Relationenalgebraische Analyse von Petri-Netzen: Konzepte und Implementierung

Alexander Fronk
Software-Technologie
Universität Dortmund
44221 Dortmund

Jörg Pleumann
Software-Technologie
Universität Dortmund
44221 Dortmund

Zusammenfassung

Unser Ansatz zur Analyse von Petri-Netzen basiert auf der Relationenalgebra und bietet einen neuartigen Ansatz, Petri-Netze formal zu behandeln. Der Relationenkalkül wurde in einer von uns in Zusammenarbeit mit der Universität Kiel entwickelten objektorientierten Java-Bibliothek, KURE*, operationalisiert und erlaubt damit seine Verwendung in beliebigen Werkzeugen, deren Anwendungsgebiet einer relationalen Modellierung zugänglich ist. Wir demonstrieren den Einsatz der Bibliothek am Beispiel der Erreichbarkeitsanalyse von Stellen/Transitionssystemen und zeigen damit sowohl ihre Verwendung im Allgemeinen als auch einen relationalen Ansatz zur Analyse von Petri-Netzen im Besonderen. Diesen Ansatz haben wir in einem CASE-Werkzeug,* PETRA*, implementiert.*

1. Einleitung und Motivation relationaler Analyseverfahren

Die Modellierung nebenläufiger Systeme mit Petri-Netzen ermöglicht ihre Analyse durch Verfahren, die auf statische und dynamische Aspekte des Netzes abzielen. Während es statischen Aspekten genügt, allein den zugrunde liegenden Netz-Graphen zu betrachten, tragen dynamische Aspekte insbesondere der Menge erreichbarer Markierungen Rechnung, da sie den Zustandsraum des modellierten Systems reflektieren.

Die Erreichbarkeitsanalyse von Petri-Netzen gibt Antworten etwa auf die folgenden Fragen: Ist eine Markierung M von einer Markierung N aus erreichbar? Welches ist die Menge aller von einer erreichbaren Markierung erreichbaren Markierungen? Gibt es Heimatmarkierungen und welche sind dies? Gibt es Paare erreichbarer Markierungen, die auf einer Menge von Stellen übereinstimmen?

Die lineare Algebra sieht hier einen ersten Zugang vor: Eine Markierung M' ist von M aus erreichbar, falls das Gleichungssystem $C * x = M' - M$ eine Lösung in x hat. Dabei ist C die Inzidenzmatrix des betrachteten Netzes. Das Lösen dieses Systems ist jedoch nicht hinreichend, den Erreichbarkeitsgraphen zu charakterisieren (vgl. [7]). Hierzu müssen erreichbare Markierungen ermittelt und als Knoten eines Graphen verwaltet werden, dessen Kanten das Schalten einer aktivierten Transition modellieren.

Für viele Analyseverfahren ist es notwendig, diesen Erreichbarkeitsgraphen zu konstruieren, um etwa die letzten beiden der oben gestellten Fragen beantworten zu können. Im Falle unendlich vieler Markierungen ist der Erreichbarkeitsgraph jedoch unendlich. In der Literatur finden sich daher viele Verfahren, die spezielle Aussagen und Erkenntnisse über Netze heranziehen, um den Erreichbarkeitsgraphen in solchen Fällen zu approximieren. Dienlich sind beispielsweise Unfolding-Techniken [8], die Simulation des Netzes oder das Berechnen eines Überdeckbarkeitsgraphen [14].

Die Berechnung des Erreichbarkeitsgraphen erfordert also mindestens zwei Schritte, nämlich das Ermitteln einer Folgemarkierung und das Codieren der Information, welche Markierung Folgemarkierung welcher anderen ist. Da sowohl ein Petri-Netz als auch sein Erreichbarkeitsgraph Graphen sind, liegt die Überlegung nahe, beide mit Hilfe einer einzigen Datenstruktur zu repräsentieren und zu analysieren, um damit sowohl den Realisierungsaufwand so gering als auch die Analyseverfahren so nachvollziehbar wie möglich zu halten. Die Relationenalgebra [13] bietet einen reichhaltigen Methodenkanon an, Graphen zu behandeln. Wir nutzen diesen Kalkül zur Codierung von Petri-Netzen, zur Konstruktion eines Erreichbarkeitsgraphen und zur Herleitung von Analyseverfahren [9]. Mit Hilfe mathematischer Umformungsregeln gelingt es, aus netztheoretischen Formulierungen etwa der oben gestellten Fragen relationenalgebraische Programme abzuleiten, die

per Konstruktion korrekt sind und sich mit Hilfe der die Relationenalgebra operationalisierenden Bibliothek KURE elegant implementieren lassen. Die entwickelten Analyseverfahren umzusetzen ist wegen ihrer einfachen Implementierbarkeit weniger fehleranfällig als die Verwendung von Techniken, die zusätzlicher Überlegungen zur Implementierung, etwa der Abhängigkeit möglicher Code-Optimierung von der gewählten Datenstruktur, bedürfen. Da wir ausschließlich Relationen als Datenstruktur verwenden, bleibt die Implementierung übersichtlich, verständlich und damit wartbar, gleichzeitig jedoch schränken wir die Möglichkeiten ein, durch Code-Optimierung die Performanz eines Programmes zu verbessern.

Neben der Performanz der Ermittlung des Erreichbarkeitsgraphen ist auch dessen Platzbedarf problematisch. Die sog. *sweep line*-Methode [11] ermöglicht eine speichereffiziente Handhabung unendlicher Erreichbarkeitsgraphen durch eine geeignete Reduktion des Zustandsraumes, so dass eine Analyse des Netzes noch immer sinnvoll möglich ist. Zur Ausführung eines relationales Programmes sind stets endlich-dimensionale Relationen erforderlich. Wir untersuchen daher beschränkte Netze und erzwingen eine Beschränkung durch die Wahl einer geeigneten oberen Schranke für die maximale Kapazität der Stellen des Netzes. Damit ist der Zustandsraum zwingend endlich, kann aber dennoch beliebig groß gewählt werden.

Ebenso können Verfahren zur symbolischen Erzeugung des Erreichbarkeitsgraphen genutzt werden, um große Zustandsräume handhabbar zu machen. In [6] wird ein solches Verfahren vorgestellt, das BDDs zur Verwaltung des Erreichbarkeitsgraphen einsetzt. Wir verwalten Relationen in ROBDDs [5] und ermöglichen damit ebenfalls eine speichereffiziente Darstellung von Relationen, so dass selbst große Zustandsräumen effizient behandelt werden können.

Damit ist das Herstellen und Beherrschen großer Zustandsräume in unserem Ansatz möglich.

Das Papier ist wie folgt aufgebaut: Abschnitt 2 gibt einen kurzen Überblick über relationenalgebraische Grundlagen sowie die Mechanisierung dieses Kalküls in der Bibliothek KURE. Die Erreichbarkeitsanalyse von Petri-Netzen wird in Abschnitt 3 dargestellt. Abschnitt 4 erklärt den Aufbau von PETRA. Das Papier schließt mit einer Zusammenfassung in Abschnitt 5.

2. Präliminarien

Wir führen hier lediglich Konzepte und Notationen der Relationenalgebra ein, sofern sie in unserem Papier von Bedeutung sind. Eine detaillierte Einführung findet sich beispielsweise in [13].

2.1. Relationenalgebraische Grundlagen

Wir notieren eine Relation R zwischen zwei stets endlichen Mengen X und Y durch $R : X \leftrightarrow Y$ und repräsentieren R durch eine Boole'sche Matrix der Größe $\#X \times \#Y$. Wir schreiben $R_{x,y}$ anstelle von $\langle x, y\rangle \in R$ und meinen damit, dass der Eintrag (x, y) in der Matrix von R den Wert 1 hat. Wir notieren die Vereinigung, den Schnitt und die Inklusion zweier Relationen R und S wie üblich durch $R \cup S$, $R \cap S$ und $R \subseteq S$, die Konverse und Negation von R durch R^T und $\overline{R}$ sowie die Konkatenation von R und S durch $R; S$ bzw. RS. Für je zwei Mengen X und Y existieren die leere Relation $\mathbb{O}$, die universelle Relation $\mathbb{L}$ sowie die Identität $\mathbb{I}$, falls $X = Y$ ist.

Der Domain einer Relation R ist definiert durch $R\mathbb{L}$, ihr Co-Domain durch $R^\mathsf{T}\mathbb{L}$.

Wir benutzen eine spezielle ein-elementige Menge $\mathbb{1} := \{\diamond\}$ und repräsentieren eine Relation $v : X \leftrightarrow \mathbb{1}$ als einen Spaltenvektor mit der Eigenschaft $v = v\mathbb{L}$. Dieser Vektor beschreibt die Teilmenge $\{x \in X \mid v_x\}$ von X. In Ausdrücken der Form $v_{x\diamond}$ und $v_{\diamond x}$ ignorieren wir $\diamond$ und schreiben v_x. Ist der Domain von v gleich einer Teilmenge $N \subseteq X$, ist v_x äquivalent zu $(v\mathbb{L})_{x,N}$ für $\mathbb{L} : \mathbb{1} \leftrightarrow 2^X$. Vektoren bilden einen Unterverband der Relationen. Ein injektiver und surjektiver Vektor ist ein Punkt. Punkte sind die Atome im Verband der Vektoren und modellieren damit ein-elementige Mengen bzw. Elemente von Mengen. Es gibt genau zwei Relationen über $\mathbb{1} \times \mathbb{1}$, nämlich $\mathbb{O} : \mathbb{1} \leftrightarrow \mathbb{1}$ und $\mathbb{L} : \mathbb{1} \leftrightarrow \mathbb{1}$, die die Boole'schen Wahrheitswerte *false* und *true* modellieren.

Die Enthaltensein-Relation auf Mengen, d.h. die Relation $\in$, wird durch eine Relation $\varepsilon : X \leftrightarrow 2^X$ auf einer Menge X und ihrer Potenzmenge modelliert, so dass $\varepsilon_{x,S} \iff x \in S$ für jede Teilmenge $S \subseteq X$ gilt.

Für die Inklusion $RQ \subseteq S$ wird Q als das *Rechtsresiduum* $R \backslash S$ von S über R bezeichnet, falls Q die größte Lösung ist, die die Inklusion erfüllt, und durch $\overline{R^\mathsf{T}\overline{S}}$ berechnet. Für die Inklusion $QR \subseteq S$ wird Q analog als das *Linksresiduum* R / S von S über R bezeichnet und durch $\overline{\overline{S}R^\mathsf{T}}$ berechnet. Der *symmetrische Quotient* $\mathsf{syQ}(R, S)$ zweier Relationen R und S ist die größte Lösung, die beide Inklusionen erfüllt, und ist definiert als $(R \backslash S) \cap (S^\mathsf{T} / R^\mathsf{T})$. Für jeden Vektor $R : X \leftrightarrow \mathbb{1}$ be-

schreibt $\mathsf{syQ}(\varepsilon, R)$ denjenigen Punkt in der Potenzmenge 2^X, der R repräsentiert.

Für eine Relation $R : X \cup Y \leftrightarrow X \cup Y$ und einen Punkt $v : X \cup Y \leftrightarrow \mathbb{1}$, der ein Element ν aus $X \cup Y$ modelliert, liefert die Relation $Rv : X \cup Y \leftrightarrow \mathbb{1}$ die Menge aller Vorgänger und $R^{\mathsf{T}}v : X \cup Y \leftrightarrow \mathbb{1}$ die Menge aller Nachfolger von ν in R.

2.2. Mechanisierung mit Kure

Die Grundoperationen der Relationenalgebra, also Vereinigung `|`, Schnitt `&`, Negation `-`, Komposition `*` und Transposition `^`, sowie Tests auf Gleichheit `eq`, auf Inklusion `incl` und auf die leere Relation `empty` sind in Kure vordefiniert. `O(R)` und `L(R)` liefern die leere bzw. die universelle Relation mit den gleichen Dimensionen wie R; `L1n(R)` liefert einen Zeilenvektor mit derselben Spaltenzahl wie R (`O1n(R)` und `On1(R)` analog); `I(R)` liefert die Identitätsrelation mit derselben Dimension wie R; `dom(R)` und `ran(R)` berechnen den Domain bzw. den Co-Domain von R als Vektoren. Mit `point(v)` selektiert man einen Punkt, der im nichtleeren Vektor v enthalten ist.

Relationale Funktionen und Programme erlauben eine Erweiterung der Funktionalität. Eine relationale Funktion F ist dabei eine Klausel der Form `F(`X_1`,...,`X_n`) = t.`, wobei `F` der Funktionsname ist, X_i, $1 \leq i \leq n$, die formalen Parameter repräsentieren und `t` ein relationaler Term über den definierten Relationen ist. Ein relationales Programm in Kure ist ein while-Programm, das lediglich auf Relationen als einzigem Datentyp operiert. Es hat folgenden selbsterklärenden Aufbau (Schlüsselwörter werden unterstrichen); Anweisungen umfassen Zuweisungen, eine while-Schleife und eine if-Anweisung:

```
<NAME>"("<PARAMETERLIST>")"
    DECL <DECLARATIONS>
    BEG <STATEMENTS>
        RETURN <TERM>
    END.
```

In Kure werden Relationen als Objekte behandelt, deren Elemente durch *get*- und *set*-Methoden angesprochen werden. Relationen werden in einem sog. Relationenmanager verwaltet, der die in Funktionen und Programmen verwendbaren Relationen umfasst und diese auszuwerten erlaubt. Ein CASE-Werkzeug nutzt Kure, um mit den aus einem proprietären Datenmodell erzeugten Relationen rechnen zu können. Dazu stehen im Wesentlichen die Methoden *createFunction* und *loadProgramm* zur Definition einer relationalen Funktion bzw. zum Laden eines Programmes sowie die Methode *evaluateTerm* zur Ausführung von Funktionen und Programmen zur Verfügung. Wir zeigen die Umsetzung von relationalen Analysemethoden mit Kure im folgenden Abschnitt.

3. Erreichbarkeitsanalyse

Wir betrachten die Aktivierung von Transitionen, die Berechnung einer unmittelbaren Folgemarkierung sowie einen Algorithmus zur Berechnung der Erreichbarkeitsrelation, die dann einer relationalen Erreichbarkeitsanalyse offen steht. Diese umfasst einen Test auf Erreichbarkeit einer Markierung, die Berechnung der Menge aller Nachfolgemarkierungen einer Markierung sowie das Finden von Heimatmarkierungen.

3.1. Relationenalgebraische Codierung eines Petri-Netzes

Ein Petri-Netz ist ein 6-Tupel der Form $\mathcal{N} = (R, S, W\_t, Wt\_, C, M)$ mit

- $R : P \leftrightarrow T$ die Flussrelation Stellen/Transitionen
- $S : T \leftrightarrow P$ die Flussrelation Transitionen/Stellen
- $W^{\bullet t} : P \times T \leftrightarrow N$ eine univalente Gewichtsrelation für Kanten aus R
- $W^{t\bullet} : P \times T \leftrightarrow N$ eine univalente Gewichtsrelation für Kanten aus S
- $C : P \leftrightarrow N$ eine totale und univalente Kapazitätsrelation
- $M_0 : P \leftrightarrow N$ eine totale und univalente Markierungsrelation

Wir bezeichnen die Stellenmenge eines Netzes mit P und die Transitionsmenge mit T. Die übliche Flussrelation F wird durch $R \cup S$ aufgespannt, analog die Kantengewichtung W. Die natürlichen Zahlen repräsentieren wir durch eine lineare Ordnung N zusammen mit einer injektiven Abbildung $succ : N \leftrightarrow N$ und einem Punkt $zero : N \leftrightarrow \mathbb{1}$ mit der Eigenschaft $(succ^{\mathsf{T}})^*; zero = \mathbb{L}$, d.h. jedes $n \in N$ kann von *zero* aus über die Nachfolgerrelation *succ* erreicht werden (die transitive Hülle einer Relation R notieren wir als R^*).

3.2. Aktivierte Transitionen

Die unter einer Markierung M aktivierten Transitionen t bestimmt man als eine Teilmenge aller Transitionen T mit den üblichen Konzessionseigenschaften $\forall p \in \bullet t : M(p) \geq W^{\bullet t}(p, t)$ und $\forall p \in t\bullet : M(p) \leq C(p) - W^{t\bullet}(t, p)$. Relationenalgebraisch modellieren wir

diese Menge durch einen Vektor $CN : T \leftrightarrow \mathbb{1}$, der durch die Relation

$$\mathbb{L}\backslash\,(\,(\overline{R} \cup (M; \geq; W^{\bullet t\,\mathsf{T}} \cap \pi^{\mathsf{T}}); \rho) \\ \cap\, (\overline{S^{\mathsf{T}}} \cup ((M; \leq; \mathsf{sub}^{\mathsf{T}} \cap C; \sigma^{\mathsf{T}}); \tau; W^{t\bullet\,\mathsf{T}} \cap \beta^{\mathsf{T}}); \alpha))$$

definiert ist. Diese Relation entspricht der Umformung der Konzessionseigenschaften (zum Beweis siehe [9]) und verwendet die natürlichen Projektionen $\pi : P \times T \leftrightarrow P$, $\rho : P \times T \leftrightarrow T$, $\alpha : T \times P \leftrightarrow T$, $\beta : T \times P \leftrightarrow P$, $\sigma : N \times N \leftrightarrow N$ und $\tau : N \times N \leftrightarrow N$. Zur Umsetzung in ein KURE-Programm müssen dann lediglich geeignete Produktdomains generiert und obiger Term in die KURE-Notation umgesetzt werden.

```
CN(R, S, C, W_t, Wt_, M)
    DECL
        PTdom = PROD(R*S, S*R);
        TPdom = PROD(S*R, R*S);
        NNdom = PROD(M^*M, M^*M);
        pi, rho, alpha, beta, sigma, tau, cn
    BEG
        pi = p-1(PTdom);
        rho = p-2(PTdom);
        alpha = p-1(TPdom);
        beta = p-2(TPdom);
        sigma = p-1(NNdom);
        tau = p-2(NNdom);
        cn = (-R | (M*ge*W_t^ & pi^)*rho)
                &
             (-S^ | ((M*le*sub^ & C*sigma^)
                      *tau*Wt_^ & beta^) *alpha)
        RETURN Ln1(R) \ cn
    END.
```

Das relationale Programm `CN` wird mit Hilfe von KURE durch folgende Sequenz aufgerufen, nachdem die nötigen Relationen zur Repräsentation eines konkreten Petri-Netzes durch die *createRelation*-Methode erzeugt wurden:

```
new RelManager manager = createRelManager("PNMan");
// initialisiere Relationenmanager mit Namen "PNMan"

new Relation R = rM.createRelation("R", dimX, dimY);
// erzeuge Relation R in Groesse dimX x dimY
// sowie alle anderen Relationen

manager.loadProgram(CN.prog);
// mache Programm CN in PNMan verfuegbar

new Relation actTrans =
       manager.evaluateTerm("CN(R, S, C, W_t, Wt_, M)");
// werte CN fuer gegebenes Netz aus
// speichere aktivierte Transitionen in actTrans
```

3.3. Bestimmung einer Nachfolgemarkierung

Ein relationes Programm, das zu einer Markierung M und einer darunter aktivierten Transition t die Nachfolgemarkierung M' ermittelt, wird analog zu `CN` durch Umformung der üblichen Schaltregel hergeleitet. Für den Fall $M'(p) := M(p) - W^{\bullet t}(p,t)$ mit $p \in \bullet t \setminus t\bullet$ ergibt sich beispielsweise ein relationaler Term der Form $(R; t \cap \overline{S^{\mathsf{T}}; t}); \mathbb{L} \cap ((M; \sigma^{\mathsf{T}} \cap w^{\bullet t}; \tau^{\mathsf{T}}); \mathsf{sub})$. Das relationale Programm `IRM` ermittelt eine solche Folgemarkierung (siehe [9]). Dabei ist $w^{\bullet t} : P \leftrightarrow N$ eine Einschränkung von $W^{\bullet t}$ auf eine betrachtete Transition t.

```
UNCHANGED(R, S, t, M) = DOMRES(M, -(R*t | S^*t)).

IRM(R, S, W_t, Wt_, M, t)
DECL
     NNdom = PROD(M^*M, M^*M);
     PTdom = PROD(R*S, S*R);
     TPdom = PROD(S*R, R*S);
     w_t, wt_,
     pi, rho, alpha, beta, sigma , tau,
     updt_, upd_t, upd_t_, hlp
BEG
     sigma = p-1(NNdom);
     tau = p-2(NNdom);
     pi = p-1(PTdom);
     rho = p-2(PTdom);
     alpha = p-1(TPdom);
     beta = p-2(TPdom);
     w_t = (R*rho^ & pi^)*W_t;
     wt_ = (S^*alpha^ & beta^)*Wt_;
     upd_t =
        DOMRES((M*sigma^ & w_t*tau^)*sub, R*t &-(S^*t));
     updt_ =
        DOMRES((M*sigma^ & wt_*tau^)*add, S^*t & -(R*t));
     hlp =
        ((M*sigma^ & w_t*tau^)*sub*sigma^ & wt_*tau^)*add;
     upd_t_ = DOMRES(hlp, R*t & S^*t)
     RETURN updt_ | upd_t | upd_t_
           | UNCHANGED(R, S, t, M)
END.
```

Auch dieses Programm wird in KURE einfach durch die folgende Sequenz geladen und ausgeführt:

```
manager.loadProgram(IRM.prog);
new Relation followMark =
       manager.evaluateTerm("IRM(R, S, W_t, Wt_, M, t)");
```

Ein gegebenes Petri-Netz kann mit den Programmen `CN` und `IRM` bequem und einfach simuliert werden.

3.4. Erzeugung des Erreichbarkeitsgraphen

Der folgende rekursive Algorithmus ermittelt eine Erreichbarkeitsrelation *depth-first*, die Paare unmittelbar erreichbarer Markierungen enthält:

1. erstelle eine leere Relation *emptyIRM* der Größe $2^{P\times N} \times 2^{P\times N}$
2. für jede noch nicht behandelte erreichbare Markierung M (beginne mit M_0):
   (a) wandle $M : P \leftrightarrow N$ in einen Punkt der Form $m : 2^{P\times N} \leftrightarrow \mathbb{1}$ um (dieser repräsentiert M als Element in der Potenzmenge aller Markierungen)

(b) trage (m, m) in *emptyIRM* ein (jede Markierung ist Nachfolgemarkierung von sich selbst)

(c) berechne die Menge CN der unter M aktivierten Transitionen

(d) für jede Transition t aus CN:

i. berechne die Nachfolgemarkierung N von M bzgl. t

ii. wandle $N : P \leftrightarrow N$ in einen Punkt der Form $n : 2^{P \times N} \leftrightarrow \mathbb{1}$ um

iii. trage (m, n) in *emptyIRM* ein

iv. falls (n, n) noch nicht in *emptyIRM* eingetragen ist (die Markierung N muss noch behandelt werden):

A. verfahre ab Punkt 2b mit N als zu behandelnde Markierung

3. wenn keine noch nicht behandelte Markierung erzeugt werden kann, enthält $REACH := emptyIRM^*$ die Menge aller Paare transitiv erreichbarer Markierungen.

Die Umsetzung diese Verfahrens in ein relationales Programm `IREACH` geschieht direkt durch die Verwendung der in KURE bereitstehenden relationalen Konzepte. Das Bilden ihrer transitiven Hülle erzeugt dann den vollständigen Erreichbarkeitsgraphen, erzeugt also eine Relation $REACH$, die Paare von transitiven erreichbaren Markierungen beherbergt (siehe [9]).

```
IREACH(R, S, C, W_t, Wt_, M)
DECL
    mvec, mpoint, allMarks
BEG
    mvec = RelToVec(M);
    mpoint = syq(epsi(mvec), mvec);
    allMarks = 0(mpoint * mpoint^)
    RETURN IRMARKS(M, mpoint, allMarks)
END.

IRMARKS(M, mpoint, AM)
DECL
    allMarks, cnc, t, irm, irmvec, irmpoint
BEG
    allMarks = AM | mpoint * mpoint^;
    cnc = CN(R, S, C, W_t, Wt_, M);
    WHILE -empty(cnc) DO
       t = point(cnc);
       irm = IRM(R, S, W_t, Wt_, M, t);
       irmvec = RelToVec(irm);
       irmpoint = syq(epsi(irmvec), irmvec);
       allMarks = allMarks | mpoint * irmpoint^;
       IF -incl(irmpoint * irmpoint^, allMarks) THEN
           allMarks = IRMARKS(irm, irmpoint, allMarks)
       FI;
       cnc = cnc & -t
    OD
    RETURN allMarks
END.
```

3.5. Relationale Analyseverfahren

Die Menge aller von einer Markierung M aus erreichbaren Markierungen, $[M>$, entspricht der Menge aller Nachfolger von M bzgl. $REACH$ und wird durch einen Vektor $REACHABLE : 2^{P \times N} \leftrightarrow \mathbb{1}$ realisiert, der durch

$$REACH^{\mathsf{T}}; \mathsf{syQ}(\varepsilon, \tilde{m})$$

definiert ist. Dabei ist $\tilde{m}$ die Umwandlung von M in einen Vektor der Form $2^{P \times N} \times \mathbb{1}$. Der symmetrische Quotient der Relation ε und des Vektors $\tilde{m}$ bezeichnet denjenigen Punkt in $2^{P \times N}$, der $\tilde{m}$ repräsentiert.

```
REACHABLE(R, S, C, W_t, Wt_, M)
DECL
    mvec
BEG
    mvec = RelToVec(M)
    RETURN REACH^ * syq(epsi(mvec), mvec)
END.
```

Ein Erreichbarkeitstest der Form „$N \in [M>?$“ wird relationenalgebraisch durch die Inklusion

$$\mathsf{syQ}(\varepsilon, \tilde{m}); \mathsf{syQ}(\varepsilon, \tilde{n})^{\mathsf{T}} \subseteq REACH$$

beschrieben. Er ist erfolgreich, falls das Paar (M, N) in $REACH$ enthalten ist.

```
isREACHABLE(R, S, C, Wt_, W_t, M, N)
DECL
    mvec, nvec
BEG
    mvec = RelToVec(M);
    nvec = RelToVec(N)
    RETURN incl(syq(epsi(mvec), mvec)
                * syq(epsi(nvec), nvec)^, REACH)
END.
```

Eine Markierung ist eine Heimatmarkierung, wenn sie von jeder erreichbaren Markierung aus erreichbar ist, also wenn $\forall N\, REACHABLE_N \rightarrow REACH_{N,M}$ gilt. Die Menge aller Heimatmarkierungen wird als Vektor über $2^{P \times N} \times \mathbb{1}$ durch Bilden des transponierten Rechtssresiduums

$$(REACHABLE \backslash REACH)^{\mathsf{T}}$$

beschrieben, der als relationale Funktion implementiert werden kann.

```
homeState(R, S, C, W_t, Wt_, M) =
       (REACHABLE(R, S, C, W_t, Wt_, M) \ REACH)^.
```

3.6. Bewertung der Verfahren

Wir untersuchten unter anderem ein Netz mit 17 Stellen und 14 Transitionen mit einer maximalen Stellenkapazität von 10 Token. Die Relation $REACH$ hat

dann die Größe $2^{187} \times 2^{187}$. Das betrachtete Netz besaß 184 Markierungen. In *REACH* gab es 704 Paare unmittelbar erreichbarer Markierungen, die gesamte Erreichbarkeitsrelation besaß 33856 Paare. Auf einem Pentium 4 mit 2.66 GHz und 512 MB RAM benötigten wir etwa 85 Sekunden zur Erzeugung der Erreichbarkeitsrelation *REACH*. Die Ergebnisse der obigen Analyseverfahren auf dieser Relation lagen dann nach ca. 0,002 Sekunden vor. Nachdem *REACH* aufgebaut ist, können wir mit vergleichbaren Werkzeugen wie PIPE oder INA mithalten.

4. Das Werkzeug PetRA

Das an der Christian-Albrechts-Universität zu Kiel entwickelte Werkzeug RelView [2] liefert die Grundlage für die Java-Bibliothek Kure. In ihr ist die für das Rechnen mit Relationen notwendige Funktionalität von RelView extrahiert und zusätzlich an das Paradigma der Objektorientierung angepasst worden [15]. Kure entstand in Zusammenarbeit mit der Kieler RelView-Gruppe. Im Gegensatz zu RelView ermöglicht Kure das Behandeln von Relationen in proprietären Werkzeugen, ohne RelView selbst heranziehen zu müssen.

Die händische Eingabe speziell großer Petrinetze unmittelbar durch Relationen ist jedoch aufwendig und fehleranfällig, mithin für ein praktisches Arbeiten nicht ausreichend. Es wird ein graphisches Modellierungswerkzeug benötigt, dass die Erstellung von Petrinetzen auf komfortablere Weise unterstützt. Über eine geeignete Schnittstelle zu Kure können anschließend auf einem derart modellierten Netz relationale Analysen ausgeführt werden. Eine naheliegende Möglichkeit, diese Schnittstelle zu realisieren, ist die Unterstützung des Speicherformates eines existierenden Petrinetzwerkzeuges oder der Petri Net Markup Language (PNML) innerhalb von Kure. Diese Realisierung erlaubt jedoch nur die Überführung des Petrinetzes nach Kure, nicht aber die Rückführung von Analyseergebnissen in das graphische Werkzeug. Wir haben uns deshalb für eine engere Anbindung entschieden, bei der Kure unmittelbar in ein existierendes Petrinetz-Werkzeug [1] integriert wird. Das Ergebnis dieser Bemühungen ist das Werkzeug PetRA zur Arbeit mit Petrinetzen auf der Basis relationaler Algebra.

Abbildung 1 zeigt ein Bildschirmphoto von der Arbeit mit PetRA. Zentrales Element der Benutzeroberfläche ist der Modellierungs- und Anzeigebereich für das Petrinetz. Oberhalb dieses Bereiches befindet sich eine Werkzeugleiste, mit Hilfe derer unter anderem neue Stellen, Transitionen oder Flusskanten in das Netz eingefügt werden. Am linken Rand finden sich Komponenten zum Inspizieren und Bearbeiten der Eigenschaften von Netzelementen und zur Navigation in größeren Netzen.

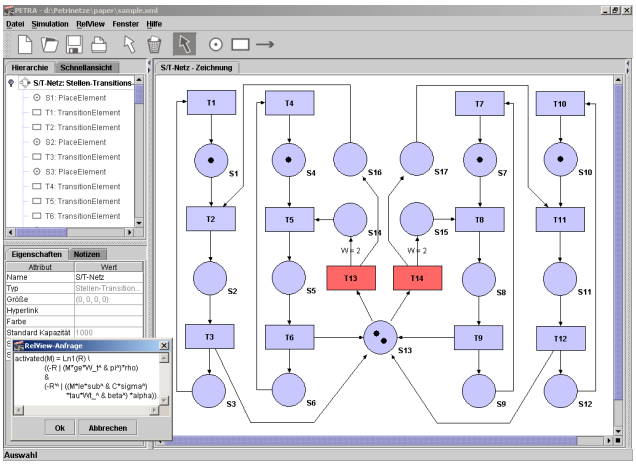

Abbildung 1. Das Werkzeug PetRA

Die Funktionalität zur relationalen Analyse eines Petrinetzes wird über die Menüleiste angeboten. Beim Starten einer Analysesitzung wird das Netz in die relationale Repräsentation von Kure übersetzt. Gleichzeitig wird die Bildschirmdarstellung in einen nur lesbaren Modus versetzt, um Inkonsistenzen zwischen beiden Repräsentationen zu verhindern. Anschließend können über das Menü Analysen statischer oder dynamischer Eigenschaften des Netzes angestoßen werden. Diese werden von Kure ausgewertet. Das Ergebnis jeder Analyse ist wieder in Form einer Relation kodiert, die mit geringem Aufwand in die graphische Darstellung zurückübersetzt wird. In der Abbildung erkennt man eine entsprechende Hervorhebung von aktivierten Transitionen, die zuvor durch Kure ermittelt wurden.

PetRA ist in Java/Swing implementiert und macht sich neben Kure ein an der Universität Dortmund entwickeltes Framework zunutze, das die Entwicklung von leichtgewichtigen Modellierungswerkzeugen unterstützt. Zur Entwicklung eines konkreten Werkzeugs für eine gegebene graphische Sprache wird deren Metamodell mit Hilfe einiger Java-Klassen implementiert, und es wird festgelegt, wie die einzelnen Modellelemente graphisch repräsentiert werden. Sämtliche vom konkreten Metamodell unabhängige Funktionalität ist bereits auf generische Weise im Framework implementiert. Dazu zählen insbesondere die gesamte Benutzungsschnittstelle, der Persistenzmechanismus und Möglichkeiten zum Drucken von Modellen.

Das Framework dient primär dazu, einfache und intuitiv nutzbare Werkzeuge für den Einsatz in der Lehre zu entwickeln [12], ist jedoch auch eine geeignete Basis für komplexere Werkzeuge wie PetRA. Neben dem Metamodell für Petrinetze und der generellen Schnittstelle zu Kure werden bei PetRA auch die einzelnen Analysen in Java implementiert. Jede Analyse findet sich in einer einfachen Java-Klasse wieder, die im wesentlichen die Auswertung eines oder mehrerer relationaler Ausdrücke durchführt und anschließend das Ergebnis auf geeignete Weise in die graphische Darstellung übersetzt. Die verfügbaren Analysen werden dem Werkzeug über eine Konfigurationsdatei bekannt gemacht, so dass neue Analysen ohne großen Aufwand hinzugefügt werden können.

5. Zusammenfassung

Die objektorientierte Java-Bibliothek Kure bietet eine Mechanisierung des Relationenkalküls und kann in jedem CASE-Werkzeug verwendet werden, dessen Anwendungsbereich mit Relationen modelliert werden kann. Wir haben mit ihrer Hilfe den Einsatz der Relationenalgebra zur Analyse von Petri-Netzen diskutiert und stellen eine neue relationenalgebraische Schnittstelle zur Erreichbarkeitsanalyse von Stellen/Transitionssystemen zur Verfügung.

Unser Ansatz konstruiert den Erreichbarkeitsgraphen und ermöglicht seine Analyse durch einfache relationale Konzepte. Wir benötigen lediglich Relationen als einzige Datenstruktur. Die Entwicklung relationaler Programme ist dank der Einbettung des Relationenkalküls in eine imperative Programmiersprache recht einfach und kann meist direkt aus netztheoretischen Formeln durch mathematische Umformungsregeln hergeleitet werden. Diese Programme sind damit korrekt per Konstruktion.

Unser Werkzeug PetRA kombiniert Relationenalgebra mit der Visualisierung von Petri-Netzen und ermöglicht eine graphische Interaktion mit dem Relationenkalkül, indem sowohl Eingaben an diesen Kalkül als auch Ausgaben von Berechnungen im modellierten Petri-Netz angezeigt werden können. Dem Modellierer steht ein Petri-Netz-Werkzeug zur Verfügung, dessen offene Systemarchitektur eine leichte Erweiterbarkeit um weitere relationale Analyseverfahren ermöglicht. Wir bieten einige Tests zur Ermittlung der Netzklassenzugehörigkeit an und erlauben das Berechnen von Fallen und Co-Fallen (vgl. [4]). Ein Test zur Ermittlung der Deadlock/Falle-Eigenschaft ist ebenfalls integriert. PetRA bietet damit eine solide Basis für einen alternativen Zugang zur Analyse von Petri-Netzen.

Danksagung Unser Dank gilt Zahir Amiri, Oliver Szymanski, Ernst-Erich Doberkat und Rudolf Berghammer.

Literatur

[1] Z. Amiri. Entwicklung eines Petrinetz-Editors und -Simulators auf Basis des LiMo-Frameworks. Diplomarbeit, Lehrstuhl für Software-Technologie, Universität Dortmund, 2004.

[2] R. Behnke, R. Berghammer, E. Meyer, and P. Schneider. RelView – A system for calculating with relations and relational programming. In *Proc. 1st Intl. Conf. FASE*, volume 1382 of *LNCS*, pages 318 – 321. Springer, 1998.

[3] R. Berghammer. The clique problem. Private communication; paper in progress, 2004.

[4] R. Berghammer, B. Karger, and C. Ulke. Relational-algebraic analysis of Petri nets with RelView. In *Proc. 2nd Workshop TACAS*, volume 1055 of *LNCS*, pages 49–69. Springer, 1996.

[5] R. Berghammer, B. Leoniuk, and U.Milanese. Implementation of relational algebra using binary decision diagrams. In *Proc. 6th Intl. Workshop RelMiCS*, volume 2561 of *LNCS*, pages 241–257. Springer, 2002.

[6] G. Ciardo, G. Lüttgen, and R. Siminiceanu. Efficient symbolic state-space construction for asynchronous systems. In *Proc. Intl. Conf. ATPN*, number 1825 in LNCS, pages 103–122. Springer, 2000.

[7] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.

[8] J. Esparza and C. Schröter. Unfolding based algorithms for the reachability problem. *Fundamenta Informaticae*, 47(3–4):231–245, 2001.

[9] A. Fronk. Using relation algebra for the analysis of Petri nets in a CASE tool based approach. In *Proc. 2nd IEEE Intl. Conf. SEFM*, 2004. Accepted Paper.

[10] Th. Hoffmann. *Fallstudien relationaler Programmentwicklung am Beispiel ausgewählter Graphdurchlaufstrategien*. Doktorarbeit, Universität Kiel, 2002.

[11] Th. Mailund. Analysing infinite-state systems by combining equivalence reduction ad the the sweep-line method. In *Proc. Intl. Conf. ATPN*, number 2360 in LNCS, pages 314–334. Springer, 2002.

[12] J. Pleumann, J. Schröder, and K. Alfert. Software engineering education needs adequate modeling tools. In *Proc. 17th IEEE Intl. CSEET*, pages 72–77. IEEE Computer Society, March 2004.

[13] G. Schmidt and Th. Ströhlein. *Relations and graphs*. EATCS Monographs on Theoretical Computer Science. Spinger, 1993.

[14] P. H. Starke. *Analyse von Petri-Netz-Modellen*. Teubner, 1990.

[15] O. Szymanski. Relationale Algebra im dreidimensionalen Software-Entwurf – ein werkzeugbasierter Ansatz. Diplomarbeit, Universität Dortmund, 2003.