

Studienarbeit

Repräsentation von Bedienungsanleitungen durch BDDs

Kathrin Kaschner

30. Januar 2006



Humboldt-Universität zu Berlin
Institut für Informatik
Unter den Linden 6
10099 Berlin

Inhaltsverzeichnis

1	Einleitung	3
2	Hintergrund	5
2.1	Binäre Entscheidungsdiagramme (BDD)	5
2.2	Bedienungsanleitungen	9
3	BDD-Repräsentationen für Bedienungsanleitungen	16
3.1	Erster Ansatz: Explizite Darstellung der Strategien im Shared BDD	16
3.1.1	Vorgehensweise	16
3.1.2	Auswertung	18
3.2	Zweiter Ansatz: Repräsentation von Struktur und Annotation in einem BDD . . .	20
3.2.1	Vorgehensweise	20
3.2.2	Auswertung	21
3.3	Dritter Ansatz: Trennung von Struktur und Annotation	23
3.3.1	Vorgehensweise	23
3.3.2	Auswertung	24
4	Zusammenfassung	26

1 Einleitung

Die Globalisierung mit ihrer zunehmenden weltweiten Vernetzung aller gesellschaftlichen Bereiche führt auf dem Gebiet der Wirtschaft zu einer immer stärkeren internationalen Arbeitsteilung. Der weltweite Wettbewerb zwingt die Unternehmen zu einer raschen Reaktion auf ständig veränderte Marktbedingungen, um gegen die Konkurrenz bestehen zu können. Das bedeutet auch, dass die Anforderungen an die eingesetzte Software sich ständig ändern. Eine Anpassung durch nachträgliches Umprogrammieren ist bei klassischen Anwendungen i.a. sehr aufwendig und kostspielig. Um die geforderte Flexibilität dennoch zu erreichen, wird vermehrt angestrebt, Anwendungen auf der Basis verteilter Softwarekomponenten zusammenzusetzen.

Jede der Komponenten ist ein Service, der über sein *Interface* eine wohl definierten Funktionalität zur Verfügung stellt und durch seinen *Namen* (ID) netzwerkweit identifiziert werden kann. Die Kommunikation mit einem Service ist nur durch Nachrichtenaustausch über das Interface möglich. Hierfür werden standardisierte Nachrichtenprotokolle verwendet. Auf diese Weise bleibt die *interne Struktur* eines Service verborgen und seine Nutzung ist unabhängig von der verwendeten Programmiersprache und Plattform. Innerhalb der Anwendung stellen nun Services die kleinste logische Einheit dar.[Col04a]

Mit dem Paradigma der *Serviceorientierten Architektur* (SOA) wird ein Mechanismus bereitgestellt, wie Services zusammengefügt werden können: Der *Service Provider* (Anbieter) stellt einen Service zur Verfügung. Damit sein Service von einem *Service Requester* (Kunden) gefunden und genutzt werden kann, publiziert er eine Beschreibung seines Services in einem zentralen Verzeichnis, das vom *Service Broker* (Vermittler) verwaltet wird. Ein Service Requester kann nun beim Broker anfragen, ob es im Verzeichnis einen zu ihm passenden Service gibt. Ist das der Fall, so können sich die Services aneinander binden und miteinander interagieren.[Col04b]

Ein Service ist nur bei seinem Besitzer gespeichert und kann dort ggf. mehrfach aufgerufen werden. Auf diese Weise wird garantiert, dass alle darauf zugreifenden Anwendungen stets dieselbe (aktuelle) Version nutzen und Änderungen ggf. nur an einer Stelle vorgenommen werden müssen. Funktionserweiterungen können einfach durch Hinzunahme bereits bestehender Services realisiert werden. So kann ein Unternehmen ohne großen technischen Aufwand flexibel auf neue Marktanforderungen reagieren.

Die SOA bietet jedoch nur ein Konzept zur Zusammenstellung von Services. Fragen zur konkreten Realisierung bleiben offen. Ungeklärt ist insbesondere, welche Informationen publiziert werden müssen. Einerseits benötigt der Service Broker genug Informationen über einen angebotenen Service P , um entscheiden zu können, ob der Service R eines Requesters zu P passt. Das heißt: Im Vorfeld muss sichergestellt werden, dass die Services während ihrer Interaktion weder in ein Deadlock geraten noch sich unerwartete Nachrichten senden. Andererseits soll die innere Struktur eines Services gekapselt werden. Oft verbergen sich dahinter gut gehütete Firmengeheimnisse. Der Service Provider ist also daran interessiert, möglichst wenig von seinem Service preiszugeben.

In [Lyn96] und [Mar04] wird vorgeschlagen, zu einem Service einen *Public View* zu veröffentlichen. Ein Public View ist eine abstrakte Version P' eines Services P , so dass jeder Service, der mit P' interagieren kann, sich an P binden darf. Jedoch wird weder in [Lyn96] noch in [Mar04] ein Algorithmus angegeben, wie ein Public View automatisch berechnet werden kann [MS05b].

In [MS05b] wird dagegen der Ansatz verfolgt, dass der Service Provider nicht Eigenschaften seines Services P an den Service Broker übergibt, sondern eine Bedienungsanleitung bereitstellt, die das Kommunikationsverhalten aller möglichen Partner zu P beschreibt. Erfüllt ein Service R die Anforderungen der Bedienungsanleitung, so wird die Interaktion zwischen R und P ohne Probleme verlaufen. In [MS05a] wird gezeigt, wie für beliebige azyklische Services eine Bedienungsanleitung generiert werden kann.

Bedienungsanleitungen und ihre Eigenschaften wurden bisher „nur“ theoretisch untersucht. Soll das Konzept in der Praxis angewendet werden, muss es eine effiziente Implementierung der Datenstruktur geben. In [MS05a] wird die Bedienungsanleitung als ein Automat definiert, dessen Zustände mit einer Annotation versehen sind. Der Automat ist jedoch zu groß, um all seine Zustände mit den Annotationen explizit speichern zu können. Deshalb ist es notwendig nach einer kompakteren Repräsentation zu suchen.

In der Modellprüfung ist es ebenfalls erforderlich, eine große Anzahl von Zuständen speichern zu können. Ziel ist es hier, für alle möglichen Zustände einer Software die Gültigkeit einer vorgegebene Eigenschaft, wie zum Beispiel Deadlock-Freiheit, nachzuweisen. In [McM92] schlägt McMillan vor, die Zustände in eine boolesche Funktion zu codieren und zur Repräsentation der Funktionen *binäre Entscheidungsdiagramme* (engl. Binary Decision Diagram, kurz: *BDD*) [Bra86] zu nutzen. Dank dieser Idee können die Zustände kompakt dargestellt werden und es es möglich größere Software als zuvor zu prüfen. So setzt die Industrie vor allem für Software in sicherheitskritischen Bereichen zunehmend die Modellprüfung zum Auffinden von Fehlern ein[Ref00].

Auch auf anderen Gebieten wie der Datenflussanalyse und der Hardwareverifikation werden BDDs erfolgreich eingesetzt. Daher wollen wir in dieser Arbeit untersuchen, ob sich BDDs auch zur kompakten Repräsentation von Bedienungsanleitungen eignen. Wir werden verschiedene Ansätze entwickeln und diese miteinander vergleichen. Anhand einer Testreihe wollen wir die Größe von BDDs für praktisch relevante Bedienungsanleitungen abschätzen.

Die Arbeit gliedert sich wie folgt: In Kapitel 2 werden die Grundlagen zu BDDs und Bedienungsanleitungen gelegt. Anschließend befassen wir uns im dritten Kapitel damit, wie die Informationen von Bedienungsanleitungen eindeutig durch BDDs dargestellt werden können. Es werden hier drei Ansätze vorgestellt. Anhand von Beispielen wollen wir diskutieren, wie gut sich die Ansätze für eine Anwendung in der Praxis eignen. In Kapitel 4 werden die Ergebnisse dann nochmal zusammengefasst.

2 Hintergrund

In diesem Kapitel geben wir zunächst eine kurze Einführung in Binäre Entscheidungsdiagramme (kurz: BDD). Wir zeigen, wie zu einer booleschen Funktion eine BDD-Darstellung gefunden werden kann und erläutern einige wichtige Eigenschaften der Diagramme.

Im zweiten Abschnitt definieren wir ein formales Modell für Services, das die interne Kontrollstruktur und sein Kommunikationsverhalten reflektiert. Auf dieser Grundlage wird eine formale Methode angegeben, die es erlaubt Bedienungsanleitungen automatisch zu erstellen. Ziel ist es, die möglichen Kommunikationspartner eines Services zu charakterisieren.

2.1 Binäre Entscheidungsdiagramme (BDD)

Ein *binäres Entscheidungsdiagramm* ist eine kompakte Datenstruktur zur Darstellung boolescher Funktionen. Boolesche Funktionen sind über einer endlichen Menge $X = \{x_1, \dots, x_n\}$ *boolescher Variablen* definiert. Eine *Belegung* ist ein n -Tupel von *Wahrheitswerten* (a_1, \dots, a_n) , wobei a_i ($1 \leq i \leq n$) der Wahrheitswert der Variablen x_i ist. Für den Wahrheitswert „wahr“ schreiben wir eine 1 und für „falsch“ eine 0. Eine boolesche Funktion ist eine Abbildung, die jeder Belegung einen Wahrheitswert zuordnet. Falls $f(a_1, \dots, a_n) = 1$ gilt, so nennen wir (a_1, \dots, a_n) eine *erfüllende Belegung*. Mit den zweistelligen logischen Operatoren „ \vee “ (*Disjunktion*) und „ \wedge “ (*Konjunktion*) können wir Variablen miteinander verknüpfen. Die *Negation* kennzeichnen wir mit einem „ \neg “ über der Variablen. Zur übersichtlicheren Darstellung können wir anstatt $x_1 \wedge x_2$ kurz $x_1 x_2$ schreiben.

In Beispiel 1 ist eine boolesche Funktion angegeben. Ihre erfüllenden Belegungen sind in Tabelle 1 aufgelistet.

Beispiel 1 (Boolesche Funktion)

$$f(x_1, x_2, x_3, x_4) = \overline{x_1} \overline{x_2} \overline{x_3} \overline{x_4} \vee \overline{x_1} \overline{x_2} x_3 x_4 \vee x_1 x_2 \overline{x_3} \overline{x_4} \vee x_1 x_2 x_3 x_4$$

x_1	x_2	x_3	x_4	f
0	0	0	0	1
0	0	1	1	1
1	1	0	0	1
1	1	1	1	1

Tabelle 1: Alle erfüllenden Belegungen der Funktion f aus Beispiel 1.

Jede Boolesche Funktion kann auch graphisch in einem *binären Entscheidungsbaum* dargestellt werden. Die Blätter im Baum sind entweder mit 0 oder mit 1 beschriftet. Alle anderen Knoten korrespondieren mit einer Variablen $x_i \in X$ und haben genau zwei ausgehende Kanten: eine 0-Kante und eine 1-Kante. Jede Belegung $b = (a_1, \dots, a_n)$ definiert einen eindeutig bestimmten Pfad von der Wurzel bis zu einem Blatt. Er folgt in jedem mit x_i beschrifteten Knoten der 1-Kante, falls $a_i = 1$. Anderenfalls wird er über die 0-Kante fortgesetzt. Das Blatt, in dem der Pfad endet, repräsentiert den zugehörigen Funktionswert zu b .

Die Reihenfolge, in der die Variablen auf ihren Wahrheitswert geprüft werden, ist in jedem Pfad des Baumes gleich und wird durch die *Variablenordnung* π festgelegt. Infolgedessen sind Knoten gleicher Tiefe mit derselben Variablen beschriftet.

Abbildung 1 zeigt den binären Entscheidungsbaum für die Funktion f aus Beispiel 1. Die 0-Kanten sind durch eine gestrichelten Linie und die 1-Kanten durch eine durchgezogene Linie dargestellt. Die Variablenordnung ist $\pi = x_1 < x_2 < x_3 < x_4$.

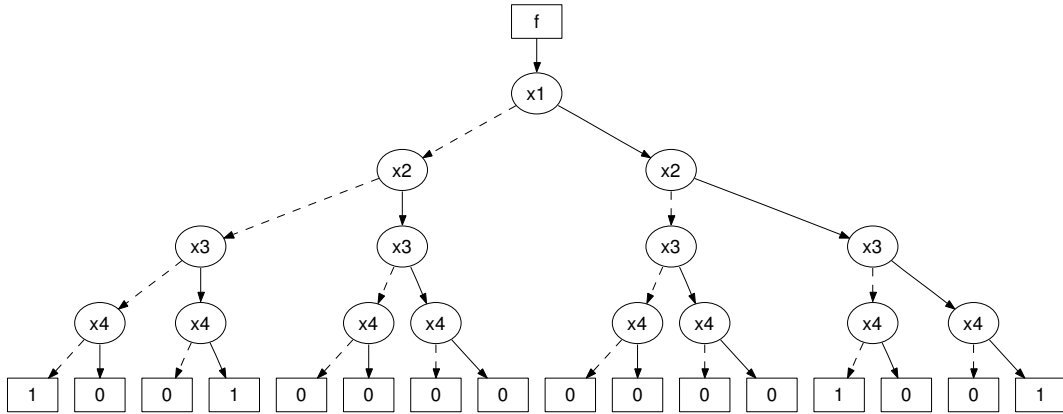


Abbildung 1: Binärer Entscheidungsbaum der Funktion f aus Beispiel 1 mit der Variablenordnung $\pi = x_1 < x_2 < x_3 < x_4$.

Wenn wir den Entscheidungsbaum aus Abbildung 1 betrachten, so stellen wir fest, dass einige Unterbäume *isomorph* zueinander sind. In diesem Fall können wir die Duplikate löschen. Die Kanten, die auf die Wurzel der gelöschten Unterbäume zeigen, werden auf die Wurzel des verbleibenden Teilbaums umgelenkt. Nach dem Entfernen gibt es u.U. Knoten, bei denen beide ausgehenden Kanten auf den selben Nachfolgeknoten gerichtet sind. Die logische Entscheidung in diesen Knoten beeinflusst offensichtlich nicht den Funktionswert. Solche Abfragen sind *irrelevant* und die betroffenen Knoten können deshalb gestrichen werden. Alle ankommenden Kanten werden dann direkt auf den jeweiligen Nachfolgeknoten umgelenkt. Haben wir alle Redundanzen entfernt, so erhalten wir ein *binäres Entscheidungsdiagramm* (Definition 1), das kleiner ist als der Entscheidungsbaum, aber dieselben Informationen beinhaltet. Wie stark die Reduktion ist, hängt von der dargestellten booleschen Funktion ab.

Definition 1 (Binäres Entscheidungsdiagramm (engl. Binary Decision Diagram, kurz: BDD))

Sei π eine totale Ordnung auf den Variablen x_1, \dots, x_n . Ein binäres Entscheidungsdiagramm bezüglich π ist ein gerichteter, zusammenhängender, azyklischer Graph mit genau einem Startknoten (Wurzel), der den folgenden Eigenschaften genügt:

- Es gibt genau zwei Terminalknoten (Senken) ohne ausgehende Kanten, die mit 0 bzw. 1 beschriftet sind.
- Die übrigen Knoten sind Variablenknoten. Sie sind mit einer Variablen beschriftet und haben genau zwei ausgehende Kanten, die mit 0 bzw. 1 beschriftet sind (in Abbildungen durch eine gestrichelte bzw. durchgezogene Linie dargestellt).

- Der Graph ist reduziert und geordnet. Das heißt:
 - Es gibt weder irrelevante Abfragen noch isomorphe Untergraphen und
 - auf allen Pfaden durch den Graphen ist die Reihenfolge der Variablen konsistent mit der Variablenordnung π , d.h. für jede Kante, die von einem mit x_i beschrifteten Knoten zu einem mit x_j beschrifteten Knoten führt, gilt $x_i <_{\pi} x_j$.

Aufgrund der Eigenschaften *reduziert* und *geordnet* wird häufig auch die Abkürzung *ROBDD* für *Reduced Ordered Binary Decision Diagram* verwendet.

Bei gegebener Variablenordnung π ist die BDD-Darstellung einer booleschen Funktion *kanonisch*. Das heißt, der Graph ist bis auf Isomorphie eindeutig bestimmt. Verändern wir die Variablenordnung in einem BDD, so kann dieses dadurch größer oder kleiner werden. Von einer *optimalen Variablenordnung* sprechen wir, wenn es keine andere Variablenordnung gibt, bei der das BDD weniger Knoten hat. In Abbildung 2 sehen wir, wie die Anzahl der Knoten in Abhängigkeit von π schwanken kann.

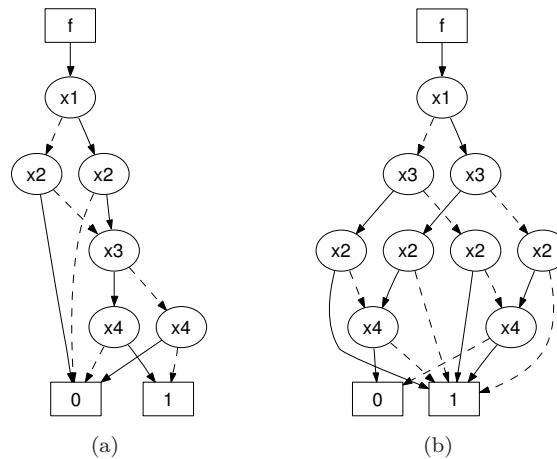


Abbildung 2: BDD-Darstellungen der Funktion f aus Beispiel 1. (a) mit einer optimalen Variablenordnung $\pi_1 = x_1 < x_2 < x_3 < x_4$ und (b) mit der Variablenordnung $\pi_2 = x_1 < x_3 < x_2 < x_4$.

In der Praxis müssen wir häufig mit mehreren booleschen Funktionen arbeiten. Anstatt nun für jede ein separates BDD zu erzeugen, gibt es die Möglichkeit, alle zusammen in einem *Shared BDD* (Definition 2) darzustellen. Dabei handelt es sich um einen Graphen mit mehreren Wurzeln. Jede Wurzel repräsentiert eine boolesche Funktion. Der Vorteil dieser Datenstruktur ist, dass Untergraphen, die in mehreren BDDs vorkommen, nur einmal gespeichert werden müssen.

Definition 2 (Shared BDD))

Sei π eine totale Ordnung auf den Variablen x_1, \dots, x_n . Ein *Shared BDD* bezüglich π ist ein gerichteter, zusammenhängender, azyklischer Graph mit k Startknoten (Wurzeln), der den gleichen Eigenschaften genügt wie ein BDD (Definition 1).

Ein Shared BDD ist wie ein BDD reduziert und geordnet. Dadurch erhalten wir im Shared BDD eine *streng* kanonische Darstellung. Das bedeutet: Äquivalente Funktionen haben den selben Wurzelknoten.

In Abbildung 3 ist ein Shared BDD mit der Funktion f aus Beispiel 1 und der Funktion $g(x_2, x_3, x_4) = x_2 \vee \overline{x_3} \overline{x_4} \vee x_3 x_4$ dargestellt. Beide Funktionen nutzen gemeinsam den am Wurzelknoten x_3 beginnenden Untergraphen.

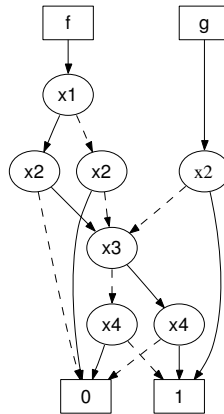


Abbildung 3: Shared BDD.

BDDs und Shared BDDs können durch Verwendung von attributierten Kanten in ihrer Größe weiter reduziert werden. Bei dieser Erweiterung wird jede Kante mit einem Attribut versehen, welches angibt, ob die Kante wie in den bisherigen Betrachtungen interpretiert wird (reguläre Kante) oder als *komplementierte* Kante: Angenommen der Knoten v repräsentiert eine Funktion h und eine Kante e ist auf v gerichtet. Falls e eine komplementierte Kante ist, so wird durch e die komplementäre Funktion \overline{h} repräsentiert. Ist e eine reguläre Kante, so wird durch e die Funktion h selbst repräsentiert. Betrachten wir zu einer gegebenen Belegung den Berechnungspfad p , ist der zugehörige Funktionswert das Komplement der Senke, falls p über eine ungerade Anzahl von komplementierten Kanten führt.

Um die Kanonizität zu erhalten, dürfen komplementierte Kanten nicht an jeder beliebigen Position eingesetzt werden und es muss die Vorgehensweise zur Beseitigung isomorpher Untergraphen und irrelevanter Abfragen geändert werden [MT98].

Abbildung 4 zeigt die Funktionen f , \overline{f} und g in einem Shared BDD mit komplementierten Kanten (gepunktete Linien). Wir sehen, dass zur Darstellung der Funktion \overline{f} keine zusätzlichen Knoten nötig sind. Es muss lediglich eine komplementierte Kante auf den Wurzelknoten von f gerichtet werden.

Wir haben in Abbildung 2 bereits gesehen, dass die Anzahl der Knoten im BDD ganz wesentlich von der Variablenordnung abhängt. Sie kann zwischen linear und exponentiell in Anzahl der booleschen Variablen schwanken. Wünschenswert wäre es nun, wenn für jede boolesche Funktion mindestens eine Variablenordnung existieren würde, die zu einem kompakten Graphen mit einer geringen Anzahl von Knoten führt. Tatsache ist, dass die Repräsentation *fast aller* Funktionen bei jeder Variablenordnung exponentiellen Platz benötigt [MT98]! Jedoch können für viele in der Praxis angewendeten booleschen Funktionen häufig kompakte Darstellungen gefunden werden. Die

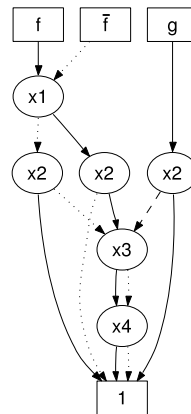


Abbildung 4: Shared BDD mit komplementierten Kanten.

größte Schwierigkeit bei der Verwendung von BDDs ist die *Bestimmung* der optimalen Variablenordnung. Das Problem ist NP-hart. Es gibt aber zahlreiche heuristische Verfahren, mit denen oft eine gute Variablenordnung ermittelt werden kann.

In dieser Arbeit wollen wir BDDs nutzen, um Bedienungsanleitungen darzustellen. Aufgabe ist es also, eine boolesche Funktion zu finden, die die Informationen einer Bedienungsanleitung vollständig repräsentiert. Dabei müssen wir drauf achten, dass es für die Funktion mindestens eine Variablenordnung mit kompakter BDD-Darstellung gibt. Praktisch relevant wird dieser Ansatz nur dann sein, wenn es uns außerdem gelingt, BDDs zu erzeugen, die kleiner sind, als die zugehörige Bedienungsanleitung.

2.2 Bedienungsanleitungen

Softwaresysteme werden seit einigen Jahren immer häufiger aus unabhängigen, verteilt laufenden Services zusammengesetzt. Jeder Service besitzt neben seiner *internen Struktur* einen *Namen* (z.B. einen Uniform Resource Identifier, kurz: URI), über den er eindeutig identifiziert werden kann und ein *Interface*. Die interne Struktur eines Services wird gekapselt – öffentlich ist nur das Interface. Services tauschen gegenseitig Daten über Nachrichtenkanäle aus, können andere Services aufrufen bzw. von einem Service aufgerufen werden oder gemeinsam einen Service koordinieren. Die gesamte Kommunikation findet über standardisierte Protokolle statt. [Col04a]

Die *Serviceorientierte Architektur* (SOA) stellt ein Konzept bereit, wie *Services* zusammengestellt werden können (Abbildung 5). Sie definiert drei Rollen (*Service Provider*, *Service Requester*, *Service Broker*) und zwischen den Rollen drei Interaktionen (*publizieren*, *finden*, *binden*): Der Service Provider stellt einen Service zur Verfügung. Er übernimmt die Implementierung und veröffentlicht eine Beschreibung (z.B. in Web Service Description Language, kurz: WSDL) zu seinem Service in einem zentralen Verzeichnis. Der Service Requester nutzt angebotene Services und integriert diese in eigene Services oder Anwendungen. Der Service Broker verwaltet das Verzeichnis mit den Beschreibungen und ermöglicht dem Service Requester das Finden von Services. Ist ein Service, der die benötigte Funktionalität zur Verfügung stellt, im Verzeichnis registriert, so können sich Service Requester und Service Broker aneinander binden und miteinander interagieren. [Col04b]

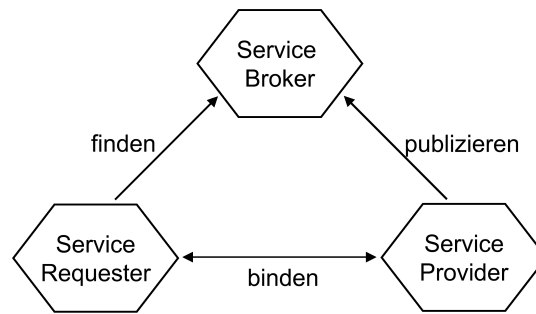


Abbildung 5: Serviceorientierte Architektur (SOA) [Col04b].

Die SOA beschreibt damit einen Ablauf, wie Services miteinander kombiniert werden können. Es existiert derzeit aber keine etablierte Methodik zur automatischen Zusammensetzung der Services. Die veröffentlichte Beschreibung enthält im wesentlichen Informationen über die verfügbaren Funktionen, deren Parameter und Rückgabewerte. Anhand ihr kann entschieden werden, ob zwei Services syntaktisch zusammen passen, nicht aber, ob auch ihr Kommunikationsverhalten zusammen passt.

Um diese Lücke zu schließen, wird in [MS05a] der Vorschlag gemacht, dass der Service Provider eine Bedienungsanleitung zu seinem Service veröffentlicht. Diese enthält eine Beschreibung, wie sich Kommunikationspartner verhalten müssen, um fehlerfrei mit dem Service interagieren zu können.

Bevor wir zeigen, wie Bedienungsanleitungen erstellt werden können, führen wir Serviceautomaten (Definition 3) als ein formales Modell für Services ein und definieren die Interaktion zweier Services auf Basis der Serviceautomaten.

Definition 3 (Serviceautomat)

Sei MC die Menge der Nachrichtenkanäle.

Ein Serviceautomat ist ein nichtdeterministischer Automat $A = [I, Q, T, q_0, \Omega]$ mit:

- einem Interface $I = [I_{in}, I_{out}]$, so dass $I_{in} \cup I_{out} \subseteq MC$ und $I_{in} \cap I_{out} = \emptyset$,
- einer Zustandsmenge Q ,
- einer Transitionsmenge $T \subseteq Q \times L \times Q$, mit $L = \{?x \mid x \in I_{in}\} \cup \{!x \mid x \in I_{out}\} \cup \{\tau\}$,
- einem Startzustand $q_0 \in Q$,
- einer Menge von Endzuständen $\Omega \subseteq Q$

L wird die Menge der Labels von A genannt.

Das Kommunikationsverhalten von Services wird mit Labels an den Transitionen des Automaten modelliert. Für eine empfangene Nachricht von Kanal x schreiben wir $?x$ und für das Senden schreiben wir $!x$.

In den folgenden Betrachtungen wollen wir uns nur auf azyklische Services beschränken. Das heißt, sie können nicht in einen Zustand zurückkehren, in dem sie schon ein einmal waren.

Den Serviceautomaten eines Service Providers wollen wir mit P und den eines Service Requesters mit R bezeichnen. Als ein Beispiel für einen Service Provider betrachten wir einen Getränkeau-

tomaten G , der durch den in Abbildung 6b dargestellten Serviceautomaten P_G beschrieben wird. Der Automat erwartet, dass ein Kunde Geld ($?€$) einwirft und die Taste für den Tee ($?T$) oder den Kaffee ($?C$) drückt. Anschließend gibt er einen Becher ($!B$) mit dem bestellten Getränk zurück. Außerdem betrachten wir einen Kunden R_T (Service Requester), der Tee möchte (Abbildung 6a).

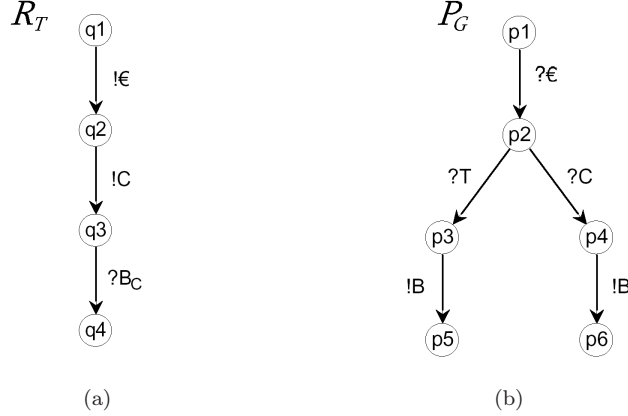


Abbildung 6: (a) Serviceautomat R_T eines Kunden und (b) Serviceautomat P_G für den Getränkeautomaten [MS05a].

Nicht jeder beliebige Service ist geeignet, mit einem gegebenen Service zu kommunizieren: Während der Interaktion dürfen die Services nicht in einem Deadlock landen und sich keine unerwarteten Nachrichten senden. Das Zusammenspiel zweier Services kann durch die *Komposition* der Serviceautomaten beschrieben werden. An dem daraus resultierenden *Transitionssystem* können wir erkennen, ob zwei Services zueinander passen.

Definition 4 (Interaktion von Serviceautomaten)

Seien P und R zwei Serviceautomaten und $\text{bags}(MC)$ die Menge aller Multimengen über MC . O.B.d.A. sei $Q_R \cap Q_P = \emptyset$.

Das Transitionssystem $(R \oplus P) = [Q, T, q_0, \Omega]$ beschreibt die Interaktion zwischen R und P bestehend aus

- einer Zustandsmenge $Q \subseteq Q_R \times Q_P \times \text{bags}(MC)$,
- einer Menge von beschrifteten Transitionen $T \subseteq Q \times (L_P \cup L_R) \times Q$,
- einem Startzustand q_0 und
- einer Menge von Endzuständen $\Omega \subseteq Q$.

Q und T werden wie folgt induktiv definiert:

Basis: $q_0 = [q_{0_R}, q_{0_P}, \{\}]$ ist ein Zustand des Transitionssystems.

Schritt: Falls $q = [q_R, q_P, M]$ ein Zustand ist und

- es eine Transition $t = [q_R, !a, q'_R] \in T_R$ gibt, so $q' = [q'_R, q_P, M + a] \in Q$ und $[q, !a, q'] \in T$.
- es eine Transition $t = [q_P, !a, q'_P] \in T_P$ gibt, so $q' = [q_R, q'_P, M + a] \in Q$ und $[q, !a, q'] \in T$.
- $a \in M$, und es eine Transition $t = [q_R, ?a, q'_R] \in T_R$ gibt, so $q' = [q'_R, q_P, M - a] \in Q$ und $[q, ?a, q'] \in T$.

- $a \in M$, und es eine Transition $t = [q_P, ?a, q'_P] \in T_P$ gibt, so $q' = [q_R, q'_P, M - a] \in Q$ und $[q, ?a, q'] \in T$.
- es eine Transition $t = [q_R, \tau, q'_R] \in T_R$ gibt, so $q' = [q'_R, q_P, M] \in Q$ und $[q, \tau, q'] \in T$.
- es eine Transition $t = [q_P, \tau, q'_P] \in T_P$ gibt, so $q' = [q_R, q'_P, M] \in Q$ und $[q, \tau, q'] \in T$.

Ein Zustand $[q_R, q_P, M] \in \Omega$ ohne Nachfolgezustände ist ein Endzustand genau dann, wenn $q_P \in \Omega_P$, $q_R \in \Omega_R$ und $M = \{\}$.

Ein Zustand $[q_R, q_P, M]$ im Transitionssystem $R \oplus P$ repräsentiert einen Zustand in R, einen Zustand in P und eine Multimenge der Nachrichten, die sich in den Kanälen befinden. Transitionen, die mit $!a$ beschriftet sind, erzeugen eine Nachricht auf dem Kanal a und Transitionen, die mit $?a$ beschriftet sind, konsumieren eine Nachricht vom Kanal a . Bei τ -Transitionen werden Nachrichten weder konsumiert noch erzeugt. Jede Nachricht, die erzeugt wird, muss auch wieder konsumiert werden, so dass im Endzustand sich keine Nachrichten mehr in den Kanälen befinden.

Definition 5 (Deadlock, weak termination)

Sei $R \oplus P = [Q, T, q_0, \Omega]$ ein Transitionssystem. Ein Zustand q , der kein Endzustand ist ($q \in Q \setminus \Omega$) und keine Nachfolger in $R \oplus P$ hat, wird Deadlock genannt.

Das Transitionssystem $R \oplus P$ hat die Eigenschaft weak termination genau dann, wenn $R \oplus P$ keine Deadlocks hat.

Abbildung 7 zeigt das Transitionssystem $R_T \oplus P_G$ der beiden Serviceautomaten aus Abbildung 6. Wie wir sehen, ist der einzige Zustand ohne Nachfolger der Endzustand. Damit hat $R_T \oplus P_G$ nach Definition 5 die Eigenschaft weak termination.

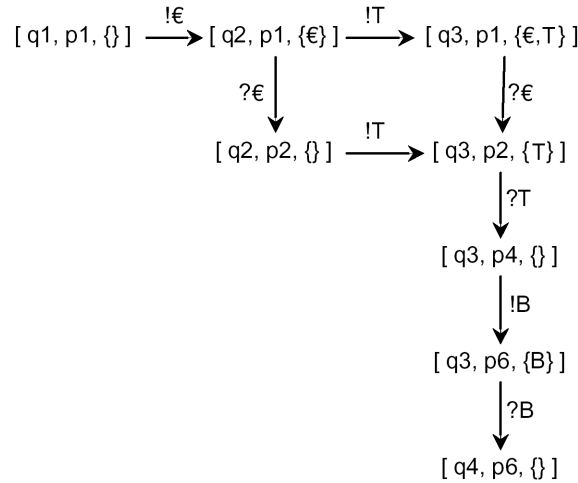


Abbildung 7: Transitionssystem $R_T \oplus P_G$ [MS05a].

Offensichtlich verläuft die Interaktion zwischen einem Serviceautomaten R und einem gegebenen Serviceautomaten P fehlerfrei, wenn ihr Transitionssystem die Eigenschaft weak termination besitzt. In diesem Fall wollen wir R eine *Strategie* von P nennen (Definition 6).

Definition 6 (Strategie)

Sei P ein Serviceautomat. Ein Serviceautomat R wird als Strategie für P bezeichnet genau dann, wenn $R \oplus P$ die Eigenschaft *weak termination* besitzt.

In unserem Beispiel ist R_T eine Strategie von P_G .

Wenn der Service Provider zu seinem Service alle Strategien veröffentlicht, so kann der Service Broker leicht entscheiden, ob im Falle einer Interaktion mit einem Service Requester beide Services immer zu einem vernünftigen Ende kommen: Der Serviceautomat des Service Requesters muss mit einer der veröffentlichten Strategien des Service Providers übereinstimmen. In [MS05a] wird nicht nur gezeigt, wie zu einem gegebenen Service alle Strategien berechnet werden können, sondern es wird darüber hinaus eine kompakte Beschreibung für die Menge aller Strategien definiert. Bei dieser Beschreibung handelt es sich um die Bedienungsanleitung OG . Wir wollen nun erläutern wie ein Service Provider zu seinem Service die zugehörige Bedienungsanleitung automatisch generieren kann.

Sei P der Serviceautomat des Service Providers und angenommen P hat die Tiefe n . Zu P erstellen wir einen vollständigen Baum R^n der Tiefe n , in dem jeder innere Knoten für jedes $x \in I_{out_P}$ bzw. für jedes $y \in I_{in_P}$ eine mit $?x$ bzw. $!y$ beschriftete ausgehende Kante hat. R^n ist auch ein Serviceautomat, der mit P komponiert werden kann. Der Baum R_G^n zu unserem Getränkeautomaten P_G aus Abbildung 6 hat in jedem inneren Knoten genau vier ausgehende Kanten, die jeweils mit $!€$, $!T$, $!C$ und $?B$ beschriftet sind.

Im nächsten Schritt notieren wir für jeden Zustand in R^n , in welchen Zuständen sich P zu diesem Zeitpunkt befinden kann und welche Nachrichten jeweils in den Kanälen vorhanden sind (Definition 7)

Definition 7 (Wissensfunktion)

Seien R und P zwei Serviceautomaten und $bags(MC)$ die Menge aller Multimengen über MC .

Die Wissensfunktion ist eine Abbildung $k_{(R,P)} : Q_R \rightarrow \wp(Q_P \times bags(MC))$, so dass $k_{(R,P)}(q_R) = \{[q_P, M] \mid [q_R, q_P, M] \in Q_{R \oplus P}\}$.

Mit Hilfe der Wissensfunktion können wir zu jedem Zustand in R^n die *Annotation* Σ nach Definition 8 bestimmen. Sie gibt für jeden Zustand $q \in R^n$ an, welche Fortführungen für einen im Zustand q befindenden Requester möglich sind, so dass bei der Interaktion Deadlocks ausgeschlossen sind. Da R^n ein vollständiger Baum bzgl. des Interfaces ist, muss jeder Zustand eines möglichen Requesters in R^n vorhanden sein.

Definition 8 (Annotation)

Seien R und P zwei Serviceautomaten. Die Annotation $\Sigma(q_R)$ ist eine boolesche Formel.

- Falls $k_{(R,P)}(q_R) = \emptyset$, dann ist $\Sigma(q_R) = \text{wahr}$.
- Sonst ist $\Sigma(q_R)$ eine Konjunktion von Teilformeln $\sigma_{(q_R, q_P, M)}$ für alle $[q_P, M] \in k_{(R,P)}(q_R)$.
Die Teilformel $\sigma_{(q_R, q_P, M)}$ ist:
 - wahr, falls $q_R \in \Omega_R$, $q_P \in \Omega$ und $M = \{\}$;
 - wahr, falls es eine Transition von P gibt, die $[q_R, q_P, M]$ im Transitionssystem $R \oplus P$ verlässt;

- falsch, falls $[q_R, q_P, M]$ im Transitionssystem $R \oplus P$ ein Deadlock ist;
- die Disjunktion aller stattfindenden Transitionen in R , die $[q_R, q_P, M]$ im Transitionssystem $R \oplus P$ verlassen.

Für Zustände, in denen $\Sigma(q_R) = \text{falsch}$ ist, gibt es offensichtlich keine vernünftige Fortsetzung der Kommunikation. Solche Zustände können deshalb (zusammen mit ihrem Untergraphen) aus R^n gestrichen werden. Vorgängerzustände, deren Annotation durch dieses Streichen den Wert falsch erhalten, können ebenfalls gelöscht werden. Dieser Prozess wird solange wiederholt, bis keine mit falsch annotierten Zustände mehr vorhanden sind. Als Resultat erhalten wir einen Serviceautomaten S , der zusammen mit den verbleibenden Annotationen die gesuchte Bedienungsanleitung OG_P repräsentiert.

Definition 9 (Bedienungsanleitung)

Sei P ein beliebiger Serviceautomat. Sei S seine liberalste Strategie und Σ die zu S gehörige Annotation. Dann ist $OG_P = (S, \Sigma)$ die Bedienungsanleitung für P .

Falls die Zustandsmenge Q_S von S nach dem Streichen leer ist, gibt es keinen passenden Serviceautomaten für P . In [MS05a] wird gezeigt, dass S eine Strategie nach Definition 6 ist und jede Strategie S' ein Teilautomat (Definition 10) von S ist. S wird auch als die *liberalste Strategie* bezeichnet, da sie das meiste Verhalten aufweist. Umgekehrt ist jeder Teilautomat S'' von S , der in jedem Zustand $q_{S''} \in Q_{S''}$ die Annotationen der Bedienungsanleitung erfüllt, eine Strategie von P . Das heißt, die Labels der ausgehenden Kanten sind in jedem Zustand $q_{S''}$ eine erfüllende Belegung der Annotation $\Sigma(q_{S''})$.

Definition 10 (Teilautomat)

Seien A und A' Automaten. A' ist ein Teilautomat von A ($A \sqsupseteq A'$) genau dann, wenn $Q_{A'} \subseteq Q_A$, $T_{A'} \subseteq T_A$, $q_{0_{A'}} = q_{0_A}$ und $\Omega_{A'} \subseteq Q_{A'} \cap Q_A$

Damit repräsentiert OG_P genau die Menge aller Strategien. In der Abbildung 8a ist die Bedienungsanleitung des Getränkeautomaten zusehen. Nicht dargestellt sind die Zustände q_R^* , deren Wissensfunktion $k_{(R^n, P)}(q_R^*) = \emptyset$ ist. Der Graph in Abbildung 8b repräsentiert ebenfalls die Bedienungsanleitung des Getränkeautomaten: Hier wurden Knoten, deren Wissensfunktion gleich ist, miteinander verschmolzen. Wir sehen, dass die Strategie R_T aus Abbildung 6 in beiden Fällen ein Teilautomat ist, der in jedem Zustand die Annotationen erfüllt.

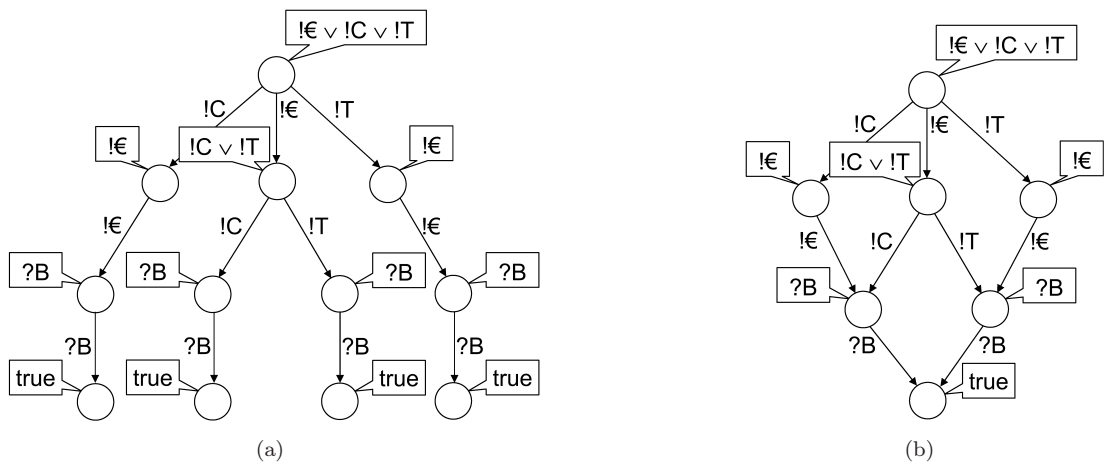


Abbildung 8: Bedienungsanleitung OG_G für den Getränkeautomaten G . Rechts in Baumstruktur und links als Graph.

3 BDD-Repräsentationen für Bedienungsanleitungen

In diesem Kapitel werden Möglichkeiten diskutiert, wie Bedienungsanleitungen durch BDDs dargestellt werden können. In allen Ansätzen wird versucht, die Informationen einer Bedienungsanleitung in eine oder mehrere boolesche Funktionen zu codieren und diese als BDD zu repräsentieren. Wie wir bereits aus Kapitel 2 wissen, ist nicht jede Funktion gleichermaßen gut für eine BDD-Darstellung geeignet. Die Schwierigkeit besteht also darin, eine "gutartige" boolesche Funktion zu finden, die eindeutig alle Informationen der Bedienungsanleitung wiedergibt. Das Resultat soll dann ein BDD sein, der kleiner ist, als die Bedienungsanleitung in Form eines annotierten Automaten.

Im Folgenden stellen wir drei Ansätze vor, wie Bedienungsanleitungen verlustfrei als BDD dargestellt werden können. Jeder der Ansätze wird am Beispiel des in Kapitel 2 eingeführten Getränkeautomaten (Abbildung 6 auf Seite 11) erläutert. Wir werden jeweils zeigen, wie zu seiner Bedienungsanleitung (Abbildung 8 auf der vorherigen Seite) die boolesche Funktion erzeugt werden kann und die zugehörige BDD-Darstellung abbilden. Um die Ansätze untereinander vergleichen zu können und um eine Aussage über die Eignung in der Praxis zu treffen, wird eine Testreihe bestehend aus vier unterschiedlich großen Bedienungsanleitungen herangezogen. Die Funktion der zugrundeliegenden Serviceautomaten (siehe Abbildung 9) besteht lediglich darin, eine bestimmte Anzahl von Nachrichten nacheinander zu senden.

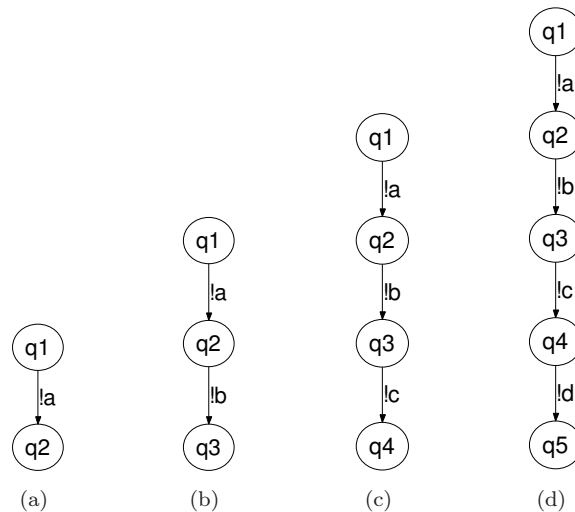


Abbildung 9: Serviceautomaten der Testbeispiele

3.1 Erster Ansatz: Explizite Darstellung der Strategien im Shared BDD

3.1.1 Vorgehensweise

Eine Bedienungsanleitung beschreibt eine Menge von Strategien. Aus Kapitel 2 wissen wir, dass es für azyklische Services immer eine liberalste Strategie gibt, und alle anderen Strategien Teilautomaten von ihr sind. Es bestehen also große Ähnlichkeiten unter den Strategien. Wir wollen nun

jede Strategie in einem BDD repräsentieren und anschließend die BDDs aller Strategien als ein Shared BDD darstellen.

Falls die BDDs von ähnlichen Strategien auch gleiche Untergraphen haben, so haben wir mit dem Shared BDD eine sehr platzsparende Darstellungsform gefunden. Wir müssen also „nur“ noch einen Weg finden, eine gegebene Strategie durch eine boolesche Formel zu beschreiben. Wie wir die zugehörige BDD-Darstellung erhalten können, wurde ausführlich in Kapitel 2 erklärt.

Jede Strategie ist ein Baum, dessen Kanten mit Channelnamen beschriftet sind (Abbildung 10). Um zu der booleschen Formel zu gelangen, beschriften wir jeden Knoten mit der Sequenz von Channels, die zu ihm führt. Die Wurzel wird mit ε , dem leeren Wort, beschriftet. Diese Beschriftung ist eindeutig, weil:

1. Jede Strategie liegt als Baum vor und es gibt deshalb zu jedem Knoten *genau einen* Pfad, der im Wurzelknoten beginnt und in *v* endet.
2. Die Channels der ausgehenden Kanten in einem Knotens sind paarweise verschieden.

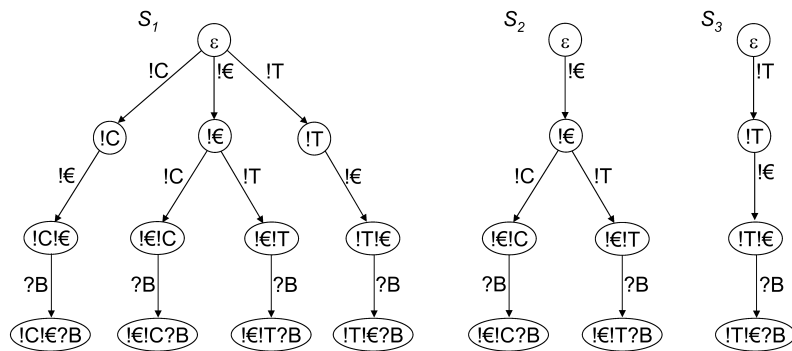


Abbildung 10: Einige Strategien des Getränkeautomaten mit der Knotenbeschriftung. Links ist die liberalste Strategie dargestellt.

In den Namen der Knoten werden so alle Interaktionen, die bis dahin stattgefunden haben, protokolliert. Folglich enthält jedes Blatt die gesamte Information des zu ihm führenden Pfades und wir müssen nur noch die Namen der Blätter speichern. Wir brauchen jetzt also anstatt der Baumstruktur lediglich die Menge der Channelsequenzen, mit denen die Blätter beschriftet sind, zu codieren.

Abbildung 10 zeigt drei der 14 Strategien des Getränkeautomaten mit den entsprechenden Knotennamen. Anhand des Blattes $!T!€?B$ von Strategie S_3 wissen wir, dass der Kunde als erstes die Taste für den Tee drücken, dann einen Euro in unseren Automaten stecken und am Ende auf das Getränk warten muss.

Um eine boolesche Funktion zu erhalten, gehen wir wie folgt vor: Im ersten Schritt werden die Namen der Channels in eine Ordnung gebracht (zum Beispiel lexikographisch aufsteigend) und durchnummeriert. Damit haben wir jedem Channel eine eindeutige Nummer zugeordnet. Diese können wir binär darstellen (Abbildung 2).

Channel	!C	!€	!T	?B
Binärdarstellung	00	01	10	11

Tabelle 2: Codierung der Channels vom Getränkeautomaten.

Jetzt können wir im zweiten Schritt jedes Blatt durch eine boolesche Funktion darstellen. Entsprechend der Binärcodierung der Channel, erhalten wir aus der Channelsequenz eine Zeichenfolge der Länge n bestehend aus Nullen und Einsen. Diese interpretieren wir als eine Belegung. Die boolesche Funktion ist nun ein Konjunktion bestehend aus n negierten und nichtnegierten Variablen, entsprechend der Belegung.

Beispielsweise wird im Getränkeautomaten nach Tabelle 2 der Channel !€ durch „01“ repräsentiert. Das Blatt „!€!T?B“ hat demnach die binäre Codierung „01 10 11“, die durch die Funktion $f_{!€!T?B} = \overline{x_1} x_2 x_3 \overline{x_4} x_5 x_6$ repräsentiert wird. Die Funktion $f_{!€!T?B}$ hat bei der Belegung $(0, 1, 1, 0, 1, 1)$ den Wert 1 und bei allen anderen Belegungen den Wert 0.

Im dritten und letzten Schritt erstellen wir mit Hilfe der Funktionen für die Blätter für jede Strategie S_i eine Funktion f_{S_i} . Da wir von jeder Strategie nur die Blätter speichern, muss f_{S_i} lediglich die Menge aller Blätter der Strategie S_i repräsentieren. Das erreichen wir, indem wir die Disjunktion über den entsprechenden Blattfunktionen bilden. Damit haben wir für jede Strategie eine BDD-Darstellung gefunden.

In Tabelle 3 sind die Belegungen der Blätter der Bedienungsanleitung des Getränkeautomaten aufgelistet. Um zum Beispiel die boolesche Funktion der Strategie S_2 aus Abbildung 10 zu erhalten, tragen wir in den Zeilen für die beiden Blätter „!€!T?B“ und „!€!C?B“ den Funktionswert 1 ein und in den anderen Zeilen (auch in denen, die hier nicht dargestellt sind) den Wert 0. Die gesuchte Funktion f_{S_2} ist dann eine Disjunktion der Funktionen $f_{!€!T?B}$ und $f_{!€!C?B}$: $f_{S_2} = \overline{x_1} x_2 x_3 \overline{x_4} x_5 x_6 \vee \overline{x_1} x_2 \overline{x_3} \overline{x_4} x_5 x_6$.

Abbildung 11 zeigt die BDD-Darstellungen aller Strategien des Getränkeautomaten im Shared BDD. Wir sehen, dass wie erwartet viele Untergraphen von mehreren Funktionen gemeinsam genutzt werden.

Blatt	x_1	x_2	x_3	x_4	x_5	x_6	f_{S_1}	f_{S_2}	f_{S_3}
!€!T?B	0	1	1	0	1	1	1	1	0
!€!C?B	0	1	0	0	1	1	1	1	0
!T!€?B	1	0	0	1	1	1	1	0	1
!C!€?B	0	0	0	1	1	1	1	0	0

Tabelle 3: Boolesche Funktionen f_{S_1} , f_{S_2} und f_{S_3} der Strategien S_1 , S_2 und S_3 aus Abbildung 10.

3.1.2 Auswertung

Als nächstes wollen wir untersuchen, ob das Shared BDD tatsächlich so kompakt ist, wie eingangs vermutet. Das Shared BDD des Getränkeautomaten (Abbildung 11) hat 33 Knoten. Das erscheint im ersten Moment viel gegenüber den 12 Knoten der Bedienungsanleitung. Wir müssen

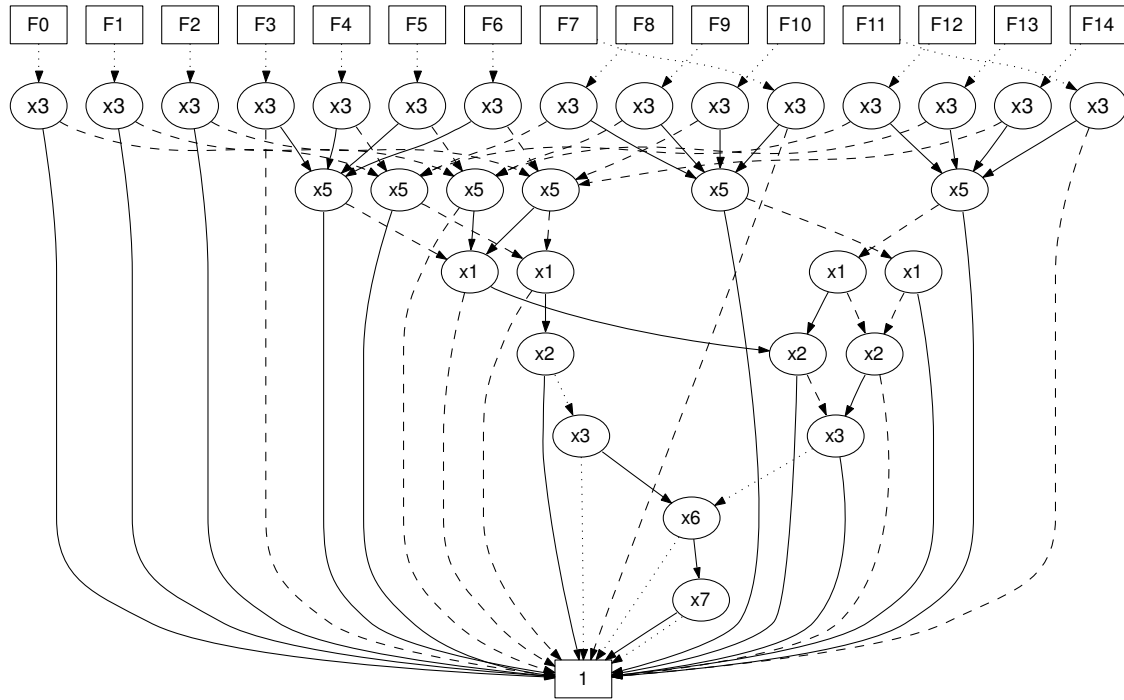


Abbildung 11: Shared BDD des Getränkeautomaten nach Ansatz 1 (optimale Variablenordnung).

aber bedenken, dass im Shared BDD bereits die Kantenbeschriftungen und die Annotationen der Bedienungsanleitung enthalten sind. Vor diesem Hintergrund sind 33 Knoten eine akzeptable Zahl.

Anhand der Testbeispiele wird dennoch schnell klar, dass dieser Ansatz nicht praxistauglich ist. Die Anzahl der inneren Knoten im Shared BDD ist zwar relativ gering, aber wir benötigen für jede Strategie einen separaten Wurzelknoten. Und die Anzahl der Strategien ist riesig. In Tabelle 4 sehen wir, dass die Anzahl der Strategien – und damit die Größe des Shared BDD – in Testbeispiel c stark zunimmt. Schon für das Testbeispiel d war es nicht mehr möglich, das Shared BDD (in einer angemessenen Zeit) zu berechnen.

In der Realität haben wir es mit deutlich größeren Bedienungsanleitungen als in unserer Testreihe zu tun, so dass der Ansatz, jede Strategie in einem BDD darzustellen, zu viel Zeit und zu viel Platz benötigt.

	Beispiel a	Beispiel b	Beispiel c	Beispiel d	Getränkeautomat
Anzahl der Knoten im Shared BDD (optimale Variablenordnung)	2	5	100	–	33
Anzahl der Strategien	1	3	63	–	15

Tabelle 4: Testreihe für Ansatz 1.

3.2 Zweiter Ansatz: Repräsentation von Struktur und Annotation in einem BDD

3.2.1 Vorgehensweise

Wir wollen nun anstatt jede Strategie explizit in einem Shared BDD darzustellen, nur noch einen BDD bauen, der die Funktion f_{OG} darstellt und der alle Informationen der Bedienungsanleitung repräsentiert. Bei der Repräsentation der Bedienungsanleitung halten wir an der Idee fest, die Knoten mit der Sequenz von Channels zu beschriften. Im Gegensatz zum vorherigen Ansatz werden wir jedoch auch die inneren Knoten binär codieren und jedem Knoten die Annotation hinzufügen.

Tabelle 5 zeigt die Wahrheitstabelle der Annotation von Knoten „! € “ in der Bedienungsanleitung des Getränkeautomaten (Abbildung 8a). Wir erkennen anhand der Annotation $!C\vee!T$ bzw. deren Wahrheitstabelle, dass es drei vernünftige Fortsetzungen gibt: die Taste T für den Tee drücken oder die Taste C für den Kaffee drücken oder eine der beiden Tasten wählen (letzte Zeile in der Tabelle 5).

! C	! T	! $C\vee!T$
0	0	0
0	1	1
1	0	1
1	1	1

Tabelle 5: Wahrheitstabelle der Annotation $!C\vee!T$.

Um die Belegungen einer Annotation eindeutig zuordnen zu können, wird die Binärcodierung des jeweiligen Knoten an die Belegung angehängt. Bei Knoten unterschiedlicher Tiefe sind die Namen und damit ihre Codierung von ungleicher Länge. Wir brauchen daher einen zusätzlichen „Reserve“-Channel. Diesen hängen wir an kurze Namen ein- oder mehrmals an, so dass alle Knotennamen gleich lang sind. Die Codierung des „Reserve“-Channels ist beliebig, darf jedoch von keinem anderen Channel genutzt werden.

Im Beispiel des Getränkeautomaten benötigen wir nun für die Channeldarstellung drei Bits, da wir im Vergleich zum vorherigen Ansatz nicht mehr vier, sondern fünf Channels codieren müssen. In Tabelle 6 sind für die Channels die neuen Codierungen angegeben. Zum Beispiel hat der Channel $!\text{€}$ die Binärdarstellung „001“. Die Codierung des Knotens „! € “ ist „001 111 111“. (Wir haben zweimal der „Reserve“-Channel hinzugefügt, da der längste Knotenname in der Bedienungsanleitung sich aus drei Channel zusammensetzt.)

Channel	! C	! €	! T	? B	„Reserve“-Channel
Binärdarstellung	000	001	010	011	111

Tabelle 6: Codierung der Channels vom Getränkeautomaten.

Nun erstellen wir wieder eine Tabelle mit einer festen Anzahl boolescher Variablen: Die ersten booleschen Variablen stehen für je einen Channel. Durch sie wird in jeder Zeile der Tabelle eine erfüllende Belegung der Annotation repräsentiert. Mit Hilfe der übrigen Variablen werden die Knotennamen codiert. Hat die Annotation eines Knotens k erfüllende Belegungen, so werden zu

seiner Repräsentation k Zeilen in der Tabelle benötigt. Auf diese Weise wird jeder Knoten und seine Annotation binär codiert. Jede Zeile in der Tabelle stellt eine Belegungen b_i dar, bei der die gesuchte boolesche Funktion f_{OG} den Wert 1 liefern soll. Jede der Belegungen b_i kann durch die Konjunktion von negierter und nichtnegierter Variablen beschrieben werden. Die Disjunktion über diesen Konjunktionen repräsentiert die Menge aller Belegungen b_i und damit die Funktion f_{OG} .

In den ersten drei Zeilen der Tabelle 7 ist zu sehen, wie der Knoten „!€“ mit seiner Annotation dargestellt wird: Die ersten vier Felder enthalten eine erfüllende Belegung der Annotation ! $CV!T$. In den übrigen Feldern ist der Knotenname „!€“ codiert. Für jede erfüllende Belegung von ! $CV!T$ benötigen wir eine Zeile. In den Zeilen vier und fünf ist die Codierung der Knoten „!€!T“ und „!€!T?B“ aufgeführt. Knoten „!€!T?B“ hat keine ausgehenden Kanten. Deshalb ist in allen Feldern für die Channels eine Null eingetragen. Das gesamte BDD des Getränkeautomaten ist in Abbildung 12 zu sehen.

Belegung				Knotenname									
!C	!€	!T	?B	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	f_{OG}
0	0	1	0	0	0	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1	1
1	0	1	0	0	0	1	1	1	1	1	1	1	1
0	0	1	0	0	0	1	0	1	0	1	1	1	1
0	0	0	0	0	0	1	0	1	0	0	1	1	1
...													...

Tabelle 7: Codierung der Knoten „!€“ (ersten drei Zeilen), „!€!T“ (vierte Zeile) und „!€!T?B“ (fünfte Zeile) der Bedienungsanleitung des Getränkeautomaten.

	Beispiel a	Beispiel b	Beispiel c	Beispiel d	Getränkeautomat
Anzahl der Knoten im BDD (optimale Variablenordnung)	3	10	40	125	26
Anzahl der Variablen im BDD	2	6	9	16	13
Anzahl der Knoten in der Bedienungsanleitung (Baumdarstellung)	2	5	16	65	12

Tabelle 8: Testreihe aus Abbildung 9 mit Ansatz 2.

3.2.2 Auswertung

Das BDD des Getränkeautomaten (Abbildung 12) hat insgesamt 26 Knoten – weniger als im ersten Ansatz. In Tabelle 8 sind die BDD-Größen für unsere Testreihe aufgeführt. Wir sehen, dass die Knotenanzahl deutlich geringer steigt als im ersten Ansatz. Das Verhältnis zwischen den Knoten der Bedienungsanleitungen und der BDDs ist in etwa eins zu zwei. Berücksichtigen wir wieder, dass in den Knoten einer Bedienungsanleitung weder Informationen der Kantenbeschriftungen noch der Annotationen enthalten sind, sind die BDDs sehr kompakt. Dennoch ist anzunehmen, dass auch dieser Ansatz für praxisrelevante Bedienungsanleitungen ungeeignet ist. Die Gründe dafür sind:

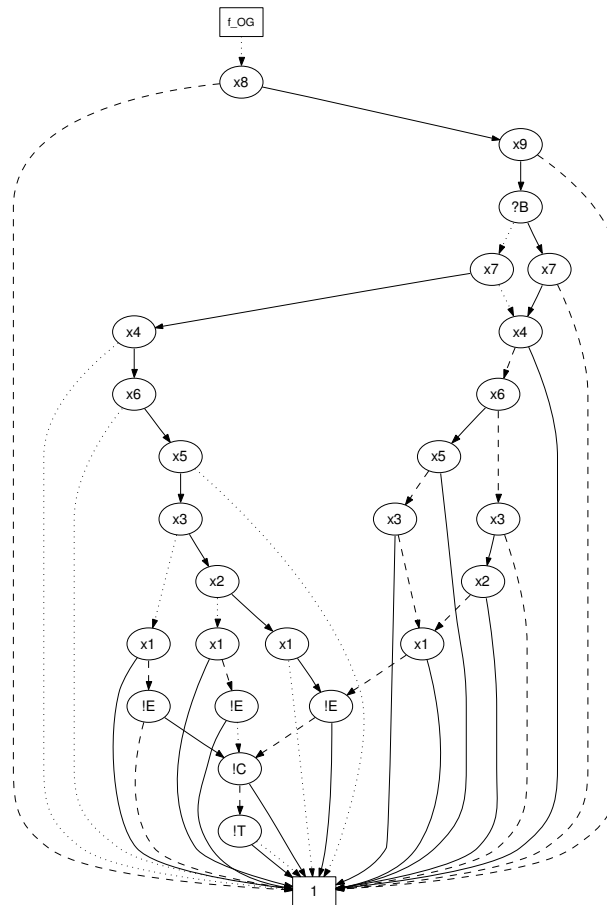


Abbildung 12: BDD des Getränkeautomaten nach Ansatz 2 (optimale Variablenordnung).

1. Die große Variablenanzahl. Bei n Channels (und einem „Reserve“-Channel) benötigen wir $\lceil \log_2(n + 1) \rceil$ Variablen zu ihrer Codierung. Da die Pfade der Bedienungsanleitung explizit codiert werden, kommen pro Ebene in der Bedienungsanleitung $\lceil \log_2(n + 1) \rceil$ Variablen im BDD hinzu.
2. Die Bedienungsanleitung wächst selbst sehr stark. Ihre Größe ist exponentiell gemessen an der Anzahl der Channels. Es ist daher nicht sinnvoll, sich allein an ihr zu messen bzw. das Ziel muss sein, deutlich kleinere Graphen zu erzeugen als es die Bedienungsanleitung ist.

Wir werden es also bei großen Beispielen mit sehr vielen Variablen zu tun haben, die wir u.U. nicht mehr repräsentieren können und falls sich das Verhältnis zwischen den Knoten der Bedienungsanleitungen und BDD nicht ändert, wird in der Praxis der Speicher nicht ausreichen.

3.3 Dritter Ansatz: Trennung von Struktur und Annotation

3.3.1 Vorgehensweise

Ein Nachteil im vorherigen Ansatz war die große Variablenanzahl, die sich durch das Codieren der langen Channelsequenzen ergibt. In diesem Ansatz soll das Problem beseitigt werden, indem wir die Knoten anders codieren: Die Knoten werden einfach durchnummeriert. Für die boolesche Funktion verwenden wir dann die Binärdarstellung der Zahlen. So benötigen wir für m Knoten nur noch $\lceil \log_2 m \rceil$ boolesche Variablen. Gleichzeitig wollen wir die Bedienungsanleitung als einen Graphen betrachten, der durch die Verschmelzung von Knoten mit gleicher Wissensfunktion entstanden ist (siehe Abbildung 8b auf Seite 15). Die Annotationen sollen analog zum zweiten Ansatz codiert werden.

In den ersten beiden Ansätzen waren die Knotennamen immer eindeutig. Jetzt haben wir die Knoten willkürlich durchnummeriert. Uns fehlen deshalb Informationen über die genaue Anordnung der Knoten im Graphen. Daher brauchen wir noch ein zweites BDD, das uns die fehlenden Informationen über die Struktur liefert.

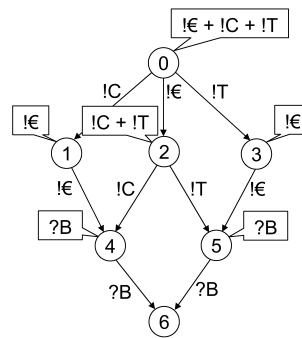


Abbildung 13: Bedienungsanleitung des Getränkeautomaten mit der Knotennummerierung.

Knotennummer	0	1	2	4	5	6
Binärdarstellung	000	001	010	100	101	110

Tabelle 9: Codierung der Knoten.

Channel	!C	!€	!T	?B
Binärdarstellung	00	01	10	11

Tabelle 10: Codierung der Channels.

Für das erste BDD erstellen wir wieder eine Tabelle. Mit den ersten Variablen werden wie im vorherigen Ansatz die Annotationen codiert und mit den übrigen $\lceil \log_2 m \rceil$ Variablen die Knotennummern. Auf diese Weise erhalten wir die Funktion $f_{\text{Annotation}}$.

Tabelle 9 zeigen die Binärdarstellungen für die Knoten der Bedienungsanleitung des Getränkeautomaten. Die Codierungen für die Knoten 1, 4 und 6 und ihre Annotation sind in der oberen Tabelle

in Abbildung 14 dargestellt. Das $BDD_{\text{Annotation}}$ für den Getränkeautomaten ist in Abbildung 15b zusehen.

Das zweite BDD repräsentiert alle Kanten der Bedienungsanleitung. Jede Kante, bestehend aus Startknoten, Channelnamen und Endknoten, bekommt eine eindeutige Codierung. Die Menge aller Kanten kann dann wieder durch eine boolesche Funktion f_{Struktur} beschrieben werden.

In der unteren Tabelle in Abbildung 14 ist die binäre Repräsentation der beiden Kanten $1 \text{!} \in 4$ und $4 \text{?} B 6$ zu sehen. Die verwendete Binärdarstellung für die Knoten bzw. Channels, kann aus den Tabellen 9 und 10 entnommen werden. Abbildung 15a zeigt das zugehörige BDD.

Belegung				Knoten			
$!C$	$! \in$	$!T$	$?B$	x_1	x_2	x_3	$f_{\text{Annotation}}$
0	0	1	0	0	0	1	1
1	0	0	0	0	0	1	1
1	0	1	0	0	0	1	1
0	0	1	0	1	0	0	1
0	0	0	0	1	1	0	1

	Startknoten			Channel		Endknoten			
Kante	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	f_{Struktur}
$1 \text{!} \in 4$	0	0	1	0	1	1	0	0	1
$4 \text{?} B 6$	1	0	1	1	1	1	1	0	1

Abbildung 14: Codierung der Knoten 1, 4 und 6 der Bedienungsanleitung des Getränkeautomaten.

3.3.2 Auswertung

Das BDD der Funktion $f_{\text{Annotation}}$ hat für den Getränkeautomaten 11 Knoten. Für die Codierung der Struktur (f_{Struktur}) werden 24 Knoten im BDD benötigt. Das sind zusammen mehr Knoten als in den ersten beiden Ansätzen. In der Testreihe (Tabelle 11) sehen wir, dass die BDDs für die beiden kleinsten Beispiele ebenfalls größer sind als in den bisherigen Versuchen. Im Gegensatz dazu ist die BDD-Größe von Beispiel 4 kleiner als bisher. Das bedeutet, dass die Knotenanzahl im Vergleich zu den anderen Ansätzen langsamer zunimmt. Falls sich dieser Trend auch für größere Beispiele fortsetzt, könnten mit diesem Vorgehen Bedienungsanleitungen praxisrelevanter Services codieren werden.

	Beispiel a	Beispiel b	Beispiel c	Beispiel d	Getränkeautomat
Anzahl der Knoten im BDD (optimale Variablenordnung)	7 (4+3)	16 (9+7)	41 (28+13)	91 (61+30)	35 (24+11)
Anzahl der Knoten in der Bedienungsanleitung (Graphdarstellung)	2	4	8	16	7

Tabelle 11: Testreihe für Ansatz 3.

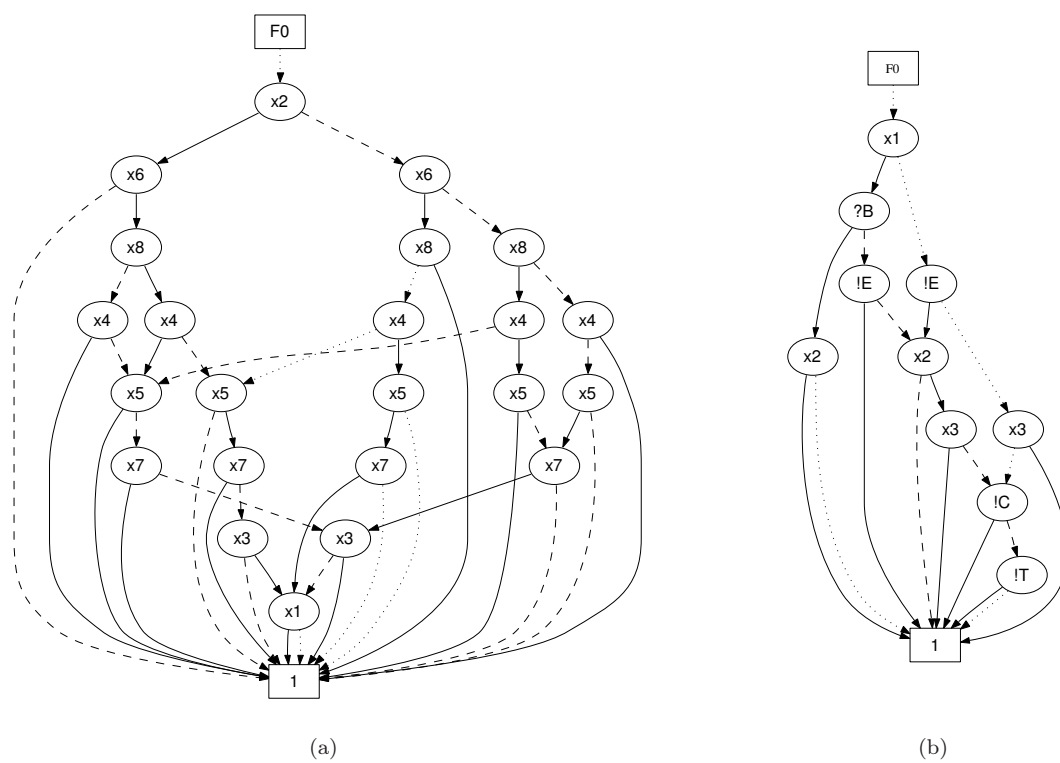


Abbildung 15: BDDs des Getränkeautomaten nach Ansatz 3: (a) BDD_{Struktur} und (b) $BDD_{\text{Annotation}}$ (jeweils optimale Variablenordnung).

4 Zusammenfassung

In dieser Studienarbeit wurden Möglichkeiten diskutiert, wie Bedienungsanleitungen durch BDDs repräsentiert werden können. In den vorgestellten Ansätzen haben wir die Bedienungsanleitung entweder als Baum oder als Graphen betrachtet und die Knoten entweder durchnummeriert oder mit der Sequenz von Channels, die zu ihnen führt, beschriftet. Auf dieser Grundlage ergeben sich mehrere Varianten (Tabelle 12). Drei von ihnen wurden in Kapitel 3 beschrieben. Die Übrigen wurden aus den folgenden Gründen nicht näher erläutert:

- (a) Der Shared BDD-Ansatz könnte auch mit einer Knotennummerierung durchgeführt werden. Es ist zu erwarten, dass die BDD-Größe dadurch geringfügig reduziert werden kann. Das Grundproblem des Ansatzes, dass für jede Strategie ein Wurzelknoten im Shared BDD nötig ist, bleibt jedoch bestehen. Das gilt auch, wenn wir die Bedienungsanleitungen als Graph codieren würden.
- (b) Der dritte Ansatz, kann auch dahingehend abgewandelt werden, dass wir die Bedienungsanleitung nicht als Graphen sondern als einen Baum betrachten. In Tests bewirkte die größere Anzahl darzustellender Knoten auch größere BDDs.
- (c) In einem Graphen können wir die Knoten nicht nach den zu ihm führenden Channelsequenzen benennen, da es mehrere Pfade zu einem Knoten geben kann und so die Namen nicht mehr eindeutig sind.

Art	Knotencodierung	Shared BDD	BDD
Baum	Channelsequenzen Nummern	1. Ansatz (a)	2. Ansatz (b)
Graph	Channelsequenzen Nummern	(a)(c) (a)	(c) 3. Ansatz

Tabelle 12: Übersicht der Ansätze.

Wir haben in Kapitel 3 gezeigt, dass Bedienungsanleitungen durch BDDs repräsentiert werden können. Für den ersten Ansatz mussten wir feststellen, dass die BDDs in der Praxis zu groß sind. Eine endgültige Beurteilung, ob sich einer der beiden anderen beschriebenen Ansätze eignet, kann hier nicht gegeben werden. Die Tests an den sehr kleinen Beispielen reichen nicht aus, um einschätzen zu können, ob mit BDDs eine kompakte Darstellung für Bedienungsanleitungen möglich ist. Versuche mit größeren Bedienungsanleitungen waren nicht möglich, da es zur Zeit keine Werkzeuge zur automatischen Generierung gibt und für die Berechnung der BDDs alle Knoten zusammen mit ihrer Annotation manuell eingeben werden müssen. Viel versprechend ist jedoch der dritte Ansatz, da bei ihm der Anstieg der Knotenanzahl in der Testreihe am geringsten ausfiel. Zudem können hier noch Knoten eingespart werden, wenn die beiden BDDs für Struktur und Annotation gemeinsam in einem Shared BDD repräsentiert werden.

Bei der Bewertung der Ergebnisse ist zu berücksichtigen, dass bei allen durchgeführten Tests eine optimale Variablenordnung für die BDDs ermittelt werden konnte. Schon bei mehr als 16 Variablen ist die Berechnung zu zeitaufwendig [Som05]. In der Praxis muss daher auf Heuristiken zurückgegriffen werden. Welche Heuristiken gut funktionieren, muss weiteren Untersuchungen vorbehalten bleiben.

Literatur

- [Bra86] Randal E. Bryant. *Graph-based algorithms for boolean function manipulation*. IEEE Transactions on Computers, C-35(8):677-691, August 1986.
- [Lyn96] Nancy A. Lynch, *Distributed Algorithms*. Morgan Kaufmann Publishers, San Mateo 1996.
- [Col04a] Mark Colan. *Service-Oriented Architecture expands the vision of Web services, Part 1 – Characteristics of Service-Oriented Architecture*. IBM developerWorks. 2004. <http://www.ibm.com/developerworks/library/ws-soaintro.html>
- [Col04b] Mark Colan. *Service-Oriented Architecture expands the vision of Web services, Part 2 – The SOA connection architecture*. IBM developerWorks. 2004. <http://www.ibm.com/developerworks/webservices/library/ws-soaintro2.html>
- [Mar04] Axel Martens. *Verteilte Geschäftsprozesse - Modellierung und Verifikation mit Hilfe von Web Services*. Dissertation, Institut für Informatik, Humboldt-Universität zu Berlin, 2004.
- [McM92] Kenneth L. McMillan. *Symbolic Model Checking - An approach to the state explosion problem*. PhD Thesis. 1992.
- [MS05a] Peter Massuthe and Karsten Schmidt. *Operating Guidelines - an Automata-Theoretic Foundation for Service-Orientated Architectures*. In Kai-Yuan Cai, Atsushi Ohnishi, and M.F. Lau, editors, *Proceedings of the Fifth International Conference on Quality Software (QSIC 2005)*, Melbourne, Australia, pages 452-457, September 2005. IEEE Computer Society.
- [MS05b] Peter Massuthe and Karsten Schmidt. *Operating Guidelines - an Alternative to Public View*. 2005.
- [MT98] Christoph Meinel und Thorsten Theobald. *Algorithmen und Datenstrukturen im VLSI Design: OBDD - Grundlagen und Anwendungen*. SpringerVerlag, Heidelberg. 1998.
- [Ref00] Frank Reffel. *Speichereffiziente Modellprüfung mit zyklischen binären Entscheidungsdiagrammen*. Dissertation, Fakultät für Informatik, Universität Karlsruhe, 2000.
- [Som05] Fabio Somenzi. *CUDD: CU Decision Diagram Package*. 2005.