

Symbolische Repräsentation von Bedienungsanleitungen für Services

Kathrin Kaschner¹, Peter Massuthe² und Karsten Wolf¹

¹ Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
{kathrin.kaschner, karsten.wolf}@informatik.uni-rostock.de

² Humboldt-Universität zu Berlin, Institut für Informatik, 10099 Berlin, Germany
massuthe@informatik.hu-berlin.de

1 Einleitung

Services sind selbstständige, ausführbare Softwarekomponenten. Sie werden jedoch im Allgemeinen nicht in Isolation ausgeführt, sondern kommunizieren durch Nachrichtenaustausch über ihr *Interface* mit anderen *Services*.

Mit dem Paradigma der *Serviceorientierten Architektur* (SOA) [Got00] 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 *Service* in einem zentralen Verzeichnis, das vom *Service Broker* (Vermittler) verwaltet wird. Ein *Service Requester* kann nun beim *Service 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.

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 einen *Deadlock* geraten noch sich unerwartete Nachrichten senden. Andererseits soll die innere Struktur von P gekapselt werden, so dass nicht einfach der komplette *Service* P veröffentlicht werden kann.

In [MRS05,MW06] haben wir einen Ansatz vorgestellt, in dem der *Provider* eine *Bedienungsanleitung* von seinem *Service* P bereitstellt. Diese beschreibt, wie sich ein *Partner* verhalten muss. Für einen *Service* R muss der *Broker* nun lediglich prüfen, ob R die Anforderungen der *Bedienungsanleitung* erfüllt. Dieser Vorgang wird *Matching* genannt. Ist das *Matching* erfolgreich, so wird die Interaktion zwischen R und P ohne Probleme verlaufen.

Eine *Bedienungsanleitung* ist ein gerichteter Graph, dessen Knoten mit Annotationen versehen sind. Da der Graph in der Praxis zu groß ist, um explizit gespeichert werden zu können, ist es erforderlich, nach einer kompakten Darstellung zu suchen. In diesem Artikel werden wir dazu einen Ansatz vorstellen. Wir werden zeigen, wie *Bedienungsanleitungen* symbolisch durch binäre Entscheidungsdiagramme (engl. Binary Decision Diagram, kurz: BDD) kodiert werden können und das *Matching*, basierend auf BDDs, erläutern.

Diese Arbeit gliedert sich wie folgt: Zunächst führen wir offene Workflownetze (oWFN) als ein formales Modell für *Services* ein und definieren die Interaktion zweier *Services* auf Basis von oWFN. Im dritten Abschnitt erläutern wir, wie für ein oWFN eines *Service* die *Bedienungsanleitung* berechnet wird. Wir werden uns dabei auf azyklische *Services* beschränken. Das heißt, sie können nicht in einen Zustand zurückkehren, in dem sie schon einmal waren. Anschließend werden wir zeigen, wie eine *Bedienungsanleitung* ihre BDD-Darstellung überführt wird und wie das *Matching* mit BDDs ausgeführt werden kann.

2 Modellierung von Services

Zur Darstellung von Services nutzen wir *offene Workflownetze* (oWFN) [MRS05], eine spezielle Klasse von Petrinetzen. Sie sind ähnlich den in [Aal98] vorgestellten Workflownetzen, besitzen jedoch spezielle Kommunikationsplätze, um das Senden und Empfangen von Nachrichten zu modellieren. Dabei abstrahieren wir vom Nachrichteninhalt und betrachten nur das Sende- bzw. Empfangsereignis. Formal ist ein oWFN ein Petrinetz $N = (S, T, F)$ mit:

- einer Menge $in \subseteq S$ von *Inputplätzen* und einer Menge $out \subseteq S$ von *Outputplätzen*, so dass $in \cap out = \emptyset$ und für alle Transitionen $t \in T$ gilt: falls $p \in in$ ($p \in out$), so $(t, p) \notin F$ ($(p, t) \notin F$),
- einer *Anfangsmarkierung* m_0 und
- einer Menge Ω von *Endmarkierungen*.

Die Menge $I = in \cup out$ ist das *Interface* von N und $J = S \setminus I$ ist die Menge der *internen* Plätze. Das *Innere* von N ist das Teilnetz $Inner_N =_{def} (J, T, F \cap ((J \times T) \cup (T \times J)))$. Transitionen, die mit einem Inputplatz x bzw. mit einem Outputplatz y verbunden sind, beschriften wir im oWFN mit $?x$ bzw. $!y$. Das Empfangen einer Nachricht über den Kanal x wird durch das Konsumieren einer Marke vom Inputplatz x durch die Transition $?x$ modelliert. Analog wird das Senden einer Nachricht über den Kanal y durch das Produzieren einer Marke von der Transition $!y$ auf den Outputplatz y beschrieben.

Als ein Beispiel für einen Service Provider betrachten wir einen Getränkeautomaten, der durch das in Abbildung 1b dargestellte oWFN G beschrieben wird. Der Automat erwartet, dass ein Kunde Geld ($?\epsilon$) einwirft und die Taste für den Tee ($?T$) oder den Kaffee ($?C$) drückt. Anschließend gibt er einen Becher ($!B$) mit dem bestellten Getränk zurück, sofern ausreichend Geld bezahlt wurde. Wurde zu wenig eingeworfen, so erhält der Kunde kein Getränk. Stattdessen wird das Geld wieder zurückgegeben ($!\epsilon'$). Außerdem wollen wir einen Kunden betrachten, der Tee am Getränkeautomaten bestellt (Abbildung 1a).

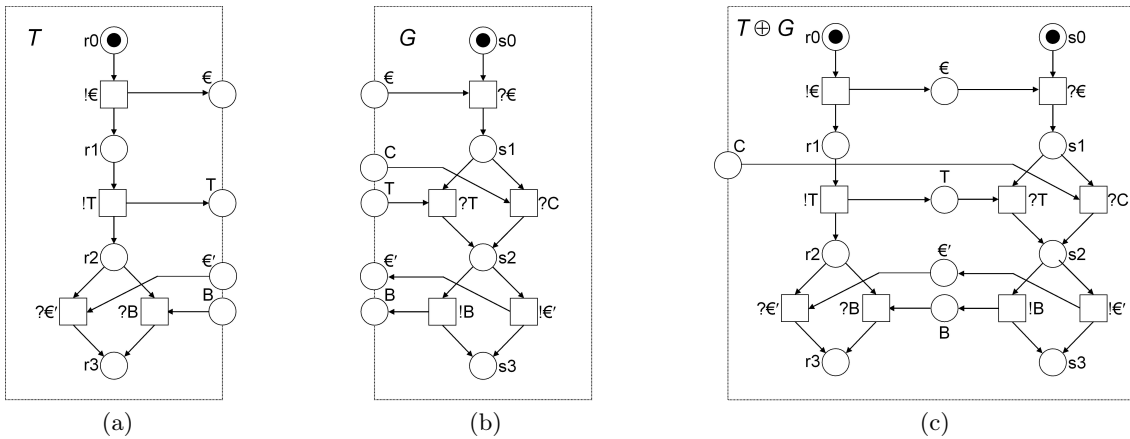


Abbildung 1. (a) Das oWFN T eines Kunden, der Tee bestellt, (b) das oWFN G , das einen Getränkeautomaten modelliert und (c) das komponentierte oWFN $T \oplus G$.

Das Zusammenspiel zweier Services kann durch die *Komposition* ihrer oWFN beschrieben werden. Die Komposition zweier oWFN N und M ergibt das oWFN $M \oplus N =_{def} (S_M \cup S_N, T_M \cup T_N, F_M \cup F_N)$ mit $in_{M \oplus N} =_{def} (in_M \setminus out_N) \cup (in_N \setminus out_M)$, $out_{M \oplus N} =_{def} (out_M \setminus in_N) \cup (out_N \setminus in_M)$ und der Anfangsmarkierung $m_{0_{M \oplus N}} = m_{0_M} + m_{0_N}$. Eine Markierung $m_{M \oplus N} = m_M + m_N$ ist eine Endmarkierung ($m_{M \oplus N} \in \Omega_{M \oplus N}$) genau dann,

wenn $m_M \in \Omega_M$ und $m_N \in \Omega_N$. Plätze in $out_M \cap in_N$ oder in $in_M \cap out_N$ werden in $M \oplus N$ zu internen Plätzen.

Das komponierte System $T \oplus G$ der beiden oWFN T und G ist in Abbildung 1c zu sehen. Der einzige Interfaceplatz in $T \oplus G$ ist $C \in in_{T \oplus G}$. Die anderen Interfaceplätze ϵ , ϵ' , T und B wurden in $T \oplus G$ zu einem internen Platz „verschmolzen“. Die (einzige) Endmarkierung ist m mit $m(r3) = m(s3) = 1$ (alle übrigen Plätze sind in m unmarkiert).

Nicht jeder beliebige Service ist geeignet, mit einem gegebenen Service zu interagieren. Die Kommunikation zwischen zwei oWFN R und P verläuft genau dann fehlerfrei, wenn in $R \oplus P$ alle Deadlocks Endmarkierungen sind. In diesem Fall nennen wir R eine *Strategie* für P . In unserem Beispiel ist T eine Strategie für G .

3 Bedienungsanleitungen

Veröffentlicht ein Provider für seinen Service alle Strategien, so muss der Broker lediglich prüfen, ob der Service eines anfragenden Requesters mit einer der Strategien übereinstimmt. Im Folgenden werden wir eine kompakte Beschreibung für die Menge aller Strategien eines Service angeben. Diese nennen wir *Bedienungsanleitung*.

Die Bedienungsanleitung OG_P eines oWFN P beschreibt, wie sich ein oWFN R verhalten muss, um eine Strategie für P zu sein. Das *Verhalten* eines gewöhnlichen Petrinetzes kann durch seinen Erreichbarkeitsbaum dargestellt werden. Das Teilnetz $Inner_P$ von P ist ein Petrinetz, von dem wir den Erreichbarkeitsbaum berechnen können. Dieser ist endlich, da wir nur azyklische Services betrachten.

Zusätzlich werden die Kanten im Erreichbarkeitsbaum, deren entsprechende Transition in P mit einem Output- oder Inputplatz verbunden sind, mit dem Label $!x$ bzw. $?x$ versehen. Die übrigen Kanten erhalten das Label τ (interne Transitionen). Falls die Labels der ausgehenden Kanten eines jeden Knoten paarweise verschieden sind und es keine mit τ beschrifteten Kanten gibt, ist ein Verhalten *deterministisch*. Zur Vereinfachung wollen wir im Folgenden nur oWFN mit deterministischem Verhalten betrachten.

In Abbildung 2 ist das Verhalten B_T und B_G der in Abbildung 1 dargestellten oWFN T und G zu sehen. Sowohl B_T als auch B_G sind deterministisch.

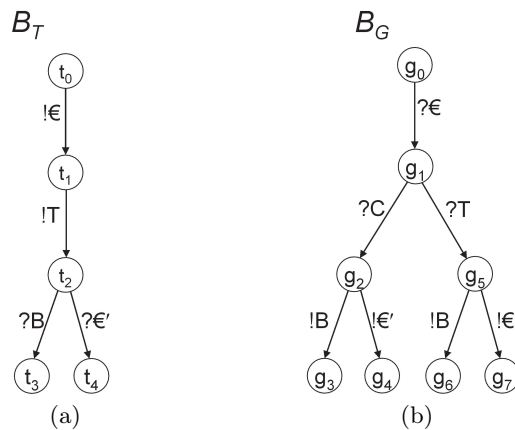


Abbildung 2. (a) Das Verhalten B_T des oWFN T und (b) das Verhalten B_G des oWFN G .

Eine Strategie mit deterministischem Verhalten bezeichnen wir als eine *deterministische Strategie*. Sei \mathcal{B}_P die Menge der Verhalten aller deterministischen Strategien für P . Ein Verhalten B nennen wir *liberaler* als ein Verhalten B' , falls B' isomorph zu einem bei der

Wurzel beginnenden Teilbaum von B ist. In [Sch05] haben wir gezeigt, dass es für jedes oWFN P ein eindeutiges *liberalstes Verhalten* B^* in \mathcal{B}_P gibt. Demzufolge nennen wir jedes oWFN R , dessen Verhalten gleich dem liberalsten Verhalten von P ist ($B_R = B^*$), eine *liberalste Strategie* für P .

In [MW06] haben wir bereits bewiesen, dass das Verhalten jeder (deterministischen) Strategie für P isomorph zu einem bei der Wurzel beginnenden Teilbaum des liberalsten Verhalten B^* ist. Damit liefert B^* die Grundlage für die Bedienungsanleitung.

Leider ist nicht jeder Teilbaum von B^* das Verhalten einer Strategie für P . Um unerlaubte Teilbäume erkennen und ausschließen zu können, annotieren wir jeden Knoten von B^* mit einer booleschen Formel. Diese besagt, welche der ausgehenden Kanten im Verhalten B_R eines oWFN R vorhanden sein müssen, so dass R eine Strategie für P ist.

Die Bedienungsanleitung $OG_P = (B^*, \Sigma)$ des oWFN P setzt sich also aus zwei Teilen zusammen: dem Verhalten B^* , das die Struktur der Bedienungsanleitung enthält und der Funktion Σ , die jedem Knoten in B^* eine Annotation zuordnet.

Abbildung 3a zeigt für das oWFN G die Bedienungsanleitung OG_G . Wir können nun leicht erkennen, dass das oWFN T eine Strategie für G ist: Zum einen ist das Verhalten B_T (vgl. Abbildung 2) ein bei der Wurzel beginnender Teilbaum von OG_G und zum anderen besitzt B_T alle in den Annotationen vorgegebenen Kanten. Das Verhalten eines zweiten Kunden, der Geld einwirft, die Taste ?T drückt und das bestellte Getränk entgegen nimmt, ist ebenfalls ein Teilbaum vom OG_G . Dennoch ist dieser kein Partner für den Getränkeautomaten, da er nicht das eingeworfene Geld wieder zurück nehmen kann und damit sein Verhalten die Annotation $?B \wedge ?\epsilon'$ verletzt.

Der Graph in Abbildung 3b ist ebenfalls die Bedienungsanleitung des oWFN G : Hier wurden Knoten mit gleichen Fortführungen miteinander verschmolzen.

Den Algorithmus zur automatischen Berechnung von Bedienungsanleitungen haben wir in [MW06] ausführlich beschrieben und seine Korrektheit bewiesen.

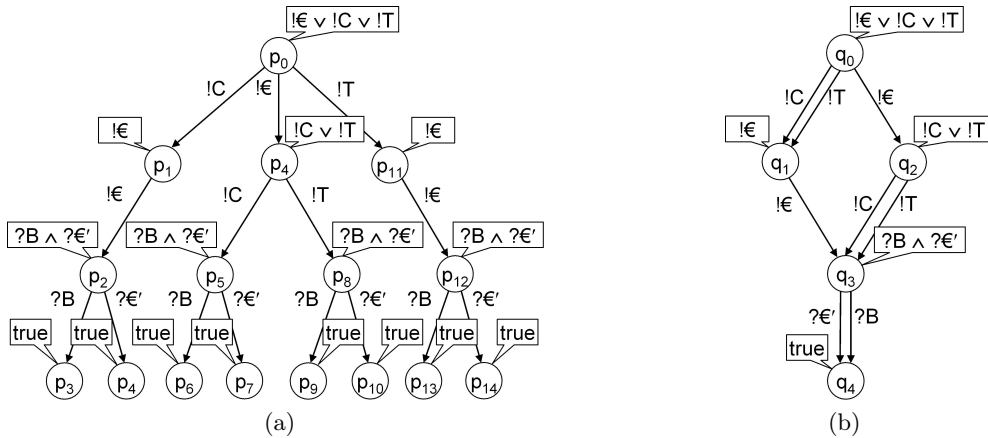


Abbildung 3. Bedienungsanleitung OG_G des oWFN G : (a) in Baumstruktur und (b) als Graph.

4 BDD-Darstellung von Bedienungsanleitungen

Die Größe von Bedienungsanleitungen steigt jedoch exponentiell in der Größe des Interfaces des zugehörigen oWFN, so dass in der Praxis Bedienungsanleitungen nicht explizit gespeichert werden können. Daher ist es notwendig, sie in eine kompakte Darstellung zu

überführen. Hierfür wollen wir BDDs (binary decision diagram) [Bry86] nutzen. Sie ermöglichen eine kompakte Repräsentation boolescher Funktionen und erlauben eine effiziente Implementierung boolescher Operatoren wie Konjunktion und Disjunktion.

Unser Vorschlag ist, die Struktur und die Annotation der Bedienungsanleitung separat in je einem BDD darzustellen. Damit erhalten wir ein $BDD_{Struktur}$ und ein $BDD_{Annotation}$, die zusammen die Informationen der Bedienungsanleitung enthalten. In [Bry86] wurde bewiesen, dass es zu jeder booleschen Funktion eine BDD-Darstellung gibt. Deshalb werden wir nur beschreiben, wie für die beiden BDDs die entsprechenden booleschen Funktionen $f_{Struktur}$ und $f_{Annotation}$ berechnet werden.

Um die Struktur der Bedienungsanleitung eindeutig zu kodieren, genügt es, die Menge aller Kanten von B^* in der Funktion $f_{Struktur}$ zu repräsentieren. Jede Kante $[q_1, l, q_2]$, bestehend aus den beiden Knoten q_1 und q_2 und dem Label l , kann durch das Wort „ $q_1 l q_2$ “ dargestellt werden. Ordnen wir den Knoten und Labels von B^* jeweils eine eindeutige binäre Nummer zu, so können wir jede Kante durch eine Zeichenfolge bestehend aus Nullen und Einsen darstellen. Diese interpretieren wir als eine Belegung. Die zu $[q_1, l, q_2]$ gehörige boolesche Funktion $f_{q_1 l q_2}$ ist damit eine Konjunktion von negierten und nichtnegierten Variablen entsprechend der Belegung. Die gesuchte Funktion $f_{Struktur}$ erhalten wir schließlich, indem wir die Disjunktion über alle Kantenfunktionen $f_{q_i l_j q_k}$ bilden.

Abbildung 4 zeigt für die Bedienungsanleitung OG_G aus Abbildung 3b eine mögliche Kodierung der Knoten und Labels. Demnach wird z.B. die Kante $[q_0, !T, q_1]$ durch die boolesche Funktion $f_{q_0 !T q_1} = \overline{x_1} \overline{x_2} \overline{x_3} \overline{x_4} x_5 \overline{x_6} \overline{x_7} \overline{x_8} x_9$ repräsentiert. Entsprechend können auch die übrigen Kanten kodiert werden und wir erhalten mit der Funktion $f_{Struktur, G} = f_{q_0 !C q_1} \vee f_{q_0 !T q_1} \vee f_{q_0 !\in q_2} \vee f_{q_1 !\in q_3} \vee f_{q_2 !C q_3} \vee f_{q_2 !T q_3} \vee f_{q_3 ?B q_4} \vee f_{q_3 ?\in' q_4}$ eine binäre Kodierung der Struktur von OG_G .

Knoten	q_0	q_1	q_2	q_3	q_4
Binärdarstellung	000	001	010	011	100

(a)

Label	!C	!\in	!T	?\in'	?B
Binärdarstellung	000	001	010	011	100

(b)

Abbildung 4. Kodierung der Knoten (a) und Labels (b) der Bedienungsanleitung OG_G .

In der Funktion $f_{Annotation}$ werden die Annotationen der Bedienungsanleitung zusammen mit den zugehörigen Knoten kodiert. Zur Darstellung der Knoten nutzen wir die gleiche Binärcodierung wie für die Funktion $f_{Struktur}$. So erhalten wir für jeden Knoten q die Knotenfunktion f_q . Im nächsten Schritt bilden wir durch Konjunktion von f_q und der Annotation $\Sigma(q)$ eine boolesche Funktion $f_{q, \Sigma(q)}$. Sie repräsentiert den Knoten q und seine Annotation. Die Labels werden jetzt also nicht mehr durch eine Binärzahl kodiert, sondern sind selbst boolesche Variablen in $f_{q, \Sigma(q)}$. Durch die Disjunktion aller Funktionen $f_{q_i, \Sigma(q_i)}$ erhalten wir schließlich die Funktion $f_{Annotation}$.

In der Bedienungsanleitung des oWFN G aus Abbildung 3 wird beispielsweise die Annotation $\Sigma(q_0)$ vom Wurzelknoten q_0 durch die Funktion $f_{q_0, \Sigma(q_0)} = \overline{x_1} \overline{x_2} \overline{x_3} (\!\in \vee !C \vee !T)$ kodiert. Entsprechend erhalten wir mit der Funktion $f_{Annotation, G} = f_{0, \Sigma(0)} \vee f_{1, \Sigma(1)} \vee f_{2, \Sigma(2)} \vee f_{3, \Sigma(3)} \vee f_{4, \Sigma(4)}$ eine binäre Kodierung der Annotationen von OG_G .

Die Bedienungsanleitung können wir nun in eine BDD-Darstellung überführen, indem aus den beiden Funktionen $f_{Struktur}$ und $f_{Annotation}$ die entsprechenden BDDs erstellt werden.

Im Folgenden wollen wir anhand einer Testreihe exemplarisch die Effizienz von BDDs zeigen. Untersucht wurden die Bedienungsanleitungen von oWFN, deren Verhalten lediglich darin besteht, eine bestimmte Anzahl von Nachrichten zu senden. Beispielsweise sen-

det Sequenz3 drei Nachrichten (vgl. Abbildung 5b), Sequenz4 vier Nachrichten usw. In Abbildung 5a ist für Sequenz3 bis Sequenz15 angegeben, wie viele Knoten die Bedienungsanleitungen in der expliziten Darstellung und in der BDD-Darstellung haben. Bei diesem Vergleich ist zu berücksichtigen, dass die angegebene Knotenanzahl der BDD-Darstellung die Kantenbeschriftungen (Labels) und Annotationen bereits umfasst und die Knoten eine feste Größe von 21 Byte beanspruchen [MT98]. Im Gegensatz dazu variiert der Speicherbedarf eines Knotens in der expliziten Darstellung in Abhängigkeit von der Anzahl seiner Nachfolgeknoten und seiner Annotation. Zusätzlich ist noch Speicherbedarf für die Kantenbeschriftungen hinzuzurechnen. Damit sind die Knoten i.d.R. größer als die BDD-Knoten.

Abbildung 5a zeigt für die Bedienungsanleitungen in expliziter Darstellung eine exponentiell zunehmende Knotenanzahl. Die BDD-Kurve hingegen steigt nur langsam, fast linear, an. Damit ist für die großen Bedienungsanleitungen eine kompakte Darstellung durch BDDs möglich.

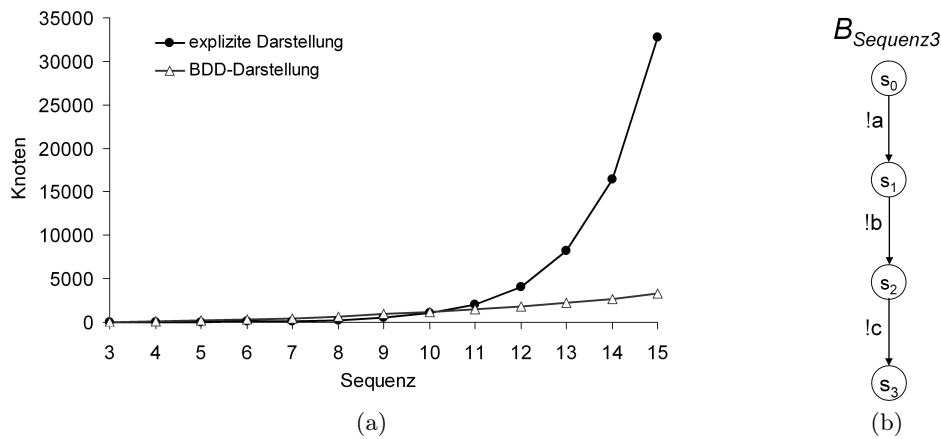


Abbildung 5. (a) Vergleich von Bedienungsanleitungen in expliziter Darstellung und in BDD-Darstellung, (b) Verhalten $B_{Sequenz3}$.

5 Matching

Um entscheiden zu können, ob ein oWFN R eine Strategie für ein gegebenes oWFN P ist, muss lediglich überprüft werden, ob das Verhalten B_R ein Teilgraph der Bedienungsanleitung OG_P ist und ob B_R alle von den Annotationen in OG_P vorgegebenen Kanten aufweist (vgl. Abschnitt 3). Ist die Bedienungsanleitung als boolesche Formel kodiert bzw. liegt sie in BDD-Darstellung vor, so sind die Kanten und Annotationen nicht mehr einfach ablesbar, wie in der expliziten Darstellung als annotierter Graph. Um das Matching dennoch ausführen zu können, überführen wir B_R ebenfalls in eine Binärdarstellung: Die Struktur wird in der Funktion f_S kodiert und eine zweite Funktion, f_L , repräsentiert für jeden Knoten k die Menge der Labels, mit denen die von k ausgehenden Kanten beschriftet sind.

Die Knoten und Labels in B_R erhalten wieder eine binäre Nummer, so dass f_S nach dem gleichen Vorgehen berechnet werden kann, wie die Funktion $f_{Struktur}$. Um das Matching effizient durchführen zu können, wird die Kodierung jedoch nicht willkürlich gewählt, sondern richtet sich nach der zu matchenden Bedienungsanleitung OG_P : gleiche Labels und Knoten erhalten die gleiche binäre Nummer. Dabei ist ein Knoten k in B_R gleich einem

Knoten q in OG_P , falls der zu k führende Weg auch ein Weg ist, der zu q führt. Die Kodierung der Labels kann leicht aus der Tabelle entnommen werden, nach der die Labels der Bedienungsanleitung kodiert wurden. Für die Knoten ist diese Vorgehensweise nicht möglich. Hier kann jedoch die Kodierung der Knoten durch eine beim Startknoten beginnende Tiefensuche durch B_R unter Anwendung boolescher Operatoren auf $f_{Struktur}$ ermittelt werden [Kas06].

Um die Funktion f_L zu berechnen, annotieren wir auch in B_R jeden Knoten k mit einer booleschen Funktion $\Psi(k)$. Diese ist immer von den Labels der Bedienungsanleitung abhängig. Die Labels (negiert oder nichtnegiert) werden dabei durch Konjunktion miteinander verknüpft. Ein Label l ist in $\Psi(k)$ nichtnegiert, falls es eine Kante $[k, l, k']$ in B_R gibt; anderenfalls ist l negiert. Die Knoten zusammen mit ihrer Annotation können nun wie bei den Bedienungsanleitungen durch die Funktion f_L repräsentiert werden.

Als ein Beispiel wollen wir das Verhalten B_T aus Abbildung 2a bzgl. der Bedienungsanleitung OG_P aus Abbildung 3b kodieren. Für die Labels in B_T nutzen wir die Tabelle aus Abbildung 4b, nach der wir bereits die Labels der Bedienungsanleitung kodiert haben. Die Kodierung der Knoten ist in Abbildung 6 aufgeführt. Damit ergibt sich beispielsweise für die Kante $[t_0, !\epsilon, t_1]$ die boolesche Funktion $f_{t_0! \epsilon t_1} = \overline{x_1} \overline{x_2} \overline{x_3} \overline{x_4} \overline{x_5} x_6 \overline{x_7} x_8 \overline{x_9}$. Analog können die übrigen Kanten kodiert werden und wir erhalten schließlich die Funktion $f_{S,T} = f_{t_0! \epsilon t_1} \vee f_{t_1! \tau t_2} \vee f_{t_2? B t_3} \vee f_{t_2? \epsilon' t_4}$. Die Annotation für den Startknoten t_0 ist zum Beispiel $\Psi(t_0) = \overline{!C} !\epsilon \overline{!T} ?\overline{B} ?\epsilon'$, da nur eine mit $!\epsilon$ beschriftete Kante t_0 verlässt. Auf diese Weise ergibt sich die Funktion $f_{t_0, \Psi(t_0)} = \overline{x_1} \overline{x_2} \overline{x_3} \overline{!C} !\epsilon \overline{!T} ?\overline{B} ?\epsilon'$ und wir können die Funktion $f_L = f_{t_0, \Psi(t_0)} \vee f_{t_1, \Psi(t_1)} \vee f_{t_2, \Psi(t_2)} \vee f_{t_3, \Psi(t_3)} \vee f_{t_4, \Psi(t_4)}$ berechnen.

Knoten in B_T	t_0	t_1	t_2	t_3	t_4
Knoten in OG_P	q_0	q_2	q_3	q_4	q_4
Binärdarstellung	000	010	011	100	100

Abbildung 6. Kodierung der Knoten in B_T .

Aufgrund der Tatsache, dass B_R bzgl. der zu matchenden Bedienungsanleitung OG_P kodiert wird, reduziert sich das Matching auf die Frage, ob f_S eine Teilfunktion von $f_{Struktur}$ und f_L eine Teilfunktion von $f_{Annotation}$ ist und damit auf die Überprüfung der beiden Gleichungen $\overline{f_S} \vee f_{Struktur} = 1$ und $\overline{f_L} \vee f_{Annotation} = 1$. Diese kann effizient auf der BDD-Darstellung ausgeführt werden.

6 Zusammenfassung

Eine Bedienungsanleitung OG_P charakterisiert alle Strategien für einen Service P . Mit ihr kann der Broker leicht entscheiden, ob ein Service R zu P passt oder nicht, ohne dass die (geheime) innere Struktur von P veröffentlicht werden muss. Da Bedienungsanleitungen im Allgemeinen sehr groß sind, ist eine effiziente Speicherung erforderlich. Wir haben einen Vorschlag gemacht, wie Bedienungsanleitungen symbolisch durch BDDs repräsentiert werden können. Anhand einer Testreihe konnten wir zeigen, dass eine kompakte Darstellung durch BDDs möglich ist. Darüber hinaus haben wir erläutert, wie auf der BDD-Darstellung passende Partner für R effizient gefunden werden können.

Literatur

- [Aal98] AALST, W. M. P. VAN DER: *The application of Petri nets to workflow management*. Journal of Circuits, Systems and Computers, 8(1):21–66, 1998.

- [Bry86] BRYANT, RANDAL E.: *Graph-Based Algorithms for Boolean Function Manipulation*. IEEE Transactions on Computers, C-35(8):677–691, August 1986.
- [Got00] GOTTSCHALK, KARL: *Web Services Architecture Overview*. IBM Whitepaper, IBM developerWorks, September 2000. <http://ibm.com/developerWorks/web/library/w-ovr/>.
- [Kas06] KASCHNER, KATHRIN: *BDD-basiertes Matching von Services*. Diplomarbeit, Humboldt-Universität zu Berlin, März 2006.
- [MRS05] MASSUTHE, PETER, WOLFGANG REISIG und KARSTEN SCHMIDT: *An Operating guideline approach to the SOA*. Annals of Mathematics, Computing & Teleinformatics, 1(3):35–43, 2005. To appear.
- [MT98] MEINEL, CHRISTOPH und THORSTEN THEOBALD: *Algorithms and Data Structures in VLSI Design*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.
- [MW06] MASSUTHE, PETER und KARSTEN WOLF: *An Algorithm for Matching Nondeterministic Services with Operating Guidelines*. Informatik-Berichte 202, Humboldt-Universität zu Berlin, 2006.
- [Sch05] SCHMIDT, KARSTEN: *Controllability of Open Workflow Nets*. In: DESEL, JÖRG und ULRICH FRANK (Herausgeber): *Enterprise Modelling and Information Systems Architectures*, Band P-75 der Reihe *Lecture Notes in Informatics (LNI)*, Seiten 236–249, Bonn, 2005. Entwicklungsmethoden für Informationssysteme und deren Anwendung (EMISA, RWTH Aachen), Köllen Druck+Verlag GmbH.