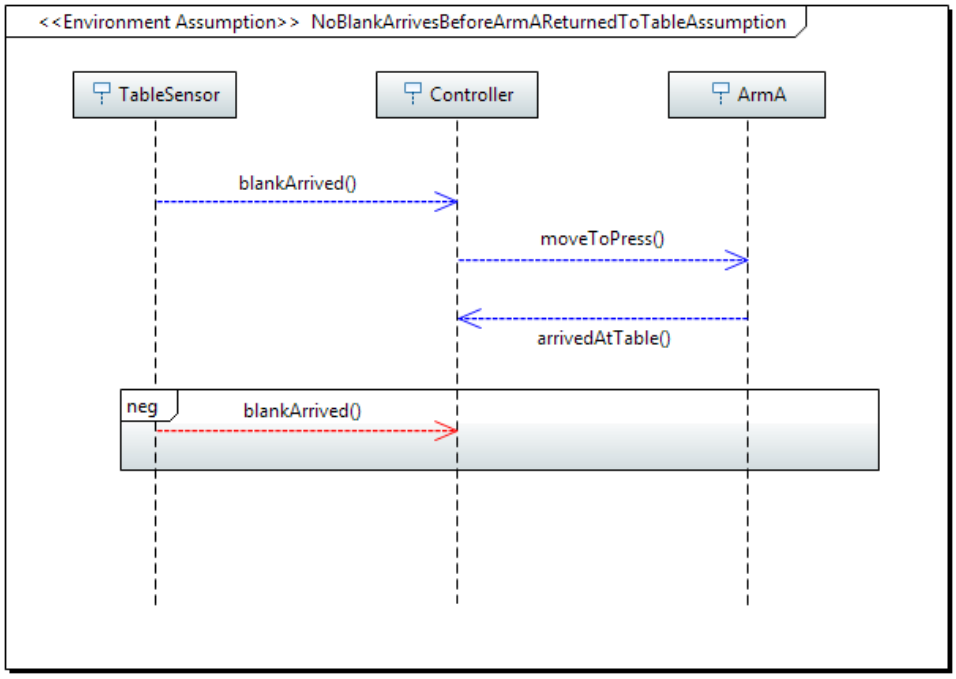


Abbildung 3: Assumption MSD mit einer Negation. (In Anlehnung an [13])

2.2.2 Violations

Wenn eine Nachricht auftritt, die in einem MSD nicht enabled ist und der dazugehörige Cut *hot* ist, dann tritt eine sogenannte s*afety Violation* auf. So eine Violation darf niemals auftreten. Ist der Cut hingegen *cold*, dann wird dies eine c*old Violation genannt*. Dieser Fall darf zwar auftreten, aber er führt dazu, dass das entsprechende MSD sofort beendet wird.

Befindet sich ein aktives MSD in einem *executed Cut*, den er nicht mehr verlassen kann, dann wird dies eine *liveness Violation* genannt.

Wenn eine dieser Violations in einem Requirement MSD auftaucht, dann wurde eine Anforderung verletzt. Dies darf demnach nicht passieren. Tritt so eine Violation hingegen in einem Assumption MSD auf, bedeutet es, dass die Umwelt sich gegen die Annahmen verhalten hat. Es wird jedoch davon ausgegangen, dass die Umwelt sich in der Praxis an die Annahmen hält und das System somit trotzdem funktioniert. Dies gilt auch für Situationen in denen sowohl eine Violation in Assumption als auch eine Violation in Requirements auftritt.

2.2.3 Spontaneous und non-spontaneous Umwelt-Nachrichten

Bisher wurde davon ausgegangen, dass Umwelt-Events jederzeit auftauchen können. In der Veröffentlichung von Greenyer et al. [3] wurde jedoch das Konzept der sogenannten non-spontaneous Events vorgestellt. Diese Events treten nur in Reaktion auf andere Events auf. Im oben erwähnten Beispiel könnte zum Beispiel das Event *arrivedAtPress* als ein *non-spontaneous* Event modelliert werden, da davon ausgegangen werden kann, dass diese Nachricht nur auftritt, wenn zuvor das Event *moveToPress* aufgetreten ist. Auf diese Weise ist es möglich präzisere Annahmen über die Umwelt zu modellieren. Ein *non-spontaneous* Event kann nur dann auftreten, wenn es eine entsprechende Nachricht in einem aktiven assumption MSD gibt, die *enabled* und *executed* ist. In der folgenden Abbildung ist die entsprechende Situation dargestellt.

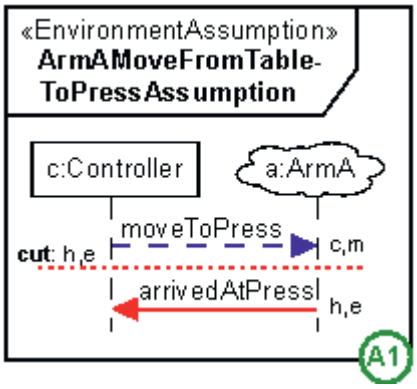


Abbildung 4: MSD Assumption. (Bild entnommen aus Greenyer et al. [4, S.391])

2.2.4 PlayOut Algorithmus

Der PlayOut Algorithmus wurde von Harel und Marelly zunächst für LSCs [9] und später auch für MSDs [10] definiert. Mit Hilfe des PlayOut Algorithmus kann das System, das durch die MSD Spezifikation modelliert wurde, analysiert bzw. simuliert werden. Wird der Algorithmus auf eine MSD Spezifikation angewandt, dann befindet sich das simulierte System zu Beginn in einem Zustand, in dem noch kein MSD aktiv ist. Aus diesem Grund muss zunächst auf ein auftretendes Umwelt-Event gewartet werden durch das mindestens ein MSD aktiviert wird. Sobald dies geschehen ist und somit mindestens ein MSD aktiv ist, wählt der Algorithmus von allen zur Zeit aktiven System-Nachrichten in den momentan aktiven MSDs zufällig eine aus. Falls die gewählte System-Nachricht zu keiner Violation führt, wird die Nachricht ausgeführt und es wird die nächste aktive System-Nachricht zufällig ausgewählt. Es wird dabei davon ausgegangen, dass das System schnell genug ist, um eine endliche Anzahl an System-Nachrichten auszuführen, bevor das nächste Umwelt-Event auftritt. Falls es keine aktiven System-Nachrichten mehr gibt, wartet der Algorithmus wieder darauf, dass ein auftretendes Umwelt-Event dazu führt, dass eine System-Nachricht aktiv wird.
Der Algorithmus läuft solange bis alle aktiven MSDs wieder beendet worden sind oder bis eine Situation entsteht, in der alle möglichen Events zu einer Violation führen.
Die Ausführung des PlayOut Algorithmus und somit die Auswahl der auftretenden Nachrichten kann entweder automatisch oder durch einen Benutzer geschehen. Auf diese Weise ist es zum Beispiel möglich, eine MSD Spezifikation zu debuggen und zu analysieren.

2.2.5 PlayOutButWaitingPossibleWithActiveEvents Algorithmus

Der PlayOutButWaitingPossibleWithActiveEvents Algorithmus basiert auf dem normalen PlayOut Algorithmus. Er ist jedoch ein bisschen flexibler als der PlayOut Algorithmus. Während beim normalem PlayOut Algorithmus, immer zunächst alle aktiven System-Nachrichten ausgeführt werden, bis keine aktiven System-Nachrichten mehr aktiv sind, gibt es beim PlayOutButWaitingPossibleWithActiveEvents Algorithmus hingegen auch die Möglichkeit, dass die aktiven System-Nachrichten nicht sofort ausgeführt werden, sondern das zunächst auf das nächste Umwelt-Event gewartet wird.
Im weiteren Verlauf dieser Arbeit wird nur noch der PlayOutButWaiting-PossibleWithActiveEvents Algorithmus verwendet.

2.2.6 Deterministische Ablaufsteuerung

Wird bei der Anwendung des PlayOutButWaitingPossibleWithActiveEvents Algorithmus auf eine MSD Spezifikation in jeder immer wiederkehrenden Situation immer die gleiche Aktion ausgeführt, und tritt dabei nie eine Violation auf, dann stellt dies eine **gültige** deterministische Ablaufsteuerung für die MSD Spezifikation dar. Das heißt vor allem, dass eine gültige deterministische Ablaufsteuerung auf jede mögliche Folge von Umwelt-Events so reagieren kann, dass das System nie in eine Situation gelangt, in der gegen die Spezifikation verstoßen wird. So eine deterministische Ablaufsteuerung kann sich dazu in jeder Situation für genau eine der folgenden Aktionen entscheiden. Entweder sie führt genau eine der aktiven System-Nachrichten aus oder sie tut nichts und wartet ab bis das nächste Umwelt-Event auftritt.

Wird im Folgenden von einer Ablaufsteuerung gesprochen, dann ist damit immer eine deterministische Ablaufsteuerung gemeint.

2.2.7 MSD Zustands Graph

Wird der PlayOutButWaitingPossibleWithActiveEvents Algorithmus auf eine MSD Spezifikation angewendet, ist es möglich, die aufgetretenen Situationen und die ausgewählten Aktionen auch als Zustandsgraphen aufzufassen und darzustellen. Jeder Zustand steht dabei für genau eine Situation, die bei der Anwendung aufgetreten ist. Die jeweilige Situation wird dabei eindeutig anhand aller aktiven MSDs und deren Cuts beschrieben. Die dargestellten Zustände enthalten auch Informationen darüber, ob eine Violation aufgetreten ist. Die Transitionen zwischen den Zuständen stellen die entsprechenden Events dar, die zur neuen Situation (Zustand) geführt haben. Umwelt-Events werden dabei mit gestrichelten Pfeilen dargestellt und die System-Nachrichten mit durchgezogenen Pfeilen.
Auf die gleiche Weise kann auch eine komplette Ablaufsteuerung als Zustandsgraphen dargestellt werden. So ein Graph beinhaltet alle Situationen (Zustände), die in dem System durch die Ablaufsteuerung auftreten können und die entsprechenden Aktionen (Transitionen), die in den jeweiligen Situationen ausgeführt werden. Wenn eine Ablaufsteuerung zum Beispiel in einem Zustand eine System-Transition besitzt, bedeutet das, dass in diesem Zustand immer die entsprechende System-Nachricht gesendet wird. Wenn eine Ablaufsteuerung hingegen in einem Zustand auf das nächste Umwelt-Event wartet, dann besitzt der Zustand für jedes

Umwelt-Event das in dieser Situation auftreten kann eine ausgehende Umwelt-Transition. Da von einer deterministischen Ablaufsteuerung ausgegangen wird, besitzt ein Zustand immer entweder genau eine System-Transition oder für jedes mögliche Umwelt-Event eine Umwelt-Transition.

In der folgenden Abbildung ist ein Ausschnitt eines Zustandsgraphen für eine Ablaufsteuerung zu sehen. Der Zustand 1 stellt den Startzustand dar, indem keine MSDs aktiv sind.

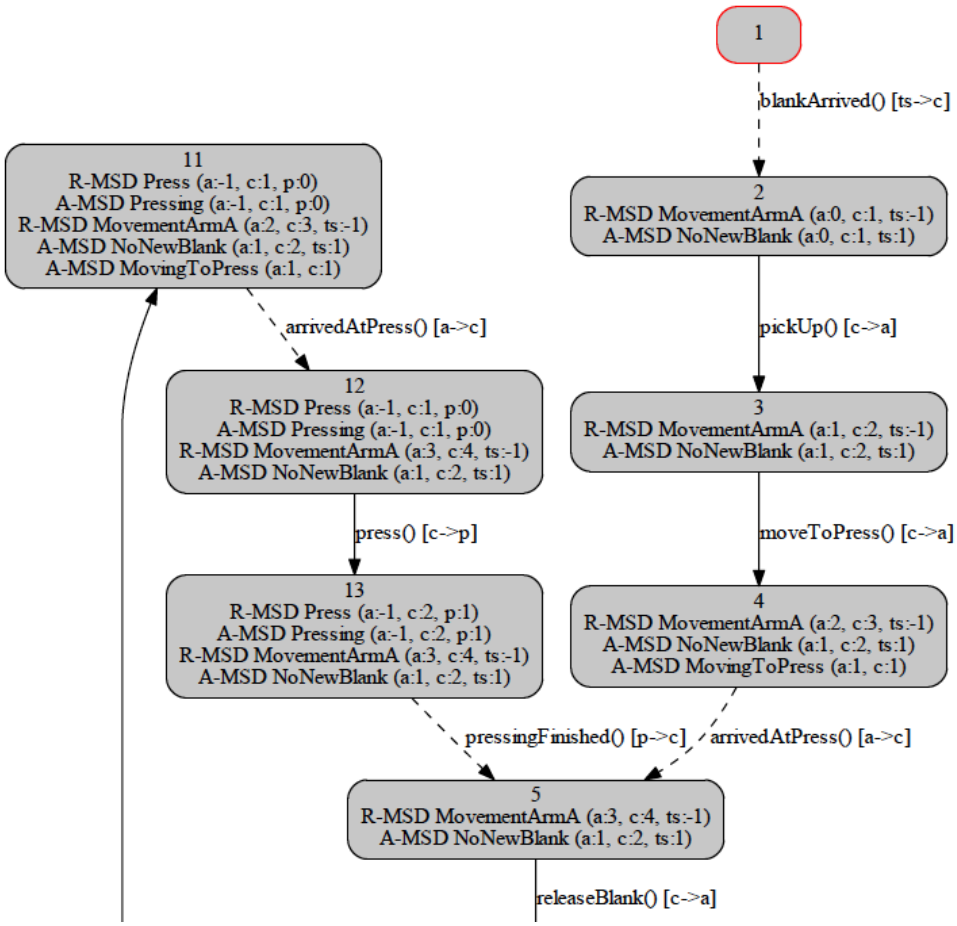


Abbildung 5: Ausschnitt eines Zustandsgraphen für eine Ablaufsteuerung.

Des Weiteren ist es auch möglich, einen Zustandsgraphen zu erstellen, der den kompletten Zustandsraum der MSD Spezifikation darstellt. Dieser Zustandsgraph enthält alle erreichbaren Zustände und alle möglichen Transitionen, die auf Basis des PlayOutButWaitingPossibleWithActiveEvents Algorithmus auftreten können. Dazu muss bei der Anwendung des PlayOutButWaitingPossibleWithActiveEvents Algorithmus in jedem Zustand immer jede System-Nachricht und jedes Umwelt-Event ausgeführt werden, die in der jeweiligen Situation auftreten können. Das heißt, dass ein Zustand in so einem Zustandsgraphen gleichzeitig sowohl System-Nachrichten als

auch Umwelt-Nachrichten als ausgehende Transitionen besitzen kann. Somit werden in so einem Zustandsgraphen alle zur Verfügung stehenden Aktionen dargestellt, die von einer theoretischen Ablaufsteuerung ausgeführt werden könnten. Durch die Darstellung einer MSD Spezifikation als Zustandsgraphen ist es möglich die MSD Spezifikation mit Hilfe von Algorithmen für Zustandsgraphen zu analysieren.
Im Folgenden wird ein Zustandsgraph, der den kompletten Zustandsraum darstellt, auch als **kompletter Zustandsgraph** bezeichnet.
In der folgenden Abbildung ist ein Ausschnitt eines kompletten Zustandsgraphen zu sehen.

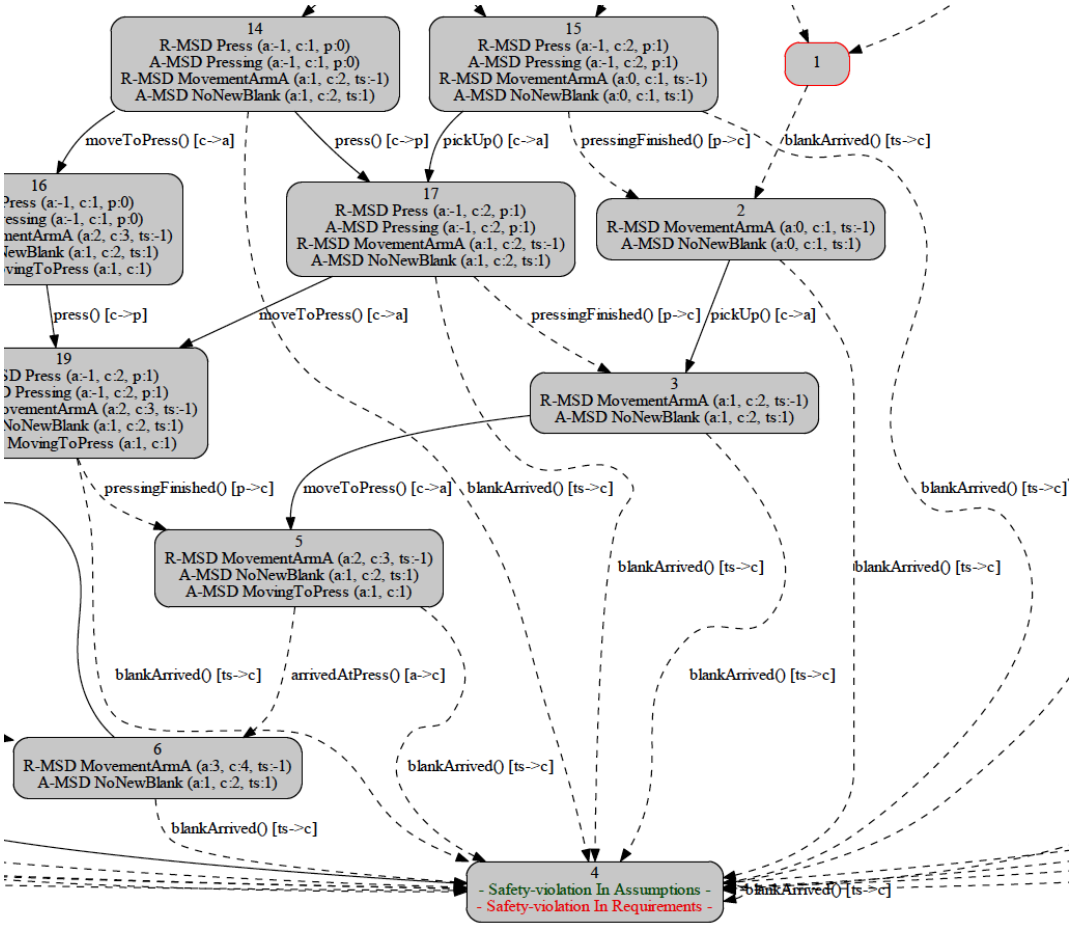


Abbildung 6: Ausschnitt eines kompletten Zustandsgraphen.

Der Zustand mit der Nummer 1 ist der Startzustand des Graphen. Dieser Zustand stellt die Situation dar, in der kein MSD aktiv ist. Das bedeutet, dass eine theoretische Ablaufsteuerung in dieser Situation nur die Möglichkeit hat auf das nächste Umwelt-Event zu warten. In diesem Fall kann nur das Event

blankArrived auftreten, da die entsprechende MSD Spezifikation so modelliert wurde, dass nur *blankArrived* ein *spontaneous* Umwelt-Event ist und alle anderen Umwelt-Events *non-spontaneous* sind. Sobald *blankArrived* in Zustand 1 auftritt, geht der Graph in den Zustand 2 über, in dem das Requirement MSD (im Graph als R-MSD MovementArmA bezeichnet), das in Kapitel 2.2 erklärt wurde, aktiv ist. Zudem ist in Zustand 2 auch das Assumption MSD aktiv, das besagt, dass kein weiterer Rohling erscheint, solange ArmA den Rohling noch nicht zur Presse gebracht hat und wieder zum Tisch zurückgekehrt ist. Aufgrund der Annahme führt die Umwelt-Transition *blankArrived* in Zustand 2 auch zu Zustand 4, in der eine Safety Violation in Assumptions auftritt. Das bedeutet, dass davon ausgegangen wird, dass dieses Event in dieser Situation auf Basis der getroffenen Annahmen nicht auftreten kann. Das heißt auch, dass in dieser Situation aufgrund der getroffenen Annahmen kein Umwelt-Event auftreten kann und eine theoretische Ablaufsteuerung in dieser Situation die System-Nachricht *pickUp* senden muss, damit das modellierte System voranschreitet.

Der Zustand 15 in dem Graphen besitzt sowohl eine System-Transition als auch eine Umwelt-Transition, die jeweils zu einem Zustand führen, in denen keine Violation auftritt. Das heißt, dass eine theoretische Ablaufsteuerung in diesem Zustand sich entscheiden könnte, ob sie entweder die System-Nachricht verschickt oder auf das nächste Umwelt-Event wartet, welches in dieser Situation auf jeden Fall *pressingFinished* sein wird, da das einzige andere mögliche Umwelt-Event in diesem Zustand zu einer Safety-Violation in Assumptions führt.

2.3 Markov-Kette

Eine Markov-Kette [1, S.744ff] ist ein stochastischer Prozess. Mit Hilfe einer Markov-Kette können stochastische Aussagen über den zukünftigen Zustand eines Systems in Abhängigkeit von seiner Vorgeschichte gemacht werden. Bei einer Markov-Kette erster Ordnung ist der zukünftige Zustand des Systems dabei nur von dem aktuellen Zustand abhängig.
Eine Markov-Kette kann als gerichteter Zustandsgraph dargestellt werden. In der folgenden Abbildung ist ein Beispiel für eine Markov-Kette in Form eines Zustandsgraphen dargestellt.

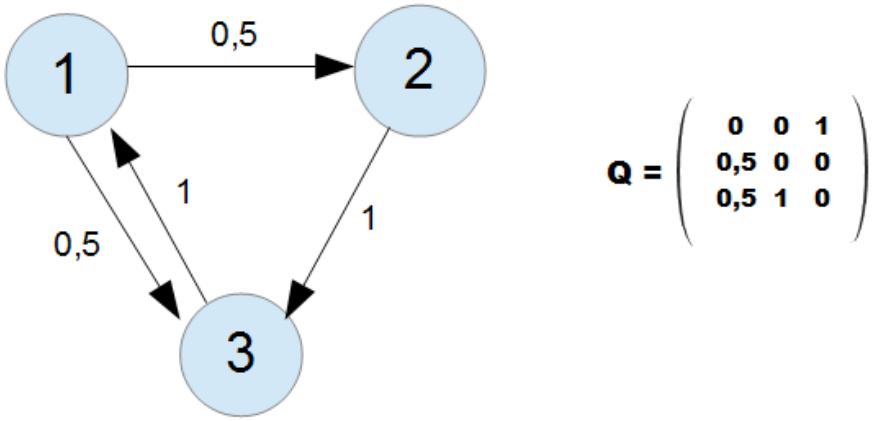


Abbildung 7: Zustandsgraph einer Markov-Kette und die Übergangsmatrix.

Auf der linken Seite der Abbildung 7 ist ein Zustandsgraph einer Markov-Kette zu sehen. An den Transitionen befinden sich die Übergangswahrscheinlichkeiten. Die Übergangswahrscheinlichkeiten können auch in einer Übergangsmatrix (Abbildung 7, rechte Seite) angegeben werden.
Mit Hilfe der Übergangsmatrix kann das folgende Gleichungssystem erstellt werden. Durch das Lösen des Gleichungssystems können die stationären Wahrscheinlichkeiten der Zustände berechnet werden. Die stationären Wahrscheinlichkeiten stellen dabei die Auftrittswahrscheinlichkeiten der Zustände dar, wenn unendlich viele Schritte durchgeführt werden.

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0{,}5 & 0 & 0 \\ 0{,}5 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \quad \text{und} \quad \sum_{i=1}^{3} p_i = 1$$

Abbildung 8: Gleichungssystem für die Berechnung der stationären Wahrscheinlichkeit.

2.4 Markov-Entscheidungsprozess

Ein Markov-Entscheidungsprozess [1, S.832ff] ist ein stochastischer Prozess. Ähnlich wie bei einer Markov-Kette können bei einem Markov-Entscheidungsprozess Aussagen über den zukünftigen Zustand des Systems gemacht werden. Bei einem Markov-Entscheidungsprozess ist der zukünftige Zustand des Systems jedoch nicht nur vom aktuellen Zustand abhängig, sondern auch von der gewählten Aktion, die in diesem Zustand getroffen wurde. Des Weiteren besitzen die Transitionen noch einen Wert, der die Belohnung darstellt, die bei der Ausführung der Transition erlangt wird.

In der folgenden Abbildung ist ein Markov-Entscheidungsprozess abgebildet. Er besteht aus zwei Zuständen, die jeweils zwei verschiedene Aktionen besitzen.

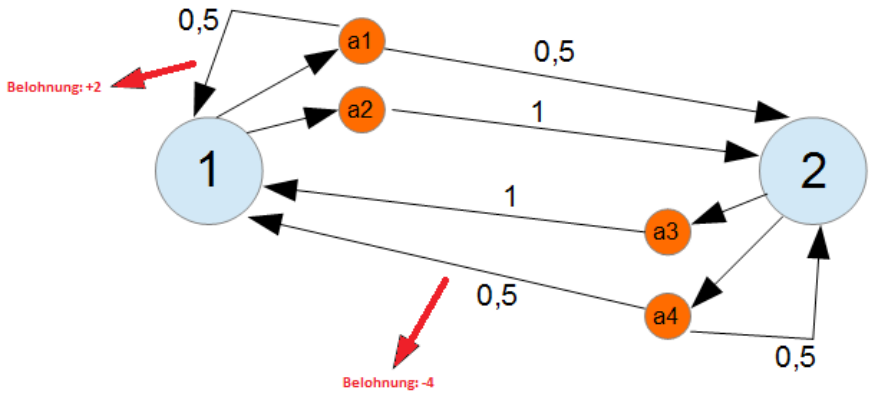


Abbildung 9: Markov-Entscheidungsprozess als Zustandsgraphen dargestellt.

Wird jedes Mal die gleiche Aktion ausgeführt, wenn ein Zustand erneut auftritt, verhält sich ein Markov-Entscheidungsprozess wie eine Markov-Kette. Das zu lösende Problem bei einem Markov-Entscheidungsprozess ist, für jeden Zustand die beste Aktion auszuwählen, damit die insgesamt erhaltene Belohnung maximiert wird.

2.5 Bremsenergie-Rückgewinnung

In der Arbeit von Greenyer et al. [4] wurde sich mit der Wiederverwendung von Bremsenergie befasst, indem sie unter anderem eine Simulation durchgeführt haben, bei der zwei Roboterarme jeweils eine Bewegung durchführen. Die Roboterarme waren dabei jeweils mit einem Servoantrieb verbunden, die fähig waren Energie bei der Abbremsung zurückzugewinnen. Die folgende Abbildung zeigt die Ergebnisse der Simulation:

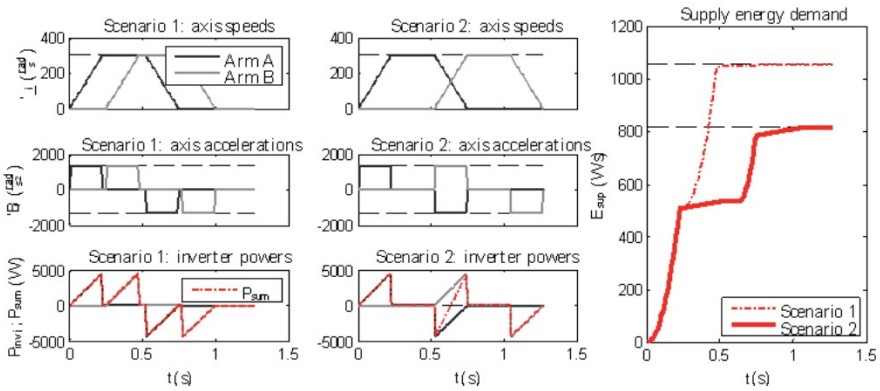


Abbildung 10: Simulationsergebnisse. (Entnommen aus [4, S.394])

In der Abbildung sind zwei verschiedene Szenarien zu sehen, die bei der Simulation durchgeführt worden sind. In beiden Szenarien gibt es zwei Roboterarme die jeweils eine Bewegung ausführen. Beim ersten Szenario sind die Bewegungsabläufe nicht aufeinander abgestimmt worden und die Beschleunigungsphasen und Abbremsphasen finden zu unterschiedlichen Zeitpunkten statt. Deswegen konnte beim ersten Szenario keine Bremsenergie wiederverwendet werden und der Energiebedarf konnte somit auch nicht reduziert werden. Im Szenario 2 hingegen wurden die Bewegungsabläufe so aufeinander abgestimmt, dass die Abbremsphase vom ersten Roboterarm und die Beschleunigungsphase vom zweiten Roboterarm sich zeitlich komplett überdeckten. Bei Szenario 2 konnte auf diese Weise der Energiebedarf des Systems um etwa 20 % reduziert werden. Dies zeigt, wie viel Potential für die Wiederverwendung von Bremsenergie in der Optimierung einer Ablaufsteuerung stecken kann.

3 Verwandte Arbeiten

Es gibt bereits mehrere Arbeiten, die sich mit dem Thema der Energieeffizienz von Produktionsanlagen beschäftigt haben. Darunter gibt es auch Arbeiten, die die Energieeffizienz einer Produktionsanlage durch die Wiederverwendung von Bremsenergie, verbessern. In den Arbeiten von Hansen et al. [6, 7, 8] wird zum Beispiel versucht für Punkt-zu-Punkt Bewegung eines mehrachsigen Industrieroboters die optimalen Bewegungsabläufe für die einzelnen Achsen zu finden. Dafür wird eine detaillierte Kostenfunktion aufgestellt, die unter anderem auch die mögliche Wiederverwendung von Bremsenergie berücksichtigt. Für die Kostenfunktion wird dann eine minimale Lösung für die jeweilige Bewegung berechnet. Die Berechnung findet dabei auf Basis einer gegebenen Ablaufsteuerung statt. Das heißt, dass dieses Verfahren aufgrund der gegebenen Möglichkeiten, die minimale Lösung berechnet. Das Verfahren versucht nicht, die gegebene Ablaufsteuerung dahingehend zu optimieren, dass so viel Energie wie möglich wiederverwendet werden kann.
Für die Synthese einer Ablaufsteuerung aus einer MSD Spezifikation wurde von Greenyer et al. [3] bereits ein Algorithmus entwickelt. Der Algorithmus ist darauf ausgelegt zu zeigen, ob eine MSD Spezifikation realisierbar ist oder nicht. Dabei durchsucht der Algorithmus den Graphen nur so lange bis eine Ablaufsteuerung gefunden worden ist, die gültig ist. Dieser Algorithmus berücksichtigt bei der Synthese einer Ablaufsteuerung jedoch nicht die Energie, die in den verschiedenen Vorgängen generiert bzw. verbraucht wird.
Bei der durchgeführten Recherche nach verwandten Arbeiten wurde keine Arbeit gefunden, die versucht hat, die Wiederverwendung von Bremsenergie zwischen verschiedenen Vorgängen zu optimieren, indem die Ablaufsteuerung dahingehend angepasst wird, dass energieverbrauchende und energieerzeugende Vorgänge so oft wie möglich gleichzeitig ablaufen. Durch die Kombination der optimierten Ablaufsteuerung und der detaillierten Kostenfunktion, die in den erwähnten Arbeiten beschrieben wird, könnte die Energieeffizienz einer Produktionsanlage weiter verbessert werden.

4 Modellierung von Energie behafteten MSDs

In diesem Kapitel wird beschrieben, wie eine MSD Spezifikation modelliert werden muss, damit die entwickelten Algorithmen eine Ablaufsteuerung synthetisieren können, die so viel Energie wie möglich wiederverwenden kann.

4.1 Erweiterung des einführenden Beispiels

In dem Beispiel aus Kapitel 2.1 wurde die Bewegung des Roboterarms bisher nur durch zwei Nachrichten modelliert. Bei der Bewegung vom Tisch zur Presse zum Beispiel wurde die Bewegung nur durch die Nachrichten *moveToPress* und *arrivedAtPress* dargestellt. Bei dieser Art der Modellierung ist nicht zu erkennen, wann sich der Roboterarm in einer Beschleunigungsphase befindet und Energie verbraucht und wann er anfängt abzubremsen und Energie erzeugt. Im Allgemeinen kann eine Bewegung in drei Phasen aufgeteilt werden. In der ersten Phase wird der Roboterarm beschleunigt. In der zweiten Phase bewegt sich der Roboterarm mit einer etwa konstanten Geschwindigkeit weiter und in der letzten Phase wird der Roboterarm schließlich wieder abgebremst. Wie bereits erwähnt, sind für die Modellierung dabei besonders Phase 1 und Phase 3 wichtig, denn in diesen Phasen wird Energie verbraucht bzw. generiert. Wird davon ausgegangen, dass die Phase 2 nur extrem kurz andauert, kann diese Phase bei der Modellierung auch weggelassen werden. Das würde bedeuten, dass der Roboterarm nach der Beschleunigungsphase direkt in die Abbremsphase übergeht. Generell sollte eine Bewegung jedoch detaillierter modelliert werden, indem mehrere Phasen verwendet werden. Damit die folgenden Beispiele aber alle übersichtlich bleiben, wird im folgenden Verlauf jedoch immer nur die Beschleunigungsphase und die Abbremsphase einer Bewegung modelliert.

Soll nur die Beschleunigungs- und die Abbremsphase modelliert werden, dann benötigt man dafür mindestens 3 Nachrichten. Die erste Nachricht zeigt an, dass der Roboterarm beschleunigt wird, die zweite Nachricht zeigt an, dass die Beschleunigungsphase beendet ist und der Roboterarm ab jetzt abgebremst wird und die letzte Nachricht zeigt an, dass die Abbremsphase beendet und das Ziel erreicht wurde.

Des Weiteren muss auch noch modelliert werden, wie viel Energie in den jeweiligen Phasen verbraucht bzw. generiert wird. Zu Beginn der Erstellung einer MSD Spezifikation ist zwar oft noch nicht bekannt, wie viel Energie genau ein Vorgang verbraucht oder generiert, aber es ist oft schon möglich

aufgrund von Erfahrungen gewisse Annahmen zu treffen. Der Einfachheit halber wird ein Integerwert verwendet, der angeben soll, wie viel Energie bei dem Vorgang verbraucht bzw. generiert wird. Die Integerwerte haben dabei keine spezielle Maßeinheit und werden deswegen im weiteren Verlauf einfach nur als Energie oder Energieeinheiten bezeichnet. Somit benötigt jede Nachricht, die so einen Vorgang modelliert, einen Integer-Parameter, dem dann der entsprechende Energiewert zugewiesen werden kann.

Damit es im Folgendem leichter zu unterscheiden ist, ob durch einen Vorgang Energie erzeugt oder verbraucht wird, erhalten Vorgänge bei denen Energie verbraucht wird positive Integer-Werte und Vorgänge bei denen Energie erzeugt wird negative Integer-Werte. Alle anderen Vorgänge erhalten keinen Integer-Parameter und somit auch keinen Integer-Wert.

Eine beispielhafte Modellierung der eben beschriebenen Phasen und Energiewerte sieht man in der folgenden Abbildung.

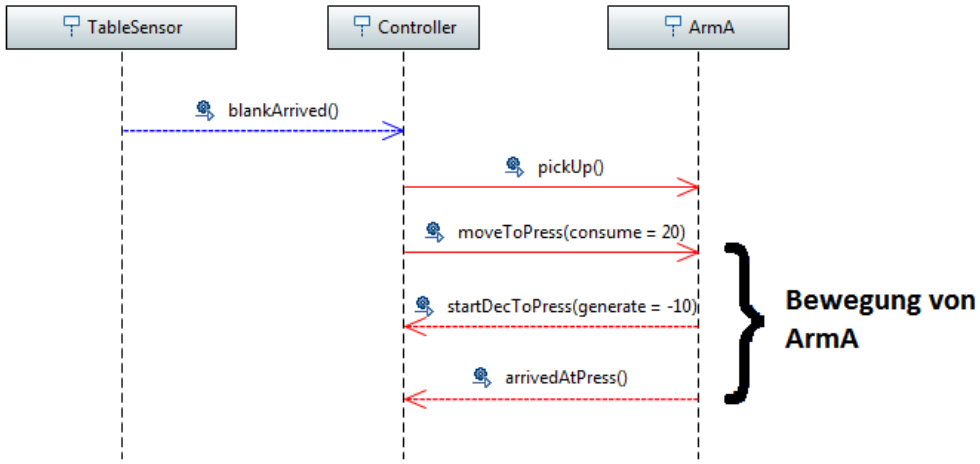


Abbildung 11: Modellierung einer Bewegung mit Energiewerten. In Anlehnung an [3, S.437].

In der Abbildung 11 ist die Modellierung der Bewegung von ArmA zur Presse dargestellt. Die Nachricht *moveToPress* leitet die Beschleunigungsphase ein und verbraucht währenddessen 20 Energieeinheiten. Dies wird durch den Parameter *consume* zum Ausdruck gebracht. Die Nachricht *startDecToPress* beendet die Beschleunigungsphase und leitet die Abbremsphase ein und generiert währenddessen 10 Energieeinheiten. Die Nachricht *arrivedAtPress* beendet wiederum die Abbremsphase und signalisiert, dass der Arm bei der Presse angekommen ist. Der Übersicht halber wird im weiteren Verlauf der Name des Integer-Parameters weggelassen und nur noch der Integerwert dargestellt.

Da die Energieeffizienz einer Ablaufsteuerung anhand des Energiebedarfs pro Zielerreichung gemessen werden soll, muss zudem noch eine Nachricht als Ziel-Nachricht festgelegt werden. Diese Nachricht soll dann später signalisieren, wenn ein Ziel erreicht wurde. Dafür erhält die entsprechende Nachricht einen Boolean-Parameter mit dem Namen isGoal. In dem Beispiel aus Kapitel 2.1 könnte zum Beispiel die Nachricht *pressingFinished* als Ziel-Nachricht modelliert werden.

In der folgenden Abbildung ist ein Beispiel für die Modellierung einer Ziel-Nachricht dargestellt.

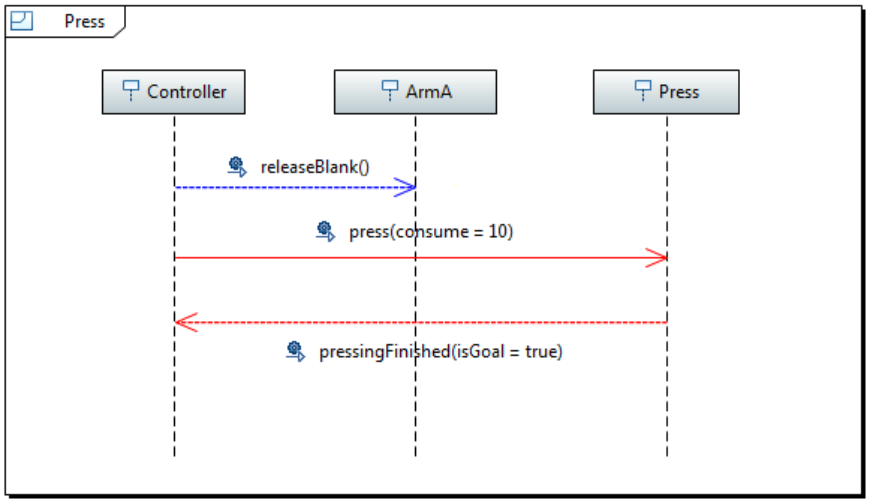


Abbildung 12: Modellierung einer Ziel-Nachricht. In Anlehnung an [3, S.437].

Auch hier wird der Übersicht halber im weiteren Verlauf auf den Parameter-Namen verzichtet und nur der Parameter-Wert dargestellt.

5 Lösungsvorschlag Brute-Force-Methode

Der einfachste Ansatz, um eine optimale Ablaufsteuerung zu finden, die so wenig Energie wie möglich von einer externen Energiequelle beziehen muss, ist, jede mögliche Ablaufsteuerung herzuleiten und dessen durchschnittlichen Energiebedarf zu berechnen. Die Ablaufsteuerungen können dazu aus dem kompletten Zustandsgraphen einer MSD Spezifikation abgeleitet werden.
In diesem Kapitel wird beschrieben welche Schritte für diesen Lösungsvorschlag umgesetzt werden müssen.

5.1 Gekürztes Beispiel

Da dieser Lösungsvorschlag auf Basis eines kompletten Zustandsgraphen arbeitet und der komplette Zustandsgraph für das einführende Beispiel aus Kapitel 2.1 mehr als 400 Zustände besitzt und somit sehr unübersichtlich ist, wird für die Beschreibung des Lösungsvorschlags eine stark reduzierte Modellierung der Produktionsanlage verwendet. Die reduzierte Modellierung besitzt nur noch den RoboterarmA, die Presse und den Tisch-Sensor. Der RoboterarmB hingegen gehört nicht mehr zur Produktionsanlage. Des Weiteren wird vom RoboterarmA auch nur noch die Bewegung von dem Tisch zur Presse modelliert. Die Bewegung von der Presse zurück zum Tisch fällt somit auch weg. Da diese Bewegung nicht mehr modelliert wird, wird davon ausgegangen, dass der RoboterarmA, nachdem er die Presse erreicht hat, automatisch wieder beim Tisch ist und somit den nächsten Rohling abholen kann. Das Aufheben und Ablegen der Rohlinge wird auch nicht mehr modelliert.

Diese Modellierung der Produktionsanlage ist zwar extrem abstrakt, aber dadurch besitzt der komplette Zustandsgraph nur noch 15 Zustände und es ist möglich die einzelnen Schritte, die für den Lösungsvorschlag durchgeführt werden müssen, zu erklären.

Des Weiteren wird festgelegt, dass die Bewegung von ArmA vom Tisch zur Presse in der Beschleunigungsphase 20 Energie verbraucht und in der Abbremsphase -10 Energie erzeugt. Die Presse dagegen verbraucht 10 Energie bei der Durchführung seines Pressvorgangs. Die Werte wurden an dieser Stelle vollkommen willkürlich gewählt und entsprechen nicht unbedingt der Realität.

5.2 Zustandsgraph „säubern“

Der erste Schritt dieses Lösungsvorschlags besteht darin den kompletten Zustandsgraphen der MSD Spezifikation aufzubauen, in dem der PlayOutButWaitingPossibleWithActiveEvents Algorithmus auf die MSD Spezifikation angewendet wird und dabei jede mögliche Aktion in jedem Zustand ausgeführt wird. Dies wurde auch in Kapitel 2.2.7 beschrieben.

In der folgenden Abbildung ist der komplette Zustandsgraph für die gekürzte Version der Produktionsanlage zu sehen.

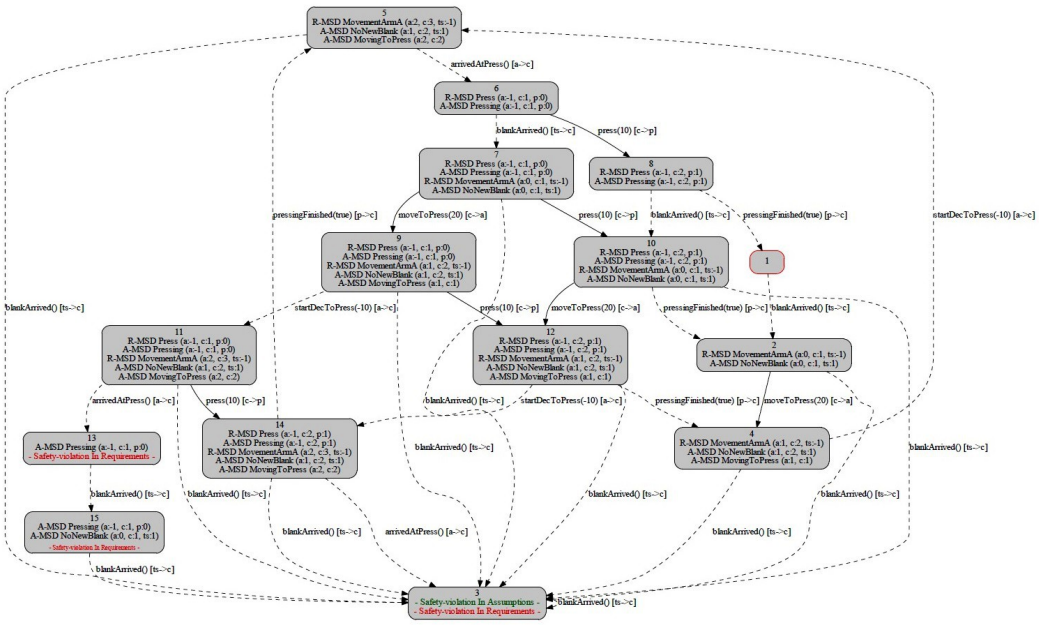


Abbildung 13: Kompletter Zustandsgraph.

Der nächste Schritt besteht darin, den kompletten Zustandsgraphen zu „säubern“, damit der Zustandsgraph am Schluss nur noch die Zustände und Transitionen bzw. Aktionen beinhaltet, die zu einer gültigen Ablaufsteuerung gehören können.

Deswegen werden zunächst alle Zustände, in denen eine **SafetyViolation in Assumptions** auftritt und alle Transitionen, die zu so einem Zustand führen, entfernt. Denn diese Transitionen und Zustände können aufgrund der getroffenen Annahmen nicht auftreten. In der Abbildung 14 ist dazu ein Beispiel dargestellt.

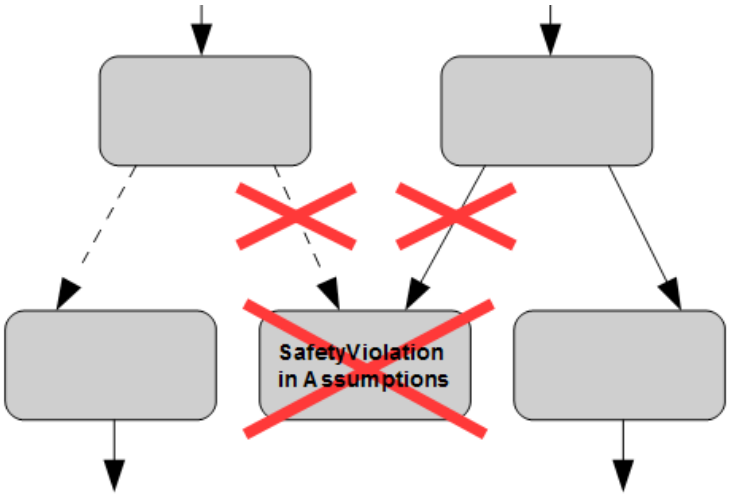


Abbildung 14: Safety Violation in Assumption: Entfernung von Transitionen und Zuständen.

Als nächstes werden alle Zustände entfernt, in denen eine **Safety Violation in Requirements** auftritt. Anschließend werden auch hier alle Transitionen entfernt, die zu so einem Zustand geführt haben. Wird auf diese Weise eine ausgehende Umwelt-Transition von einem Zustand entfernt, bedeutet dies, dass eine theoretische Ablaufsteuerung in diesem Zustand nicht auf das nächste Umwelt-Event warten darf, da es sonst zu einer Violation in Requirements führen könnte. Deswegen werden in diesem Zustand zusätzlich auch noch alle weiteren Umwelt-Transitionen entfernt. In der Abbildung 15 ist dazu ein Beispiel dargestellt.

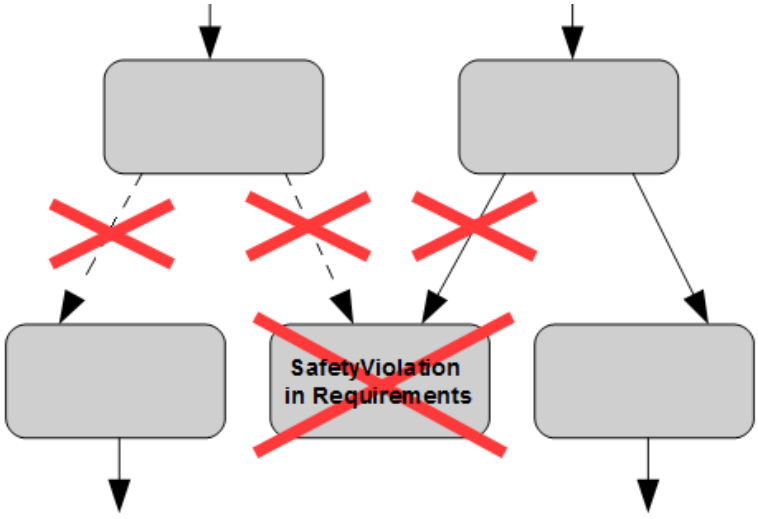


Abbildung 15: Safety Violation in Requirements: Entfernung von Transitionen und Zuständen.

Durch die Entfernung von Transitionen kann es dazu kommen, dass es anschließend Zustände gibt, die keine eingehenden und/oder ausgehenden Transitionen mehr besitzen. Deswegen werden alle Zustände, die keine eingehenden Transitionen mehr besitzen, ebenfalls entfernt, da diese Zustände niemals erreicht werden können. Hiervon ausgenommen ist nur der Startzustand. Dieser bleibt auch bestehen, wenn er keine eingehenden Transitionen besitzt. Wird ein Zustand auf diese Weise entfernt, werden auch all seine ausgehenden Transitionen entfernt.

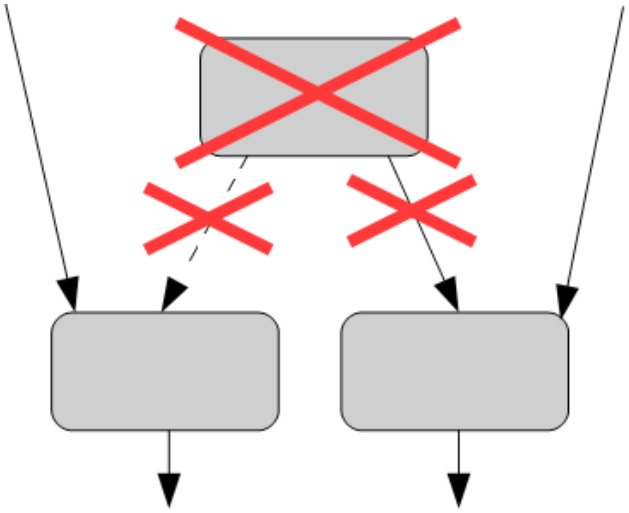


Abbildung 16: Entfernung eines Zustands ohne eingehende Transitionen.

Alle Zustände, die keine ausgehenden Transitionen mehr besitzen, können nicht mehr verlassen werden und das würde dazu führen, dass eine Produktionsanlage in diesem Zustand nicht mehr weiter arbeiten könnte. Deswegen darf so ein Zustand niemals erreicht werden. Aus diesem Grund werden für so einen Zustand die gleichen Schritte durchgeführt, wie bei einem Zustand, in dem eine Safety Violation in Requirements aufgetreten ist.

Diese Schritte werden solange wiederholt, bis der Graph keine Zustände mehr besitzt, die auf diese Weise entfernt werden müssen, oder bis der Startzustand keine ausgehenden Transitionen mehr besitzt.

Tritt der zweite Fall auf, dann gibt es für die gegebene MSD Spezifikation keine gültige Ablaufsteuerung und die MSD Spezifikation ist nicht realisierbar.

Wenn der erste Fall auftritt, dann gibt es nur dann eine gültige Ablaufsteuerung, wenn der Graph mindestens einen Zyklus hat, der auch eine Ziel-Transition besitzt. Denn nur dann kann das definierte Ziel, wie zum Beispiel die Fertigstellung eines Produkts, immer wieder erreicht werden.

In der folgenden Abbildung ist der „gesäuberte“ Zustandsgraph der gekürzten Produktionsanlage zu sehen.

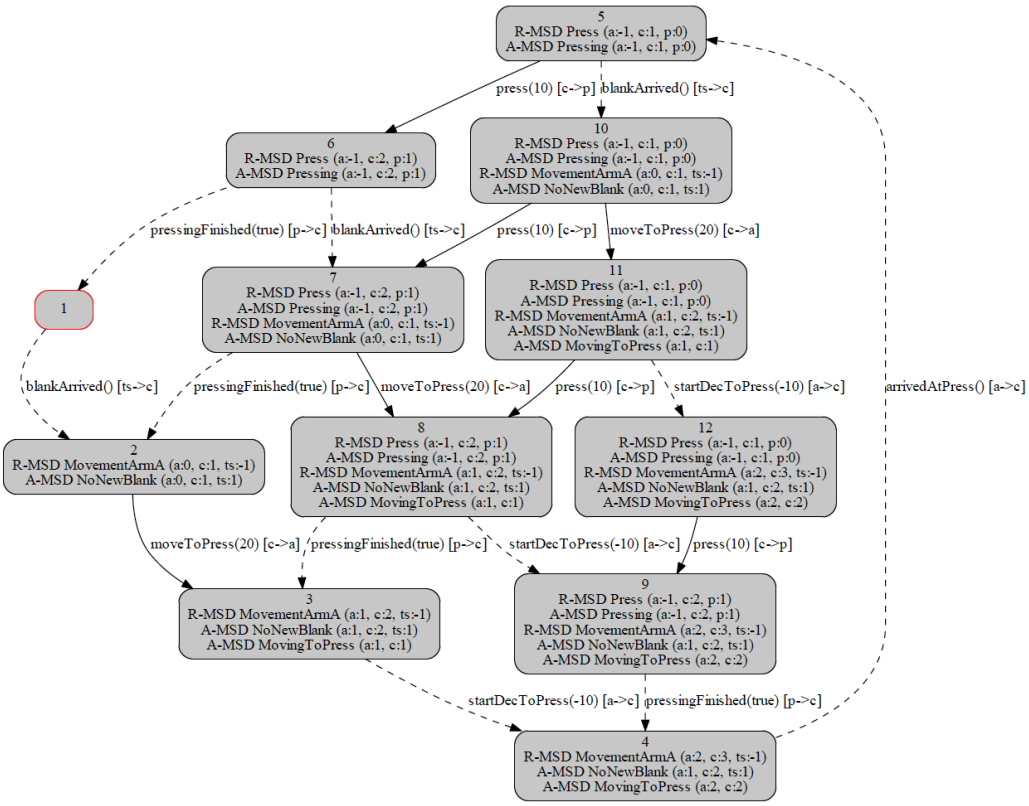


Abbildung 17: „Gesäuberter“ Zustandsgraph.

5.3 „Gesäuberten" Zustandsgraph mit Energiewerten erweitern

Der nächste Schritt besteht nun darin zu ermitteln, wann Energie wiederverwendet werden kann und ob dadurch die verbrauchte Energie von Vorgängen reduziert werden kann. Das Ziel ist es den „gesäuberten" Zustandsgraphen so zu erweitern, dass am Schluss jede Transition einen Wert besitzt, welcher den Energiebedarf darstellt, der von einer externen Quelle bezogen werden muss, wenn die entsprechende Transition auftritt. Das heißt, dass dieser Wert auch die Wiederverwendung von Bremsenergie berücksichtigen soll. Dafür muss zunächst für jeden Zustand hergeleitet werden, welche Energiewerte durch aktive Vorgänge, wie zum Beispiel eines Beschleunigungsvorgangs oder eines Bremsvorgangs, noch verfügbar sind.

Für diesen Zweck wird in jedem Zustand für jedes Umwelt-Objekt ein Integerwert abgespeichert, der die restliche Energie des jeweiligen Objektes in dem jeweiligen Zustand darstellen soll.

Dafür wird als Erstes der Startzustand ausgewählt. Alle Energiewerte des Startzustandes erhalten den Wert 0, da in diesem Zustand keine MSDs und somit auch keine aktiven Vorgänge aktiv sind. Anschließend wird nach dem Verfahren der Breitensuche der Graph vom Startzustand aus einmal komplett durchlaufen. Dabei werden für jeden Zustand, der so erreicht wird, seine Energiewerte ermittelt. Die Energiewerte setzen sich dabei aus den Energiewerten des vorherigen Zustands und dem Nachrichten-Event der Transition zusammen, die die beiden Zustände verbindet. Für die Herleitung werden dabei immer die folgenden Schritte durchgeführt:

Wenn alle Energiewerte des vorherigen Zustandes gleich 0 sind, dann wird im Folgezustand nur der Energiewert von dem Umwelt-Objekt angepasst, das an dem Nachrichten-Event der Transition, die die beiden Zustände verbindet, beteiligt ist. In der folgenden Abbildung 18 ist dafür ein Beispiel zu sehen. In dem dargestelltem Beispiel sind zwei Zustände und eine Transition dargestellt. Die Werte in den Zuständen stellen dabei die Energiewerte der aktiven Vorgänge der verschiedenen Umwelt-Objekte dar. Im oberen Zustand sind die Energiewerte der Presse und von ArmA gleich 0. Die Energiewerte des unteren Zustands können dann durch die Energiewerte des oberen Zustands und durch das Nachrichten-Event **press(10)** von der Transition, welche die beiden Zustände verbindet, hergeleitet werden. Da das Nachrichten-Event in diesem Fall einen Integer-Parameter besitzt und der Parameter einen positiven Wert besitzt, kann erkannt werden, dass es sich hierbei, um ein Nachrichten-Event handelt, das einen energieverbrauchenden Vorgang startet. Deswegen wird im Folgezustand dem Energiewert von der Presse der Wert 10 zugewiesen. Der

Energiewert von ArmA wird nicht verändert und wird deswegen vom Vorgänger-Zustand übernommen.

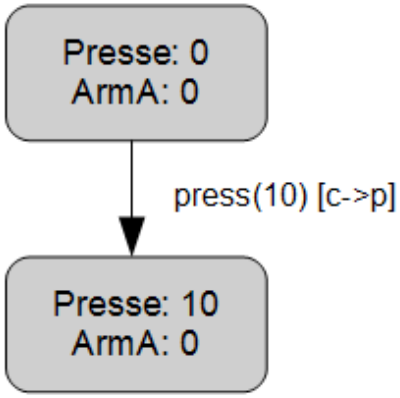


Abbildung 18: Energiebewertung eines Zustands.

Wenn es in einem Zustand mindestens einen Energiewert gibt der ungleich 0 ist, dann wird überprüft, ob durch das Nachrichten-Event von der Transition ein Vorgang beendet wird, der Energie verbraucht. Dies wird ermittelt, indem überprüft wird welches Umwelt-Objekt an dieser Nachricht beteiligt ist und ob der Energiewert des Umwelt-Objekts in dem vorherigen Zustand positiv war. Falls dies der Fall ist, wird anschließend überprüft, ob der Energiebedarf durch die generierte Energie eines anderen Vorgangs reduziert werden kann, indem überprüft wird, ob es einen Energiewert gibt der negativ ist. Wenn zum Beispiel für die Presse im vorherigen Zustand der Energiewert 10 gespeichert ist und die Nachricht der Transition *pressingFinished* lautet, dann wird als Nächstes überprüft, ob momentan Vorgänge aktiv sind, die Energie erzeugen. Dafür wird überprüft, ob ein anderer Energiewert des vorherigen Zustandes einen negativen Wert besitzt. Ist dies der Fall, dann kann der Energiebedarf des energieverbrauchenden Vorgangs reduziert werden. Wenn zum Beispiel für ArmA ein Energiewert von -8 im vorherigen Zustand gespeichert worden ist, dann ist bekannt, dass ArmA momentan einen Vorgang durchführt, der Energie erzeugt. Somit kann der Energiebedarf des Pressvorgangs aus einer externen Energiequelle auf 10-8 = 2 reduziert werden. Wenn es keine weiteren Vorgänge mehr gibt, durch den der Energieverbrauch noch weiter reduziert werden kann, dann wird der entsprechenden Transition der reduzierte Wert 2 zugewiesen. Dieser Wert spiegelt dann den Energiebedarf des Vorgangs wieder der noch von einer externen Energiequelle bezogen werden muss, wenn das Nachrichten-Event der Transition in diesem Zustand auftritt. Anschließend werden die Energiewerte des Zielzustandes berechnet. Als Grundlagen dienen dafür wieder die Energiewerte des vorherigen Zustandes. Wurde, wie

in dem Beispiel beschrieben, generierte Energie von einem Umwelt-Objekt verbraucht, wird der entsprechende Energiewert des entsprechenden Umwelt-Objekts angepasst. In dem genannten Beispiel wurde die komplette Energie aufgebraucht, deswegen wird dem Energiewert von ArmA der Wert 0 zugewiesen. Als Letztes wird dem Energiewert des Umwelt-Objekts, das an der Nachricht beteiligt war, in dem neuen Zustand der entsprechende Energiewert des Nachrichten-Events zugewiesen. Falls das Nachrichten-Event keinen Energiewert besitzt, wie in dem genannten Beispiel, dann wird dem entsprechenden Energiewert der Wert 0 zugewiesen.

In der folgenden Abbildung ist das eben beschriebene Beispiel auf der linken Seite zu sehen. Auf der rechten Seite ist ein weiteres Beispiel zu sehen, bei dem der Betrag des Energiewertes von ArmA größer ist als der Energiewert von der Presse.

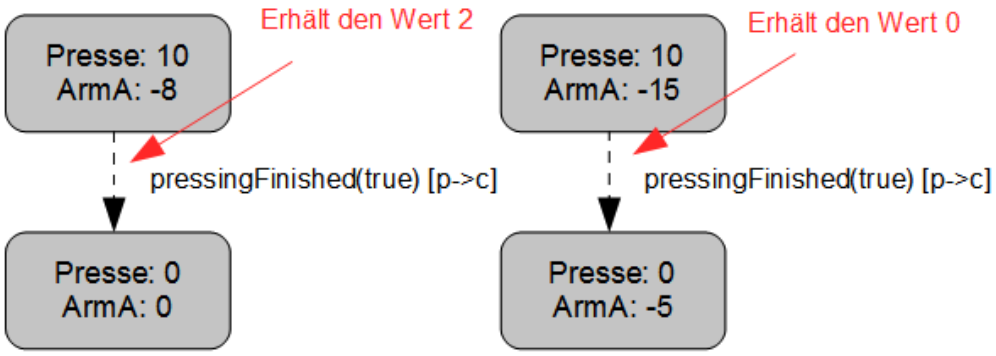


Abbildung 19: Energiebewertung eines Zustands: Energieverbrauchender Vorgang wird beendet.

In der rechten Abbildung hat der obere Zustand für die Presse den Energiewert 10 und für den ArmA den Wert -15 abgespeichert. Da durch pressingFinished der Pressvorgang beendet wird, wird die generierte Energie von ArmA verwendet um den Energiebedarf von der Presse zu reduzieren. Da die generierte Energie von ArmA größer ist als die benötigte Energie von der Presse, bekommt ArmA im folgenden Zustand -5 als Energiewert zugewiesen. Die Transition erhält den Wert 0, da der Energiebedarf komplett durch die generierte Energie von ArmA abgedeckt werden konnte.

Wird durch eine Transition ein Vorgang beendet, der Energie generiert, dann wird überprüft, ob es einen anderen Vorgang gibt, der von der generierten Energie profitieren kann. Kann auf diese Weise nicht die komplette Energie des Vorgangs wiederverwendet werden, verfällt die Energie. In der folgenden Abbildung ist dazu ein Beispiel zu sehen.

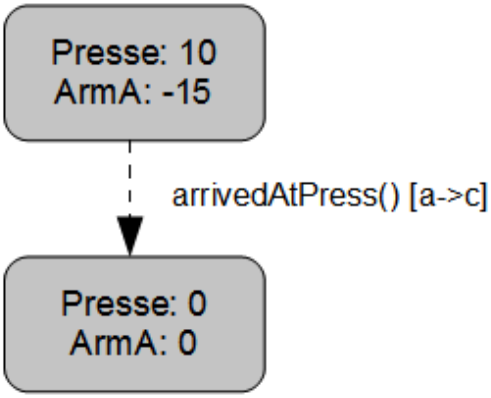


Abbildung 20: Energiebewertung eines Zustands: Energieerzeugender Vorgang wird beendet.

In dem oberen Zustand hat der Energiewert der Presse den Wert 10 und der Energiewert von ArmA den Wert -15. Durch das Nachrichten-Event der dargestellten Transition, wird der Vorgang von ArmA beendet und somit wird überprüft, ob es einen anderen Vorgang gibt, der von der Energie profitieren kann. In diesem Fall ist es die Presse. Deswegen wird der Energiewert der Presse in dem Folgezustand auf **max(0, 10-15) = max(0, -5) = 0** gesetzt. Die übrig gebliebene Energie von ArmA hingegen verfällt, da es keinen weiteren Vorgang gibt, der von der generierten Energie profitieren kann. Der Energiewert von ArmA wird dann ebenfalls auf 0 gesetzt, da durch die Nachricht *arrivedAtPress* kein neuer Vorgang gestartet wurde, der Energie erzeugt oder verbraucht.

Bei komplexeren Systemen kann es auch vorkommen, dass in einem Zustand mehrere Vorgänge aktiv sind, die Energie verbrauchen oder generieren. Wenn in so einem Zustand, ein Vorgang beendet wird, der Energie generiert, und es zudem noch in diesem Zustand mehrere aktive Vorgänge gibt, die diese Energie verbrauchen könnten, dann wird die so generierte Energie nicht direkt verrechnet, sondern zunächst in einer Extra-Datenstruktur abgespeichert. Diese Extra-Energie wird erst dann verrechnet, wenn einer dieser energieverbrauchenden Vorgänge beendet wird. Auf diese Weise kann die bestmögliche Verteilung der Energie berechnet werden. Dafür werden in der Extra-Datenstruktur die generierte Energie und alle Umwelt-Objekte

abgespeichert, die in diesem Zustand von der generierten Energie hätten profitieren können. In der folgenden Abbildung ist ein Beispiel zu sehen.

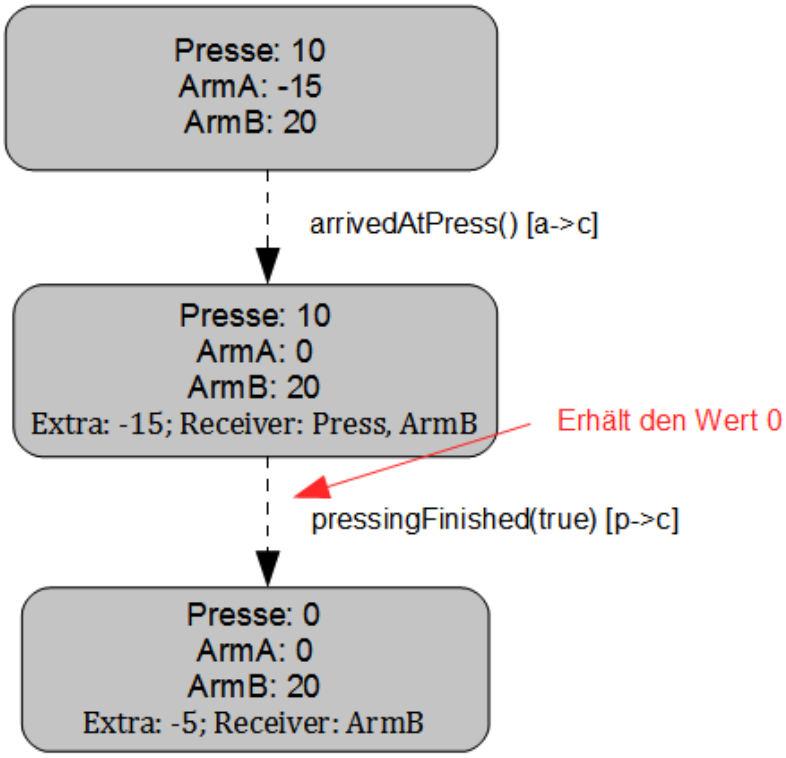


Abbildung 21: Energiebewertung eines Zustands: Verwendung der Extra-Datenstruktur.

In Zustand 1 besitzt der Energiewert von der Presse, den Wert 15. Der Energiewert von ArmA besitzt den Wert -10 und der Energiewert von ArmB besitzt den Wert 20. Von Zustand 1 aus wird der Zustand 2 durch das Nachrichten-Event *arrivedAtPress* erreicht. Durch das Nachrichten-Event wird der energieerzeugende Vorgang von ArmA beendet. Da sowohl die Presse als auch ArmB gerade einen energieverbrauchenden Vorgang aktiv haben, wird die Energie nicht direkt verrechnet, sondern in der erwähnten Extra-Datenstruktur abgespeichert. In Zustand zwei wird dann durch das Auftreten der Nachricht pressingFinished der Vorgang der Presse beendet und der Energiebedarf der Presse wird durch die Extra-Energie reduziert werden. Die restliche Energie bleibt in der Extra-Datenstruktur gespeichert. Die Presse wird jedoch aus der Datenstruktur als möglicher Empfänger entfernt. Das heißt, dass sie von der restlichen Energie nicht mehr profitieren kann. Die Extra-Datenstruktur wird solange an die nächsten Zustände weitergegeben, bis die Energie aufgebraucht worden ist, oder bis die Extra-Energie keine Empfänger (Receiver) mehr besitzt. Die Extra-Datenstruktur kann auf diese Weise auch mehrere Extra-Energiewerte abspeichern.

Durch die Verwendung der Extra-Datenstruktur soll die bestmögliche Wiederverwendung von Bremsenergie hergeleitet werden. Würde in dem oben erwähnten Beispiel die Energie zum Beispiel zufällig verteilt werden, könnte es dazuführen, dass der Energiebedarf von ArmB auf 20-15 = 5 reduziert wird und der Energiebedarf von der Presse bei 10 bleibt. Der gesamte Energiebedarf wäre zwar auf diese Weise identisch zu dem Energiebedarf aus dem Beispiel. Jedoch könnte der restliche Energiebedarf so nur noch um die übrig gebliebenen 5 Energie reduziert werden. Dadurch, dass durch die Verwendung der Extra-Datenstruktur die Energie erst später einem Vorgang zugewiesen wird, kann in den Folgezuständen von Zustand 3 der Energiebedarf jedoch noch um 20-5 = 15 Energie reduziert werden. Mit der Extra-Datenstruktur kann somit indirekt in die Zukunft geguckt werden, um dann rückwirkend die Energie optimal zu verteilen.
Die Extra-Datenstruktur speichert auf die gleiche Weise auch die verbrauchte Energie eines Vorgangs, wenn bei der Beendigung des Vorgangs, mehrere energieerzeugenden Vorgänge aktiv sind, von denen profitiert werden könnte.
Da die meisten Zustände mehrere eingehende Transitionen besitzen, können diese Zustände auf unterschiedlichen Pfaden erreicht werden. So kann es sein, dass beim Durchlaufen des Zustandsgraphen einem Zustand mehrere unterschiedliche Energiewerte zugewiesen werden, weil zum Beispiel bei dem einen Pfad die eigentlich verfügbare Energie schon wiederverwendet wurde, und bei dem anderen Pfad noch nicht. Tritt so ein Fall auf, dann wird dieser Zustand in mehrere Zustände aufgesplittet. Jeder Zustand erhält dann einen der ermittelten Energiewerte und die entsprechenden eingehenden Transitionen, die zu diesem Energiewert geführt haben. Jeder der gesplitteten Zustände erhält jedoch alle der ausgehenden Transitionen des Ursprungs-Zustands. In der Abbildung 22 ist hierfür ein Beispiel zu sehen.

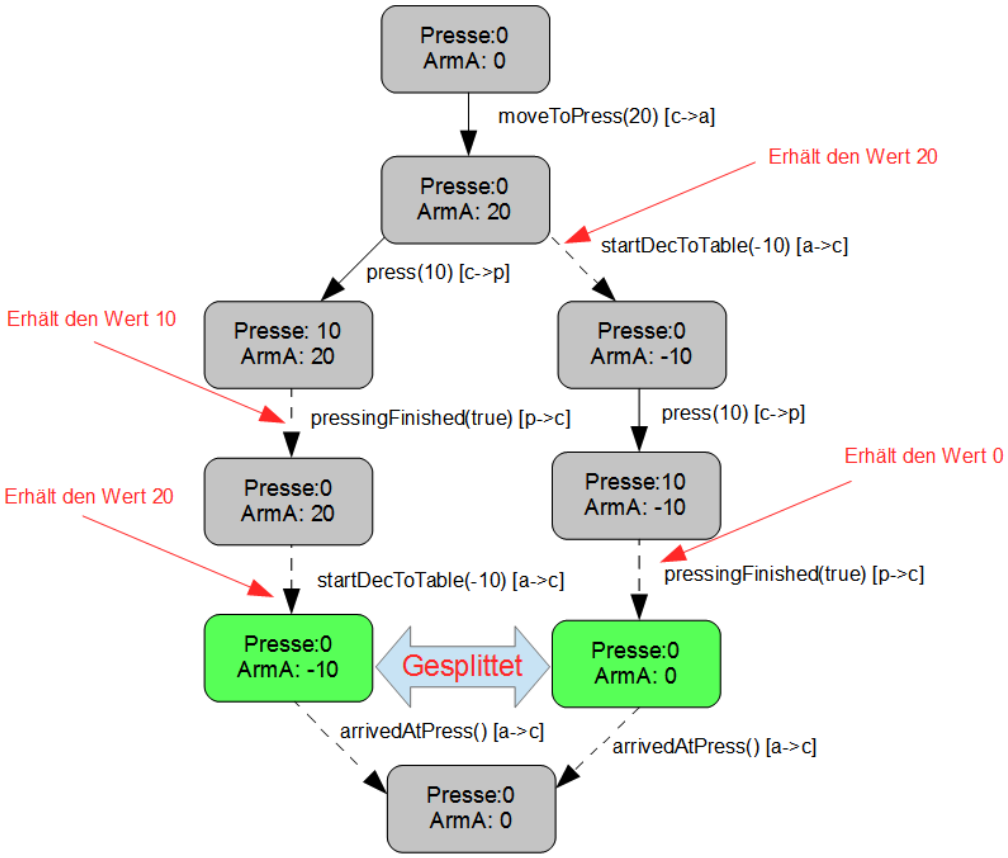


Abbildung 22: Aufgesplitteter Zustand.

In dem dargestellten Beispiel sind zwei Pfade abgebildet. Die grün dargestellten Zustände besitzen die selben Cuts und stellten eigentlich den selben Zustand dar. Da jedoch auf dem rechten Pfad die Energie wiederverwendet werden konnte und auf dem linken Pfad nicht, besitzen sie unterschiedliche Energiewerte und wurden deswegen an dieser Stelle in zwei Zustände aufgesplittet. In dieser Abbildung ist auch zu sehen, dass die beiden Zustände im nächsten Schritt trotzdem wieder zum gleichen Zustand führen, da durch die Beendigung des Bremsvorgangs von ArmA der Unterschied der Energiewerte nicht weiter besteht. Es kann jedoch auch sein, dass durch die Aufsplittung von Zuständen auch die Folgezustände aufgesplittet werden müssen. Diese Aufsplittung wird jedoch spätestens dann wieder unterbrochen, wenn ein Zustand erreicht wird, in dem kein Vorgang aktiv ist, der Energie verbraucht bzw. generiert, wie zum Beispiel der Startzustand oder der Zustand 6 in den oben gezeigtem Beispiel.

All diese Schritte werden solange durchgeführt bis der komplette Zustandsgraph bewertet wurde. Alle Transitionen die keinen Energiewert zugeordnet bekommen haben, erhalten den Wert 0.

Im Folgenden wird ein Zustandsgraph, der auf die beschriebene Weise mit Energiewerten erweitert wurde, als **erweiterter Zustandsgraph** bezeichnet.

In der folgenden Abbildung ist der erweiterte Zustandsgraph von der gekürzten Produktionsanlage zu sehen. In dem Bild werden die Energiebedarfswerte der Transitionen rot dargestellt. Die beiden grünen Zustände wurden aufgesplittet.

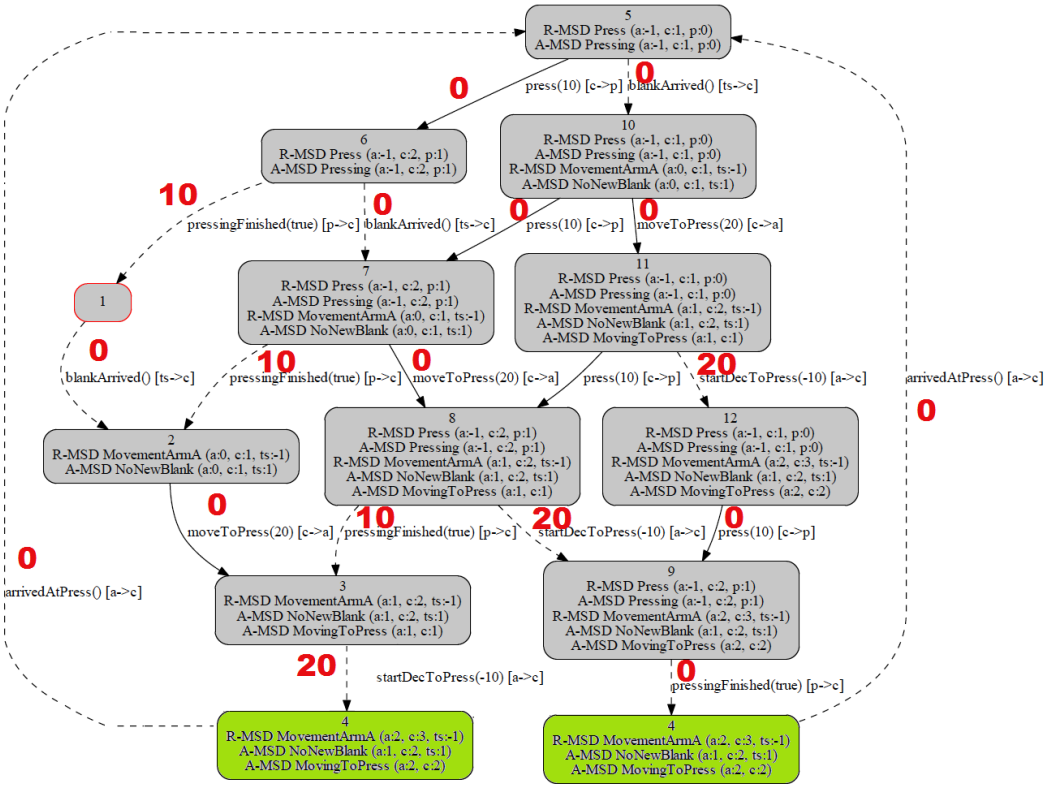


Abbildung 23: Erweiterter Zustandsgraph.

5.4 Alle möglichen Ablaufsteuerungen ableiten

Nachdem der Graph erweitert wurde, müssen nun alle möglichen Ablaufsteuerungen abgeleitet werden. Dafür wird der erweiterte Graph vom Startzustand aus einmal komplett durchlaufen und währenddessen werden dabei alle möglichen Ablaufsteuerungen extrahiert. Dazu wird der folgende Algorithmus verwendet:

1. Erstelle einen neuen Graphen und füge dem neu erstellten Graphen den Startzustand des erweiterten Graphen hinzu. Erstelle zudem noch eine Noch-Zu-Erforschen-Warteschlange und füge der Warteschlange ebenfalls den Startzustand zu.
2. Wähle den ersten Zustand der Warteschlange aus und entferne ihn aus der Warteschlange.
3. Lies die Anzahl der möglichen Aktionen des gewählten Zustands von dem erweiterten Graphen ab. (Jede System-Transition einzeln und alle Umwelt-Transitionen zusammen zählen jeweils als eine Aktion).
    - Wenn es nur eine mögliche Aktion gibt, dann füge alle Transitionen, die zu dieser Aktion gehören, dem Graphen hinzu. Füge zusätzlich noch die Zielzustände der Transitionen dem Graphen hinzu, falls die entsprechenden Zustände noch nicht Teil des Graphen sind. Speichere zusätzlich die neu hinzugefügten Zustände in der Warteschlange ab. Fahre anschließend mit Schritt 2 fort.
    - Wenn es mehr als eine mögliche Aktion gibt, dann erstelle für jede weitere Aktion eine Kopie von dem bis dahin erforschten Graphen und von der dazugehörigen Noch-Zu-Erforschen-Warteschlange (bei n möglichen Aktionen werden (n-1) Kopien erstellt). Weise anschließend jedem der Graphen genau eine Aktion zu. Füge dann die entsprechenden Transitionen der Aktionen dem jeweiligen Graphen hinzu. Füge zusätzlich jedem Graphen noch die entsprechenden Zielzustände der Transitionen hinzu, falls die entsprechenden Zustände noch nicht Teil des jeweiligen Graphen sind. Speichere die neu hinzugefügten Zustände in der Warteschlange des entsprechenden Graphen ab. Führe nun für jeden dieser Graphen Schritt 2 aus.

Das Ergebnis ist eine Liste von allen gültigen Ablaufsteuerungen für die entsprechende MSD Spezifikation.

5.5 Ablaufsteuerung als Markov-Kette auffassen

Der nächste Schritt besteht dann darin, die einzelnen Ablaufsteuerungen nach Ihrem Energiebedarf pro Zielerreichung zu bewerten. Der Zustandsgraph einer Ablaufsteuerung kann als Markov-Kette (siehe auch Kapitel 2.3) angesehen werden. Dafür müssen den Transitionen des Zustandsgraphen nur noch angemessene Übergangswahrscheinlichkeiten zugewiesen werden.

Jeder System-Transition kann eine Übergangswahrscheinlichkeit von 100% zugewiesen werden, da sie in einer Ablaufsteuerung immer die einzige ausgehende Transition in einem Zustand ist. Die Schwierigkeit ist es jedoch den Umwelt-Nachrichten eine angemessene Übergangswahrscheinlichkeit zuzuweisen, denn in einer Ablaufsteuerung kann ein Zustand mehrere von ihnen als ausgehende Transition besitzen. Da es auch keinerlei Informationen gibt, welche der möglichen Umwelt-Events am ehesten als nächstes auftritt, muss an dieser Stelle eine Annahme über die Übergangswahrscheinlichkeiten getroffen werden. Im Rahmen dieser Arbeit wurden die folgenden Varianten in Betracht gezogen:

Eine Variante wäre es von einer Gleichverteilung auszugehen, wodurch jede Umwelt-Transition die gleiche Wahrscheinlichkeit erhält. Das heißt das bei zwei möglichen Umwelt-Nachrichten, jede Nachricht eine Übergangswahrscheinlichkeit von 50% erhält. Bei 3 Nachrichten würde dementsprechend jede Transition eine Übergangswahrscheinlichkeit von 33,33% erhalten, usw. Somit spiegelt dieser Ansatz einen Art Average-Case wieder.

Eine zweite Variante wäre spontaneous und non-spontaneous Nachrichten mit unterschiedlichen Übergangswahrscheinlichkeiten zu versehen. Dies könnte entweder durch eine Benutzereingabe oder basierend auf einer fest vorgegebenen Wahrscheinlichkeitsverteilung geschehen. Non-spontaneous Nachrichten könnten somit zum Beispiel eine höhere Übergangswahrscheinlichkeit erhalten als spontaneous Nachrichten. Dies könnte damit begründet werden, dass in einer Situation in der sich ein Roboterarm gerade zur Presse bewegt und schon in der Abbremsphase ist, es wahrscheinlicher ist, dass als nächstes das non-spontaeous Nachrichten-Event *arrivedAtPress* auftreten wird und nicht das spontaneous Nachrichten-Event *blankArrived*.

Eine weitere Variante wäre, dass die MSD Spezifikation um die Möglichkeit erweitert wird Annahmen über Übergangswahrscheinlichkeiten einzelner Events zu modellieren. Wie so eine entsprechende Erweiterung aussehen könnte wurde in dieser Arbeit jedoch nicht weiter verfolgt.

Im Folgenden wird die Variante 1 verwendet und somit wird davon ausgegangen, dass das Auftreten der Umwelt-Events in jedem Zustand gleichverteilt ist. Somit wird von einer Art Average-Case ausgegangen.

In der folgenden Abbildung ist eine mögliche Ablaufsteuerung für die gekürzte Produktionsanlage zu sehen, bei der die Transitionen eine Übergangswahrscheinlichkeit entsprechend Variante 1 erhalten haben.

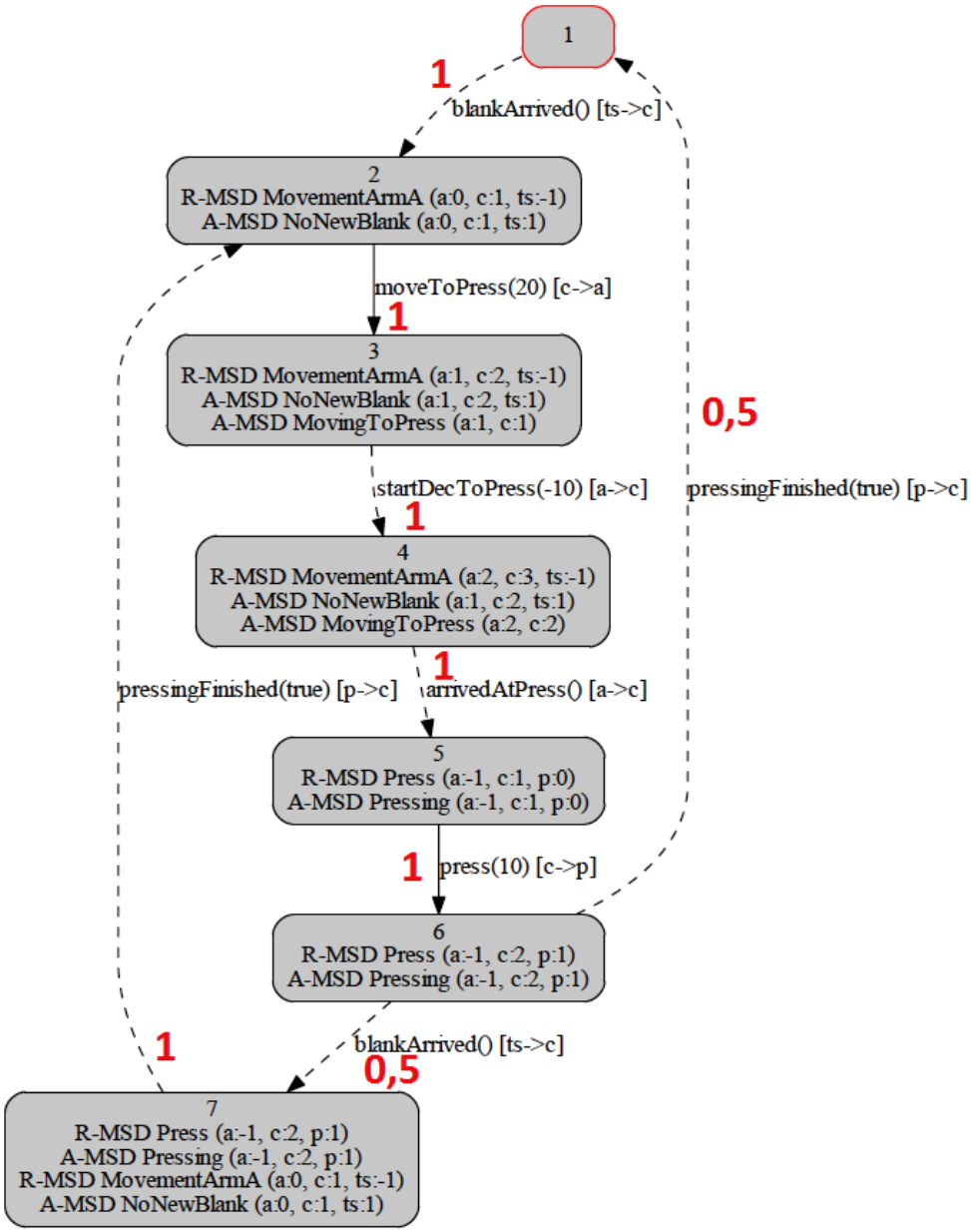


Abbildung 24: Ablaufsteuerung mit Übergangswahrscheinlichkeiten.

Wenn die Übergangswahrscheinlichkeiten aller Transitionen einer Markov-Kette bekannt sind, dann können die Auftrittswahrscheinlichkeiten von den verschiedenen Zuständen einer Ablaufsteuerung mit Hilfe eines linearen Gleichungssystems berechnet werden, wie es Kapitel 2.3 beschrieben worden ist.

Anhand der Auftrittswahrscheinlichkeit der Zustände kann wiederum abgeleitet werden, wie hoch die Auftrittswahrscheinlichkeit von den verschiedenen Transitionen ist. Besitzt ein Zustand zum Beispiel nur eine ausgehende Transition, dann ist die Auftrittswahrscheinlichkeit dieser Transition gleich der Auftrittswahrscheinlichkeit des Zustands. Besitzt ein Zustand mehrere ausgehende Transitionen, dann wird die Auftrittswahrscheinlichkeit einer Transition berechnet, indem die Auftrittswahrscheinlichkeit des Zustandes mit der Übergangswahrscheinlichkeit der Transition multipliziert wird.

5.6 Bewertung einer Ablaufsteuerung

Durch die vorher durchgeführten Schritte wurde für jede Transition eine Auftrittswahrscheinlichkeit (Kapitel 5.5) ermittelt und ein Wert der den Energiebedarf darstellt, der von einer externen Energiequelle bezogen werden muss (Kapitel 5.3). Falls die Ablaufsteuerung mindestens eine Transition besitzt, die eine Ziel-Nachricht darstellt und diese Transitionen eine Auftrittswahrscheinlichkeit größer 0 besitzt, dann kann der durchschnittliche Energiebedarf pro Zielerreichung des kompletten Systems mit Hilfe der folgenden Gleichung berechnet werden. Alle Transitionen der Ablaufsteuerung werden zuvor durchnummeriert.

$$Bewertung = \frac{\sum_{i=1}^{N} \left(Auftrittswahrscheinlichkeit_i * Energiebedarf_i \right)}{\sum_{i=1}^{N} \left(Auftrittswahrscheinlichkeit_i * isGoal_i \right)}$$

N : *Anzahl der Transitionen*,

$Energiebedarf_i$: *Energiebedarfswert von Transition* i,

$$isGoal_i = \begin{cases} 0, & \text{wenn Transition i keine Ziel-Nachricht darstellt,} \\ 1, & \text{wenn Transition i eine Ziel-Nachricht darstellt} \end{cases}$$

$Auftrittswahrschenilichkeit_i$: *Auftrittswahrscheinlichkeit von Transition* i

Als Letztes muss nur noch die Ablaufsteuerung mit der niedrigsten Bewertung ausgewählt und ausgegeben werden.

5.7 Zusammenfassung der Brute-Force-Methode

Der Vorteil dieses Lösungsvorschlags besteht darin, dass alle Ablaufsteuerungen und deren Bewertung hergeleitet werden. Somit ist es am Ende zum Beispiel möglich, dass die besten 10 Ablaufsteuerungen ausgegeben werden und nicht nur die beste Ablaufsteuerung. Dies ist vor Allem deswegen sehr hilfreich, da die Bewertungen der Ablaufsteuerungen auf gewissen Annahmen beruhen, wie zum Beispiel den Wahrscheinlichkeiten für das Auftreten der Umwelt-Events. Somit kann es sein, dass die Ablaufsteuerung mit der besten Bewertung in der Praxis nicht unbedingt die beste Ablaufsteuerung ist. Ein Ingenieur könnte somit die besten 10 Ablaufsteuerung analysieren und anhand seiner eigenen Erfahrungen die beste von ihnen auswählen.

Dass der Algorithmus jede mögliche Ablaufsteuerung erstellt und bewertet ist jedoch zugleich auch sein größter Nachteil, denn die Anzahl der möglichen Ablaufsteuerungen steigt exponentiell mit der Tiefe des Zustandsgraphen, wodurch auch gleichzeitig die Laufzeit des Algorithmus steigt. Besonders extrem steigt die Anzahl der möglichen Ablaufsteuerungen in Zuständen die mehrere ausgehende Umwelt-Transitionen besitzen. In der folgenden Abbildung soll dies verdeutlicht werden.

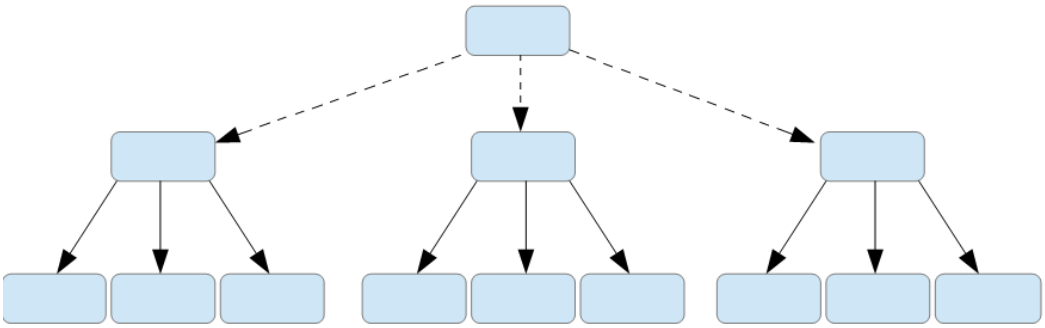


Abbildung 25: Exponentielles Wachstum der Ablaufsteuerungen.

Die Abbildung 25 zeigt einen Zustandsgraph, bei dem der Zustand auf der ersten Stufe, drei Umwelt-Transitionen besitzt. Auf der zweiten Stufe besitzt jeder Zustand 3 System-Transitionen. Für diese zwei Stufen des Zustandsgraphen gibt es insgesamt 27 verschiedene mögliche Ablaufsteuerungen, da für jede Umwelt-Transition auf der ersten Stufe es drei Möglichkeiten auf der zweiten Stufe gibt. Somit gibt es 3*3*3 = 27 verschiedene Kombinationen und alle davon stellen eine theoretische Ablaufsteuerung dar. Das heißt, dass dieser Algorithmus nur für MSD Spezifikationen mit sehr geringem Umfang geeignet ist.

6 Lösungsvorschlag Markov-Entscheidungsprozess

In diesem Kapitel wird eine weitere Möglichkeit vorgestellt, mit der die optimale Ablaufsteuerung aus einer MSD Spezifikation abgeleitet werden kann. Die Grundidee dabei ist es, den kompletten **erweiterten** Zustandsgraphen als Markov-Entscheidungsprozess anzusehen. Dazu muss zunächst wieder der erweiterte Zustandsgraph erstellt werden, indem als Erstes der komplette Zustandsgraph mit dem PlayOutButWaiting-PossibleWithActiveEvents Algorithmus aufgebaut wird, wie es in Kapitel 2.2.7 beschrieben worden ist. Anschließend muss der Graph wieder gesäubert werden, wie es in Kapitel 5.2 beschrieben worden ist und als Letztes muss jeder Transition des Zustandsgraphen wieder ein Wert zugewiesen werden, der den Energiebedarf aus einer externen Energiequelle darstellt, wie es in Kapitel 5.3 beschrieben worden ist. Der so erhaltene Zustandsgraph stellt dann den Graphen für den Markov-Entscheidungsprozess dar. Die Werte der Transitionen, die den Energiebedarf darstellen, werden dabei als **Kosten** angesehen. Die möglichen Aktionen, die in einem Zustand ausgeführt werden können, werden wieder durch die ausgehenden Transitionen eines Zustandes dargestellt. Das heißt, jede System-Transition einzeln und alle Umwelt-Transitionen zusammen stellen jeweils eine Aktion dar. Des Weiteren erhalten alle System-Transitionen wieder die Übergangswahrscheinlichkeit 100% und der Einfachheit halber erhalten alle Umwelt-Transitionen wieder eine Übergangswahrscheinlichkeiten auf Basis einer Gleichverteilung. Wie auch schon in Kapitel 5.5 beschrieben wurden ist, könnte jedoch auch eine andere Wahrscheinlichkeitsverteilung für die Umwelt-Transitionen verwendet werden. Das Ziel ist es nun, die Aktionen in jedem Zustand so zu wählen, dass in der Folge die Kosten (Energiebedarf) pro Zielerreichung minimiert werden.
Für die Lösung eines Markov-Entscheidungsprozess gibt es bereits ein bekanntes Wert-Iteration-Verfahren [1, S.854], das auf den Bellmanschen Optimalitätsprinzip basiert. Dieses Verfahren kann auf das hier vorliegende Problem angewendet werden.
Das Verfahren basiert darauf, dass für jeden Zustand die geschätzten minimal Kosten iterativ berechnet werden. Die geschätzten Kosten eines Zustandes in einem Iterationsschritt setzten sich dabei aus den unmittelbaren Kosten, die durch die Wahl der Aktion und die damit verbundenen Transitionen auftreten, und den geschätzten Kosten der Zielzustände der gewählten Aktion, zusammen, indem die unmittelbaren Kosten und die geschätzten Kosten einer Aktion summiert werden. In jedem Iterationsschritt wird für jeden Zustand die Aktion ausgewählt, die zu den geringsten geschätzten Kosten führt. Zu Beginn des Verfahrens werden die geschätzten Kosten aller Zustände auf 0 gesetzt. Das heißt, im ersten

Iterationsschritt, werden nur die Kosten, die durch die gewählten Transitionen auftreten berücksichtigt. In der folgenden Abbildung ist ein kleines Beispiel zu sehen:

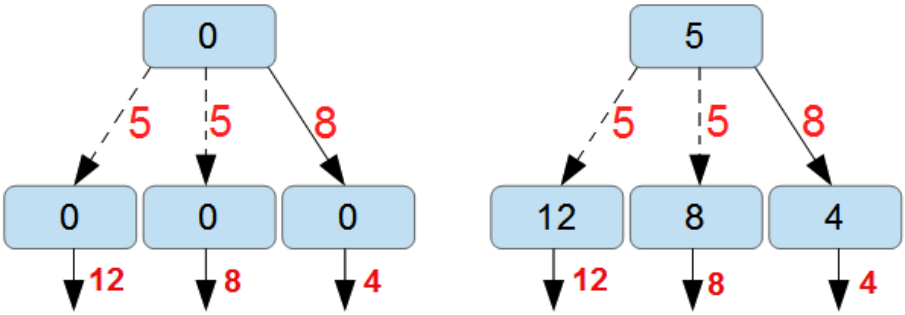


Abbildung 26: Iterationsschritt 1 und 2 des Wert-Iterations-Verfahren.

In der Abbildung 26 ist ein Zustand mit seinen direkten Folgezuständen zu sehen. Die Werte in den Zuständen stellen die geschätzten Minimalkosten des letzten Iterationsschritts dar und die Werte an den Transitionen stellen die unmittelbaren Kosten (Energiebedarf) dar. Auf der linken Seite ist die Situation vor dem ersten Iterationsschritt dargestellt. Alle Zustände besitzen deswegen noch geschätzte Kosten von 0. Im Zustand 1 gibt es nun zwei mögliche Aktionen. Entweder es werden die beiden Umwelt-Transitionen gewählt oder die System-Transition. Da die geschätzten Kosten der Zustände im Ersten Schritt für die Folgezustände alle 0 sind, werden nur die Kosten der Transition berücksichtigt. Somit sind die Kosten, wenn die Umwelt-Transitionen gewählt werden: (0,5 * 5) + (0,5 * 5) = 5. Die Kosten für die System-Transition beträgt 1* 8 = 8. Somit wird im ersten Iterationsschritt, die Aktion mit den Umwelt-Transitionen gewählt und die geschätzten Kosten für den Zustand werden für den nächsten Iterationsschritt auf 5 gesetzt. Auf die gleiche Weise werden auch die geschätzten Kosten der anderen Zustände berechnet. Auf der rechten Seite ist dann die Situation im nächsten Iterationsschritt dargestellt. In diesem Schritt sind die geschätzten Kosten der Folgezustände ungleich 0 und somit werden auch sie mit einbezogen. Die Kosten für die Umwelt-Transitionen beträgt dann 0,5 * (5 + 12) + 0,5 * (5 + 8) = 15. Die geschätzten Kosten für die System-Transition beträgt 1 * (8 + 4) = 12. Somit wird im zweiten Iterationsschritt die System-Transition als die bevorzugte Aktion ausgewählt und die geschätzten Minimalkosten des Zustandes auf 12 gesetzt.
Mit Hilfe des iterativen Verfahrens werden somit für jeden Zustand die minimal möglichen Kosten berechnet, wenn genau k Aktionen durchgeführt werden dürfen. Dabei steht k für die Anzahl der Iterationen. Das würde bedeuten, dass die geschätzten Minimalkosten mit jedem weiteren Iterationsschritt immer größer werden würden. Deswegen wird auf die gleiche Weise in jedem Iterationsschritt auch die Anzahl der Erreichung einer Ziel-Transition berechnet. Dafür erhalten alle Transitionen, die eine

Ziel-Transition darstellen, den Wert 1 und alle anderen den Wert 0. In jedem Iterationsschritt wird dann sowohl die geschätzten Minimalkosten als auch die Anzahl der Zielerreichung für jede Aktion berechnet. Dabei wird in jedem Zustand, die Aktion gewählt, die den kleinsten Quotienten der geschätzten Minimalkosten und der Anzahl an Zielerreichungen aufweist. Da der Quotient sich nicht berechnen lässt, solange die Anzahl an Zielerreichungen bei einer Aktion gleich 0 ist, wird in diesen Situationen der Wert des Quotienten als unendlich angenommen. Somit wird für jeden Zustand indirekt der minimale Energiebedarf pro Zielerreichung berechnet. Da der Graph zyklisch ist, konvergiert der berechnete Quotient auch gegen einen Erwartungswert und somit ist es möglich, sobald das Ergebnis hinreichend genau ist und somit nur noch eine geringe Rest-Fehlerwahrscheinlichkeit besteht, die Iterationen zu beenden.
Wann ein Ergebnis hinreichend genau ist, kann zum Beispiel ermittelt werden, indem der Quadratische Fehler nach jedem Iterationsschritt zwischen dem aktuellem und dem vorherigen Iterationsschritt berechnet wird. Sobald der Quadratische Fehler einen vorher festgelegten Wert unterschreitet, kann die Iteration abgebrochen werden und die gewählten Aktionen als Ergebnis ausgelesen werden.
In der folgenden Abbildung ist ein Zustandsgraph dargestellt, bei dem das Iterationsverfahren angewandt worden ist. Die gewählten Aktionen der Zustände sind rot markiert.

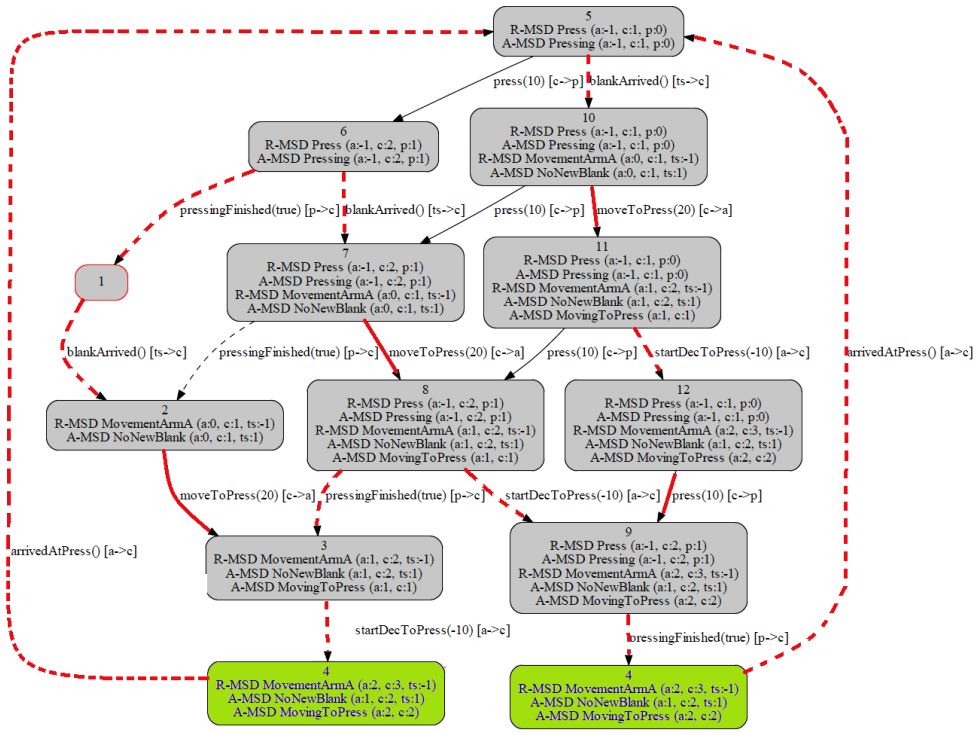


Abbildung 27: Erweiterter Zustandsgraph mit markierten Aktionen.

Nachdem die Iteration beendet wurde, muss nur noch eine Ablaufsteuerung aus dem kompletten Zustandsgraphen abgelesen werden. Dazu wird der Graph vom Startzustand aus durchlaufen und alle besuchten Transitionen und Zustände der Ablaufsteuerung hinzugefügt. Beim Durchlaufen des Zustandsgraphen werden jedoch nur die ausgehenden Transitionen verwendet, die zu den gewählten Aktionen in den jeweiligen Zuständen gehören. Zustände, die auf diese Weise nicht vom Startzustand aus erreicht werden können und alle Transitionen, die zu einer Aktion gehören, die in einem Zustand nicht gewählt worden sind, werden verworfen. In der folgenden Abbildung ist die Ablaufsteuerung zu sehen, die aus dem Zustandsgraphen aus Abbildung 27 extrahiert worden ist.

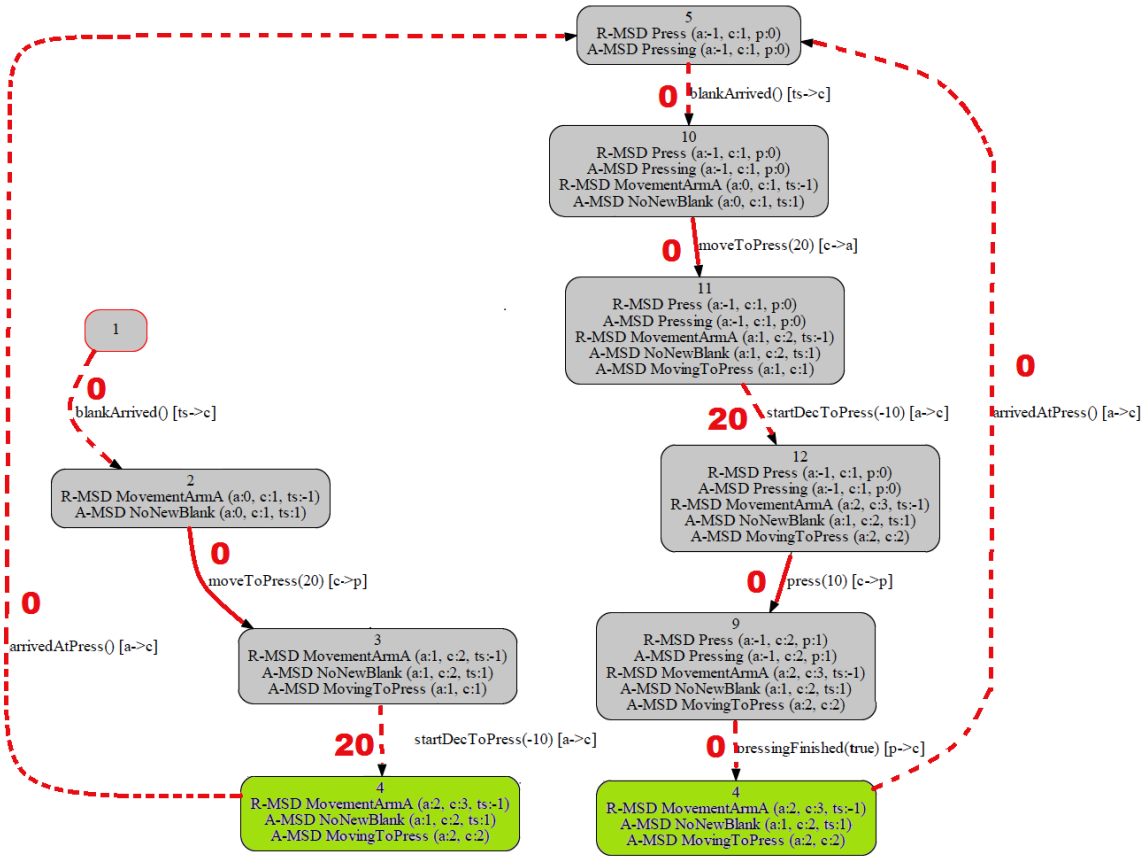


Abbildung 28: Ermittelte Ablaufsteuerung mit Hilfe Wert-Iterations-Verfahrens.

6.1 Zusammenfassung von Markov-Entscheidungsprozess

Der Vorteil bei diesem Verfahren ist, dass im Gegensatz zu dem vorgestellten Verfahren aus Kapitel 5, nicht alle Ablaufsteuerungen einzeln bewertet werden müssen und dass das Verfahren dadurch deutlich schneller ist. Die Laufzeit ist bei diesem Verfahren nur noch von der Anzahl an Zuständen im erweiterten Zustandsgraphen und der Anzahl an benötigten Iterationsschritten abhängig.

Der Nachteil bei diesem Verfahren ist jedoch, dass am Ende nur die Ablaufsteuerung ausgegeben wird, die die beste Bewertung besitzt und es somit nicht möglich ist, zum Beispiel, die besten 10 Ablaufsteuerungen auszugeben. Somit gibt es für einen Ingenieur nicht mehr die Möglichkeit seine eigenen Erfahrungen einfließen zu lassen, indem er Ablaufsteuerungen ausschließt, die nur schwer oder gar nicht umgesetzt werden können. Das heißt, wenn er mit der erhaltenen Ablaufsteuerung unzufrieden ist, müsste er die Modellierung dementsprechend anpassen.

7 Implementierung des Prototypen

Im Rahmen dieser Masterarbeit wurde auch ein Prototyp erstellt. Der Prototyp wurde als Plugin für das Werkzeug ScenarioTools [12] umgesetzt. ScenarioTools ist ein Werkzeug für die Erstellung von MSD Spezifikationen. Für die Anfertigung von MSDs stellt ScenarioTools einen grafischen Editor bereit und zudem ist es unter anderem möglich eine MSD Spezifikation auf Basis des PlayOut bzw. PlayOutButWaitingWithActiveEvents zu analysieren. Unter anderem wurde für ScenarioTools auch schon ein Algorithmus umgesetzt mit dem es möglich ist die Realisierbarkeit einer MSD Spezifikation zu überprüfen.

In dem erstellten Prototyp wurde der Lösungsvorschlag mit dem iterativen Verfahren zur Lösung eines Markov-Entscheidungsprozesses umgesetzt. Es wurde bisher jedoch noch keine Abbruchbedingung für das Verfahren implementiert. Die Anzahl der Iterationsschritte müssen somit manuell angegeben werden. Am Ende jedes Iterationsschrittes wird der Energiebedarf der bis dahin besten gefundenen Ablaufsteuerung ausgegeben. Dafür wird die Ablaufsteuerung mit Hilfe des beschriebenen Verfahrens in Kapitel 5.5 und Kapitel 5.6 bewertet. Für die Lösung des linearen Gleichungssystem für die Markov Kette, wurde das Java-Package Jama [11] verwendet. Zudem wurde in dem Prototyp auch noch nicht das Konzept der Extra-Datenstruktur implementiert. Das heißt, dass in Situationen, in denen zum Beispiel zwei energieverbrauchende Vorgänge aktiv sind und ein energieerzeugender Vorgang gerade beendet wird, die Energie direkt verrechnet und nicht in der Extra-Datenstruktur abgespeichert wird. Beim Prototypen erhält in so einer Situation der Vorgang die Energie, der als Erstes in der Liste der Umwelt-Objekte auftaucht. Das heißt, dass der Prototyp für komplexere Systeme noch nicht die optimale Ablaufsteuerung synthetisieren kann.

8 Evaluierung des Prototypen

Mit Hilfe des erstellten Prototypen wurde auch eine Evaluierung des zweiten Lösungsvorschlags (Markov-Entscheidungsprozess) durchgeführt. Für diesen Zweck wurde das einführende Beispiel aus Kapitel 2.1 als MSD Spezifikation in zwei verschiedenen Varianten modelliert.

Für beide Varianten wurde dann mit Hilfe des Prototypen versucht eine optimale Ablaufsteuerung zu finden. Dabei wurde zum einen überprüft, wie gut die gefundene Ablaufsteuerung ist und zum anderen nach wie vielen Iterationsschritten dieses Verfahren gegen eine feste Ablaufsteuerung konvergiert.

8.1 Evaluierung anhand einer einfachen Modellierung

Die erste Variante der Modellierung des einführenden Beispiels aus Kapitel 2.1 besitzt nur den ArmA, die Presse und den TischSensor. ArmB wurde nicht modelliert. Dabei wurden die folgenden energieverbrauchenden und energieerzeugenden Vorgänge modelliert:

ArmA	**Presse:**
moveToPress(20) startDecToPress(-10) moveToTable(10) startDecToTable(-5)	press(10)

Da aufgrund der Modellierung ArmA das einzige Umwelt-Objekt ist, dass Energie erzeugt, kann der Energiebedarf von ArmA nicht reduziert werden. Der Energiebedarf von der Presse hingegen könnte theoretisch durch die generierte Energie von ArmA auf 0 reduziert werden. Das heißt, dass die bestmögliche Ablaufsteuerung für diese MSD Spezifikation keine bessere Bewertung als **20 + 10 + max(0, 10 - 5 - 10) = 30** erhalten kann. Zudem ist aufgrund der Modellierung davon auszugehen, dass in der bestmöglichen Ablaufsteuerung der Pressvorgang der Presse immer gestartet wird, wenn ArmA sich in der Abbremsphase zur **Presse** befindet. Denn in dieser Phase werden -10 Energie generiert. In der Abbremsphase zum Tisch hingegen werden nur -5 Energie generiert.
Bei der Anwendung des Prototypen auf diese Modellierung wurde auch genau dieses Ergebnis nach 7 Iterationsschritten erzielt. In der Abbildung 29 ist die entsprechende Ablaufsteuerung zu sehen:

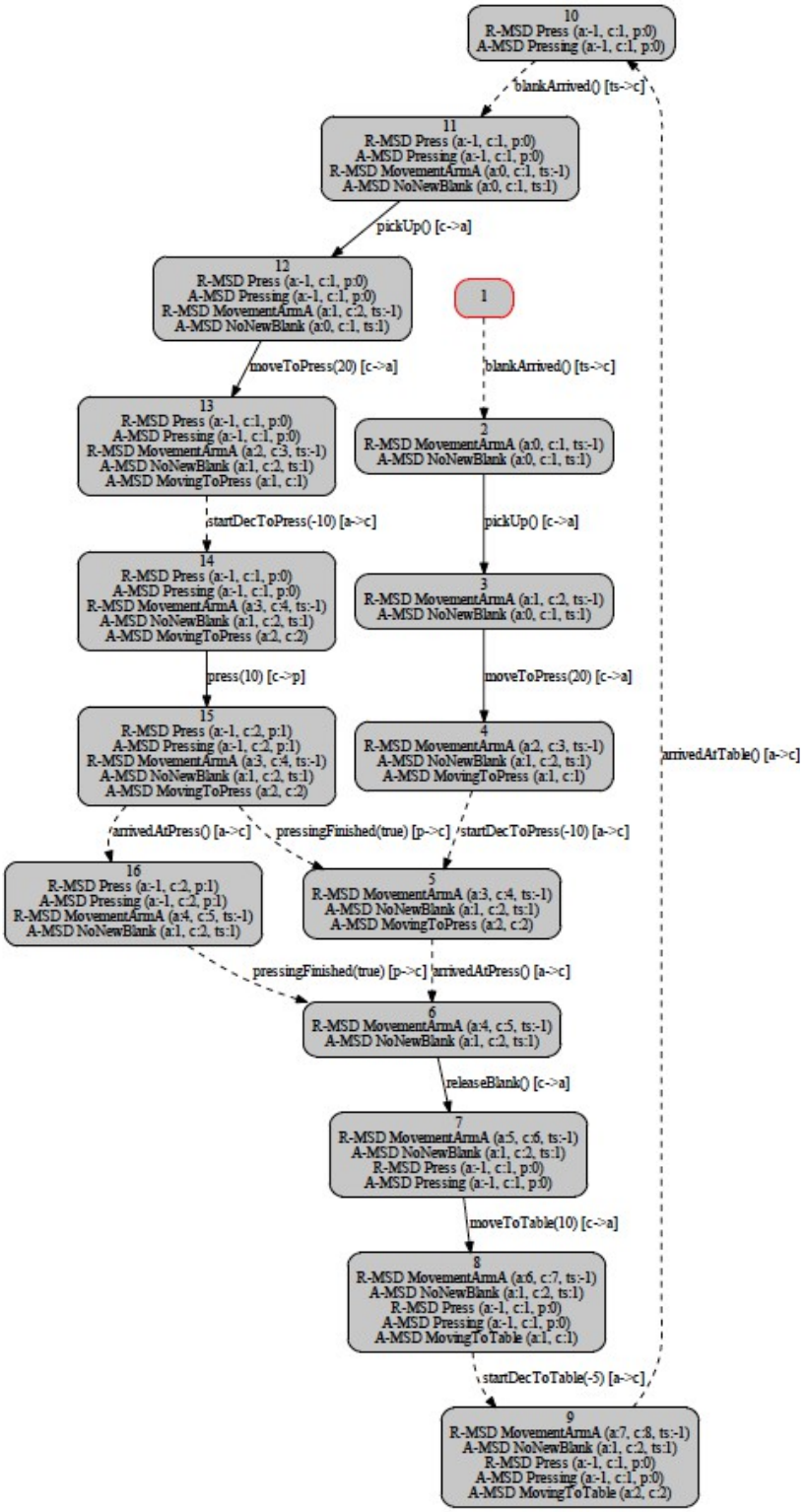


Abbildung 29: Ablaufsteuerung, bei der der Pressvorgang gestartet wird, sobald ArmA in die Abbremsphase zur Presse übergeht.

Die erste Variante der Modellierung wurde auch nochmal mit den folgenden Energiewerten umgesetzt, um zu testen, ob der Prototyp auch bei anderen Werten die optimale Ablaufsteuerung findet:

ArmA	**Presse:**
moveToPress(10) startDecToPress(-5) moveToTable(20) startDecToTable(-10)	press(10)

Aufgrund der geänderten Werte für die energieverbrauchenden und energieerzeugenden Vorgänge von ArmA sollte der Prototyp nun eine Ablaufsteuerung finden, bei der der Pressvorgang der Presse immer gestartet wird, wenn ArmA sich in der Abbremsphase zum **Tisch** befindet.

Bei der Anwendung des Prototypen auf die Modellierung wurde die erwartete Ablaufsteuerung nach 4 Iterationsschritten gefunden. In der Abbildung 30 ist die entsprechende Ablaufsteuerung zu sehen.

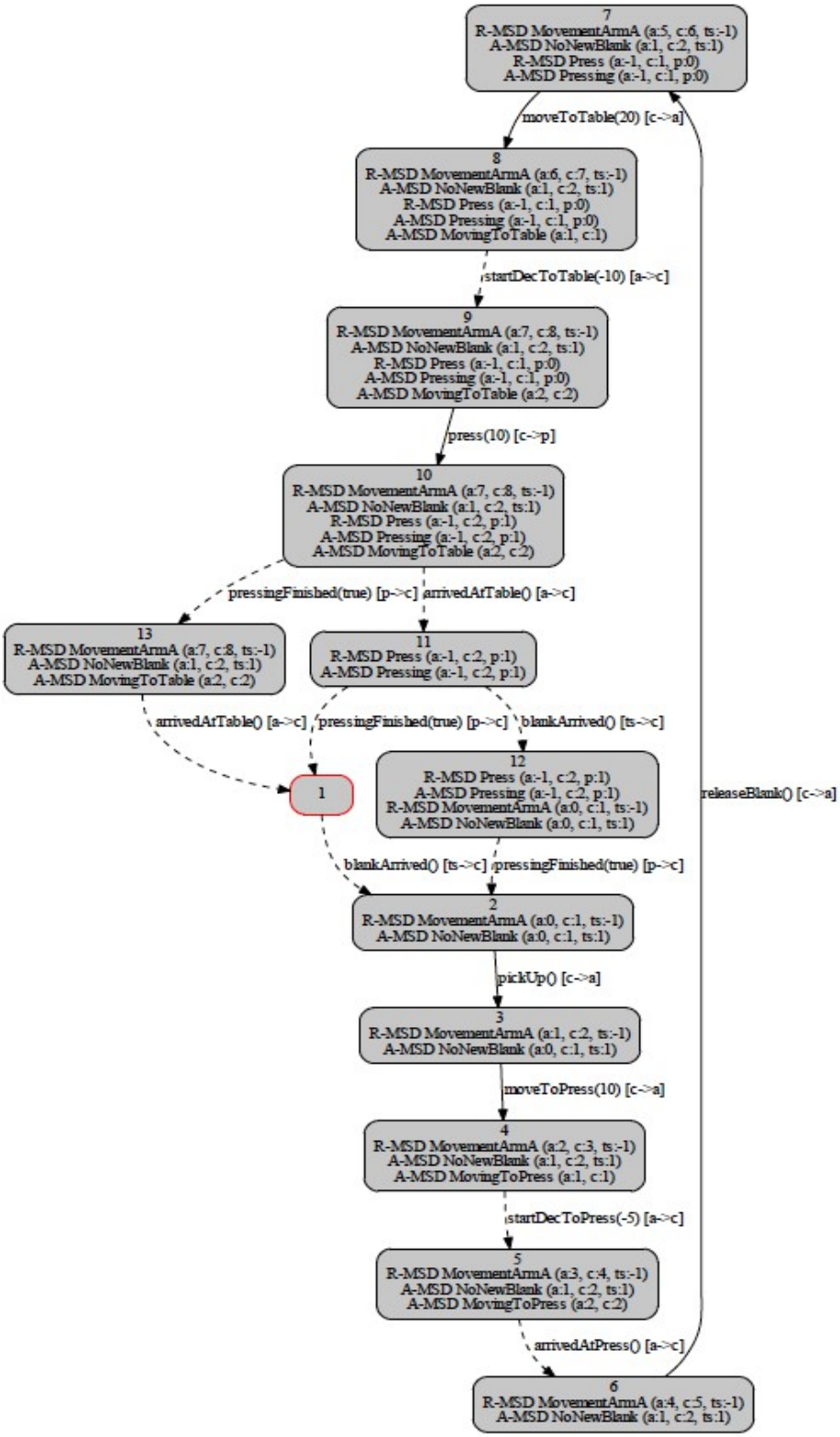


Abbildung 30: Ablaufsteuerung, bei der der Pressvorgang gestartet wird, sobald ArmA in die Abbremsphase zum Tisch übergeht.

8.2 Evaluierung anhand einer komplexeren Modellierung

Als Letztes wurde der Prototyp auch noch an dem vollständig modellierten Beispiel aus Kapitel 2.1 getestet. In der MSD Spezifikation wurden dazu folgende energieverbrauchenden und energieerzeugenden Vorgänge modelliert:

ArmA:	**ArmB:**	**Presse:**
MoveToPress(20) startDecToPress(-10) moveToTable(20) startDecToTable(-10)	MoveToPress(20) startDecToPress(-10) moveToDepositBelt(20) startDecToDepositBelt(-10)	press(10)

Bei dieser MSD Spezifikation ist es theoretisch möglich, dass die komplette Energie wiederverwendet werden kann. Somit ist die bestmögliche Bewertung, die eine Ablaufsteuerung für diese MSD Spezifikation erhalten kann 20+20+20+20+10-10-10-10-10 = **50.** Ob eine Ablaufsteuerung synthetisiert werden kann, die eine Bewertung von 50 besitzt, hängt jedoch auch von den spezifizierten Anforderungen und Annahmen ab.

Für diese MSD Spezifikation wurde nach **80** Iterationsschritten eine Ablaufsteuerung gefunden, die eine Bewertung von **56,40625** besaß (In den folgenden Iterationsschritten hat sich die Bewertung nicht mehr verändert). Das heißt, dass mit dieser Ablaufsteuerung nicht die komplette Energie wiederverwendet werden kann. Es kann jedoch immerhin bis zu 33,59375/40 = **83,98%** der Energie wiederverwendet werden. Da beim Prototypen das Konzept der Extra-Datenstruktur noch nicht umgesetzt worden ist, könnte es sein, dass es noch eine bessere Ablaufsteuerung gibt. Es kann jedoch auch sein, dass aufgrund der Anforderungen und Annahmen die gefundene Ablaufsteuerung schon das Optimum darstellt.

9 Fazit

Das Ziel dieser Arbeit war es einen Algorithmus zu finden, der aus einer MSD Spezifikation eine energieeffiziente Ablaufsteuerung synthetisiert, die so viel Bremsenergie, wie möglich wiederverwendet. In dieser Arbeit wurden dafür zwei mögliche Lösungsvorschläge vorgestellt. Der Vorteil des ersten Lösungsvorschlags ist, dass am Ende mehrere energieeffiziente Ablaufsteuerung ausgegeben werden können und ein Ingenieur somit aus verschiedenen Varianten eine auswählen kann. Der Nachteil dieses Vorschlags ist, dass er eine exponentielle Laufzeit besitzt und somit nur für Systeme mit geringem Umfang geeignet ist. Der Vorteil des zweiten Lösungsvorschlags ist, dass er eine deutlich bessere Laufzeit als der erste Vorschlag besitzt. Der Nachteil ist, dass am Ende nur die Ablaufsteuerung mit der besten Bewertung ausgegeben wird. Das heißt, dass ein Ingenieure keine Alternativen zur Verfügung hätte, falls diese Ablaufsteuerung in der Praxis nicht umgesetzt werden kann.

Der zweite Lösungsvorschlag wurde im Rahmen dieser Arbeit auch als Prototyp für das Werkzeug ScenarioTools umgesetzt und auch evaluiert. Bei der Evaluierung wurden zum einen die Anzahl der benötigten Iterationsschritte und zum anderen die Güte der synthetisierten Ablaufsteuerung betrachtet. Für einfache MSD Spezifikationen konnte auch gezeigt werden, dass der Prototyp die optimale Ablaufsteuerung findet. Bei komplexeren MSD Spezifikationen kann jedoch nicht mit Sicherheit gesagt werden, ob die optimale Ablaufsteuerungen gefunden wird, da es keine effektive Möglichkeit gibt dies zu überprüfen. Die größte Schwachstelle der beiden Lösungsvorschläge sind die Annahmen, die getroffen werden mussten, wie zum Beispiel die Wahrscheinlichkeitsverteilung für die Umwelt-Nachrichten bei der Markov-Kette und dem Markov-Entscheidungsprozess. Vor allem bei komplexeren Systemen, kann es deswegen sein, dass aufgrund der getroffenen Annahmen eine Ablaufsteuerung synthetisiert wird, die in der Praxis nicht die energieeffizienteste Ablaufsteuerung darstellt. Auch wenn die vorgestellten Lösungsvorschläge noch Schwächen haben, stellen sie eine gute Basis dar, auf der zukünftigen Arbeiten, die die gleiche Problemstellung lösen möchten, aufbauen können.

10 Ausblick

Die größte Schwäche der vorgestellten Lösungsvorschläge sind die getroffenen Annahmen für die Wahrscheinlichkeitsverteilung der Umwelt-Nachrichten. Um die Lösungsvorschläge zu verbessern, könnte deswegen versucht werden ein Konzept zu erarbeiten, mit dem es möglich ist, genauere Informationen über die Wahrscheinlichkeitsverteilung der Umwelt-Nachrichten in einer MSD Spezifikation zu modellieren. Des Weiteren könnte in einer zukünftigen Arbeit versucht werden einen Algorithmus zu entwickeln, der bei der Synthese einer Ablaufsteuerung nicht nur auf Energieeffizienz, sondern auch auf die benötigte Zeit, die für die Erreichung eines Ziels benötigt wird, achtet. Denn eine Ablaufsteuerung, die zwar 10% weniger Energie als eine vergleichbare Ablaufsteuerungen benötigt, zugleich aber doppelt so lange für die Fertigstellung eines Produkts braucht, ist in der Praxis im Allgemeinen ungeeignet. Eine weitere Möglichkeit, um die Lösungsvorschläge zu erweitern besteht darin, den Energiebedarf und die generierte Energie nicht mehr als Integerwert anzugeben, sondern in einer detaillierteren Form.

Literaturverzeichnis

[1] Baier C, Katoen J-P. 2008. Principles of Model Checking (Representation and Mind Series). The MIT Press.

[2] Damm W, Harel D. LSCs: Breathing Life into Message Sequence Charts. Formal Methods in System Design; Vol. 19; 2001. p. 45-80

[3] Greenyer J, Brenner C, Cordy M, Heymans P, Gressi E. Incrementally Synthesizing Controllers from Scenario-Based Product Line Specifications. Proc. of the 2013 9th Joint Meeting on Foundations of Software Engineering; 2013. p.433-443

[4] Greenyer J, Hansen C, Jens Kotlarski, Tobias Ortmaier: Towards Synthesizing Energy-Efficient Controllers for Modern Production Systems from Scenario-Based Specifications, In Proceedings of the 2nd International Conference on System-integrated Intelligence (SysInt 2014), 2014.

[5] Harel D, Maoz S. Assert and negate revisited: Modal semantics for UML sequence diagrams. Software and Systems Modeling (SoSyM); Vol.7; 2008. p. 237-252

[6] Hansen C, Kotlarski J, Ortmaier T. Path Planning Approach for the Amplification of Electrical Energy Exchange in Multi Axis Robotic Systems. Proc.of the 2013 IEEE Int. Conf. on Mechatronics and Automation, 2013,p.44-50

[7] Hansen C, Kotlarski J, Ortmaier T. Experimental Validation of Advanced Minimum Energy Robot Trajectory Optimization. Proc. of the 16th Int. Conf. On Advanced Robotics, 2013

[8] Hansen C, Öltjen J, Meike D, Ortmaier T. Enhanced Approach for Energy" Efficient Trajectory Generation of Industrial Robots. Proc. of the 2012 IEEE Int. Conf. on Automation Science and Engineering, 2012, p. 1–7

[9] Harel D, Marelly R, Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine, Springer-Verlag, 2003.

[10] Harel D. , Marelly R. Specifying and Executing Behavioral Requirements: The Play-In/Play-Out Approach. Software and System Modeling (SoSyM) 2:2003, 2002.

[11] JAMA : A Java Matrix Package, http://math.nist.gov/javanumerics/jama/, letzter Zugriff: Juni 2015

[12] ScenarioTools, http://scenariotools.org/, letzter Zugriff: Juni 2015

[13] ScenarioTools – MSD Specification, http://scenariotools.org/msd-specifications/, letzter Zugriff : Juni 2015

Abbildungsverzeichnis

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 3.06.2015 _____

Florian Werner