Der Petrinetz-Würfel im Petrinetz-Kern

Michael Weber*

Humboldt-Universität zu Berlin, Institut für Informatik

1 Motivation

Eine Möglichkeit, ein theoretisches Konzept zu validieren, ist seine Implementierung in einem Werkzeug. In diesem Papier skizzieren wir die Implementierung des Petrinetz-Würfels mit Hilfe des Petrinetz-Kerns.

Mit dem Petrinetz-Würfel [KW98] und dem Petrinetz-Kern [HHK+98, KW99] wurden zwei Konzepte entwickelt, die jeweils einen allgemeinen Petrinetz-theoretischen Ansatz verfolgen. Der Petrinetz-Kern dient als Infrastruktur zum Bau von Werkzeugen für Petrinetze beliebiger Typen. Und der Petrinetz-Würfel strukturiert und klassifiziert die Welt der klassischen Petrinetze. Beiden gemeinsam ist die Verallgemeinerung von Petrinetz-Typen. Unter einem Petrinetz-Typ verstehen wir die Festlegung der zulässigen Markierungen, der zulässigen Kanteninschriften sowie des Schaltverhaltens für Netze diesen Typs. Beide Verallgemeinerungen unterscheiden sich jedoch grundlegend voneinander.

Das Ziel des Petrinetz-Würfels ist die Klassifizierung von Petrinetz-Typen derart, dass ihre Unterschiede und Gemeinsamkeiten hervortreten. Insbesondere die intuitive (klassische) Schaltregel kann mit diesem Konzept für alle klassischen Petrinetz-Typen gemeinsam definiert werden. Außerdem genügen 3 voneinander unabhängige Aspekte, um die klassischen Petrinetz-Typen zu beschreiben. Ein besonderer Nutzen des Petrinetz-Würfels besteht darin, dass Techniken zu Petrinetzen, die nur von ganz bestimmten Aspekten abhängen, leicht von einem Petrinetz-Typ auf einen anderen übertragen werden können, wenn sich beide in den bestimmten Aspekten nicht unterscheiden.

Der Petrinetz-Kern hat zum Ziel, den Bau von Petrinetz-Werkzeugen zu erleichtern. Dazu stellt er Standardfunktionen wie das Einfügen von Elementen in ein Petrinetz oder bestimmte Anfragen an Netze zur Verfügung. Damit der Petrinetz-Kern den Bau von Werkzeugen für beliebige Petrinetz-Typen unterstützt, besitzt er eine Schnittstelle zur Definition eines Petrinetz-Typs. Mit ihr können alle Petrinetz-Typen beschrieben werden, die auf klassische Petrinetz-Typen aufbauen und Stellen, Transitionen sowie Kanten zwischen Stellen und Transitionen bzw. umgekehrt zulassen.[1] Im Gegensatz zum Petri-

*Email: mweber@informatik.hu-berlin.de

[1] Eine zukünftige Version des Petrinetz-Kerns wird auch Petrinetz-Typen ermöglichen, die über klassische Petrinetz-Typen weit hinausgehen und z. B. Signalkanten zwischen Transitionen zulassen.

netz-Würfel werden Petrinetz-Typen im Petrinetz-Kern so beschrieben, wie es intuitiv erscheint, ohne Abhängigkeiten aufzulösen.

Im folgenden werden wir eine weitere Schnittstelle des Petrinetz-Kerns vorstellen, so dass Petrinetz-Typen wie im Petrinetz-Würfel beschrieben werden können. Zunächst werden wir beide Konzepte separat einführen; anschließend (in Abschnitt 4) wird ihre Verbindung im Petrinetz-Kern dargestellt.

2 Petrinetz-Würfel

Der Petrinetz-Würfel wurde eingeführt, um Petrinetz-Typen zu strukturieren und zu klassifizieren. Er verfolgt einen didaktischen Ansatz: die Unterschiede und Gemeinsamkeiten der verschiedenen Petrinetz-Typen werden deutlich.

Die zentrale Idee des Petrinetz-Würfels [KW98] ist, dass drei Aspekte ausreichen, um klassische Petrinetz-Typen vollständig zu beschreiben. Diese Aspekte sind zudem voneinander unabhängig und werden deshalb als *Dimensionen* bezeichnet. Ausgangspunkt der Überlegungen war, dass ein Petrinetz jeglichen Typs Stellen, Transitionen und Kanten enthält. Die Unterschiede der klassischen Petrinetz-Typen betreffen lediglich die Art der zulässigen Token, die Struktur, mit der Token zu Markierungen zusammengesetzt werden und den zulässigen Markenfluss, der über eine Kante beim Schalten einer Transition fließen kann.

Mit der ersten Dimension – der Tokenstruktur – wird festgelegt, welche Token in einem Netz des betreffenden Typs auftreten dürfen. An dieser Stelle genügt es zu wissen, dass diese Dimension durch wenigstens zwei Werte instantiiert werden kann: einerseits durch eine Klasse mit der einelementigen Menge, die das *black token* enthält, andererseits durch eine Klasse, die beliebige Mengen (*high-level* Token) enthält. Für *low-level* Netze wie B/E- bzw. S/T-Netze wird die erste Klasse verwendet. Und die zweite Klasse findet Verwendung in *high-level* Netzen. Selbstverständlich ist es denkbar, noch weitere Klassen als Werte dieser Dimension zuzulassen. Sie können beispielsweise verschiedene Datentyptechniken zur Strukturierung verwenden.

Die zweite Dimension des Petrinetz-Würfels beschreibt, wie Token zu Markierungen auf Stellen zusammengefasst werden. Markierungen sind meistens Mengen, Multimengen oder Sequenzen. Die mathematische Technik der Freien Algebren (siehe [EM85] für Details) ermöglicht es, die verschiedenen Petrinetz-Typen nach der Eigenschaft der Additionsoperation für Token zu Markierungen zu klassifizieren, ohne auf konkrete Token einzugehen. Mengen korrespondieren mit einer idempotenten Operation, d. h., das Hinzufügen eines bereits vorhandenen gleichen Elementes verändert die Menge nicht. Eine derartige Markierung kann also nur ein Exemplar eines bestimmten Tokens aufnehmen. Diese Eigenschaft wird beispielsweise in B/E-Netzen und Prädikat/Ereignis-Systemen gefordert. Eine kommutative Additionsoperation korrespondiert mit Multimengen; die Reihenfolge einzelner Operationen spielt keine Rolle, wogegen für eine nicht kommutative Additionsoperation die Reihenfolge wichtig ist. Letztere korrespondiert mit Sequenzen. Markierungen auf Stellen von S/T-Netzen und *Coloured Petri Nets* können als Multimengen von Token aufgefasst werden. Markierungen auf Stellen von FIFO-Netzen dagegen sind als Sequenzen zu betrachten, da die Reihenfolge des Erscheinens eines Tokens auf einer Stelle wichtig wird.

(Konkrete) Instanzen der beiden ersten Dimensionen konstituieren den Wertebereich der Markierungen für Netze des entsprechenden Typs. Außerdem ist bereits die (intuitive) klassische Schaltregel für Petrinetze definierbar, die dabei nur einmal für alle Petrinetz-Typen festgelegt werden muss [KW98].

Die dritte Dimension des Petrinetz-Würfels schränkt lediglich die Menge der möglichen Netze bestimmter Petrinetz-Typen ein. Beispielsweise wird bisher nicht verhindert, dass Netze mit *black token* als Tokenstruktur und Multimengen als Markierungsstruktur als Kanteninschriften 0 erhalten. Für S/T-Netze ist dies zumindest ungewöhnlich. Außerdem kann bisher keine Unterscheidung zwischen *Coloured Petri Nets* und Algebraischen Netzen [Rei91] gemacht werden. Beide erhalten *high-level* Token und Multimengen als Markierungen. Für Algebraische Netze wird jedoch ein festes Kantengewicht gefordert, d. h., bei jedem Schalten der entsprechenden Transition fließt die gleiche Anzahl von Token über die Kante. Für *Coloured Petri Nets* gilt diese Einschränkung nicht. Die dritte Dimension legt deshalb die Bedingungen des Kantenflusses fest.

Die drei unabhängigen Dimensionen spannen einen Raum auf, in dem die klassischen Petrinetz-Typen jeweils einen Punkt belegen. Inwieweit es möglich ist, weitere Dimensionen zu definieren, mit denen auch nicht klassische Petrinetz-Typen wie Zeit-, stochastische oder Fuzzy-Petrinetze beschrieben werden können, ist Gegenstand weiterer Forschungen.

3 Der Petrinetz-Kern

Während der Petrinetz-Würfel als theoretisch didaktisches Konzept entwickelt wurde, bei dem Übersichtlichkeit und Eleganz im Vordergrund standen, ist der Petrinetz-Kern mit den Prämissen von Vollständigkeit und praktischer Handhabbarkeit entwickelt worden. Der Petrinetz-Kern ist eine Infrastruktur zum Bau von Petrinetz-Werkzeugen. Als solche erspart er einem Werkzeugentwickler die Implementierung von Standardfunktionen für Petrinetze wie Laden oder Speichern von Netzen, Zugriff auf und Modifizierung der Netzstruktur. Außerdem muss der Entwickler keine graphische Benutzerschnittstelle programmieren, sondern kann sich auf die Implementierung seines Petrinetz-Algorithmus konzentrieren. Unter einem Petrinetz-Algorithmus verstehen wir einen Algorithmus zur Analyse, Simulation, Verifikation o. ä. von Petrinetzen.

Der Petrinetz-Kern wird verwendet, um Petrinetze beliebiger Typen zu verwalten. Der in einem Werkzeug schließlich konkret verwendete Petrinetz-Typ wird über einen Parameter dem Petrinetz-Kern bekannt gemacht. Ein solcher Parameter beschreibt das Spezifische eines Petrinetz-Typs (kurz: PN-Typ-Spezifik) so, wie es einem Werkzeugentwickler intuitiv erscheint: Stellen können Markierungen aufnehmen, Transitionen (insbesondere in *high-level* Netzen) können in verschiedenen Modi schalten und Kanten lassen Inschriften zu. Darüber hinaus können Erweiterungen sowohl lokal zu einzelnen Petrinetz-Elementen (Zeitannotationen, *transition guards*, Codefragmente etc.) als auch global zu Petrinetzen (Deklarationen, Signatur etc.) definiert werden. Mit diesem Konzept ist es möglich, jeden beliebigen Petrinetz-Typ als Parameter des Petrinetz-Kerns zu implementieren.

Die Implementierung einer PN-Typ-Spezifik muss eine bestimmte Schnittstelle erfüllen. Transformationsfunktionen sorgen für ein Austauschformat, so dass Editor sowie

Speicher- und Ladeoperation Zugang zu den entsprechenden Werten haben. Es wird jeweils eine Funktion gefordert, die explizit aufgerufen werden kann, um die Syntax eines entsprechenden Wertes im Netzkontext zu überprüfen. Für Markierungen müssen außerdem verschiedene Operationen (z. B. Addieren und Subtrahieren anderer Markierungen) definiert werden, und auch Modi müssen eine erweiterte Schnittstelle erfüllen. Überdies wird der Modus eingesetzt, um aus dem Term einer Kanteninschrift eine Markierung zu berechnen. Beispielsweise werden in algebraischen Netzen eventuell vorhandene Variablen an konkrete Werte des aktuellen Modus gebunden und entsprechende Operationen im Term ausgeführt.

Wir sehen, dass dieses Konzept von dem oben beschriebenen für den Petrinetz-Würfel abweicht, weil es hier um rasche Implementierung und praktische Anschaulichkeit und weniger um Wiederverwendung von Programmcode für Petrinetz-Typen geht.

4 Konnexion

Zuweilen ist es jedoch wünschenswert, Teile eines Petrinetz-Typs in einem anderen wiederzuverwenden. Beispielsweise könnte für ein *high-level* Netz das Verhalten von B/E-Systemen gewünscht sein, oder ein vorhandener Petrinetz-Typ soll um Techniken zur Beschreibung des Datentyps ergänzt werden. Mit der Schnittstelle CubeInterface steht nun eine solche Möglichkeit für den Petrinetz-Kern zur Verfügung. Außerdem sollte gezeigt werden, dass ein theoretisch didaktisches Konzept wie der Petrinetz-Würfel ohne größeren Aufwand implementiert werden kann.

Für die eben genannten Beispiele muss im Petrinetz-Würfel jeweils nur der Wert einer Dimension geändert werden. Für das erste Beispiel bleibt die Tokenstruktur unverändert, und die Markierungsstruktur wird von Multimenge zu Menge geändert. Für das zweite Beispiel wird der Wert der Tokenstruktur verändert. Um diese Beispiele mit einer vorhandenen PN-Typ-Spezifik des Petrinetz-Kerns umzusetzen, ist vergleichsweise viel Aufwand nötig, da die Abhängigkeiten ihrer verschiedenen Teile zu beachten sind. Schon für das erste Beispiel müssen Änderungen in der Beschreibung für Markierungen und für Modi (dort in der Termauswertungsfunktion) vorgenommen werden.

Wir gehen vom Petrinetz-Kern aus und werden die Implementierung einer PN-Typ-Spezifik skizzieren, die selbst Parameter aufnehmen kann. Wir nennen diese PN-Typ-Spezifik CubeSpecification; die Parameter, die sie aufnehmen kann, fassen wir mit CubeInterface zusammen. Je Dimension des Petrinetz-Würfels wird ein Parameter für CubeSpecification gefordert. CubeInterface umfasst vorerst zwei Dimensionen des Petrinetz-Würfels – die Tokenstruktur und die Markierungsstruktur. Die Bedingungen des Kantenflusses wurden noch nicht implementiert; eine Implementierung sollte jedoch unproblematisch sein.

CubeSpecification erzeugt nun aus den konkreten Parametern für CubeInterface eine PN-Typ-Spezifik des Petrinetz-Kerns. Eine Markierung besteht dann auch implementationstechnisch aus Token. Ihre Operationen werden durch den entsprechenden Parameter aus CubeInterface festgelegt.

Die Tokenstruktur wird in CubeInterface durch TokenInterface implementiert. Da die Definition von *high-level* Token unterstützt werden soll, wird hier eine umfangreiche

Schnittstelle zur Verfügung gestellt. Sie ermöglicht die Definition von Sorten und einer Deklaration. Außerdem können Schaltmodi und Terme über Token beschrieben werden. Besondere Anforderungen haben Schaltmodi und die Beschreibung für Token zu erfüllen. Schaltmodi müssen aus Termen Token erzeugen können, und Token müssen mit anderen Token verglichen werden. Aus diesen Angaben zur Beschreibung von Token kann CubeSpecification Strukturen erzeugen, mit denen der Petrinetz-Kern umgehen kann. Ein Problem stellen lediglich Kanteninschriften dar: im allgemeinen werden an Kanten Terme über Markierungen notiert. Es war also nötig, bei der Implementation Terme über Markierungen in eine Struktur von Termen über Token zu zerlegen.

Die zweite Dimension des Petrinetz-Würfels – die Markierungsstruktur – wird in CubeInterface durch Collection implementiert. Aufbauend auf der Vergleichsfunktion der einzelnen Elemente wird eine Struktur implementiert, die die Reihenfolge des Hinzufügens beachtet (Sequenz), Elemente mehrfach enthalten kann (Multimenge) oder jedes Element höchstens einmal enthält (Menge). Funktionen von Collection sind u. a. das Hinzufügen eines Elementes bzw. einer gesamten Collection oder die Überprüfung auf das Enthaltensein einer Collection bzw. auf Leerheit.

Mit der Schnittstelle CubeInterface wird die Menge der möglichen Petrinetz-Typen auf jene beschränkt, die auch mit dem Petrinetz-Würfel abgedeckt werden. Der Entwicklung des Petrinetz-Würfels folgend, wird auch CubeInterface weitere Dimensionen bzw. Parameter erhalten.

Im folgenden zeigen wir an Beispielen, wie die neue Schnittstelle im Petrinetz-Kern verwendet wird. Im Petrinetz-Kern wird eine Anwendungsfunktion `function` mit einer PN-Typ-Spezifik `Specification` zu einer Anwendung verbunden:

```
Build_Application(Specification, function)
```

Damit wird der Editor des Petrinetz-Kerns gestartet. Ein Netz der angegebenen PN-Typ-Spezifik kann editiert werden. Das Drücken eines entsprechenden Buttons löst die Funktion `function` angewendet auf das aktuelle Netz aus.

Wir nehmen an, dass wir einige konkrete Parameter für CubeInterface implementiert haben. Für TokenInterface seien `BlackTokenInterface`, das ein *black token* beschreibt, und `HLTokenInterface`, das *high-level* Token beschreibt, implementiert. Als Collection seien `Set` für Mengen und `MultiSet` für Multimengen implementiert. Mit der PN-Typ-Spezifik CubeSpecification und einer generischen Funktion `test`, die höchstens Funktionalität von CubeSpecification ausnutzt, stehen nun die folgenden Anwendungen zur Verfügung:

```
Build_Application(CubeSpecification(BlackTokenInterface, Set), test)
Build_Application(CubeSpecification(BlackTokenInterface, MultiSet), test)
Build_Application(CubeSpecification(HLTokenInterface, Set), test)
Build_Application(CubeSpecification(HLTokenInterface, MultiSet), test)
```

Wiederum startet der Editor des Petrinetz-Kerns, der nun Netze aus der Kombination der jeweiligen Werte für Dimensionen des Petrinetz-Würfels editieren und „testen“ lässt.

5 Zusammenfassung

Mit der Schnittstelle CubeInterface ist ein theoretisch didaktisches Konzept in den Petrinetz-Kern integriert worden. Der Editor des Petrinetz-Kerns stellt ohne weiteres eine graphische Benutzerschnittstelle zum Modellieren von Petrinetzen zur Verfügung, deren Typen ihrer Beschreibung im Petrinetz-Würfel folgen. Durch die generische Schaltregel des Petrinetz-Würfels ist es insbesondere möglich, einen generischen Simulator für Netze, deren Typ CubeInterface implementiert, zu programmieren. Eine derartige Anwendung wird demnächst entwickelt.

Die Implementierung von CubeInterface durch die PN-Typ-Spezifik CubeSpecification ist gewiss nicht effizient. Effizienz ist jedoch auch nicht das vordringliche Ziel des Petrinetz-Kerns. Vielmehr soll mit CubeInterface der Rahmen erweitert werden, in dem (insbesondere neue) Petrinetz-Algorithmen erprobt werden können.

Literatur

[EM85] Ehrig, Hartmut und Bernd Mahr: *Fundamentals of Algebraic Specifications 1, Equations and Initial Semantics*, Bd. 6 von *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag. 1985.

[HHK+98] Hauptmann, Jens; Bodo Hohberg; Ekkart Kindler; Ines Schwenzer und Michael Weber: Der Petrinetz-Kern – Dokumentation der Anwendungs-Schnittstelle. *Informatik-Berichte 98*, Humboldt-Universität zu Berlin. Febr. 1998.

[KW98] Kindler, Ekkart und Michael Weber: The Dimensions of Petri Nets: The Petri Net Cube. *Bulletin of EATCS 66*:155–166. Okt. 1998.

[KW99] Kindler, Ekkart und Michael Weber: *The Petri Net Kernel. Documentation of the Application Interface. PNK Version 2.0.* Humboldt-Universität zu Berlin, Institut für Informatik. `http://www.informatik.hu-berlin.de/~kindler/PN-Kern/`. Jan. 1999.

[Rei91] Reisig, Wolfgang: Petri Nets and Algebraic Specifications. *Theoretical Computer Science 80*:1–34. Mai 1991.