Effiziente Abschätzung von Datenflussfehlern in strukturierten Geschäftsprozessen

Thomas S. Heinze[1], Wolfram Amme[1], Simon Moser[2]

[1] Friedrich-Schiller-Universität Jena
{T.Heinze,Wolfram.Amme}@uni-jena.de
[2] IBM Entwicklungslabor Böblingen
smoser@de.ibm.com

Zusammenfassung. Neben dem Kontrollfluss von Geschäftsprozessen kann auch der Datenfluss Ursache einer fehlerhaften Prozessausführung sein, daher ist die Überprüfung eines Prozessmodells auf Datenflussfehler ebenfalls wesentlich. Wir schlagen in diesem Beitrag eine Methode zur Abschätzung von Datenflussfehlern für strukturierte Geschäftsprozesse vor. Auf Grundlage der durch eine Datenflussanalyse abgeleiteten Datenflussinformation geben wir Fehlermengen für mögliche und sichere Datenflussfehler eines Geschäftsprozesses an. Der Vorteil dieses Ansatzes besteht zum einen in der Effizienz der Analyse, andererseits aber auch in der Identifikation und Lokalisation von Fehlern in einem Schritt. Als Nachteil ergibt sich hingegen der Verlust absoluter Präzision.

1 Einführung

Neben der Verifikation von Geschäftsprozessen hinsichtlich Kontrollflussfehlern, wie Verklemmungen oder fehlender Synchronisation, ist die Analyse der Verwendung von Prozessdaten zur Gewährleistung einer fehlerfreien Prozessausführung von Interesse [5]. Typische Fehler in diesem Zusammenhang sind beispielsweise der lesende Zugriff auf noch uninitialisierte oder bereits gelöschte Daten, das parallele Schreiben und Lesen von Daten oder das Überschreiben ungelesener Daten. Enthält ein Prozessmodell auch Informationen zur Verwendung der Prozessdaten, in Form der durch Prozessaktivitäten geschriebenen, gelesenen und gelöschten Daten, kann eine Überprüfung auf derartige Datenflussfehler erfolgen.

Ein insbesonders für die Analyse des Datenflusses geeignetes Verfahren ist die *statische Datenflussanalyse* [2, 3]. Im Gegensatz zu Verifikationstechniken die auf einer vollständigen Modellprüfung beruhen erlaubt die Datenflussanalyse eine effiziente Ableitung von konservativer Datenflussinformation, verzichtet aber im Gegenzug auf exakte Ergebnisse. Auf diese Weise kann der exponentielle Verifikationsaufwand vermieden werden, der sich sonst bei einer präzisen Analyse ergibt. Der hohe Aufwand ist dabei auf die zur Überprüfung des Datenflusses notwendige Identifikation parallel ausführbarer Prozessaktivitäten zurückzuführen, die schon für strukturierte Prozesse und unter Ausschluß von Schleifen exponentielle Kosten verursachen kann [1]. Zusätzlich wird auch keine endliche Abstraktion für die in einem Geschäftsprozess auftretenden Daten benötigt, da lediglich der Datenfluss zwischen den Prozessaktivitäten berücksichtigt werden muss.

Fehlende Daten	Zugriff auf ein uninitialisiertes oder gelöschtes Datum
Redundante Daten	Schreiben eines Datums durch eine Prozessaktivität auf das im weiteren Verlauf nicht lesend zugegriffen wird
Überschriebene Daten	Überschreiben eines Datums durch eine Prozessaktivität auf das noch nicht lesend zugegriffen wurde
Inkonsistente Daten	Zugriff einer Prozessaktivität auf ein Datum und dazu paralleles Schreiben oder Löschen desselben Datums
Nicht gelöschte Daten	Fehlendes Löschen für ein geschriebenes Datum
Doppelt gelöschte Daten	Zweimaliges Löschen ein und desselben Datums
Zu spät gelöschte Daten	Letzter lesender Zugriff einer Prozessaktivität auf ein Datum ohne sich unmittelbar anschließendes Löschen

Tabelle 1. Datenflussfehler (Anti-Muster) nach [5]

Im vorliegenden Beitrag zeigen wir, wie das Verfahren der Datenflussanalyse zur Überprüfung eines strukturierten Geschäftsprozesses auf Datenflussfehler genutzt werden kann. In Abschnitt 2 wird dazu zunächst ein Überblick zu Datenflussfehlern und dem hier verwendeten Prozessmodell gegeben. Danach erfolgt in Abschnitt 3 eine Beschreibung der Bestimmung von fehlenden, gelöschten sowie definierenden Daten mit Hilfe einer *statischen Datenflussanalyse*. Auf Grundlage der so für einen Prozess effizient ableitbaren Datenflussinformationen können wir dann in Abschnitt 4 *Fehlermengen* einführen, die Abschätzungen zu den im Prozess enthaltenen Datenflussfehlern in Form *sicherer* und *möglicher* Fehler bilden. Schließlich wird der Beitrag in Abschnitt 5 kurz zusammengefasst.

2 Grundbegriffe

2.1 Datenflussfehler

In der Arbeit [5] wird eine Sammlung der in Geschäftsprozessen auftretenden Datenflussfehler beschrieben. Dabei werden mehrere Anti-Muster vorgestellt, die Schwachstellen hinsichtlich einer fehlerfreien Verwendung von Prozessdaten darstellen. Wir beziehen uns im Folgenden auf diese, in Tabelle 1 angegebenen, Anti-Muster, wenn wir von Datenflussfehlern sprechen. Im Gegensatz zu [5] wird hier für die Anti-Muster *Redundante Daten* und *Überschriebene Daten* keine Unterscheidung zwischen Fehlern, die immer auftreten, und solchen, die nur in bestimmten Ausführungsszenarien auftreten, vorgenommen. Stattdessen werden für alle Anti-Muster die Begriffe *sicherer* und *möglicher Fehler* definiert:

Definition 1 (Sicherer/Möglicher Datenflussfehler). *Ein sicherer Fehler tritt unabhängig vom zur Ausführungszeit tatsächlich gewählten Kontrollfluss eines Prozesses immer auf. Ein möglicher Fehler ist ein Kandidat für einen Fehler der in mindestens einer Prozessausführung auftreten kann.*

2.2 Prozessrepräsentation

Zur Durchführung unserer Methode benötigen wir ein Prozessmodell, dass die Wiedergabe des Datenflusses innerhalb eines Prozesses gestattet. Zu diesem Zweck werden *erweiterte Workflow-Graphen* genutzt, die gewöhnliche Workflow-Graphen [4] mit Datenflussannotationen versehen. Auf der rechten Seite von

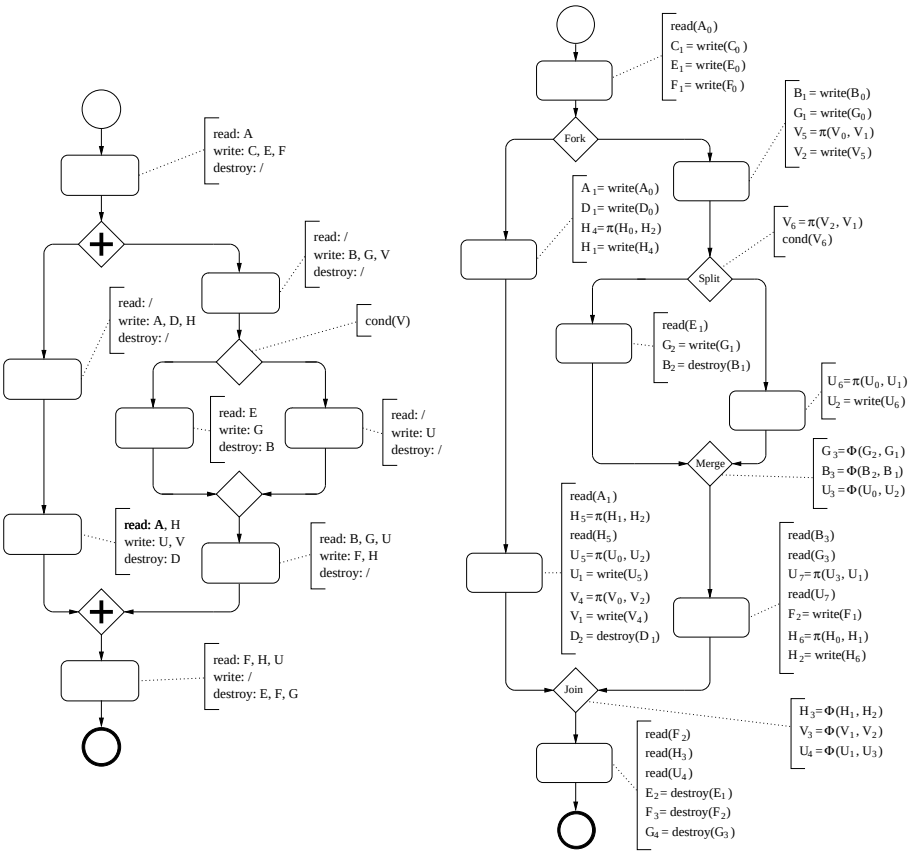

Abb. 1. Beispielprozess in BPMN-Notation (l.) und als Workflow-Graph (r.)

Abbildung 1 ist der erweiterte Workflow-Graph für den aus [5] übernommenen Beispielprozess dargestellt. Zum Vergleich bildet die linke Seite den Prozess auch in BPMN-Notation ab. In der BPMN-Darstellung sind die Prozessknoten mit Annotationen versehen, die in den Knoten gelesene, geschriebene oder gelöschte Daten in Form von Variablen (`A,B,...`) bezeichnen. Ferner wurde die bedingte Aufspaltung des Kontrollflusses mit der Verzweigungsbedingung versehen.

Im erweiterten Workflow-Graphen sind die Annotationen übernommen, nur dass diese nun im Format der *Concurrent Static Single Assignment Form (CSSA-Form)* [2] vorliegen. Zu diesem Zweck wurden die auf Daten operierenden Prozessaktivitäten auf vier Instruktionstypen abgebildet:

- $read(V_i)$ liest das Datum in Variable V_i,
- $cond(V_i)$ bestimmt den Wert einer Verzweigungsbedingung über Variable V_i,
- $V_i = write(V_j)$ überschreibt die alte Definition des Datums in Variable V_j mit einem neuen Datum und legt dieses in der Variablen V_i ab,
- $V_i = destroy(V_j)$ löscht das Datum in Variable V_j und setzt dadurch den Wert der Variablen V_i auf undefiniert.

Charakteristische Eigenschaft der *CSSA-Form* ist, dass jede Variable statisch einmal definiert ist, so dass für jede Variablendefinition durch die Instruktionen *write* oder *destroy* ein eigener Name eingeführt, und Variablenzugriffe entsprechend angepasst wurden (beispielsweise $G_1, \ldots, G_4$ für Variable `G`). Treffen mehrere Definitionen einer Variablen auf verschiedenen Pfaden des Kontrollflusses in einem Knoten zusammen, wurden spezielle Instruktionen mit wie folgt definierten *Φ-Funktionen* eingefügt, um die Definitionen zusammenzufassen:

Definition 2 (Φ-Funktion). *Eine Φ-Funktion für Variable V hat die Form $\Phi(V_1, \ldots, V_n)$, wobei die Operanden V_i den im Knoten der Funktion zusammenfließenden Definitionen von V entsprechen. Der Wert der Funktion ist der Operand V_i, der die zur Prozesslaufzeit tatsächlich, beziehungsweise als letztes, ausgeführte Instruktion mit einer Definition der Variablen V repräsentiert.*

Neben den Instruktionen mit *Φ-Funktionen* enthält die CSSA-Form weitere spezielle Instruktionen mit *π-Funktionen*, um Schreib-/Lese-Konflikte zwischen parallelen Prozessaktivitäten modellieren zu können:

Definition 3 (π-Funktion). *Eine π-Funktion für Variable V hat die Form $\pi(V_1, \ldots, V_n)$, wobei die Operanden V_i den im Knoten der Funktion konkurrierenden Definitionen von V entsprechen. Der Wert der Funktion ist der Operand V_i, der die zur Prozesslaufzeit letzte Definition von V repräsentiert.*

3 Datenflussinformation

Auf Grundlage der Repräsentation eines strukturierten Geschäftsprozesses durch erweiterte Workflow-Graphen können dann Informationen zum Datenfluss abgeleitet werden. Für die Bestimmung der sicheren und möglichen Fehler zu den in Tabelle 1 aufgeführten Fehlerarten werden Datenflussinformationen über die *fehlenden*, *gelöschten* und *definierenden Daten* des untersuchten Prozesses benötigt.

Als *fehlende Daten* werden Variablen bezeichnet, die uninitialisiert sind oder gelöscht wurden. Dabei kann unterschieden werden, ob eine Variable für mindestens ein Ausführungsszenario ein fehlendes Datum beschreibt, oder für alle möglichen Prozessausführungen. In unserem Beispiel aus Abbildung 1 entspricht die Variable A_0 immer einem fehlenden Datum, die Variable B_3 hingegen nur dann, falls die Instruktion $B_2 = destroy(B_1)$ ausgeführt wurde. Zur Darstellung der Datenflussinformation werden Wahrheitswerte genutzt, die angeben ob eine Variable ein fehlendes Datum beschreibt. Bedingt durch die Eigenschaft der CSSA-Form dass jede Variable statisch nur einmal definiert ist, können diese Werte den die Variablen definierenden Instruktionen zugewiesen werden:

Definition 4 (Fehlende Daten). *Für Instruktion s enthält $MISS^{MUST}(s)$ einen Wahrheitswert, der anzeigt ob die durch s definierte Variable auf allen Kontrollflusspfaden uninitialisiert/gelöscht ist und $MISS^{MAY}(s)$ einen Wert, ob die Variable auf mindestens einem Pfad uninitialisiert/gelöscht ist. Ist die Instruktion s nicht vorhanden, gilt $MISS^{MUST}(s) = MISS^{MAY}(s) = true$.*

Analog ergibt sich die Datenflussinformation zu *gelöschten Daten*, die angibt ob eine Variable im Prozess bereits gelöscht wurde:

Definition 5 (Gelöschte Daten). *Für Instruktion s enthält $DEL^{MUST}(s)$ einen Wahrheitswert, der anzeigt ob die durch s definierte Variable auf allen Kontrollflusspfaden gelöscht wurde und $DEL^{MAY}(s)$ einen Wahrheitswert, der anzeigt ob diese Variable auf mindestens einem Pfad gelöscht wurde. Ist die Instruktion s nicht vorhanden, gilt $DEL^{MUST}(s) = DEL^{MAY}(s) = false$.*

Die Datenflussinformation zu *definierenden Daten* entspricht hingegen der Menge von Daten, in Form von durch *write*-Instruktionen definierten Variablen, die den Wert einer Variablen in mindestens einer Prozessausführung festlegen (beispielsweise $\{H_1, H_2\}$ für Variable H_3 in Abbildung 1), oder in allen:

Definition 6 (Definierende Daten). *Für Instruktion s enthält die Menge $DATA^{MUST}(s)$ alle Daten, welche den Wert der durch s definierten Variablen auf allen Kontrollflusspfaden festlegen und die Menge $DATA^{MAY}(s)$ alle Daten, welche den Wert dieser Variablen auf mindestens einem Pfad festlegen. Ist die Instruktion s nicht vorhanden, gilt $DATA^{MUST}(s) = DATA^{MAY}(s) = \emptyset$.*

Um die so definierten Datenflussinformationen für einen Geschäftsprozess exakt bestimmen zu können, ist eine Analyse der parallel ausführbaren Prozessaktivitäten notwendig. Grundsätzlich ist eine solche Analyse Co-NP-schwer [1]. Daher verzichten wir auf die exakte Bestimmung und ermitteln stattdessen konservative Abschätzungen. Zu diesem Zweck wird das Verfahren der *statischen Datenflussanalyse* angewendet. Dieses erlaubt für die Charakterisierung eines Datenflussproblems durch ein System rekursiver Gleichungen eine Fixpunktlösung zu berechnen, die eine Abschätzung zur gesuchten Information bildet. Das Gleichungssystem zu den *definierenden Daten* ergibt sich beispielsweise wie folgt:

$$
\begin{array}{ll}
DATA^{MUST}(s) = \bigcap_{i\in\{1,\ldots,n\}} DATA^{MUST}(def(V_i)) & \text{für } s : V = \Phi(V_1, \ldots, V_n) \\
DATA^{MUST}(s) = \bigcap_{i\in\{1,\ldots,n\}} DATA^{MUST}(def(V_i)) & \text{für } s : V = \pi(V_1, \ldots, V_n) \\
DATA^{MAY}(s) = \bigcup_{i\in\{1,\ldots,n\}} DATA^{MAY}(def(V_i)) & \text{für } s : V = \Phi(V_1, \ldots, V_n) \\
DATA^{MAY}(s) = \bigcup_{i\in\{1,\ldots,n\}} DATA^{MAY}(def(V_i)) & \text{für } s : V = \pi(V_1, \ldots, V_n) \\
DATA^{MUST}(s) = DATA^{MAY}(s) = \{V_i\} & \text{für } s : V_i = write(V_j) \\
DATA^{MUST}(s) = DATA^{MAY}(s) = \emptyset & \text{sonst}
\end{array}
$$

Wie zu erkennen, werden darin jeder Instruktion s eines Prozesses Gleichungen $DATA^{MUST}(s)$ und $DATA^{MAY}(s)$ zugeordnet. Die Datenflussinformation für eine *write*-Instruktion s bildet gerade die Menge, die als einziges Element die durch s definierte Variable enthält. Für eine Instruktion mit Φ- oder π-Funktion ergeben sich die Mengen $DATA^{MUST}$ und $DATA^{MAY}$ als Schnitt beziehungsweise Vereinigung der Datenflussinformation zu den die Operanden definierenden Instruktionen (Instruktion $def(V_i)$ für Operand V_i). Da das Gleichungssystem über endlichen Mengen und monotonen Funktionen definiert ist, ist dessen Konvergenz sichergestellt. Für die Fixpunktbestimmung kann dann ein Algorithmus zur Datenflussanalyse auf CSSA-Form genutzt werden [2, 3], der diesen in höchstens quadratischer Zeit bezüglich der Anzahl von Prozessinstruktionen berechnet. Aufgrund des beschränkten Platzes wird hier auf die Angabe der Fixpunktgleichungen zu *fehlenden* und *gelöschten Daten* verzichtet.

Fehlerart	**Fehlermenge**
Fehlende Daten (sichere Fehler)	$\{\ s \mid (\ s:\ V_i = destroy(V_j)\ \vee\ s:\ read(V_j)$ $\vee\ s:\ cond(V_j)\ )\ \wedge\ MISS^{MUST}(def(V_j)) = true\ \}$
Fehlende Daten (mögliche Fehler)	$\{\ s \mid (\ s:\ V_i = destroy(V_j)\ \vee\ s:\ read(V_j)$ $\vee\ s:\ cond(V_j)\ )\ \wedge\ MISS^{MAY}(def(V_j)) = true\ \}$
Redundante oder überschriebene Daten (sichere Fehler)	$\{\ s \mid s:\ V_i = write(V_j)$ $\wedge\ V_i \notin (\ \bigcup_{s':\ read(V_k)} DATA^{MAY}(def(V_k))$ $\cup\ \bigcup_{s':\ cond(V_k)} DATA^{MAY}(def(V_k))\ )\ \}$
Redundante oder überschriebene Daten (mögliche Fehler)	$\{\ s \mid s:\ V_i = write(V_j)$ $\wedge\ V_i \notin (\ \bigcup_{\substack{s':\ read(V_k) \\ s'\ postdominiert\ s}} DATA^{MUST}(def(V_k))$ $\cup\ \bigcup_{\substack{s':\ cond(V_k) \\ s'\ postdominiert\ s}} DATA^{MUST}(def(V_k))\ )\ \}$
Inkonsistente Daten (mögliche Fehler)	$\{\ s \mid (\ s:\ V_i = destroy(V_j)\ \vee\ s:\ read(V_j)$ $\vee\ s:\ cond(V_j)\ \vee\ s:\ V_i = write(V_j)\ )$ $\wedge\ V_j\ ist\ definiert\ durch\ \pi{-}Funktion\ mit\ mehr$ $als\ einem\ Operanden\ \}$
Nicht gelöschte Daten (sichere Fehler)	$\{\ s \mid s:\ V_i = write(V_j)$ $\wedge\ V_i \notin (\ \bigcup_{s':\ V_l = destroy(V_k)} DATA^{MAY}(def(V_k))$ $\cup\ \bigcup_{s':\ V_l = write(V_k)} DATA^{MAY}(def(V_k))\ )\ \}$
Nicht gelöschte Daten (mögliche Fehler)	$\{\ s \mid s:\ V_i = write(V_j)$ $\wedge\ V_i \notin (\ \bigcup_{\substack{s':\ V_l = destroy(V_k) \\ s'\ postdominiert\ s}} DATA^{MAY}(def(V_k))$ $\cup\ \bigcup_{\substack{s':\ V_l = write(V_k) \\ s'\ postdominiert\ s}} DATA^{MAY}(def(V_k))\ )\ \}$
Doppelt gelöschte Daten (sichere Fehler)	$\{\ s \mid s:\ V_i = destroy(V_j)$ $\wedge\ DEL^{MUST}(def(V_j)) = true\ \}$
Doppelt gelöschte Daten (mögliche Fehler)	$\{\ s \mid s:\ V_i = destroy(V_j)$ $\wedge\ DEL^{MAY}(def(V_j)) = true\ \}$
Zu spät gelöschte Daten (mögliche Fehler)	$\{\ s \mid (\ s:\ read(V_i)\ \vee\ s:\ cond(V_i)\ )$ $\wedge\ \nexists\ s':\ V_j = destroy(V_i)\ in\ Basisblock\ von\ s$ $\wedge\ V_i \notin (\ \bigcup_{\substack{s'':\ read(V_k)\ \wedge\ s'' \neq s \\ s''\ postdominiert\ s}} DATA^{MUST}(def(V_k))$ $\cup\ \bigcup_{\substack{s'':\ cond(V_k)\ \wedge\ s'' \neq s \\ s''\ postdominiert\ s}} DATA^{MUST}(def(V_k))\ )\ \}$

Tabelle 2. Fehlermengen (s, s', s'' bezeichnen Instruktionen des Prozesses)

4 Abschätzung sicherer und möglicher Fehler

Nachdem Abschätzungen für die fehlenden, gelöschten und definierenden Daten für einen Prozess bestimmt wurden, können dessen *sichere* und *mögliche Datenflussfehler* abgeleitet werden. Zu diesem Zweck definieren wir für jeden der in Tabelle 1 aufgeführten Fehler zugehörige *Fehlermengen* (vergleiche Definition 1): Die *Menge sicherer Fehler* enthält Instruktionen, die den Fehler sicher und in allen Prozessausführungen aufweisen, ist also eine Teilmenge der tatsächlichen Fehler. Die *Menge möglicher Fehler* enthält Instruktionen, die den Fehler in einer Ausführung aufweisen können, ist also eine Obermenge der tatsächlichen Fehler. In Tabelle 2 sind die *Fehlermengen* dargestellt. Wie zu erkennen, konnten für alle Fehler, bis auf *Inkonsistente Daten* und *Zu spät gelöschte Daten*, sowohl die Menge der sicheren, als auch die Menge der möglichen Fehler angegeben werden.

Fehlerart	**sichere Fehler**	**mögliche Fehler**
Fehlende Daten	A_0	A_0, B_3, U_7, U_4
Redundante oder überschriebene Daten	C_1, F_1, D_1 davon redundant: C_1, D_1	$C_1, F_1, D_1, H_1, H_2, U_1, U_2, V_1, V_2, G_1,$ G_2, E_1, B_1 davon redundant: C_1, D_1, G_2, E_1, B_1
Inkonsistente Daten	/	$H_4, H_5, H_6, U_5, U_6, U_7, V_4, V_5, V_6$
Nicht gelöschte Daten	C_1, A_1	$C_1, A_1, H_1, H_2, U_1, U_2, V_1, V_2, B_1$
Doppelt gelöschte Daten	$\emptyset$	$\emptyset$
Zu spät gelöschte Daten	/	$H_3, H_5, A_1, B_3, E_1, G_3, U_4, U_7, V_6, A_0$

Tabelle 3. Abgeleitete Fehler für den Beispielprozess aus Abbildung 1

Die Menge *Fehlende Daten (sichere Fehler)* enthält Instruktionen s der Instruktionstypen $V_i = \mathit{destroy}(V_j)$, $\mathit{read}(V_j)$ und $\mathit{cond}(V_j)$, die für alle Kontrollflusspfade auf ein fehlendes Datum V_j zugreifen. Dazu wird überprüft, ob $MISS^{MUST}$ für die V_j definierende Instruktion $def(V_j)$ dem Wahrheitswert *true* entspricht. Die Menge *Fehlende Daten (mögliche Fehler)* umfasst Instruktionen, die für einen Pfad auf ein fehlendes Datum zugreifen können, und wurde analog über die Wahrheitswerte in $MISS^{MAY}$ definiert. Auf gleiche Weise ergeben sich die Fehlermengen *Doppelt gelöschte Daten*, nur das diese *destroy*-Instruktionen enthalten und als Datenflussinformation DEL^{MUST}, DEL^{MAY} genutzt wird.

Die Fehlermenge *Redundante oder überschriebene Daten (sichere Fehler)* umfasst *write*-Instruktionen, die Daten schreiben auf die im Prozess nie lesend, also durch Instruktionen *read* oder *cond* zugegriffen wird. Zu diesem Zweck wird die Menge aller im Prozess auf mindestens einem Kontrollflusspfad gelesenen Daten bestimmt, als Vereinigung der Mengen $DATA^{MAY}(def(V_k))$ für alle durch Instruktionen $s' : \mathit{read}(V_k)$ und $s' : \mathit{cond}(V_k)$ gelesenen Variablen V_k. Ist eine durch Instruktion $s : V_i = \mathit{write}(V_j)$ definierte Variable V_i kein Element dieser Menge, wird nie lesend auf V_i zugegriffen und die Instruktion erfüllt den Fehler. Die Menge *Redundante oder überschriebene Daten (mögliche Fehler)* ergibt sich analog, nur dass nun überprüft wird, ob eine durch Instruktion $s : V_i = \mathit{write}(V_j)$ definierte Variable V_i nicht auf allen Kontrollflusspfaden gelesen wird, unter Ausnutzung der Datenflussinformation $DATA^{MUST}$ und der Postdominanz-Relation.

In [5] wird zusätzlich eine Unterscheidung zwischen *redundanten* und *überschriebenen Daten* vorgenommen. Um auch eine solche Unterscheidung durchzuführen, kann die Menge $\bigcup_{s:\ V_l = \mathit{write}(V_k)} DATA^{MAY}(def(V_k))$ aller auf mindestens einem Kontrollflusspfad überschriebenen Variablen bestimmt werden. Ist die durch eine *write*-Instruktion definierte Variable kein Element dieser Menge, wird sie in keinem Fall überschrieben und muss daher redundant sein.

Die Menge *Inkonsistente Daten (mögliche Fehler)* umfasst Instruktionen, die auf ein Datum zugreifen, auf das auch eine parallel ausgeführte Instruktion schreibend oder löschend zugreift. Da solche Schreib-/Lese-Konflikte im erweiterten Workflow-Graphen bereits mittels π-Funktionen gekennzeichnet sind, ergibt sich die Fehlermenge als Menge aller Instruktionen, die auf eine durch π-Funktion mit mehr als einem Operanden definierte Variable zugreifen. In ähnlicher Weise zu den hier näher erläuterten Fehlermengen ergeben sich dann auch die Mengen zu den Datenflussfehlern *Nicht gelöschte Daten* und *Zu spät gelöschte Daten*.

Eine auf diese Weise durchgeführte Abschätzung für die Datenflussfehler im Beispielprozess aus Abbildung 1 ist in Tabelle 3 dargestellt. Aus Gründen der Übersichtlichkeit wurden nicht Instruktionen, sondern die zugehörigen Variablen angegeben. So enthalten die Mengen zum Fehler *Fehlende Daten* Variablen, die fehlende Daten beschreiben und auf die durch eine Instruktion zugegriffen wird, und die Mengen zum Fehler *Nicht gelöschte Daten* Variablen, die durch eine Instruktion geschrieben aber später nicht gelöscht werden. Die Fehlermengen beschreiben offenbar recht gut die im Prozess enthaltenen Fehler. Die Mengen sicherer Fehler repräsentieren so nur tatsächlich immer im Prozess auftretende Fehler. Die Mengen möglicher Fehler repräsentieren nahezu nur Fehler, die für mindestens eine Prozessausführung auch tatsächlich auftreten. Lediglich bei U_4 in der Fehlermenge *Fehlende Daten* und G_2 in der Fehlermenge *Redundante oder überschriebene Daten* handelt es sich um keine tatsächlichen Fehler.

5 Zusammenfassung

In der vorliegenden Arbeit haben wir eine Methode vorgestellt, die es für strukturierte Geschäftsprozesse erlaubt, Abschätzungen für Datenflussfehler effizient zu bestimmen. Zu diesem Zweck werden erweiterte Workflow-Graphen als Prozessmodell genutzt, die die Durchführung einer Datenflussanalyse begünstigen. Basierend auf den durch Datenflussanalyse ableitbaren Informationen zu fehlenden, gelöschten und definierenden Daten konnten dann Fehlermengen für sichere und mögliche Datenflussfehler angegeben werden. Die Methode bietet neben ihrer Effizienz den weiteren Vorteil, dass alle in einem Prozess enthaltenen Datenflussfehler in einem Schritt abgeschätzt werden können. Im Gegensatz dazu liefert ein Ansatz auf Grundlage einer Modellprüfung immer nur einen Fehler als Gegenbeispiel zur untersuchten Eigenschaft. Zukünftige Arbeiten sollen die praktische Relevanz dieser Vorteile anhand von Fallstudien weiter untersuchen.

Literatur

[1] Callahan, David ; Subhlok, Jaspal: Static Analysis of Low-level Synchronization. In: *ACM SIGPLAN Notices* 24 (1989), Nr. 1, S. 100–111

[2] Lee, Jaejin ; Midkiff, Samuel P. ; Padua, David A.: Concurrent Static Single Assignment Form and Constant Propagation for Explicitly Parallel Programs. In: *Languages and Compilers for Parallel Computing, 10th International Workshop, LCPC'97, Minneapolis, Minnesota, USA, August 7-9, 1997, Proceedings*, Springer, 1998 (LNCS 1366), S. 114–130

[3] Moser, Simon ; Martens, Axel ; Görlach, Katharina ; Amme, Wolfram ; Godlinski, Artur: Advanced Verification of Distributed WS-BPEL Business Processes Incorporating CSSA-based Data Flow Analysis. In: *2007 IEEE International Conference on Services Computing (SCC 2007), 9-13 July 2007, Salt Lake City, Utah, USA*, IEEE Computer Society Press, 2007, S. 98–105

[4] Sadiq, Wasim ; Orlowska, Maria E.: Analyzing Process Models Using Graph Reduction Techniques. In: *Information Systems* 25 (2000), Nr. 2, S. 117–134

[5] Trčka, Nikola ; van der Aalst, Wil M. ; Sidorova, Natalia: Data-Flow Anti-patterns: Discovering Data-Flow Errors in Workflows. In: *Advanced Information Systems Engineering, 21st International Conference, CAiSE 2009, Amsterdam, The Netherlands, June 8-12, 2009, Proceedings*, Springer, 2009 (LNCS 5565), S. 425–439