KfK 5246
September 1993

Konzeption und Entwicklung eines graphischen Analysewerkzeugs für Höhere Petri-Netze mit zustandsabhängiger Schaltregel

W. Süß
Institut für Angewandte Informatik

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE

Institut für Angewandte Informatik

KfK 5246

Konzeption und Entwicklung eines graphischen Analysewerkzeugs für Höhere Petri-Netze mit zustandsabhängiger Schaltregel *)

Wolfgang Süß

*) vom Fachbereich 4: Informatik der Universität Koblenz-Landau genehmigte Dissertation

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Kernforschungszentrum Karlsruhe GmbH
Postfach 3640, 76021 Karlsruhe

ISSN 0303-4003

Konzeption und Entwicklung eines graphischen Analysewerkzeugs für Höhere Petri-Netze mit zustandsabhängiger Schaltregel

Zusammenfassung

Zur Spezifikation und Modellierung komplexer Softwaresysteme benötigt man formale Verfahren, die sowohl komplexe sequentielle Aktionen (Funktionen auf Objekten) als auch komplexe parallele Aktionen formal beschreibbar machen. Dabei sollte ein Anwender solcher Methoden durch ein computergestütztes Werkzeug unterstützt werden, das ihm neben der reinen Eingabe auch die automatische Analyse von Modellen erlaubt. Zur Beschreibung komplexer sequentieller Aktionen auf Datenstrukturen kann man die Methode der algebraischen Spezifikation abstrakter Datentypen verwenden. Komplexe parallele Aktionen lassen sich mit der Petri-Netz-Theorie beschreiben. Unser Spezifikations- und Modellierungsansatz und das zugehörige Werkzeug SMARAGD (Specification, Modelling and Reachability Analysis Graphical Development System for High-level Petri Nets) basieren daher auf einer Kombination von algebraischen Spezifikationsmethoden und der Petri-Netz-Theorie.

Die formale Beschreibung des Netz-Modells erfolgt zunächst in Form der Netzbeschreibungssprache SNL (SMARAGD Net Language) und wird dann anschließend auf eine durch graphische Konstrukte unterstützte Beschreibungsform abgebildet, die eine weitgehend natürliche Eingabe der Netz-Modelle über einen graphischen Editor erlaubt, wie er dann auch im Werkzeug SMARAGD implementiert wurde. Über diese Sprache SNL und den SMARAGD-Editor, mit dem SNL-Systeme erstellt werden können, wird ein kurzer Überblick gegeben.

Die Analyse von SNL-Systemen ist Gegenstand dieser Arbeit. Bei der Analyse eines Petri-Netzes werden Eigenschaften des Netzes, wie z.B. Verklemmungen, bestimmt. Solche Eigenschaften können über Invarianten- und Erreichbarkeitsanalyse gewonnen werden. Um SNL-Systeme analysieren zu können, müssen wir zuerst zeigen, daß Invarianten- und Erreichbarkeitsanalyse für SNL-Systeme möglich ist. Dafür werden in der Arbeit die theoretischen Grundlagen erarbeitet. Für ein Werkzeug ist außerdem wichtig, daß diese Analysen von einem Automaten ausgeführt werden können. Während automatische Invariantenberechnung für Höhere Petri-Netze im allgemeinen und auch für die hier eingeführten SNL-Systeme nicht möglich ist, kann die Erreichbarkeitsanalyse vom Rechner übernommen werden.

Bei der Erreichbarkeitsanalyse werden zuerst alle Zustände des Netzes, die von einer gegebenen Anfangsmarkierung aus möglich sind, generiert. Dies führt zu dem sogenannten Erreichbarkeitsgraphen. Aus dem Erreichbarkeitsgraphen wird der Graph der starken Zusammenhangskomponenten generiert. Aus diesen beiden Graphen werden die Eigenschaften eines SNL-Systems bestimmt. Anhand dieser Eigenschaften kann ein Benutzer entscheiden, ob sich das System gemäß seiner Intention verhält. Zeigt das System ein Fehlverhalten, so möchte der Benutzer die Ursache dieses Fehlverhaltens finden. Die Suche nach dem Fehlverhalten eines SNL-Systems kann sehr aufwendig sein. Deshalb wird das Konzept einer regelbasierten Auswertung der Analyseergebnisse angegeben. Mit solch einer regelbasierten Auswertung von Analyseergebnissen soll der Rechner dem Benutzer helfen, ein mögliches Fehlverhalten des Systems und seine Ursachen schneller zu finden.

Weiter wird in der Arbeit das Konzept der SMARAGD-Analyseeinheit vorgestellt und ein erster Prototyp, bei dem eine rechnergestützte Erreichbarkeitsanalyse realisiert ist, beschrieben. Die SMARAGD-Analyseeinheit erleichtert dem Benutzer die Analyse eines Systems durch eine übersichtliche Darstellung der Eigenschaften eines Systems. SMARAGD kombiniert dazu graphische und textuelle Ausgabemöglichkeiten zu einer zusammenhängenden Einheit.

Am Ende der Arbeit wird unser Werkzeug mit anderen Werkzeugen verglichen und ein Ausblick auf den weiteren Ausbau von SMARAGD gegeben. Ein Liste mit den verwendeten Bezeichnungen sowie ein Literaturverzeichnis schließen die Arbeit ab.

A Graphical Analyzer for High-Level Petri Nets with State Dependant Firing Rule

Abstract

The specification and modelling of complex software systems request for formal methods. We use these methods to describe sequentielle and parallel actions. The method of algebraic specification of abstract datatypes is suitable for the description of sequentielle actions of datastructures. Parallel actions can be very good described with petri nets. Therefore in our approach and our tool SMARAGD (Specification, Modelling and Reachability Analysis Graphical Developmentsystem for High-level Petri Nets) we combine the methods of algebraic specification and Petri Nets.

The formal description of a net model is done in the net description language SNL (SMARAGD Net Language). This is called SNL-specification. We transform these SNL-specifications into a graphical description form, the so called SNL-systems. It is possible to develop SNL-systems with a graphical editor. Such a graphical editor is implemented in the SMARAGD-tool. A short overview about the SNL-language and the SMARAGD-editor is part of this paper.

The analysis of SNL-systems is the objective of this paper. The analysis of petri nets detects various properties of the net model, such as livelocks and deadlocks. Such properties are computable by invariant or reachability analysis. Therefore the theoretical basis for the reachability and invariant analysis is given. Designing our SMARAGD-tool we found it highly recommended, that the analysis can be done automatically. But for invariants this is impossible for high-level petri nets, whereas the reachability analysis can be done automatically by a computer.

The reachability analysis generates first all possible states of the net being reachable from a given initial marking. This leads to the reachability graph. From the reachability graph we generate the graph of the strong connected components. On the basis of these two graphs we compute the properties of the SNL-system. Using these properties a user can decide the correctness of the SNL-system. Has a mistake been found in the system behaviour, the user wants to find the reason for this mistake. The search for the reason of this mistake can be very extensive. Therefore the concept of a rule-based interpretation of the analyzed net properties is given.

Furthermore this paper describes the SMARAGD-analyzer which includes an automatically reachability analysis.

We conclude with a comparison of the SMARAGD-tool with other petri net tools and an overview about future developments of SMARAGD.

Inhaltsverzeichnis

1. Einleitung

In den letzten Jahren konnte man eine stürmische Entwicklung von Theorien zur Beschreibung nebenläufiger Systeme beobachten. Zuerst motiviert durch konkrete Fragen innerhalb der Informatik, finden diese Theorien immer mehr Anwendungen auch in anderen Gebieten wie z.B. der allgemeinen Systemtheorie. Die Theorie der Petri-Netze gehört bereits heute zu den klassischen Vertretern solcher Theorien.

Petri-Netze werden als Modellierungshilfsmittel für nebenläufige Systeme verwendet. In dieser Arbeit wird die rechnergestützte Analyse eines Petri-Netz-Modells beschrieben.

Am Institut für Angewandte Informatik (IAI) des Kernforschungszentrums Karlsruhe (KfK) haben mein Kollege Herr Clemens Düpmeier und ich an der Entwicklung des Petri-Netz-Werkzeuges SMARAGD (Specification, Modelling and Reachability Analysis Graphical Developmentsystem for High-level Petri Nets) gearbeitet. Ein erster Prototyp von SMARAGD wurde fertiggestellt. SMARAGD wird am IAI zur rechnergestützten Spezifikation und Modellierung komplexer Softwaresysteme eingesetzt und basiert auf der algebraischen Spezifikation abstrakter Datentypen und einer Höheren Petri-Netz-Form.

Bei der Entwicklung von SMARAGD ließen sich zwei thematisch getrennte Aufgabengebiete unterscheiden: zum einen die Entwicklung der Benutzerschnittstelle von SMARAGD, im folgenden auch Petri-Netz-Editor genannt, und zum anderen die Entwicklung der Analyseeinheit von SMARAGD. Meine Arbeit war die Entwicklung der Analyseeinheit, während Herr Düpmeier den Petri-Netz-Editor sowie die Definition des Petri-Netz-Modells bearbeitet hat. Eine genaue Beschreibung des Petri-Netz-Editors und der entwickelten Sprache SNL (SMARAGD Net Language) ist in der Dissertation von Herrn Düpmeier ([Düpmeier92]) zu finden. Die dort verwendeten Bezeichnungen, Beispiele und Ergebnisse werden hier übernommen, wobei die für diese Arbeit wichtigsten Begriffe noch einmal kurz erläutert werden.

Diese Arbeit gliedert sich im wesentlichen in zwei Abschnitte. Zuerst werden die theoretischen Grundlagen der Analyse von SNL-Systemen behandelt. Diese Analyse gliedert sich in die drei Teile Erreichbarkeitsanalyse, Invariantentheorie und regelbasierte Auswertung von Analyseergebnissen. Im zweiten Teil der Arbeit wird das Konzept der SMARAGD-Analyseeinheit vorgestellt und ein erster Prototyp beschrieben. In diesem ersten Prototyp ist die Erreichbarkeitsanalyse von SNL-Systemen realisiert.

Am Ende der Arbeit wird unser Werkzeug mit anderen Werkzeugen verglichen und ein Ausblick auf den weiteren Ausbau von SMARAGD gegeben. Ein Liste mit den verwendeten Bezeichnungen sowie ein Literaturverzeichnis schließen die Arbeit ab.

2. Problemstellung und Lösungsansatz

Unsere Aufgabe am IAI war, ein auf Petri-Netzen basierendes System für die Beschreibung nebenläufiger Systeme zu entwickeln, das für die vielfältigen Entwicklungen im IAI verwendet werden kann.

Unser Netzmodell basiert auf PrT-Netzen, wobei einige wichtige Einschränkungen der klassischen PrT-Netz Definition nach Genrich und Lautenbach ([Genrich78]) notwendig waren. Das Netzmodell baut auf Spezifikationen von abstrakten Datentypen innerhalb einer geeigneten Spezifikationssprache auf. Wir beschreiben im Sinne einer Spezifikation ein System zunächst durch eine SNL-Netz-Spezifikation. Eine Realisierung dieser SNL-Netz-Spezifikation liefert ein ausführbares Modell des Systems, das die Anwendung der Invariantentheorie und Erreichbarkeitsanalyse ermöglicht.

Eine SNL-Netz-Spezifikation besteht aus drei Teilen: einer der Netzanschrift zugrundeliegenden Spezifikation eines abstrakten Datentypes, einem Netz und einer Beschriftung des Netzes, die auf die in der Spezifikation gegebenen Elemente aufbaut. Die Beschriftung ordnet Stellen Mengen von Variablen, Kapazitäten für Sorten und optional eine Formel zu. Sie ordnet Kanten formale Summen über Terme der Spezifikation und ordnet Transitionen eine Formel der Spezifikationssprache zu. Die Anfangsmarkierung des Netzes ist eine formale Summe über geschlossenen Termen.

Bei der Analyse eines Petri-Netzes werden Eigenschaften des Netzes, wie z.B. Verklemmungen, bestimmt. Solche Eigenschaften können über Invarianten- und Erreichbarkeitsanalyse gewonnen werden. Dazu müssen wir zuerst zeigen, daß Invarianten- und Erreichbarkeitsanalyse für unsere Systeme möglich ist. Für ein Werkzeug ist außerdem wichtig, daß diese Analysen von einem Automaten ausgeführt werden können. Während automatische Invariantenberechnung für Höhere Petri-Netze im allgemeinen und auch für unsere SNL-Systeme nicht möglich ist, kann die Erreichbarkeitsanalyse vom Rechner übernommen werden.

Eine automatische Invariantenanalyse unserer SNL-Systeme ist nicht möglich, da alle Funktionen, die als Kantenbeschriftung vorkommen, vom Rechner invertierbar sein müßten. Wir werden jedoch zeigen, daß vom Benutzer vorgegebene Invarianten verifizierbar sind. Der Rechner kann den Benutzer dabei unterstützen, potentielle Invarianten zu finden. Eine Möglichkeit dafür ist, das einem SNL-System zugrundeliegende Netz als Platz/Transitions-Netz zu beschriften und dann von diesem Netz die Invarianten zu berechnen. Solche Invarianten können ein Indiz auf entsprechende Invarianten des zugehörigen SNL-Systems sein.

Bei der Erreichbarkeitsanalyse werden zuerst alle Zustände des Netzes, die von einer gegebenen Anfangsmarkierung aus möglich sind, generiert. Dies führt zu dem sogenannten Erreichbarkeitsgraphen. Aus dem Erreichbarkeitsgraphen wird der Graph der starken Zusammenhangskomponenten generiert. Aus diesen beiden Graphen werden die Eigenschaften eines SNL-Systems bestimmt. Anhand dieser Eigenschaften kann ein Benutzer entscheiden, ob sich das System gemäß seiner Intention verhält. Zeigt das System ein Fehlverhalten, so möchte der Benutzer die Ursache dieses Fehlverhaltens finden. Dies bedingt, daß der Benutzer eine Vielzahl von Eigenschaften eines Systems in Betracht ziehen muß. Die SMARAGD-Analyseeinheit erleichtert dem Benutzer die Analyse eines Systems durch eine übersichtliche Darstellung der Eigenschaften eines Systems. SMARAGD kombiniert dazu graphische und textuelle Ausgabemöglichkeiten zu einer zusammenhängenden Einheit.

Trotz dieser übersichtlichen Darstellungsweise kann die Analyse eines Systems aufwendig sein. Wir werden deshalb das Konzept einer regelbasierten Auswertung der Analyseergebnisse angeben. Mit solch einer regelbasierten Auswertung von Analyseergebnissen soll der Rechner dem Benutzer helfen, ein mögliches Fehlverhalten des Systems und seine Ursachen schneller zu finden.

3. Kurzer Überblick über das Petri-Netz-Modell

In der Arbeit von Clemens Düpmeier [Düpmeier92] wurde das unserer Spezifikations- und Modellierungssprache SNL (SMARAGD Net Language) zugrunde liegende Petri-Netz-Modell vorgestellt und der Petri-Netz-Teil von SNL formal durch eine kontextfreie Grammatik beschrieben. In meiner Arbeit wird die Analyse der auf SNL aufbauenden Netzmodelle beschrieben. Zum besseren Verständnis der Arbeit wollen wir die wichtigsten Definitionen aus [Düpmeier92] hier noch einmal kurz wiederholen. Die Definitionen aus [Düpmeier92] sind dabei in identischer Form übernommen. Darin verwendete Bezeichnungen oder Grundlagen können in [Düpmeier92] nachgelesen werden.

Wir müssen dazu zunächst definieren, was wir unter formalen Summen verstehen, denn diese werden wir als Beschriftung von Kanten und zur Angabe von Markierungen des Netzes verwenden. Im Anschluß daran beschreiben wir, was wir unter einer SNL-Netz-Spezifikation verstehen. Die Realisierung von SNL-Netz-Spezifikationen liefert uns dann SNL-Systeme (SNL-Modelle).

3.1 Formale Summen und Multimengen

Die Markierung eines traditionellen Petri-Netzes besteht aus einer Anzahl von ununterscheidbaren Marken (black token), die auf Stellen liegen können. In Netzen mit individuellen Marken können wir nun Objekte als Individuen unterscheiden. Trotzdem ist es sinnvoll, daß man in einer Stellenmarkierung mehr als ein Exemplar ein und desselben Individuums auf einer Stelle zuläßt. Eine geeignete Formalisierung des Begriffs des Mehrfachvorkommens von Individuen innerhalb einer Menge bietet der Begriff der Multimenge.

Definition 1 **(Multimenge)**

Sei A im folgenden eine beliebige Menge.

(a) Eine Funktion $\mu: A \longrightarrow \mathbb{Z}$ nennen wir im folgenden eine (*verallgemeinerte*) *Multimenge*. Die Menge aller Multimengen über der Menge A bezeichnen wir mit $MULT(A)$.

(b) Eine Funktion $\mu: A \longrightarrow \mathbb{N}$ nennen wir (*eigentliche*) *Multimenge*. Die Teilmenge aller eigentlichen Multimengen von $MULT(A)$ bezeichnen wir mit $MULT^+(A)$.

(c) Für $\mu \in MULT(A)$ bezeichnen wir mit $Tr(\mu)$ die Trägermenge von μ, d.h. die Menge aller $x \in A$, für die $\mu(x)$ ungleich 0 ist.

(d) $MULT_{fin}(A)$ bzw. $MULT^+_{fin}(A)$ seien die Teilmengen von $MULT(A)$ bzw. $MULT^+(A)$, die nur Funktionen mit endlicher Trägermenge enthalten.

Man kann sich eine eigentliche Multimenge $\mu: A \longrightarrow \mathbb{N}$ als eine Verallgemeinerung des Mengenbegriffes so vorstellen, daß μ jedes Element $a \in A$ $\mu(a)$ mal enthält. Auf Multimengen lassen sich die folgenden Operationen einführen.

Definition 2 **(Operationen auf Multimengen)**

Sei A wiederum eine beliebige Menge.

(a) Die *Addition von Multimengen* ist eine Operation $+: MULT(A) \times MULT(A) \longrightarrow MULT(A)$, die wie folgt definiert ist: Für je zwei $\mu, \nu \in MULT(A)$ und jedes $a \in A$ ist $(\mu+\nu)(a)=\mu(a)+\nu(a)$.

(b) Die *Subtraktion von Multimengen* ist eine Operation $-: MULT(A) \times MULT(A) \longrightarrow MULT(A)$, die wie folgt definiert ist: Für je zwei $\mu, \nu \in MULT(A)$ und jedes $a \in A$ ist $(\mu-\nu)(a)=\mu(a)-\nu(a)$.

(c) Die *Skalarmultiplikation einer Multimenge mit einer ganzen Zahl* ist eine Operation $\cdot: \mathbb{Z} \times MULT(A) \longrightarrow MULT(A)$, die für jedes $\mu \in MULT(A)$, $d \in \mathbb{Z}$ und alle $a \in A$ durch $(d\,\mu)(a)=d\,\mu(a)$ definiert ist.

Wir wollen einmal einige Rechenregeln im Umgang mit Multimengen zusammenstellen.

Bemerkung 1 **(Rechenregeln im Umgang mit Multimengen)**

Sei A eine Menge und $MULT(A)$ die Menge der Multimengen über A.

(a) $MULT(A)$ ist mit der oben eingeführten Addition und Skalarmultiplikation ein $\mathbb{Z}$-Modul, d.h. es gelten für alle $\mu, \nu, \delta \in MULT(A)$ und $d, e \in \mathbb{Z}$ die Regeln:

(1) $d(\mu+\nu)=(d\mu)+(d\nu)$, $(d+e)\mu=(d\mu)+(e\mu)$

(2) $(\mu+\nu)+\delta=\mu+(\nu+\delta)$

(3) $\mu+\nu=\nu+\mu$

(4) Mit $\overline{0}$ als die Multisumme mit $\overline{0}(a)=0$ für alle $a \in A$ haben wir $\overline{0}+\mu=\mu=\mu+\overline{0}$, $0\,\mu=\overline{0}$, $\mu+(-\mu)=(-\mu)+\mu=\overline{0}$

(b) Die Subtraktion läßt sich über die Addition durch $\mu-\nu=\mu+(-1)\nu$ darstellen.

$MULT_{fin}(A)$, $MULT^{+}(A)$ bzw. $MULT^{+}_{fin}(A)$ sind in natürlicher Weise in den $\mathbb{Z}$-Modul $MULT(A)$ eingebettet. $MULT_{fin}(A)$ ist dabei ein Untermodul von $MULT(A)$. Die Restriktionen von + und $\cdot$ bilden entsprechend Operationen auf $MULT^{+}(A)$ bzw. $MULT^{+}_{fin}(A)$, die nicht aus $MULT^{+}(A)$ bzw. $MULT^{+}_{fin}(A)$ herausführen; $MULT^{+}(A)$ bzw. $MULT^{+}_{fin}(A)$ sind allerdings keine Untermodule von $MULT(A)$, da es in der Regel kein inverses Element zu einer Multimenge μ aus $MULT^{+}(A)$ bzw. $MULT^{+}_{fin}(A)$ in $MULT^{+}(A)$ bzw. $MULT^{+}_{fin}(A)$ gibt.

Wir wollen noch eine Ordnungsrelation auf Multimengen einführen.

Definition 3 **(Halbordnung auf Multimengen)**

Die Ordnungsrelation $\leq$ auf $MULT(A)$ sei wie folgt definiert: Für alle $\mu, \nu \in MULT(A)$ gilt $\mu \leq \nu$ genau dann, wenn $\mu(a) \leq \nu(a)$ für alle $a \in A$ ist. Die so eingeführte Ordnungsrelation überträgt sich analog auf $MULT_{fin}(A)$, $MULT^{+}(A)$ bzw. $MULT^{+}_{fin}(A)$.

$\leq$ ist nur eine Halbordnung auf $MULT(A)$, da nicht alle Elemente von $MULT(A)$ miteinander vergleichbar sind. Ansonsten gelten die für Ordnungsrelationen bekannten Rechenregeln.

Bemerkung 2

Für alle $\mu, \nu, \delta \in MULT(A)$ gelten die Regeln:

(1) $\mu \leq \nu$ und $\nu \leq \mu \Rightarrow \mu=\nu$ (antisymmetrisch)

(2) $\mu \leq \nu$ und $\nu \leq \delta \Rightarrow \mu \leq \delta$ (transitiv)

(3) $\mu \leq \mu$ (reflexiv)

Wir wollen nun einmal spezielle Multimengen betrachten, die wir für ein zugehöriges Element $a \in A$ mit $`a \in MULT(A)$ bezeichnen und durch $`a(a)=1$ und $`a(b)=0$ für alle $b \in A$ mit $a \neq b$ definieren wollen. Jede Multimenge aus $MULT_{fin}(A)$ läßt sich dann eindeutig in der Form $\mu = \sum_{a \in Tr(\mu)} d_a \cdot `a$ darstellen, wobei $Tr(\mu)$ die endliche Trägermenge, d.h. die Menge aller $a \in A$ mit $\mu(a) \neq 0$, ist.

Definition 4 **(Formale Summenschreibweise für Multimengen)**

Sei A eine Menge.

(a) Für jedes $a \in A$ bezeichnen wir mit $`a$ die Multimenge über $A$, die durch $`a(a)=1$ sonst 0 definiert ist.

(b) Die Darstellung einer Multimenge $\mu \in MULT_{fin}(A)$ in der Form $\mu = \sum_{a \in Tr(\mu)} d_a \cdot `a$ mit $d_a \in \mathbb{Z}$ für alle $a \in Tr(\mu)$ bezeichnen wir als *formale Summenschreibweise* der Multimenge μ.

Offensichtlich läßt sich jedes Element $\mu \in MULT_{fin}(A)$ in so einer formalen Summenschreibweise darstellen. Wir werden diese Darstellung im folgenden häufig verwenden. Für den späteren Gebrauch benötigen wir den Begriff des Schnittes zweier Multimengen. Wir geben die Definition dafür in der formalen Summenschreibweise an.

Definition 5 **(Schnitt zweier Multimengen)**

Sei A eine Menge und $\mu, \nu \in MULT_{fin}(A)$ mit $\mu = \sum_{a \in Tr(\mu)} d_a \cdot `a$ und $\nu = \sum_{a \in Tr(\nu)} e_a \cdot `a$. Wir definieren den Schnitt von μ und ν als $\mu \cap \nu = \sum_{a \in Tr(\mu) \cap Tr(\nu)} \min(d_a, e_a) \cdot `a$.

In den nachfolgenden Kapiteln benötigen wir Multimengen über den Trägermengen eines algebraischen Systems. Elemente solcher Trägermengen werden dabei zunächst nur rein syntaktisch als geschlossene Terme einer zugehörigen Signatur dargestellt und als Elemente der Trägermengen eines zugehörigen algebraischen Systems interpretiert. Diese Trennung von Syntax und Semantik wollen wir nun auf Multimengen über Trägermengen eines algebraischen Systems übertragen.

Hierzu führen wir den Begriff einer *formalen Summe* ein, der an die formale Summenschreibweise angelehnt ist. Dabei werden grundlegende Begriffe aus der Logik und algebraischen Spezifikation verwendet, die in dieser Arbeit nicht eingeführt wurden (wie z.B. *SIG* und $TERM(SIG,V_S)$). Die entsprechenden Definitionen können in einführenden Büchern zur Logik und algebraischen Spezifikation, oder in der gewählten Terminologie in [Düpmeier92] nachgelesen werden.

Definition 6 **(Formale Summen)**

Sei $SIG=(S,\Omega,\Pi,typ_\Omega,typ_\Pi)$ eine Signatur, $V_S=(V_S(s))_{s\in S}$ eine Familie von Variablen zu den Sorten $s\in S$ und $TERM(SIG,V_S)$ die Menge der Terme über der Signatur *SIG* mit Variablen aus der Familie V_S.

(a) Die *Menge der formalen Summen über* $TERM(SIG,V_S)$ (kurz: $FSM(SIG,V_S)$) ist die kleinste Sprache, für die gilt:

(1) Für alle $t\in TERM(SIG,V_S)$ ist die Zeichenfolge $`t$ in $FSM(SIG,V_S)$. Dabei bindet der Operator ` stärker, als alle Operatoren und Prädikate innerhalb des Terms *t*.

(2) Für beliebige $\omega_1, \omega_2\in FSM(SIG,V_S)$ ist die Zeichenfolge $\omega_1+\omega_2$ in $FSM(SIG,V_S)$.

(b) Wir wollen noch einige abkürzende Schreibweisen einführen:

(1) $`t+....+`t$, wobei $`t$ n-mal (mit $n\in\mathbb{N}$) wiederholt wird, schreiben wir abkürzend als $n`t$.

(2) Für $\omega_1+\omega_2+.....+\omega_n$ schreiben wir abkürzend $\sum_{i=1}^{n}\omega_i$.

(c) Den *Typ einer formalen Summe* wollen wir rekursiv wie folgt definieren:

(1) Für $`t\in FSM(SIG,V_S)$ sei $typ(`t)=\{\ typ(t)\ \}\subseteq S$.
Dabei ist $typ(t)$ der Typ des Terms t (siehe [Düpmeier92]).

(2) Für $\omega=\sum_{i=1}^{n}\omega_i\in FSM(SIG,V_S)$ sei $typ(\omega)=\bigcup_{i=1}^{n} typ(\omega_i)\subseteq S$.

(3) Sei $\omega=\sum_{i=1}^{n} d_i`t_i\in FSM(SIG,V_S)$. Dann ist die *Menge der Variablen von* ω definiert durch

$var(\omega)=\bigcup_{i=1}^{n} var(t_i)$.

Dabei is $var(t_i)$ die Menge der Variablen des Terms t_i (siehe Düpmeier92]).

(4) Sei $\omega=\sum_{i=1}^{n} d_i`t_i\in FSM(SIG,V_S)$. Dann heißt $card(\omega)=\sum_{i=1}^{n} d_i$ die *Kardinalität von* ω. Für jede Sorte $s\in S$ heißt $card_s(\omega)=\sum_{typ(t_i)=s} d_i$ die *Kardinalität von* ω *bzgl. der Sorte s.*

Die Syntax von formalen Summen ist damit analog zur formalen Summenschreibweise von Multimengen. Wie interpretieren wir nun aber formale Summen?

Wenn wir ein algebraisches System $A=(A_S,A_\Omega,A_\Pi)$ als Modell von *SIG* haben, so können wir einen geschlossenen Term *t* mit $typ(t)=s$ als ein Element der Trägermenge $A_S(s)$ des algebraischen Systems *A* interpretieren. Ist dieses Element a_t, d.h. $t_R(v)=a_t$ für alle $v\in VAL(V_S,A_S)$, so liegt es nahe, daß wir $`t$ als die Multimenge $`a$ interpretieren.

Eine formale Summe $\omega=\sum_{i=1}^{n} d_i`t_i\in FSM(SIG,V_S)$, wobei alle $t_i$ für $i=1,..,n$ geschlossen sind, können wir dann analog als Multisumme $\sum_{i=1}^{n} d_i`a_{t_i}$ interpretieren.

Im allgemeinen enthalten aber Terme innerhalb von formalen Summen Variablen, so daß sie nicht geschlossen sind. Was ist nun in diesem Fall? Zunächst bemerken wir hierzu, daß im Fall eines geschlossenen Terms der Wert t_R nicht von der Valuation abhängt und wir diesen daher als eine konstante Funktion $t_R: VAL(V_S,A_S) \longrightarrow \bigcup_{s \in S} A_S(s)$ auffassen können. Wir können daher, wie dies in der Spezifikationstheorie üblich ist, die konstante Abbildung t_R mit der Konstante a_i identifizieren und unsere formale Summe über geschlossenen Termen als $\sum_{i=1}^{n} d_i \grave{} (t_i)_R$ schreiben. Den Ausdruck $\grave{t}_R$ können wir dann (auch wenn t nicht geschlossen ist) als eine Abbildung von der Menge der Valuationen $VAL(V_S,A_S)$ in die Menge der Multimengen über der Vereinigung der Trägermengen von A $\bigcup_{s \in S} A_S(s)$ interpretieren. Um dies zu definieren, führen wir zunächst die folgende abkürzende Schreibweise ein.

Bezeichnung:

Sei $A=(A_S,A_\Omega,A_\Pi)$ ein algebraisches System über einer Signatur SIG.

Wenn es zu keinen Mißverständnissen führt, schreiben wir im folgenden einfach A für die Vereinigung der Trägermengen von A. Insbesondere steht $MULT^+_{fin}(A)$ für $MULT^+_{fin}(\bigcup_{s \in S} A_S(s))$.

Wir definieren nun:

Definition 7 (Realisation von formalen Summen)

Sei $SIG=(S,\Omega,\Pi,typ_\Omega,typ_\Pi)$ eine Signatur, $V_S=(V_S(s))_{s \in S}$ eine Familie von Variablen zu den Sorten $s \in S$, $TERM(SIG,V_S)$ die Menge der Terme über der Signatur SIG mit Variablen aus der Familie V_S und $FSM(SIG,V_S)$ die Menge der formalen Summen von Termen aus $TERM(SIG,V_S)$. Weiter sei $A=(A_S,A_\Omega,A_\Pi)$ ein algebraisches System zur Signatur SIG.

Die *Realisation der formalen Summen* aus $FSM(SIG,V_S)$ im algebraischen System A ist eine Abbildung $\mathbf{R}_{FSM}: FSM(SIG,V_S) \to [VAL(V_S,A_S) \to MULT^+_{fin}(A)]$, die wie folgt definiert ist: Für jede formale Summe $\omega \in FSM(SIG,V_S)$ ist $\mathbf{R}_{FSM}(\omega)=\omega_R: VAL(V_S,A_S) \longrightarrow MULT^+_{fin}(A)$ durch die folgenden Regeln rekursiv bestimmt:

(1) Für $\omega = \grave{t}$ ist $\omega_R(v)=\grave{t}_R(v)$ für alle $v \in VAL(V_S,A_S)$.

(2) Für $\omega=\omega_1+\omega_2$ ist $\omega_R(v)=\omega_{1R}(v)+\omega_{2R}(v)$ für alle $v \in VAL(V_S,A_S)$. Dabei steht das +-Zeichen auf der rechten Seite für die Addition von Multimengen.

Eine formale Summe $\omega = \sum_{i=1}^{n} d_i \grave{} t_i \in FSM(SIG,V_S)$ interpretieren wir also als Abbildung

$$\omega_R = \sum_{i=1}^{n} d_i \grave{} t_{i\,R}: VAL(V_S,A_S) \longrightarrow MULT^+_{fin}(A).$$

Falls alle Terme innerhalb der formalen Summe geschlossen sind, ist die Abbildung ω_R konstant, und wir können sie in diesem Falle als Element von $MULT^+_{fin}(A)$, d.h. als Multimenge, auffassen.

Die formalen Summen von Termen sind "Ausdrücke" (im wesentlichen Terme einer Signatur) mit "Argumenten" einer Operation `, die selbst Terme über einer Signatur sind. Die Semantik dieser "Ausdrücke" haben wir dabei im wesentlichen unabhängig von der speziellen Semantik der "Argumentterme" beschrieben. Um eine solche Konstruktion wirklich formal in einem Logikkalkül durchführen zu können, müßten wir Variablen haben, deren Werte Terme sind, und einen Logikkalkül benützen, der auf solchen "Höheren Variablen" basiert. Dies wäre aber eine Sprache 2. Ordnung.

Wir wollen dies hier nicht näher präzisieren, sondern nur auf dieses unangenehme Merkmal der Theorie der Multimengen hinweisen. Wir begnügen uns hier mit der intuitiv verständlichen Beschreibung der Semantik von formalen Summen, wie sie durch Definition 7 gegeben ist.

Formale Summen bilden die Grundlage für die Spezifikation der Kantenbeschriftungen und Markierungen in unserem Netz-Modell. In SNL sind formale Summen über die folgende kontext-freie Grammatik definiert:

formal_sum: *base_sum* { + *base_sum* }$_0^*$
base_sum: [**num**] ` *term*

Als Basis dienen formale Summen der Gestalt `*term*, wobei *term* ein beliebiger Term über einer gegebenen Signatur und den Variablen einer Spezifikation ist. Diese Grundelemente für formale Summen können dann optional mit einem Faktor versehen werden, der eine natürliche Zahl bezeichnet. Einzelne Faktoren können schließlich durch das Summensymbol verknüpft werden. Der Operator `, der aus einem Term eine formale Summe erzeugt, hat dabei eine höhere Priorität, als jeder Operator innerhalb von Termen.

Beispiel 1

Um ein Beispiel für formale Summen angeben zu können, wollen wir eine parametrisierte Spezifikation eines Warteschlangendatentypes FIFO angeben.

```
specification ELEMENT = spec
     sort   elem;
end
functor FIFO ( E : ELEMENT ) = spec
     include  N : NAT;
     sort     fifo;
     const    empty : fifo;
     opn      put : fifo * E.elem -> fifo;
              get : fifo -> E.elem;
              rest : fifo -> fifo;
              length : fifo -> N.nat;
     var      w : fifo;
              n : E.elem;
     eqn      if (w != empty) then get(put(w,n))=get(w)
                              else get(put(w,n))=n;
              if (w != empty) then
                     rest(put(w,n))=put(rest(w),n)
                     else rest(put(w,n))=w;
              length(put(w,n))= N.++ length(w);
              length(empty)=N.0;
end
```

Wenn wir `w` als eine Variable vom Typ `fifo` auffassen, so sind `1`rest(w)` und `1`get(w)+2`get(rest(w))` zwei formale Summen vom Typ `fifo` bzw. `elem`.

3.2 SNL-Netz-Spezifikationen

In diesem Unterkapitel wollen wir das grundlegende Petri-Netz-Modell beschreiben. Es basiert (wie alle Petri-Netzformen) auf dem Begriff des Netzes.

Definition 8 **(Netz)**

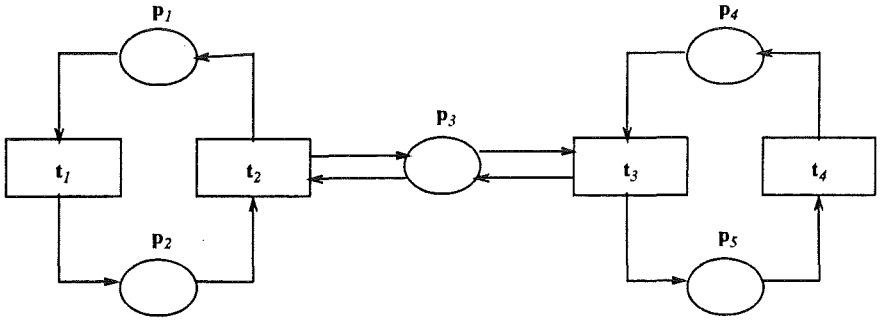
Ein Netz ist ein Tupel $N=(P,T,F)$, wobei P und T Mengen von *Stellen* und *Transitionen* mit der folgenden Eigenschaft sind:

(i) $P \cap T = \emptyset$
(ii) $P \cup T \neq \emptyset$
(iii) $F \subseteq P \times T \cup T \times P$
(iv) $dom(F) \cup cod(F) = P \cup T$

Man veranschaulicht sich ein Netz, indem man die Stellen als Kreise, die Transitionen als Rechtecke und die Elemente der Flußrelation $F \subseteq P \times T \cup T \times P$ als gerichtete Kanten darstellt. Betrachten wir hierzu ein Beispiel.

Beispiel 2

Sei $P=\{p_1,p_2,p_3,p_4,p_5\}$ die Menge der Stellen, $T=\{t_1,t_2,t_3,t_4\}$ die Menge der Transitionen und $F=\{(t_1,p_2),(p_2,t_2),(t_2,p_1),(p_1,t_1),(t_2,p_3),(p_3,t_2),(t_3,p_3),(p_3,t_3),(t_3,p_5),(p_5,t_4),(t_4,p_4),(p_4,t_3)\}$ die Flußrelation. Das folgende Bild zeigt dann das Netz in der Veranschaulichung:



Definition 9 **(Umgebungsstellen und Umgebungskanten)**

Sei $N=(P,T,F)$ ein Netz und $X=P\cup T$.

(a) Für jedes Element $x\in P\cup T$ definieren wir

$$\bullet x = \{y\in P\cup T \mid yFx\},\quad x\bullet = \{y\in P\cup T \mid xFy\}$$

Wenn $x\in T$ ist, nennen wir Elemente aus $\bullet x$ *bzw.* $x\bullet$ *Vor-* bzw. *Nachstellen der Transition x.* Die Menge der Vor- und Nachstellen einer Transition x bezeichnen wir mit $\bullet x\bullet$ und definieren $\bullet x\bullet = \bullet x \cup x\bullet$.

(b) Für jedes $x\in X$ definieren wir die *Menge der Umgebungsstellen von x* durch

$$penv(x) = \begin{cases} \{p\} & falls\ x=p \in P \\ \bullet t\bullet & falls\ x=t \in T \end{cases}$$

(c) Für jedes $x\in X$ heißt $x^\succ=\{f\in F \mid$ es gibt ein $y\in X$ mit $f=(x,y)\}$ die *Menge der Ausgabekanten* und $^\prec x=\{f\in F \mid$ es gibt ein $y\in X$ mit $f=(y,x)\}$ die *Menge der Eingabekanten von x.* $^\prec x^\succ = {}^\prec x \cup x^\succ$ heißt die *Menge der Umgebungskanten von x.*

Im folgenden werden nur solche Netze betrachtet, die die folgende Bedingung erfüllen:

- Die Mengen P und T sind endlich.

In der Praxis wünscht man sich Netze, die eine problemnahe Modellierung erlauben. Da der Anwender es mit konkreten Objekten und Abläufen auf diesen Objekten zu tun hat, wünscht er sich eine Netzform, in der "individuelle Objekte" aus seiner Problemwelt und die zugehörigen Aktionen auf den Objekten zur Modellierung verwendet werden können. Dies führte zu einer Reihe von Petri-Netzformen mit individuellen Marken (Objekten) und Beschriftungen, die Operationen und Bedingungen bzgl. der Objekte beinhalten.

In unserem Netz-Modell werden Objekte und die auf ihnen durchführbaren Operationen durch abstrakte Datentypen beschrieben. Hierbei ist es sinnvoll, die Spezifikation eines abstrakten Datentyps von einer Realisierung des Datentyps zu unterscheiden. Entsprechend wollen wir in unserem Netz-Modell zwischen der Spezifikation eines Netzes und möglicher ausführbarer Modelle dieser Spezifikation (Realisierungen der Netz-Spezifikation) unterscheiden. Wir wollen daher zunächst einmal den Begriff einer SNL-Netz-Spezifikation einführen. Diese versieht ein Netz mit einer Beschriftung, die als Bauelemente Elemente der Spezifikation eines abstrakten Datentyps verwendet.

Definition 10 **(SNL-Netz-Spezifikation)**

Es sei $N = (P,T,F)$ ein Netz, $SIG=(S,\Omega,\Pi,typ_\Omega,typ_\Pi)$ eine Signatur, $V_S=(V_S(s))_{s\in S}$ eine Familie von Variablen, $L = (ALPH, TERM(SIG,V_S), FORM(LSIG,SIG,V_S))$ eine Sprache der ersten Ordnung über SIG mit Variablen aus V_S und $SPEC(SIG,E)$ eine algebraische Spezifikation über SIG mit Formeln aus L (d.h. $E\subseteq FORM(LSIG,SIG,V_S)$).

(a) Eine Beschriftung des Netzes N in der Sprache L ist ein 3-Tupel $A_N=(A_P,A_T,A_F)$, wobei gilt:

(1) $A_P=(cap, vars, eqn)$ ist ein 3-Tupel von Abbildungen $cap:P\longrightarrow[S\longrightarrow\mathbb{N}_0]$, $vars:P\longrightarrow\wp(V_S)$ und $eqn:P\longrightarrow FORM(LSIG,SIG,V_S)$.

cap ordnet jeder Stelle $p\in P$ eine *Kapazitätsfunktion* $cap(p)=cap_p:S\longrightarrow\mathbb{N}_0$ zu, wobei $cap(p)(s)=cap_p(s)= cap(p,s)$ die *Kapazität der Stelle p bzgl. der Sorte* $s\in S$ ist. *vars* ordnet jeder Stelle $p\in P$ eine *Familie von Variablen* $vars(p)=(vars_s(p))_{s\in S}\subseteq\wp(V_S)=(\wp(V_S(s))_{s\in S}$ zu. *eqn* ordnet schließlich jeder Stelle eine *Formel* $eqn(p)$ zu.

(2) $A_T=(fvars, bvars, eqn)$ ist ein 3-Tupel von Abbildungen $fvars:T\longrightarrow\wp(V_S)$, $bvars:T\longrightarrow\wp(V_S)$ und $eqn:T\longrightarrow FORM(LSIG,SIG,V_S)$. *fvars* ordnet jeder Transition $t\in T$ eine *Familie von freien Variablen* $fvars(t)=(fvars_s(t))_{s\in S}\subseteq\wp(V_S)$ und *bvars* ordnet jeder Transition $t\in T$ eine *Familie von gebundenen Variablen* $bvars(t)=(bvars_s(t))_{s\in S}\subseteq\wp(V_S)$ zu.
eqn ordnet jeder Transition eine *Transitionsformel* $eqn(t)\in FORM(LSIG,SIG,V_S)$ zu.

(3) $A_F:F\longrightarrow FSM(SIG,V_S)$ ordnet jeder Kante des Netzes eine formale Summe zu.

(4) $typ(vars(p))\subseteq\{\, s\in S \mid cap_p(s)>0 \,\}$ für alle $p\in P$.

(5) $eqn(p)$ ist für alle $p\in P$ eine geschlossene Formel mit $vars(eqn(p))\subseteq vars(p)$. Dabei sei $vars(eqn(p))$ die Menge der Variablen in der Formel $eqn(p)$.

(6) $\forall p\in P$ und $f\in {}^\bullet p^\bullet$ gilt: $typ(A_F(f))\subseteq\{\, s\in S \mid cap_p(s)>0 \,\}$.

(7) $\forall t\in T$ und $f\in {}^\bullet t^\bullet$ gilt: $fvars(A_F(f))\subseteq fvars(t)$.

(8) $\forall t\in T$ gilt: $fvars(t)\cap bvars(t)=\emptyset$.

(9) $\forall t\in T$ gilt: $vars(t):=fvars(t)\cup bvars(t)\subseteq \bigcup_{p\in penv(t)} vars(p)$.

(10) $\forall t\in T$ gilt: $fvars(eqn(t))\subseteq fvars(t)$ und $bvars(eqn(t))\subseteq bvars(t)$. Dabei sei $fvars(eqn(t))$ die Menge der freien Variablen und $bvars(eqn(t))$ die Menge der gebundenen Variablen in der Formel $eqn(t)$.

(b) Sei A_N eine Beschriftung wie in (a) definiert. Eine Familie von formalen Summen $M_f=(m_p^f)_{p\in P}$ heißt eine *(formale) Markierung eines beschrifteten Netzes* (N,A_N), falls gilt:

(1) Für alle $p\in P$ ist $m_p^f\in FSM(SIG,V_S)$.

(2) $vars(m_p^f)=\emptyset$ für alle $p\in P$.

(3) $0\le card_s(m_p^f)\le cap_p(s)$ für alle $s\in S$ und $p\in P$.

(c) Seien A_N eine Beschriftung und M_0^f eine Markierung wie in (a) und (b) definiert. Das Tripel $NSPEC=(N,A_N,M_0^f)$ heißt dann eine *SNL-Netz-Spezifikation mit Anfangsmarkierung* M_0^f *über der algebraischen Spezifikation SPEC.*

Wir wollen im Kontext der obigen Definition noch einige Bezeichnungen einführen:

Definition 11 **(Der Begriff der Umgebung)**

Es seien die Voraussetzungen der vorigen Definition gegeben, und es sei $X=P\cup T$.

(a) Für alle $p\in P$ definieren wir die *Menge der Sorten von p* durch $sorts(p) = \{ s\in S \mid cap_p(s)>0 \}$.

(b) Für jedes $x\in X$ sei $senv(x)= \bigcup_{p \in penv(x)} sorts(p)$ die *Menge der Sorten in der Umgebung von x.*

(c) Für jedes $x\in X$ sei $venv(x)= \bigcup_{p \in penv(x)} vars(p)$ die *Familie der Variablen in der Umgebung von x.*

(d) Für jedes $p\in P$ sei *die Familie der freien Variablen von p* definiert durch $fvars(p):=\emptyset$, da Formeln auf Stellen per Definition geschlossen sind.

(e) Für alle $t\in T$ sei die Familie der Variablen von t definiert durch $vars(t)=(vars_s(t))_{s\in S}=(fvars_s(t)\cup bvars_s(t))_{s\in S}$.

(f) Für jedes $x\in X$ nennen wir die durch $penv(x)$, $^{\bullet}x^{\bullet}$, $senv(x)$ und $venv(x)$ gegebenen Informationen die *Umgebung von x in der SNL-Netz-Spezifikation NSPEC.*

Wir wollen Definition 10 nun ein wenig näher erläutern. Der Punkt (a) der Definition legt die Beschriftung eines Netzes fest, die aus einer Reihe von Abbildungen besteht. Jeder Stelle $p\in P$ ist zunächst über die Abbildung *cap* eine Menge von Sorten $sorts(p)$ (siehe Definition 11 (a)) zugeordnet. Wir können uns eine Stelle p als Speicher vorstellen, der Multimengen von Objekten enthält, wobei der Typ jedes Objektes einer der Sorten aus $sorts(p)$ entsprechen muß. Die maximale Anzahl der Objekte einer Sorte s auf der Stelle p wird durch die Kapazität $cap_p(s)$ bestimmt. Eine Kapazität 0 bedeutet, daß Objekte dieser Sorte nicht auf der Stelle vorkommen dürfen.

Weiter ist jeder Stelle p eine Menge von Variablen $vars(p)$ zugeordnet, deren Typen in $sorts(p)$ enthalten sein müssen (Bedingung 4). Diese Variablen können dann in Formeln bzgl. der Stelle p bzw. innerhalb der Formeln von Transitionen, zu deren Umgebung die Stelle gehört, verwendet werden (siehe etwas weiter unten).

In der Praxis benötigt man in Netzen in der Regel abstrakte Datentypen, deren Individuenbereiche in einer Realisierung ebenfalls endlich sind. Dies garantiert man durch Formeln innerhalb der Spezifikation, die die Individuenbereiche beschränken. In einem Beispiel könnte die Länge der erlaubten Warteschlangen z.B. durch eine Formel $length(w)<maxf$ beschränkt sein. Dabei erscheint es natürlich, daß man so eine Formel in einem Netz einer Stelle analog der Kapazitätsfunktion zuordnet. Wir erlauben es daher, daß man Stellen p optional eine geschlossene Formel $eqn(p)$ zuordnet, die nur Variablen enthalten darf, die dieser Stelle zugeordnet sind (Bedingung 5). In einer Realisierung des Netzes sind dann nur Markierungen erlaubt, die die Formeln auf den Stellen erfüllen.

Jeder Transition $t\in T$ ist eine Formel $eqn(t)$ zugeordnet. Diese Formel stellt eine Bedingung dar, die erfüllt werden muß, damit eine Transition schalten kann. Die Menge der freien Variablen einer Transitionsformel $eqn(t)$ muß dabei in der Familie der freien Variablen der Transition $fvars(t)$ und die Menge der gebundenen Variablen der Transitionsformel muß in der Familie der gebundenen Variablen $bvars(t)$ enthalten sein. Dies ist Bedingung 10. Die Familien der freien und gebundenen Variablen müssen weiter disjunkt sein, und jede Variable (frei und gebunden) einer Transition muß mindestens einer Umgebungsstelle der Transition zugeordnet sein (Bedingungen 8 und 9).

Jede Kante ist mit einer formalen Summe beschriftet, deren Variablen in der Menge aller freien Variablen der zugehörigen Transition enthalten sein müssen (Bedingung 7). Diese formalen Summen definieren den Fluß der Objekte durch das Netz, d.h. nach Substitution der Variablen liefern sie in der Realisation der Netzspezifikation Multimengen, die beschreiben, welche Objekte von einer Stelle abzuziehen bzw. welche Objekte auf eine Stelle zu legen sind. Da beim Schalten einer Transition nur Objekte auf eine Stelle gelegt bzw. von einer Stelle abgezogen werden, die zu Sorten gehören, für die die Stelle auch einen Speicher darstellt, muß Bedingung 6 gelten.

Bedingung (9) stellt eine Anforderung an das Schalten eines Netzes dar, die man als Lokalitätsprinzip bezeichnen kann. Dieses Prinzip verlangt, daß das Schalten einer Transition (also das "ob" und "wie" eine Transition schaltet) nur vom Zustand des Systems in der Umgebung der Transition, nicht aber von anderen Faktoren abhängen darf. Wir verlangen daher in (9), daß die Familien der Variablen einer Transition t in der Menge der Umgebungsvariablen $venv(t)$ von t enthalten sein muß. Wir werden die Variablen der Transitionen dann über den Inhalt der Umgebungsstellen der betreffenden Transition auswerten. Die Werte der Transitionsbeschriftungen und Kantenbeschriftungen hängen damit nur von dem Inhalt der Umgebungsstellen, also vom aktuellen Zustand (der aktuellen Markierung) in der Umgebung der betreffenden Transition, ab.

(10) und (7) geben an, daß das Schalten einer Transition t nur durch eine Auswertung (Zuweisung von Werten) der freien Variablen einer Transition t ($fvars(t)$) bestimmbar ist. Bezüglich so einer Auswertung ist die Formel $eqn(t)$ entweder *true* oder *false* und bestimmt damit, ob die Bedingung für das Schalten der Transition bezüglich dieser Auswertung erfüllt ist oder nicht. Einsetzen der Auswertung in die Kantenbeschriftung ergibt die Multimenge von Objekten, die abzuziehen bzw. auf die Nachfolgestellen zu legen sind.

Um ein Beispiel einer SNL-Netz-Spezifikation geben zu können, ist es notwendig, die mathematische Definition einer SNL-Netz-Spezifikation in die Sprache SNL zu transformieren. In [Düpmeier92] wurde diese Transformation durch Angabe einer kontextfreien Grammatik für SNL ausführlich beschrieben, weshalb wir diese Transformation hier nicht noch einmal wiederholen wollen. Um einen Eindruck über SNL zu vermitteln, verwenden wir im folgenden Beispiel SNL als Beschriftungssprache.

Beispiel 3

Sei $N = (P,T,F)$ das Netz aus Beispiel 2. Wir wollen zunächst einmal eine Spezifikation PROCESS einführen, die ein System von Prozessen beschreibt, die auf eineindeutige Weise Datenpakete produzieren.

```
specification PROCESS = spec
      sort     process; data;
      opn      mdata : process -> data;
               proc : data -> process;
      var      pr : process;
               d : data;
      eqn      mdata(proc(d))=d;
               proc(mdata(pr))=pr;
end
```

Weiter führen wir eine Spezifikation NAT ein.

```
specification NAT = spec
     sort      nat;
     const     0 : nat;
     opn       ++ : nat -> nat;
               + : nat * nat -> nat;
     pred      <= : nat * nat;
     infix     + 6; <= 4;
     var       m : nat; n : nat;
     eqn       n + 0 = n;
               n + ++ m = ++(n+m);
               m <= m + n;
               not n + (++ m) <= n;
end
```

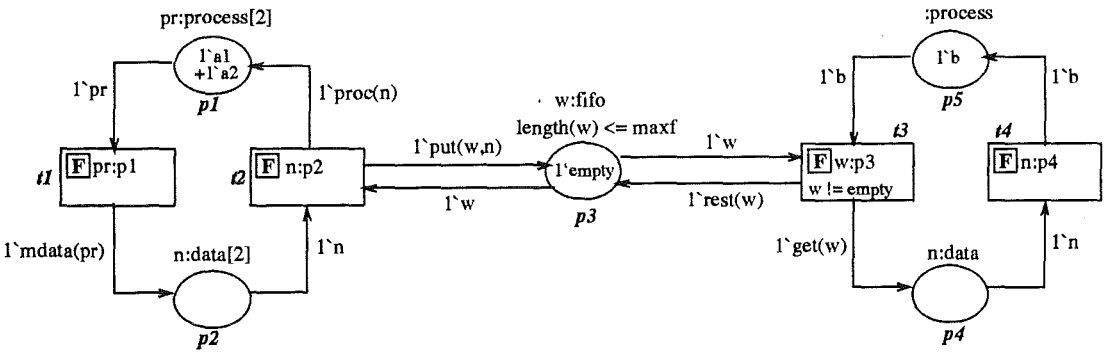
Auf Basis dieser Spezifikationen und der Spezifikation FIFO aus Beispiel 1 wollen wir dann die Spezifikation für unser Beispielnetz wie folgt definieren:

```
petrinet ERZEUGER-VERBRAUCHER = net
     open      NAT; PROCESS;
               FIFO(spec elem=PROCESS.data; end);
     const     a1 : process; a2 : process;
               b:process;
               c1 : data; c2 : data;
               maxf : nat;
     placedef  p1=place
                   var pr:process[2];
                   marking 1`a1+1`a2;
               end;
               p2=place
                   var n:data[2];
               end;
               p3=place
```

```
                    var w:fifo;
                    eqn length(w) <= maxf;
                    marking 1`empty;
                end;
                p4=place
                    var n:data;
                end;
                p5=place
                    var: process;
                    marking 1`b;
                end;
        transitiont1=trans
                    fvar pr:p1;
                    arc p1 -> 1`pr;
                        p2 <- 1`mdata(pr); end;
                t2=trans
                     fvar n:p2; w:p3;
                     arc p2 -> 1`n;
                         p1 <- 1`proc(n);
                         p3 -> 1`w;
                         p3 <- 1`put(w,n); end;
                t3=trans
                    fvar w:p3;
                    arc p3 -> 1`w;
                        p3 <- 1`rest(w);
                        p4 <- 1`get(w);
                        p5 -> 1`b;
                    eqn w != empty; end;
                t4=trans
                    fvar n:p4;
                    arc p4 -> 1`n;
                        p5 <- 1`b; end;
    end
```

Im Erzeugerzyklus haben wir zwei Prozesse `1`a1` und `1`a2`, die Daten `1`mdata(a1)` und `1`mdata(a2)` erzeugen können. Diese werden über die Transition `t2` in die Warteschlange auf der Stelle `p3` über einen Aufruf von `put` eingetragen. Auf der Verbraucherseite gibt es einen Prozeß `b`, der die Daten über einen `get` Aufruf über die Transition `t3` aus der Warteschlange auf `p3` entfernt und dann in `t4` verarbeitet.

Man beachte dabei die Formel `length(w)<= maxf` auf der Stelle `p3`. Diese blockiert automatisch das Schalten der Transition `t2`, wenn die Warteschlange `maxf` Elemente besitzt. Weiter beachte man die Formel `w != empty` auf der Transition `t3`. Diese verhindert, daß `t3` schaltet, wenn die Warteschlange leer ist. Die folgende Abbildung verdeutlicht die Netz-Spezifikation ERZEUGER-VERBRAUCHER:



In diesem Bild haben wir die Beschriftung des Netzes in der Form angegeben, die wir in unserer Sprachdefinition definiert haben. Die Angabe der freien Variablen haben wir zusätzlich durch ein Quadrat mit der Inschrift F markiert, um sie eindeutig von einer möglichen Angabe von gebundenen Variablen zu unterscheiden.

3.3 SNL-Systeme (SNL-Modelle)

In diesem Unterkapitel wollen wir nun die Semantik beschreiben, die durch eine SNL-Netz-Spezifikation gegeben ist. Die Idee dabei ist, daß wir die Netzbeschriftung bzgl. algebraischer Systeme, die Modelle der Spezifikationen innerhalb der SNL-Netz-Spezifikation sind, interpretieren. Dies wollen wir im folgenden genauer beschreiben.

Um die semantischen Eigenschaften eines SNL-Systems definieren zu können, müssen wir zunächst die Semantik der Beschriftung der SNL-Netz-Spezifikation definieren. Hierzu ist zunächst eine Realisation der algebraischen Spezifikation in der SNL-Netz-Spezifikation über ein algebraisches System A und eine logische Matrix $\overline{A}$ notwendig. Auf deren Grundlage interpretieren wir dann die Terme, formalen Summen und elementaren Formeln innerhalb der SNL-Netz-Spezifikation über die uns aus [Düpmeier92] bekannten Realisierungen $\mathbf{R}_T$, $\mathbf{R}_{FSM}$ und $\mathbf{R}_{EF}$. Die Formeln im Spezifikationsteil einer SNL-Netz-Spezifikation interpretieren wir ebenfalls über die aus [Düpmeier92] bekannte Realisation $\mathbf{R}_F$.

Im Gegensatz zur Realisation von Formeln mit Quantoren im Spezifikationsteil einer SNL-Netz-Spezifikation, die wir schon im Spezifikationsteil dieses Berichtes definiert haben, wollen wir die Quantoren in Formeln auf den Transitionen und Stellen nicht als All- und Existenzquantor auf den zugehörigen Wertebereichen von A, sondern gemäß unserem Lokalitätsprinzip als All- und Existenzquantor über der Menge der Individuen der jeweiligen Sorte, die bezüglich der aktuellen Markierung des Netzes auf den Umgebungsstellen einer Transition liegen, auffassen. Wir erhalten hiermit eine Interpretation der Formeln, die von der Markierung im Netz abhängt.

Bevor wir diese Interpretation von Netz-Formeln formal definieren, wollen wir zunächst noch einige Hilfsbegriffe einführen.

Definition 12 (Hilfsbegriffe)

Sei $NSPEC=(N,A_N,M_0)$ eine SNL-Netz-Spezifikation und $A=(A_S,A_\Omega,A_\Pi)$ ein Modell der zu $NSPEC$ gehörenden algebraischen Spezifikation $SPEC$. $M=(m_p)_{p\in P}$ sei weiter eine Familie von Multimengen, die jeder Stelle $p\in P$ des Netzes eine Multisumme $m_p\in MULT_{fin}(A)$ über den Trägermengen von A zuordnet, wobei $typ(m_p)\subseteq sorts(p)$ ist. Weiter sei $X=P\cup T$.

(a) Mit $Tr_s(m_p)$, $s\in sorts(p)$, bezeichnen wir die Teilmenge der Elemente von $Tr(m_p)$, die vom Typ s sind.

(b) Für $t\in T$, $v\in vars(t)$ sei $penv_v(t):=\{\ p\in {}^\bullet t^\bullet \mid v\in vars(p)\ \}$. Für $p\in P$, $v\in vars(p)$ sei $penv_v(p)=\{p\}$.

(c) Sei $x\in X$ und $v\in vars(x)$. Mit $obj_{v,x}(M)$ bezeichnen wir die Menge aller Objekte der Sorte $typ(v)$, die bzgl. der *Familie M in der Umgebung von x* liegen, d.h. $obj_{v,x}(M)=\bigcup_{p\in penv_v(x)} Tr_s(m_p)$.

(d) Die Menge der Formeln in der Netz-Beschriftung einer SNL-Netz-Spezifikation *NSPEC* bezeichnen wir mit *NEQN(NSPEC)*. Die Menge der Formeln im Spezifikationsteil bezeichnen wir mit *EQN(NSPEC)*.

(e) Sei $t \in T$. Die Menge der Auswertungen der freien Variablen von t in A bezeichnen wir mit $VAL_t(NSPEC,A) := \{ v := (v_s : fvars_s(t) \longrightarrow A_S(s))_{s \in S} \}$. Für $p \in P$ definieren wir $VAL_p(NSPEC,A) := \emptyset$.

(f) Sei $t \in T$ und $v \in VAL_t(NSPEC,A)$. v heißt *Auswertung der freien Variablen von t unter M in A*, falls für alle $y \in fvars(t)$ gilt: $v(y) \in obj_{y,t}(M)$. Wir bezeichnen die Menge aller Auswertungen der freien Variablen von t unter M in A mit $VAL_t(NSPEC,M)$. Für jedes $p \in P$ definieren wir $VAL_p(NSPEC,A) := \emptyset$.

Wir definieren nun die Realisation von Formeln in der Netzbeschriftung einer SNL-Netz-Spezifikation wie folgt:

Definition 13 (Realisation der Formeln einer SNL-Netz-Spezifikation)

Seien *SIG*, *LSIG*, V_S, wie in [Düpmeier92] eingeführt, gegeben. L sei eine formale Sprache 1. Ordnung über *SIG* und *LSIG*. Sei $NSPEC=(N,A_N,M_0)$ eine SNL-Netz-Spezifikation über *SIG* und *LSIG* mit Formeln aus L, $A=(A_S,A_\Omega,A_\Pi)$ ein algebraisches System über der Signatur *SIG* und $\overline{A}=(A_S,A_{\overline{\Omega}})$ eine logische Matrix zu der Signatur *LSIG*. $M=(m_p)_{p \in P}$ sei eine Familie von Multimengen, die jeder Stelle des Netzes eine Multimenge $m_p \in MULT_{fin}(A)$ über den Trägermengen von A zuordnet, wobei für alle $p \in P$ $typ(m_p) \subseteq sorts(p)$ ist. Weiter sei $\rho_\Pi = (\rho_\pi : A_S^{typ_\Pi(\pi)} \rightarrow A_{\overline{S}}(\overline{s}))_{\pi \in \Pi}$ eine Familie von Funktionen,

$\mathbf{R}_T : TERM(SIG,V_S) \longrightarrow [VAL(V_S,A) \longrightarrow \cup_{s \in S} A_S(s)]$ die durch A eindeutig bestimmte Realisation von Termen und $\mathbf{R}_{EF} : FORM(SIG,V_S) \longrightarrow [VAL(V_S,A) \longrightarrow A_{\overline{S}}(\overline{s})]$ die durch A, $\overline{A}$ und ρ_Π eindeutig bestimmte Realisation von atomaren Formeln.

Die durch A, $\overline{A}$ und ρ_Π eindeutig bestimmte *Realisation der Formeln der Netzbeschriftung einer SNL-Netz-Spezifikation NSPEC bzgl. M* ist eine Abbildung $\mathbf{R}_N$, die jeder Formel $\alpha \in NEQN(NSPEC)$ zu $x \in X$ eine Abbildung $\mathbf{R}_N(\alpha) = \alpha_\mathbf{R} : VAL_x(NSPEC,M) \longrightarrow A_{\overline{S}}(\overline{s})$ zuordnet, die wie folgt definiert ist:

Im folgenden sei $v \in VAL_x(NSPEC,M)$ eine beliebige Auswertung der freien Variablen von $x \in X$ unter M in A.

(1) Wenn $\alpha = \pi \in FORM(SIG,V_S)$ eine atomare Formel ist, dann ist $\alpha_\mathbf{R} = \mathbf{R}_{EF}(\alpha)|_{VAL_x(NSPEC,A)}$.

(2) Wenn α eine Formel der Gestalt $o^i(\beta)$ $(i \leq l)$ mit $o^i \in L_l$ ist, und der Wert $\beta_\mathbf{R}(v)$ definiert ist, dann ist $\alpha_\mathbf{R}(v) = A_{\overline{\Omega}}(o^i)(\beta_\mathbf{R}(v))$.

(3) Wenn α eine Formel der Gestalt $(\beta) o_i (\gamma)$ $(i \leq m)$ mit $o_i \in L_2$ ist, und die Werte $\beta_\mathbf{R}(v)$ und $\gamma_\mathbf{R}(v)$ definiert sind, dann ist $\alpha_\mathbf{R}(v) = A_{\overline{\Omega}}(o_i)(\beta_\mathbf{R}(v), \gamma_\mathbf{R}(v))$.

(4) Wenn α eine Formel der Gestalt $\forall_\zeta \beta$ ist, so gilt:

  (1) wenn die gebundene Variable $\zeta \in bvars(x)$ in der Formel β nicht vorkommt, so ist $(\forall_\zeta \beta)(v) = \beta_\mathbf{R}(v)$;

  (2) wenn die Variable $\zeta \in bvars(x)$ in der Formel β vorkommt, dann ist $(\forall_\zeta \beta)(v) = inf_{a \in obj_{\zeta,x}(M)} \{ \beta_\mathbf{R}(\zeta/a)(v) \}$, wobei $\beta(\zeta/a)$ eine Formel ist, die aus β durch die Substitution $\zeta \longrightarrow a$ mit $a \in obj_{\zeta,x}(M)$ entsteht, und *inf* die Operation der unteren Schranke in der Algebra $\overline{A}$ ist.

(5) Wenn α eine Formel der Gestalt $\exists_\zeta \beta$ ist, so gilt:

  (1) wenn die gebundene Variable $\zeta \in bvars(x)$ in der Formel β nicht vorkommt, so ist $(\exists_\zeta \beta)(v) = \beta_\mathbf{R}(v)$;

  (2) wenn die Variable $\zeta \in bvars(x)$ in der Formel β vorkommt, dann ist $(\exists_\zeta \beta)(v) = sup_{a \in obj_{\zeta,x}(M)} \{ \beta_\mathbf{R}(\zeta/a)(v) \}$, wobei $\beta(\zeta/a)$ eine Formel ist, die aus β durch die Substitution $\zeta \longrightarrow a$ mit $a \in obj_{\zeta,x}(M)$ entsteht, und *sup* die Operation der oberen Schranke in der Algebra $\overline{A}$ ist.

Man beachte, daß damit die Interpretation der Quantoren und damit der Formeln der Netzbeschriftung von der Familie $M=(m_p)_{p\in P}$ abhängt. Diese Interpretation der Formeln ist unseres Wissens nach neu in der Petri-Netz-Theorie, ist jedoch nur eine konsequente Anwendung des Lokalitätsprinzips, da der Zustandsübergang nur vom momentanen Zustand des Systems in der Umgebung einer Transition abhängig sein soll. Der Zustand des Systems in der Umgebung einer Transition wird aber gerade durch die Markierung der Umgebungsstellen bestimmt, und die einzigen Individuen, die die Transition kennt, sind daher die Objekte, die auf den Umgebungsstellen liegen. Logischerweise sollten die Quantoren nur über diese Objekte interpretiert werden.

Definition 14 (Realisation einer SNL-Netz-Spezifikation)

Es seien die Voraussetzungen wie in Definition 13 gegeben. Das Viertupel $\mathbf{R}=(\mathbf{R}_T, \mathbf{R}_F, \mathbf{R}_{FSM}, \mathbf{R}_N)$ nennen wir dann eine *Realisation der SNL-Netz-Spezifikation NSPEC.*

Da aus dem Kontext stets ersichtlich ist, welche der Abbildungen $\mathbf{R}_T$, $\mathbf{R}_F$, $\mathbf{R}_{FSM}$ bzw. $\mathbf{R}_N$ anzuwenden ist, werden wir im folgenden kurz **R** für $\mathbf{R}_T$, $\mathbf{R}_F$, $\mathbf{R}_{FSM}$ bzw. $\mathbf{R}_N$ schreiben.

Wenn wir eine Sprache L über der Signatur *LBOOL* haben, können wir diese in Bezug auf die klassische boolesche Matrix *A-BOOL* realisieren. Die Realisation von Formeln aus *EQN(NSPEC)* und *NEQN(NSPEC)* hängt dann nur noch von dem algebraischen System A und bei Formeln aus *NEQN(NSPEC)* noch von der aktuellen Markierung ab. Wir sprechen dann von einer *(prädikatenlogischen) SNL-Netz-Spezifikation (1. Ordnung)* und einer Realisierung bzgl. der Prädikatenlogik 1. Ordnung. Die Formeln von SNL-Spezifikationen sind stets solche prädikatenlogischen Formeln 1. Ordnung, und Realisierungen sind bzgl. unserer Sprache stets schon eindeutig durch das algebraische System A bestimmt.

Definition 15 (Erfüllbarkeit von Formeln einer SNL-Netz-Spezifikation)

Es seien die Voraussetzungen wie in Definition 13 gegeben, und $\mathbf{R}=(\mathbf{R}_T, \mathbf{R}_F, \mathbf{R}_{FSM}, \mathbf{R}_N)$ sei eine Realisation der SNL-Netz-Spezifikation *NSPEC* über A, $\overline{A}$ und ρ_Π.

(a) Sei $\alpha\in NEQN(NSPEC)$ eine beliebige Formel von $x\in X$. Die Formel α ist *vermöge* einer Auswertung $v\in VAL_x(NSPEC,M)$ *bzgl. der Realisation* **R** *unter M erfüllt*, wenn $\alpha_{\mathbf{R}}(v)=v$ gilt. Wir schreiben dies als $M \models_{v,\mathbf{R}} \alpha$.

(b) Die Formel $\alpha\in NEQN(NSPEC)$ aus der Umgebung von $x\in X$ heißt *erfüllbar unter M bzgl. der Realisation* **R**, wenn es eine Auswertung $v\in VAL_x(NSPEC,M)$ gibt, vermöge der α bzgl. **R** erfüllt ist.

(c) Die Formel $\alpha\in NEQN(NSPEC)$ aus der Umgebung von $x\in X$ heißt *gültig unter M bzgl. der Realisation* **R**, wenn für jede Auswertung $v\in VAL_x(NSPEC,M)$ gilt: α ist vermöge v bzgl. **R** unter M erfüllt. Wir schreiben dies als $M \models_{\mathbf{R}} \alpha$.

(d) *M erfüllt die SNL-Netz-Spezifikation NSPEC bzgl. der Realisation* **R** (im Zeichen $M \models_{\mathbf{R}} NSPEC$), falls gilt:

(1) Für alle $p\in P$ gilt: $M \models_{\mathbf{R}} eqn(p)$ (Formeln auf Stellen erfüllt).

(2) $0\le card_s(m_p) \le cap_p(s)$ für alle $s\in S$ und $p\in P$.

Wir lassen im folgenden bei der Schreibweise $\models$ den Suffix **R** weg, wenn die Realisation einer SNL-Netz-Spezifikation *NSPEC* aus dem Kontext ersichtlich ist.

Mit der obigen Definition können wir nun definieren, was wir unter einem SNL-System (auch SNL-Modell) und Markierungen solcher SNL-Systeme verstehen.

Definition 16 (SNL-Systeme (SNL-Modelle) und deren Anfangsmarkierung)

Es seien die Voraussetzungen wie in Definition 13 gegeben und $\mathbf{R}=(\mathbf{R}_T, \mathbf{R}_F, \mathbf{R}_{FSM}, \mathbf{R}_N)$ eine Realisation der SNL-Netz-Spezifikation *NSPEC* über A, $\overline{A}$ und ρ_Π.

Das Tupel $SNL\text{-}SYS=(NSPEC,A)$ heißt dann ein *SNL-System* (oder *SNL-Modell*) *mit der Realisierung* **R** *über* A, $\overline{A}$ und ρ_Π, falls gilt: $M_0 := \mathbf{R}_{FSM}(M_0') \models_{\mathbf{R}} NSPEC$. $M_0 = \mathbf{R}_{FSM}(M_0')$ heißt in diesem Falle *Anfangsmarkierung des SNL-Systems* (*SNL-Modells*) *SNL-SYS*. Dabei ist $\mathbf{R}_{FSM}(M_0')$ gegeben durch die Familie $\mathbf{R}_{FSM}(M_0')=(\mathbf{R}_{FSM}(m'_{0,p}))_{p\in P}$.

Wir werden im folgenden mit einem SNL-System *SNL-SYS* stets eine Realisierung **R** über A, einer logischen Matrix $\overline{A}$ und Familie von Funktionen ρ_Π voraussetzen, ohne daß wir $\overline{A}$, ρ_Π oder **R** explizit angeben werden, sofern wir nicht verschiedene Realisierungen ein und desselben SNL-Systems über verschiedene $\overline{A}$ und/oder ρ_Π betrachten, was kaum vorkommt. Insbesondere unterdrücken wir daher auch im allgemeinen das tiefgestellte **R** beim Be-

griff des Erfülltsein von Formeln. Bei prädikatenlogischen PrT-Netz-Spezifikationen ist die Realisation sowieso durch die Angabe von A in der SNL-System-Schreibweise $SNL\text{-}SYS=(NSPEC,A)$ eindeutig festgelegt, weil wir hier nur Realisierungen der Prädikate über die Relationen in A betrachten. Netz-Spezifikationen in unserer Spezifikations- und Modellsprache SNL sind daher stets schon durch Angabe eines Modells A zur Spezifikation $SPEC$ eindeutig als SNL-System interpretierbar!

3.4 Das Schalten von SNL-Systemen

Bevor wir uns den Begriffen des Schaltens von Schritten bzw. Transitionen in SNL-Systemen zuwenden, wollen wir zunächst einen linear-algebraischen Kalkül im Umgang mit Familien von Moduln über einen Ring (in unserem Fall Multimengen über den Ring $\mathbb{Z}$) vorstellen, der es uns erlaubt, Schritt- bzw. Schaltvorgänge auf mathematisch einfache und elegante Weise zu beschreiben.

3.4.1 Ein kurzer Exkurs in die lineare Algebra

Familien von Moduln über einen gemeinsamen Ring kann man in natürlicher Weise selbst als ein Modul auffassen, den man direktes Produkt der Familie von Moduln nennt.

Definition 17 **(kartesisches Produkt einer Familie von Mengen)**

Sei R ein Ring, I eine beliebige Indexmenge und $A=((A_i,+_i,\cdot_i))_{i\in I}$ eine Familie von R-Moduln. Wir definieren dann:

(a) Das *kartesische Produkt der Familie von Mengen* $(A_i)_{i\in I}$ ist definiert durch:

$$A_I := \prod_{i\in I} A_i := \{\ (a_i)_{i\in I} \mid a_i\in A_i\ \forall i\in I\ \}$$

Ein I-Tupel $a_I:=(a_i)_{i\in I}\in A_I$ nennen wir im folgenden auch einen *I-Vektor* (über der Familie A).

(b) Auf A_I definieren wir eine Operation $+:A_I\times A_I\longrightarrow A_I$ durch $a_I+b_I=(a_i)_{i\in I}+(b_i)_{i\in I}:=(a_i+_i b_i)_{i\in I}\ \forall a_I, b_I\in A_I$
Wir nennen + die Addition auf A_I.

(c) Weiter definieren wir eine *Skalarmultiplikation* $\cdot:R\times A_I\longrightarrow A_I$ durch $\lambda\cdot a_I=\lambda\cdot(a_i)_{i\in I}:=(\lambda\cdot_i a_i)_{i\in I}\ \forall\lambda\in R$, $\forall a_I\in A_I$.

Es gelten dann die folgenden, bekannten Rechenregeln. Wir geben diese Rechenregeln hier an, da wir uns später noch einmal darauf beziehen werden.

Bemerkung 3

Es seien die Voraussetzungen wie in der vorhergehenden Definition gegeben. Für alle $a_I=(a_i)_{i\in I}$, $b_I=(b_i)_{i\in I}$, $c_I=(c_i)_{i\in I}\in A_I$ und $r, s\in R$ gelten die folgenden Regeln:

(a) $r\cdot(a_I+b_I)=(r\cdot a_I)+(r\cdot b_I),\ (r+s)\cdot a_I=(r\cdot a_I)+(s\cdot a_I)$

(b) $(a_I+b_I)+c_I=a_I+(b_I+c_I)$

(c) Mit $0_I=(0_i)_{i\in I}$, wobei 0_i das Nullelement des Moduls $(A_i,+_i,\cdot_i)$ ist, gilt:
$0_I+a_I=a_I=a_I+0_I,\ 0\cdot a_I=0_I$

(d) Sei $1\in R$ das Einselement des Ringes R und -1 das inverse Element von 1 in R. Dann gilt:
$a_I+(-1)\cdot a_I=(-1)\cdot a_I+a_I=0_I$
d.h. jedes $a_I\in A_I$ besitzt ein inverses Element $-a_I:=(-1)\cdot a_I$

(e) Es läßt sich eine Operation *Subtraktion* $-:A_I\times A_I\longrightarrow A_I$ durch $a_I-b_I:=a_I+(-1)\cdot b_I$ einführen.

(f) Sind die R-Module $((A_i,+_i,\cdot_i))_{i\in I}$ alle kommutativ, so ist auch $(A_I,+,\cdot)$ kommutativ, d.h. es gilt dann:
$a_I+b_I=b_I+a_I$.

Die oben angegebenen Rechenregeln besagen gerade, daß $(A_I,+,\cdot)$ ebenfalls ein R-Modul ist. Dies führt zu der folgenden Definition.

Definition 18 **(direktes Produkt einer Familie von Moduln)**

Sei R ein Ring, I eine beliebige Indexmenge und $A=((A_i,+_i,\cdot_i))_{i\in I}$ eine Familie von R-Moduln. Dann ist $(A_I,+,\cdot)$ ebenfalls ein R-Modul, den wir das *direkte Produkt der Familie von R-Moduln A* nennen.

Der einfacheren Schreibweise wegen bezeichnen wir im folgenden die Operationen aller beteiligten Module stets mit + und $\cdot$. Wir lassen also die Indizes weg und schreiben eine Familie von Moduln kurz als $A=(A_i)_{i\in I}$. Wenn aus dem Kontext ersichtlich ist, wo eine Skalarmultiplikation zu stehen hat, lassen wir der mathematischen Konvention zu Folge, oft auch den Punkt ($\cdot$) ganz weg. Weiter schreiben wir kurz A_I für $(A_I,+,\cdot)$, d.h. A_I steht einmal für den Modul $(A_I,+,\cdot)$ und ein anderes Mal nur für die Menge $A_I:=\prod_{i\in I} A_i$.

Wenn die Indexmenge $I=\{i_1, ..., i_k\}$, $k\in \mathbb{N}, 0<k$, endlich ist, können wir sie willkürlich anordnen, wie dies in der obigen Mengenschreibweise schon implizit durch die Indizierung der Elemente erfolgt ist. Wir können dann einen I-Vektor $a_I=(a_i)_{i\in I}$ in der Form

$$\begin{pmatrix} a_{i_1} \\ \cdots \\ a_{i_k} \end{pmatrix} \begin{array}{l} \leftarrow \text{ Komponente } i_1 \\ \\ \leftarrow \text{ Komponente } i_k \end{array}$$

darstellen, wie man dies für Vektoren gewöhnt ist. Addition und Skalarmultiplikation verstehen sich dann als die bekannten komponentenweise durchgeführten Operationen innerhalb dieser Schreibweise.

Nachdem wir nun wissen, wie wir mit Familien von R-Moduln umgehen, wollen wir uns nun ansehen, wie wir Familien von Modulhomomorphismen behandeln. Hierzu definieren wir zunächst:

Definition 19 (Der Begriff einer Matrix von Modulhomomorphismen)

Sei R ein Ring, I und J Indexmengen und $A=(A_i)_{i\in I}$ und $B=(B_j)_{j\in J}$ Familien von R-Moduln.

(a) Eine $A\times B$ - *Matrix* (bzw. $A_I\times B_J$ - Matrix) ist eine Familie von Modulhomomorphismen $\Theta=(\Theta_{i,j}:A_i\longrightarrow B_j)_{i\in I,j\in J}$ (d.h. jede Abbildung $\Theta_{i,j}:A_i\longrightarrow B_j$ ist ein Modulhomomorphismus).

(b) Die Menge aller $A\times B$-Matrizen bezeichnen wir mit $[A,B]$ (bzw. $[A_I,B_J]$).

(c) Auf $[A,B]$ definieren wir eine Operation *Addition* $+:[A,B]\times[A,B] \longrightarrow [A,B]$ durch $\Theta+\Psi=(\Theta_{i,j})_{i\in I,j\in J}+(\Psi_{i,j})_{i\in I,j\in J} := (\Theta_{i,j}+\Psi_{i,j})_{i\in I,j\in J}\ \forall\Theta, \Psi\in [A,B]$.

(d) Wir definieren eine Operation *Skalarmultiplikation* $\cdot R\times[A,B] \longrightarrow [A,B]$ durch $\lambda\cdot\Theta = \lambda\cdot(\Theta_{i,j})_{i\in I,j\in J}:=(\lambda\cdot\Theta_{i,j})_{i\in I,j\in J}\ \forall\lambda\in R, \forall\Theta\in[A,B]$.

Man sieht dann leicht, daß für $([A,B],+,\cdot)$ die gleichen Rechenregeln gelten wie für direkte Produkte. Wir haben also:

Bemerkung 4

$[A,B]$ ist mit den oben eingeführten Operationen + und $\cdot$ ebenfalls ein R-Modul.

Wenn I und J endliche Indexmengen sind und wir wiederum eine willkürliche Ordnung der Elemente von I und J, wie sie z.B. durch die Mengenschreibweisen $I=\{i_1, ..., i_n\}$, $n\geq 1$, $n\in\mathbb{N}$, und $J=\{j_1, ..., j_m\}$, $m\geq 1$, $m\in\mathbb{N}$ nahegelegt wird, voraussetzen, können wir ein Element $\Theta\in[A_I,B_J]$ eindeutig in einer Matrixschreibweise

$$\begin{array}{l} i_1 \rightarrow \\ \\ i_n \rightarrow \end{array} \begin{bmatrix} \Theta_{i_1,j_1} & \cdots & \Theta_{i_1,j_m} \\ \cdots & \Theta_{i_p,j_k} & \cdots \\ \Theta_{i_n,j_1} & \cdots & \Theta_{i_n,j_m} \end{bmatrix} \begin{array}{c} \\ \\ \end{array}$$
$$\begin{array}{cc} \uparrow & \uparrow \\ j_1 & j_m \end{array}$$

darstellen. Die Addition zweier Matrizen $\Theta, \Psi\in[A,B]$ ergibt sich dann in dieser Schreibweise als die Addition korrespondierender Elemente der Matrix innerhalb der Matrixschreibweise. Die Skalarmultiplikation bedeutet die Multiplizierung jedes Elementes innerhalb der Matrixschreibweise mit dem gegebenen Faktor.

Als nächstes wollen wir nun definieren, was es heißt, eine Matrix $\Theta\in[A,B]$ auf einen I-Vektor $a_I\in A_I$ anzuwenden.

Definition 20 **(Anwendung von Matrizen auf I-Vektoren)**

Sei R ein Ring, I und J endliche Indexmengen und $A=(A_i)_{i\in I}$ und $B=(B_j)_{j\in J}$ Familien von R-Moduln. Wir definieren dann eine Operation $\cdot:[A_I,B_J]\times A_I \longrightarrow B_J$ durch

$$\Theta \cdot a_I := ((\Theta \cdot a_I)_j)_{j\in J} := \Big(\sum_{i\in I} \Theta_{i,j}(a_i)\Big)_{j\in J} \qquad \forall \Theta=(\Theta_{i,j})_{i\in I,j\in J}\in [A_I,B_J],\ \forall\ a_I=(a_i)_{i\in I}\in A_I$$

Wir sagen: $\Theta \cdot a_I$ ist die *Anwendung der Matrix* Θ *auf den I-Vektor* a_I.

Wenn wir bei der oben eingeführten Operation $\cdot:[A_I,B_J]\times A_I \longrightarrow B_J$ ein Element $\Theta=(\Theta_{i,j})_{i\in I,j\in J}\in [A_I,B_J]$ als konstant festhalten, so erhalten wir eine Abbildung $\Theta= \cdot (\Theta,\_) : A_I \longrightarrow B_J$, $\Theta(a_I)=\Theta \cdot a_I$. Auf Grund der Homomorphismuseigenschaft sämtlicher $\Theta_{i,j}$ ist auch die so eingeführte Abbildung Θ ein Modulhomomorphismus, denn es gilt:

$$\begin{aligned}(\Theta(\lambda_1 \cdot a_I^1 + \lambda_2 \cdot a_I^2))_j &= \sum_{i\in I} \Theta_{i,j}(\lambda_1 \cdot a_i^1 + \lambda_2 \cdot a_i^2)\\ &= \sum_{i\in I} (\lambda_1 \cdot \Theta_{i,j}(a_i^1) + \lambda_2 \cdot \Theta_{i,j}(a_i^2))\\ &= \lambda_1 \cdot \sum_{i\in I} \Theta_{i,j}(a_i^1) + \lambda_2 \cdot \sum_{i\in I} \Theta_{i,j}(a_i^2)\\ &= \lambda_1 \cdot (\Theta \cdot a_I^1)_j + \lambda_2 \cdot (\Theta \cdot a_I^2)_j\end{aligned}$$

Damit haben wir also: $\Theta(\lambda_1 \cdot a_I^1 + \lambda_2 \cdot a_I^2)=\lambda_1 \cdot (\Theta \cdot a_I^1) + \lambda_2 \cdot (\Theta \cdot a_I^2)$.

Analog erhalten wir bei Festhalten von a_I eine Abbildung $a_I : [A_I,B_J] \longrightarrow B_J$, $a_I(\Theta)=\Theta \cdot a_I$, die ebenfalls ein Modulhomomorphismus ist. Es gilt also die folgende Bemerkung:

Bemerkung 5

Sei R ein Ring, I und J endliche Indexmengen und $A=(A_i)_{i\in I}$ und $B=(B_j)_{j\in J}$ Familien von R-Moduln.

(a) Für jede Matrix $\Theta=(\Theta_{i,j})_{i\in I,j\in J}\in [A_I,B_J]$ ist die Abbildung $\Theta= \cdot (\Theta,\_) : A_I \longrightarrow B_J$, $\Theta(a_I)=\Theta \cdot a_I$ $\forall\ a_I=(a_i)_{i\in I}\in A$ ein R-Modulhomomorphismus zwischen A_I und B_J, d.h. es gilt die Beziehung

$\Theta(\lambda_1 \cdot a_I^1 + \lambda_2 \cdot a_I^2)=\lambda_1 \cdot (\Theta \cdot a_I^1) + \lambda_2 \cdot (\Theta \cdot a_I^2)$ $\forall a_I^1, a_I^2\in A_I$ und $\forall \lambda_1, \lambda_2\in R$

(b) Für jedes $a_I\in A_I$ ist die Abbildung $a_I : [A_I,B_J] \longrightarrow B_J$, $a_I(\Theta)=\Theta \cdot a_I$ $\forall\Theta\in [A_I,B_J]$ ein R-Modulhomomorphismus zwischen $[A_I,B_J]$ und B_J, d.h. es gilt die Gleichung

$(\lambda_1 \cdot \Theta + \lambda_2 \cdot \Psi) \cdot a_I = \lambda_1 \cdot (\Theta \cdot a_I) + \lambda_2 \cdot (\Psi \cdot a_I)$ $\forall \Theta, \Psi\in [A_I,B_J]$ und $\forall \lambda_1, \lambda_2\in R$.

Wir wollen nun noch ein Produkt von Matrizen von Familien von Modulhomomorphismen einführen. Dazu sei R wiederum ein Ring, I, J und K seien endliche Indexmengen und $A=(A_i)_{i\in I}$, $B=(B_j)_{j\in J}$ und $C=(C_k)_{k\in K}$ seien drei Familien von R-Moduln. Weiter sei $\Theta=(\Theta_{i,j})_{i\in I,j\in J}\in [A_I,B_J]$ und $\Psi=(\Psi_{j,k})_{j\in J,k\in K}\in [B_J,C_K]$. Die Summe $\sum_{j\in J} \Psi_{j,k} o \Theta_{i,j}$, wobei o die Komposition von Abbildungen ist, ist dann eine wohldefinierte Abbildung $\sum_{j\in J} \Psi_{j,k} o \Theta_{i,j} : A_i \longrightarrow C_k$ und es gilt weiter

$$\begin{aligned}\Big(\sum_{j\in J} \Psi_{j,k} o \Theta_{i,j}\Big)(\lambda_1 \cdot a_i^1 + \lambda_2 \cdot a_i^2) &= \sum_{j\in J} ((\Psi_{j,k} o \Theta_{i,j})(\lambda_1 \cdot a_i^1 + \lambda_2 \cdot a_i^2))\\ &= \sum_{j\in J} \Psi_{j,k}(\Theta_{i,j}(\lambda_1 \cdot a_i^1 + \lambda_2 \cdot a_i^2))\\ &= \sum_{j\in J} \Psi_{j,k}(\lambda_1 \cdot \Theta_{i,j}(a_i^1) + \lambda_2 \cdot \Theta_{i,j}(a_i^2))\\ &= \sum_{j\in J} (\lambda_1 \cdot \Psi_{j,k}(\Theta_{i,j}(a_i^1)) + \lambda_2 \cdot \Psi_{j,k}(\Theta_{i,j}(a_i^2)))\\ &= \lambda_1 \cdot \sum_{j\in J} (\Psi_{j,k} o \Theta_{i,j})(a_i^1) + \lambda_2 \cdot \sum_{j\in J} (\Psi_{j,k} o \Theta_{i,j})(a_i^2)\end{aligned}$$

d.h. aber, daß $\sum_{j\in J} \Psi_{j,k} o \Theta_{i,j} : A_i \longrightarrow C_k$ für alle $i\in I$ und $k\in K$ ein Modulhomomorphismus ist.

Damit ist $\Psi \cdot \Theta := (\sum_{j \in J} \psi_{j,k} \circ \Theta_{i,j})_{i \in I, k \in K} \in [A_I, C_k]$. Wir definieren daher:

Definition 21 **(Produkt von Matrizen von Familien von Modulhomomorphismen)**

Sei R wiederum ein Ring, I, J und K seien endliche Indexmengen und $A=(A_i)_{i \in I}$, $B=(B_j)_{j \in J}$ und $C=(C_k)_{k \in K}$ seien drei Familien von R-Moduln. Die Operation $\cdot : [B_J, C_K] \times [A_I, B_J] \longrightarrow [A_I, C_K]$ sei definiert durch

$$\Psi \cdot \Theta := ((\Psi \cdot \Theta)_{i,k})_{i \in I, k \in K} := (\sum_{j \in J} \psi_{j,k} \circ \Theta_{i,j})_{i \in I, k \in K} \; \forall \Psi = (\Psi_{j,k})_{j \in J, k \in K} \in [B_J, C_K], \; \Theta = (\Theta_{i,j})_{i \in I, j \in J} \in [A_I, B_J]$$

Für $\Psi \in [B_J, C_K]$ und $\Theta \in [A_I, B_J]$ heißt $\Psi \cdot \Theta$ das *Produkt der Matrizen* Ψ *und* Θ.

Für die Matrixmultiplikation gelten die folgenden Rechenregeln, wie man leicht durch Nachrechnen zeigt.

Bemerkung 6

Sei R ein Ring, I, J, K und L seien endliche Indexmengen und $A=(A_i)_{i \in I}$, $B=(B_j)_{j \in J}$, $C=(C_k)_{k \in K}$ und $D=(D_l)_{l \in L}$ seien vier Familien von R-Moduln. Dann gilt:

(a) Für jedes $\Psi \in [B_J, C_K]$ ist die Abbildung $\Psi : [A_I, B_J] \longrightarrow [A_I, C_K]$, $\Psi(\Theta) := \Psi \cdot \Theta \; \forall \Theta \in [A_I, B_J]$, ein R-Modulhomomorphismus, d.h. es gelten die Gleichungen

(1) $\Psi \cdot (\Theta^1 + \Theta^2) = (\Psi \cdot \Theta^1) + (\Psi \cdot \Theta^2) \; \forall \Theta^1, \Theta^2 \in [A_I, B_J]$

(2) $\Psi \cdot (\lambda \cdot \Theta) = \lambda \cdot (\Psi \cdot \Theta) \; \forall \lambda \in R$ und $\forall \Theta \in [A_I, B_J]$

(b) Für jedes $\Theta \in [A_I, B_J]$ ist die Abbildung $\Theta : [B_J, C_K] \longrightarrow [A_I, C_K]$, $\Theta(\Psi) := \Psi \cdot \Theta \; \forall \Psi \in [B_J, C_K]$, ein R-Modulhomomorphismus, d.h. es gelten die Gleichungen

(1) $(\Psi^1 + \Psi^2) \cdot \Theta = (\Psi^1 \cdot \Theta) + (\Psi^2 \cdot \Theta) \; \forall \Psi^1, \Psi^2 \in [B_J, C_K]$

(2) $(\lambda \cdot \Psi) \cdot \Theta = \lambda \cdot (\Psi \cdot \Theta) \; \forall \lambda \in R$ und $\forall \Psi \in [B_J, C_K]$

(c) Für alle $\Theta \in [A_I, B_J]$, $\Psi \in [B_J, C_K]$ und $\Phi \in [C_K, D_L]$ gilt: $\Phi \cdot (\Psi \cdot \Theta) = (\Phi \cdot \Psi) \cdot \Theta$.

(d) Für jedes $\Psi \in [B_J, C_K]$, $\Theta \in [A_I, B_J]$ und $a_i \in A_I$ gilt: $(\Psi \cdot \Theta)(a_i) = \Psi \cdot (\Theta \cdot a_i)$.

Damit haben wir alle Begriffe definiert, die wir benötigen, um Schritt- bzw. Schaltvorgänge in SNL-Systemen mit linear-algebraischen Mitteln zu beschreiben. Dieser Aufgabe wollen wir uns im nächsten Abschnitt widmen.

3.4.2 Lineare Algebra von SNL-Systemen

Im Abschnitt über SNL-Systeme haben wir zur Beschreibung der Anfangsmarkierung eines SNL-Systems Familien von Multimengen $M=(m_p)_{p \in P}$ mit $m_p \in MULT^+_{fin}(A)$, $typ(m_p) \subseteq sorts(p) \; \forall p \in P$, verwendet. Wir wir schon im Kapitel über Multimengen festgehalten haben, können wir $MULT_{fin}(A)$ und damit die Mengen $M_p(NSPEC,A) := \{m_p \in MULT_{fin}(A) \mid typ(m_p) \subseteq sorts(p)\} \; \forall p \in P$ als $\mathbb{Z}$-Module auffassen. Gemäß den Definitionen im vorherigen Unterkapitel sind daher Familien $M_P=(m_p)_{p \in P}$, $m_p \in M_p(NSPEC,A)$, P-Vektoren bzgl. der Familie $M(NSPEC,A) := M_P(NSPEC,A) := (M_p(NSPEC,A))_{p \in P}$ von $\mathbb{Z}$-Moduln. Dies führt zu den folgenden Definitionen.

Definition 22 **(P-Vektoren von SNL-Systemen)**

Sei *SNL-SYS*$=(NSPEC,A)$ ein SNL-System.

(a) $M_p(NSPEC,A) := \{m_p \in MULT_{fin}(A) \mid typ(m_p) \subseteq sorts(p)\}$ ist $\forall p \in P$ ein $\mathbb{Z}$-Modul.

(b) Das direkte Produkt $M(NSPEC,A) := M_P(NSPEC,A) := (M_p(NSPEC,A))_{p \in P}$ ist ebenfalls ein $\mathbb{Z}$-Modul, dessen Elemente wir *P-Vektoren des SNL-Systems SNL-SYS* nennen.

(c) Analog zu (a) definieren wir $M^+_p(NSPEC,A) := \{m_p \in MULT^+_{fin}(A) \mid typ(m_p) \subseteq sorts(p)\} \subseteq M_p(NSPEC,A)$.

(d) Analog zu (b) definieren wir $M^+(NSPEC,A) = M^+_P(NSPEC,A) = (M^+_p(NSPEC,A))_{p \in P}$.

Gemäß der obigen Definition sind die erlaubten Objektverteilungen innerhalb eines SNL-Systems *SNL-SYS*$=(NSPEC,A)$ offensichtlich P-Vektoren, die zusätzlich die Netz-Spezifikation erfüllen (d.h. $M \models_R NSPEC$).

Definition 23 **(Markierungen von SNL-Systemen)**

Sei *SNL-SYS*$=(NSPEC,A)$ ein SNL-System. Ein P-Vektor $M \in M(NSPEC,A)$ heißt *(mögliche) Markierung des SNL-Systems SNL-SYS*, falls gilt: M erfüllt $NSPEC$ (d.h. $M \models_R NSPEC$).

Die Menge aller möglichen Markierungen bezeichnen wir mit $MARK(NSPEC,A)$.

Wir haben die natürlichen Einbettungen $MARK(NSPEC,A) \subseteq M^+(NSPEC,A) \subseteq M(NSPEC,A)$. Dabei übertragen sich die Moduloperationen in natürlicher Weise von $M(NSPEC,A)$ auf $M^+(NSPEC,A)$, nicht aber auf $MARK(NSPEC,A)$. Letzteres scheitert nicht nur an den Kapazitätsbedingungen $0 \leq cap_s(m_p) \leq cap_p(s)$ $\forall p \in P$, $s \in S$, sondern in der Regel auch an den Formeln der Stellen $p \in P$.

Die Formel `(all x : (x>4)) orelse (all x : (x<2))` wird z.B. bzgl. der üblichen Semantik von natürlichen Zahlen von den Multimengen `1`5` und `1`1, nicht aber von der Multimenge `1`5+1`1` erfüllt.

Jeder Transition $t \in T$ eines SNL-Systems $SNL\text{-}SYS=(NSPEC,A)$ ist vermöge der Netzbeschriftung eine Menge von freien Variablen $fvars(t)$ zugeordnet. Dabei hängt die Formel der Transition und die Beschriftung der Kanten in der Umgebung der Transition t nur von diesen Variablen als freien Variablen ab. Jede Auswertung $v \in VAL_t(NSPEC,A)$ dieser freien Variablen im algebraischen System A bietet daher die Möglichkeit, die Transitionsformel auf Erfülltheit zu überprüfen und die Kantenbeschriftungen in der Umgebung der Transition auszuwerten, und stellt damit einen potentiellen Modus dar, in dem die Transition t schalten kann.

Wenn wir es zulassen wollen, daß eine Transition parallel (nebenläufig) zu sich selbst schalten darf, so lassen sich solche nebenläufigen Schaltvorgänge von t allgemeiner durch Multimengen von Auswertungen $MULT^+_{fin}(VAL_t(NSPEC,A))$ beschreiben. Lassen wir noch allgemeiner nicht nur die Nebenläufigkeit einer Transition sondern das nebenläufige Schalten von Transitionen, die ihrerseits jede für sich nebenläufig zu sich selbst schalten kann, zu, so werden solche allgemeinen Schaltvorgänge durch Familien von Multimengen von Auswertungen der freien Variablen der beteiligten Transitionen beschrieben, d.h. durch Familien $\tau=(\tau_t)_{t \in T}$, $\tau_t \in MULT^+_{fin}(VAL_t(NSPEC,A))$ für alle $t \in T$, beschrieben. Dies sind aber gerade T-Vektoren bzgl. der Familie von $\mathbb{Z}$-Moduln $(MULT^+_{fin}(VAL_t(NSPEC,A)))_{t \in T}$. Wir definieren daher:

Definition 24 (T-Vektoren)

Sei $SNL\text{-}SYS=(NSPEC,A)$ ein SNL-System.

(a) Die Menge $S_t(NSPEC,A) := MULT_{fin}(VAL_t(NSPEC,A))$ der Multimengen aller Auswertungen der freien Variablen einer Transition $t \in T$ ist ein $\mathbb{Z}$-Modul.

(b) Das direkte Produkt $S(NSPEC,A) := S_T(NSPEC,A) := (S_t(NSPEC,A))_{t \in T}$ ist ebenfalls ein $\mathbb{Z}$-Modul, dessen Elemente wir *T-Vektoren des SNL-Systems SNL-SYS* nennen.

(c) Analog zu (a) definieren wir die Menge $S^+_t(NSPEC,A) := MULT^+_{fin}(VAL_t(NSPEC,A))$.

(d) Analog zu (b) definieren wir die Menge $S^+(NSPEC,A) := S^+_T(NSPEC,A) := (S^+_t(NSPEC,A))_{t \in T}$.

(e) Sei $M \in MARK(NSPEC,A)$ eine Markierung.
Dann ist die Menge $S_t(NSPEC,M) := MULT_{fin}(VAL_t(NSPEC,M))$ der Multimengen von Auswertungen der freien Variablen einer Transition $t \in T$ unter der Markierung M in A ebenfalls ein $\mathbb{Z}$-Modul.

(f) Sei $M \in MARK(NSPEC,A)$ eine Markierung.
Das direkte Produkt $S(NSPEC,M) := S_T(NSPEC,A) := (S_t(NSPEC,M))_{t \in T}$ ist ebenfalls ein $\mathbb{Z}$-Modul, dessen Elemente wir *T-Vektoren des SNL-Systems SNL-SYS unter der Markierung M* nennen.

(g) Analog zu (e) definieren wir die Menge $S^+_t(NSPEC,M) := MULT^+_{fin}(VAL_t(NSPEC,M))$.

(h) Analog zu (f) definieren wir die Menge $S^+(NSPEC,M) := S^+_T(NSPEC,A) := (S^+_t(NSPEC,M))_{t \in T}$.

Es gelten offensichtlich die Teilmengenrelationen $S^+(NSPEC,M) \subseteq S(NSPEC,M) \subseteq S(NSPEC,A)$ $\forall M \in MARK(NSPEC,A)$. Weiter haben wir $S^+(NSPEC,M) \subseteq S^+(NSPEC,A) \subseteq S(NSPEC,A)$.

Sei nun $M \in MARK(NSPEC,A)$. Dann stellen die T-Vektoren $\mu \in S^+(NSPEC,A)$ potentielle Kandidaten zur Beschreibung eines Schaltvorganges ausgehend von der Markierung M im SNL-System $SNL\text{-}SYS=(NSPEC,A)$ dar. Wenn wir die Einhaltung des Lokalitätsprinzips verlangen, müssen wir uns sogar auf T-Vektoren $\mu \in S^+(NSPEC,M)$ beschränken. Als weitere Voraussetzung für einen Schaltvorgang sollten aber auch die Transitionsformeln von allen beteiligten Auswertungen der freien Variablen der Transitionen erfüllt werden. Dies führt zu der folgenden Definition.

Definition 25 **(T-Vektoren, die Transitionsformeln erfüllen)**

Sei *SNL-SYS*=(*NSPEC*,*A*) ein SNL-System und $M \in MARK(NSPEC,A)$. Weiter sei $\mu = \left(\sum_{i_t=1}^{k_t} d_{i_t} \cdot v_{i_t} \right)_{t \in T} \in S(NSPEC,A)$ ein beliebiger *T*-Vektor. Dann definieren wir:

Der *T-Vektor* μ *erfüllt die Transitionsformeln von SNL-SYS=(NSPEC,A) unter der Markierung M* (kurz: μ *erfüllt NSPEC unter M*), falls gilt:

$$\forall t \in T,\ i_t = 1, \ldots, k_t \text{ gilt: } M \models_{v_{i_t}, \mathbf{R}} NSPEC$$

Wir schreiben dies als $M \models_{\mu} NSPEC$.

Wir müssen uns nun noch der Interpretation der Kantenbeschriftungen bei einem Schaltvorgang zuwenden, um die dynamische Semantik eines SNL-Systems vollständig beschreiben zu können. Hierzu bemerken wir zunächst, daß die Kantenbeschriftungen $A_F(f)$ gemäß der Definition einer SNL-Netzspezifikation formale Summen mit den Randbedingungen $fvars(A_F(f)) \subseteq sorts(p)$ für $f=(p,t)$ bzw. (t,p) $t \in T$, $p \in P$ sind. Die Realisierung so einer Kantenbeschriftung liefert dann eine Abbildung $\mathbf{R}_{FSM}(A_F(f)) = A_F(f)_{\mathbf{R}} : VAL_t(NSPEC,A) \longrightarrow M_p(NSPEC,A)$. Die allgemeinen Regeln für die Semantik der Kantenbeschriftungen von Petri-Netzen legen nun nahe, daß $A_F(f)_{\mathbf{R}}(v)$ die Multimenge von Objekten ist, die beim Schalten von *t* mit der Variablenauswertung $v \in VAL_t(NSPEC,A)$ von Eingabestellen (d.h. $f=(p,t)$) abzuziehen bzw. auf Ausgabestellen (d.h. $f=(t,p)$) hinzuzufügen ist. Hierzu führen wir die folgende Definition ein.

Definition 26

Sei *SNL-SYS*=(*NSPEC*,*A*) ein SNL-System. Die Abbildung $\Delta t_p : VAL_t(NSPEC,A) \longrightarrow M(NSPEC,A)$ sei für alle $p \in P$ und $t \in T$ definiert durch

$$\Delta t_p = \begin{cases} 0 \in M_p(NSPEC,A) & \text{falls } (t,p) \text{ und } (p,t) \notin F \\ -A_F(f)_{\mathbf{R}} & \text{falls } f=(p,t) \in F \text{ und } (t,p) \notin F \\ A_F(f)_{\mathbf{R}} & \text{falls } (p,t) \notin F \text{ und } f=(t,p) \in F \\ -A_F((p,t))_{\mathbf{R}} + A_F((t,p))_{\mathbf{R}} & \text{falls } (p,t) \in F \text{ und } (t,p) \in F \end{cases}$$

Δt sei definiert als die Familie $\Delta t := (\Delta t_p)_{p \in P}$.

Die Folgemarkierung von *M*, die entsteht, wenn die Transition *t* mit der Variablenauswertung *v* schaltet, sollte sich dann durch die Gleichung $M' = M + \Delta t(v)$ ergeben. Hierbei sind aber noch die Randbedingungen zu beachten, daß unter der Markierung *M* die "richtigen" Objekte in genügender Anzahl auf den Eingabestellen vorhanden sein müssen, um beim Schaltvorgang abgezogen werden zu können, und weiter $M' = M + \Delta t(v)$ eine mögliche Markierung des SNL-Systems sein muß (d.h. *M'* muß den Kapazitäten und Formeln der Stellen genügen). Unter Berücksichtigung dieser Randbedingungen können wir nun definieren, wann eine Transition aktivierbar ist und was wir uns unter der Schaltregel für SNL-Systeme vorzustellen haben.

Definition 27 **(Aktivierbarkeit von Transitionen und Schaltregel in SNL-Systemen)**

Sei *SNL-SYS*=(*NSPEC*,*A*) ein SNL-System und $M \in MARK(NSPEC,A)$ eine mögliche Markierung von *SNL-SYS*.

(a) Eine Transition $t \in T$ *heißt im Modus* $v \in VAL_t(NSPEC,A)$ *unter M aktivierbar*, wenn die folgenden Bedingungen erfüllt sind.

(1) $v \in VAL_t(NSPEC,M)$ (Lokalitätsprinzip)

(2) $M \models_v eqn(t)$ (Transitionsformel erfüllt)

(3) $\forall\, p \in P$ mit $(p,t) \in F$ gilt: $A_F((p,t))_{\mathbf{R}} \leq m_p$
(die "richtigen" Objekte in genügender Anzahl auf den Eingabestellen vorhanden)

(4) $M' = M + \Delta t(v) \in MARK(NSPEC,A)$
(d.h. die Folgemarkierung ist eine mögliche Markierung des SNL-Systems)

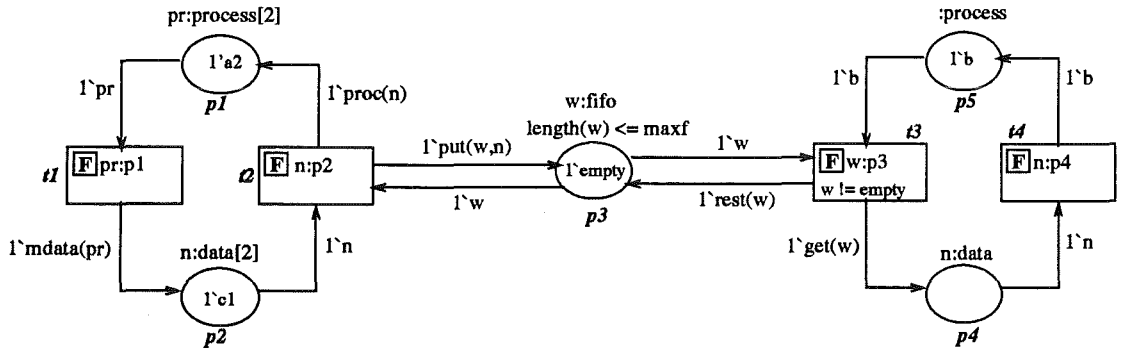
(b) Eine Transition $t \in T$ *heißt unter M aktivierbar*, falls ein $v \in VAL_t(NSPEC,A)$ existiert, so daß t im Modus v unter M aktivierbar ist.

(c) (Schaltregel)
Wenn eine Transition $t \in T$ in einem Modus $v \in VAL_t(NSPEC,A)$ unter einer Markierung M aktivierbar ist, so kann t im Modus v schalten, wobei das SNL-System in ein System mit der Folgemarkierung $M'=M+\Delta t(v)$ übergeht.

Wir haben nun definiert, wie wir uns einen dynamischen Zustandsübergang in einem SNL-System vorzustellen haben, wenn wir nur das Schalten jeweils einer Transition bzgl. einer Auswertung ihrer freien Variablen zu einer Zeit als atomare Operation auf den Stellen des Systems zulassen. Die Bedingung für die Aktivierbarkeit solcher Zustandsübergänge sind die Erfüllung des Lokalitätsprinzips (d.h. alle zur Auswertung v gehörigen Objekte sind auf den "richtigen" Umgebungsstellen der Transition vorhanden, also $v \in VAL_t(NSPEC,M)$), das Erfülltsein der Transitionsformel, das Vorhandensein einer ausreichenden Anzahl von den Objekten auf den Eingabestellen, die beim Schaltvorgang von diesen abzuziehen sind, sowie die Bedingung, daß die Folgemarkierung ebenfalls eine gültige Markierung des SNL-Systems ist. Wenn diese Bedingungen erfüllt sind, kann ein Zustandswechsel "Markierung M geht in die Markierung $M'=M+\Delta t(v)$ über" im System erfolgen. Wir wollen zum Schalten von Transitionen einmal ein Beispiel betrachten.

Beispiel 4 (Schalten einer Transition)

Betrachten wir die SNL-Netz-Spezifikation aus dem vorhergehenden Beispiel und realisieren sie mit einem geeigneten algebraischen System A, wobei wir annehmen, daß `mdata(a1)=c1` ist. Dann kann die Transition `t1` in dem Modus `pr`→ `a1` schalten, da die Formel A_T(`t1`) stets `true` ist und `1`a1` als Marke auf der Stelle `p1` vorhanden ist. Das Netz geht dann in die Markierung des folgenden Bildes über.



Wir führen mit der folgenden Definition noch einige Bezeichnungen ein.

Definition 28 (Schaltvorgang und Schaltfolge)

Sei *SNL-SYS*=(*NSPEC*,*A*) ein SNL-System.

(a) Unter einem *Schaltvorgang* verstehen wir ein Tripel $(M,t:v,M')$ mit M, $M' \in MARK(NSPEC,A)$, $t \in T$ und $v \in VAL_t(NSPEC,M)$, so daß t unter M im Modus v aktivierbar ist und $M'=M+\Delta t(v)$ die Folgemarkierung von M ist.

Um suggestiv darzustellen, daß M' aus M durch Schalten von t im Modus v entsteht, schreiben wir einen Schaltvorgang auch in der Form $M[t:v>M'$.

(b) Eine *Schaltfolge* ist eine Folge von Schaltvorgängen $(M_i,t_i:v_i,M_i')_{i=1,...,n}$ mit $M_i'=M_{i+1}$ für $i=1,..,n-1$.

Eine Schaltfolge schreiben wir suggestiv auch in der Form

$$M_1[t_{k_1}:v_{k_1}>M_2[t_{k_2}:v_{k_2}>.....M_n[t_{k_n}:v_{k_n}>M'_n = M_{n+1}$$

Bemerkung 7

In vielen Fällen wird es uns egal sein in welchem Modus eine Transition schaltet. Wichtig wird dann nur sein, daß eine Markierung M' durch Schalten einer Transition t aus einer Markierung M entsteht. Wir werden das durch die Schreibweise (M,t,M') bzw. $M[t>M'$ ausdrücken.

Man beachte jedoch, daß z.B. die zwei Schaltvorgänge $(M,t{:}v_1,M')$ und $(M,t{:}v_2,M')$, mit zwei verschiedenen Auswertungen v_1 und v_2, durch Wegfallen der verschiedenen Modi zu einem Schaltvorgang (M,t,M') verschmelzen.

In der Regel interessieren uns nicht alle möglichen Markierungen eines SNL-Systems, sondern nur die, die von der Anfangsmarkierung aus durch Schalten von Transitionen erreichbar sind. Hierzu führen wir die folgenden Definitionen ein:

Definition 29 **(Erreichbarkeitsrelation und Erreichbarkeitsmenge)**

Sei $SNL\text{-}SYS=(NSPEC,A)$ ein SNL-System und M_0 die zugehörige Anfangsmarkierung.

(a) Die *direkte Erreichbarkeitsrelation* $\Longrightarrow$ sei auf $MARK(NSPEC,A)$ wie folgt definiert:

$\forall M, M' \in MARK(NSPEC,A)$ gilt $M \Longrightarrow M'$ genau dann, wenn eine Transition $t \in T$ und eine Auswertung $v \in VAL_t(NSPEC,A)$ existiert, so daß $(M,t{:}v,M')$ ein Schaltvorgang ist.

(b) Als *Erreichbarkeitsrelation* $\overset{*}{\Longrightarrow}$ bezeichnen wir den transitiven und reflexiven Abschluß von $\Longrightarrow$, d.h. $\forall M, M' \in MARK(NSPEC,A)$ gilt $M \overset{*}{\Longrightarrow} M'$ genau dann, wenn $M=M'$ ist oder eine Schaltfolge $(M_i,t_i{:}v_i,M_i)_{i=1,\ldots,n}$ existiert, für die $M=M_1$ und $M'_n=M'$ ist.

(c) $[M\rangle := \{ M' \in MARK(NSPEC,A) \mid M \overset{*}{\Longrightarrow} M' \}$

(d) $[M_0\rangle$ heißt *Erreichbarkeitsmenge* des SNL-Systems $SNL\text{-}SYS$.

Die für uns relevanten Zustände eines SNL-Systems werden also durch die Erreichbarkeitsmenge $[M_0\rangle$ beschrieben. Entsprechend interessieren uns nur Schaltvorgänge, die von Markierungen aus $[M_0\rangle$ ausgehen können.

Definition 30

Sei $SNL\text{-}SYS=(NSPEC,A)$ ein SNL-System und M_0 die zugehörige Anfangsmarkierung.

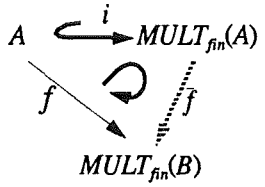
$TMOCC(NSPEC,A) := \{ (M,t{:}v,M') \mid (M,t{:}v,M')$ ist ein Schaltvorgang und $M \in [M_0\rangle \}$.

$TOCC(NSPEC,A) := \{ (M,t,M') \mid (M,t,M')$ ist ein Schaltvorgang und $M \in [M_0\rangle \}$.

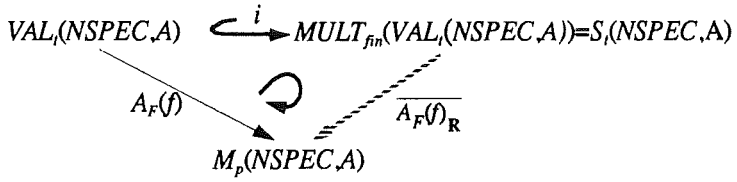
$[M_0\rangle$ und $TMOCC(NSPEC,A)$ bilden die Knoten- und Kantenmengen eines Graphen, den wir *Erreichbarkeitsgraph* nennen wollen. Der Erreichbarkeitsgraph beschreibt die dynamische Struktur eines SNL-Systems bzgl. dem Schalten von Transitionen vollständig und wird später noch näher untersucht.

Wir haben bisher nur Zustandsübergänge in SNL-Systemen betrachtet, die als atomare Operationen nur das Schalten jeweils einer Transition in einem Modus zulassen. Wir haben aber schon mehrfach im Text zum Ausdruck gebracht, daß wir allgemeiner auch das nebenläufige Schalten einer Transition bzgl. mehrerer Schaltmodi und das nebenläufige Schalten mehrerer aktivierter Transitionen zulassen wollen. Hierzu wollen wir uns zunächst einmal ansehen, wie wir das nebenläufige Schalten einer Transition $t \in T$ in mehreren Modi beschreiben können. Hierzu müssen wir nicht mehr eine Auswertung $v \in VAL_t(NSPEC,A)$, sondern Multimengen von Auswertungen $\mu = \sum_{i=1}^{k} d_i \cdot v_i \in MULT^+_{fin}(VAL_t(NSPEC,A)) = S^+_t(NSPEC,A)$ als Schaltmodi betrachten.

Wir hatten aber eine Kantenbeschriftung $A_F(f)$ in der Realisation als eine Abbildung $A_F(f)_{\mathbf{R}}{:}VAL_t(NSPEC,A) \longrightarrow M_p(NSPEC,A)$ interpretiert, so daß $A_F(f)_{\mathbf{R}}$ nicht so ohne weiteres auf eine Multisumme $\mu \in S^+_t(NSPEC,A)$ anwendbar ist. Wir können jedoch für zwei Mengen A und B jede Abbildung $f{:}A \longrightarrow MULT_{fin}(B)$ auf genau eine eindeutige Art und Weise zu einem $\mathbb{Z}$-Modulhomomorphismus $\bar{f}{:}MULT_{fin}(A) \longrightarrow MULT_{fin}(B)$ fortsetzen, so daß das folgende Diagramm kommutativ ist:



Hierzu definieren wir $\bar{f}(\sum_{k=1}^{n} d_k \cdot `a_k) = \sum_{k=1}^{n} d_k \cdot f(a_k)$ für alle $n \in \mathbb{N}$, $n \geq 1$, $d_k \in \mathbb{Z}$ und $a_k \in A$ für $1 \leq k \leq n$ und $\bar{f}(0)=0$. Es gibt damit zu $A_F(f)_R: VAL_t(NSPEC,A) \to M_p(NSPEC,A)$ genau einen $\mathbb{Z}$-Modulhomomorphismus $\overline{A_F(f)_R}: MULT^+_{fin}(VAL_t(NSPEC,A)) = S^+_t(NSPEC,A) \to M_p(NSPEC,A)$, der das Diagramm



kommutativ macht.

Wenn wir Multimengen von Auswertungen der freien Variablen einer Transition betrachten, verbinden wir im folgenden mit der Realisation einer Kantenbeschriftung $A_F(f)$ stets diesen eindeutig bestimmten Modulhomomorphismus $\overline{A_F(f)_R}$. Analog zu Δt definieren wir nun $\Theta_F=(\Theta_{p,t})_{p \in P}: S^+_t(NSPEC,A) \to M(NSPEC,A)$ durch

$$\Theta_{p,t} = \begin{cases} 0 \in M_p(NSPEC,A) & \text{falls } (t,p) \text{ und } (p,t) \notin F \\ -\overline{A_F(f)_R} & \text{falls } f=(p,t) \in F \text{ und } (t,p) \notin F \\ \overline{A_F(f)_R} & \text{falls } (p,t) \notin F \text{ und } f=(t,p) \in F \\ -\overline{A_F((p,t))_R} + \overline{A_F((t,p))_R} & \text{falls } (p,t) \in F \text{ und } (t,p) \in F \end{cases}$$

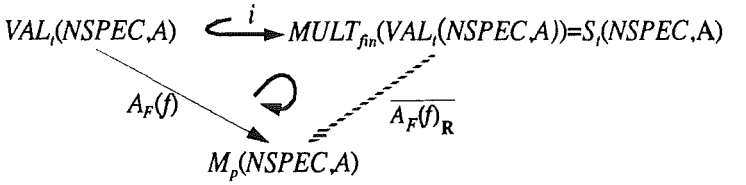
Wenn die Transition dann bzgl. der Multimenge $\mu_t \in S^+_t(NSPEC,A)$ von Auswertungen ihrer freien Variablen schaltet, wird die Folgemarkierung analog zur Gleichung $M'=M+\Delta t(v)$ durch die Gleichung $M'=M+\Theta_t(\mu_t)$ beschrieben. Wenn wir nun noch das nebenläufige Schalten von Transitionen ins Kalkül ziehen, dann wird ein möglicher Zustandsübergang nicht mehr durch eine Multimenge, sondern durch eine Familie von Multimengen $\mu=(\mu_t)_{t \in T}$ mit $\mu_t \in S^+_t(NSPEC,A)$ $\forall t \in T$, d.h. durch einen T-Vektor $\mu \in S(NSPEC,A)$, beschrieben. Die Folgemarkierung ergibt sich in diesem allgemeinen Fall durch die Gleichung $M' = M + \sum_{t \in T} \Theta_t(\mu_t)$. Wenn wir nun $(\Theta_{p,t})_{p \in P, t \in T}$ als eine Matrix $\Theta \in [S_T(NSPEC,A), M_P(NSPEC,A)] = [S(NSPEC,A), M(NSPEC,A)]$ auffassen, da die $\Theta_{p,t}$ ja alle $\mathbb{Z}$-Modulhomomorphismen sind, entspricht der Summe $(\sum_{t \in T} \Theta_t(\mu_t) = \sum_{t \in T} (\Theta_{p,t})_{p \in P}(\mu_t))$ gerade die Anwendung der Matrix Θ auf den T-Vektor μ und wir können schreiben: $M'=M+\Theta \cdot \mu$. Wir führen daher den folgenden Begriff ein.

Definition 31 **(Inzidenzmatrix eines SNL-Systems)**

Sei $SNL\text{-}SYS=(NSPEC,A)$ ein SNL-System. $\forall p \in P$ und $\forall t \in T$ sei der $\mathbb{Z}$-Modulhomomorphismus $\Theta_{p,t}: S^+_t(NSPEC,A) \to M_p(NSPEC,A)$ durch

$$\Theta_{p,t} = \begin{cases} 0 \in M_p(NSPEC,A) & \text{falls } (t,p) \text{ und } (p,t) \notin F \\ -\overline{A_F(f)_R} & \text{falls } f=(p,t) \in F \text{ und } (t,p) \notin F \\ \overline{A_F(f)_R} & \text{falls } (p,t) \notin F \text{ und } f=(t,p) \in F \\ -\overline{A_F((p,t))_R} + \overline{A_F((t,p))_R} & \text{falls } (p,t) \in F \text{ und } (t,p) \in F \end{cases}$$

definiert, wobei $\overline{A_F(f)}_R : S_t^+(NSPEC,A) \rightarrow M_p(NSPEC,A)$ für alle $f=(p,t)$ *bzw.* $f=(t,p) \in F$ der eindeutig bestimmte $\mathbb{Z}$-Modulhomomorphismus ist, der das folgende Diagramm kommutativ macht:



Die Matrix $\Theta=(\Theta_{p,t})_{p\in P,t\in T} \in [S(NSPEC,A),M(NSPEC,A)]$ heißt dann *Inzidenzmatrix des SNL-Systems SNL-SYS*.

Wenn wir im folgenden von Schaltvorgängen bzgl. T-Vektoren eines SNL-Systems sprechen, wollen wir zur Unterscheidung zum Fall des Schaltens von Transitionen die Begriffe *Schalten von Schritten* und *Schrittvorgang* verwenden. Die Definition der Schaltregel für Schritte lautet wie folgt:

Definition 32 (Aktivierbarkeit von Schritten und Schrittregel)

Sei $SNL\text{-}SYS=(NSPEC,A)$ ein SNL-System und $M=(m_p)_{p\in P} \in MARK(NSPEC,A)$ eine Markierung.

(a) Ein T-Vektor $\mu=(\mu_t)_{t\in T} \in S(NSPEC,A)$ heißt *aktivierbar unter der Markierung M*, falls die folgenden Bedingungen erfüllt sind.

(1) $\mu \in S^+(NSPEC,M)$ (Lokalitätsprinzip).

(2) $M \models_\mu NSPEC$ (μ erfüllt die Transitionsformeln).

(3) $\forall p\in P$ gilt: $\sum_{t\in T,(p,t)\in F} \overline{A_F((p,t))}_R(\mu_t) \leq m_p$ (d.h. die "richtigen" Objekte sind in genügender Anzahl auf den Eingabestellen vorhanden).

(4) $M'=M+\Theta \cdot \mu$ ist eine mögliche Markierung, d.h. $M+\Theta \cdot \mu \in M(NSPEC,A)$.

(b) (Schrittregel)
Sei der T-Vektor $\mu \in S(NSPEC,A)$ unter M aktivierbar. Das SNL-System kann dann im Modus μ schalten, wobei der Zustand des Systems von der Markierung M in die Markierung $M'=M+\Theta \cdot \mu$ übergeht.

Der Begriff der Aktivierbarkeit von T-Vektoren ergibt sich also analog zu dem Begriff der Aktivierbarkeit einer Transition t in einem Modus v. Die Aktivierbarkeit einer Transition $\bar{t}$ in einem Modus v können wir sogar als einen Spezialfall des Begriffs der Aktivierbarkeit von T-Vektoren auffassen, wenn wir als T-Vektoren $\mu=(\mu_t)_{t\in T}$ mit $\mu_t=1`v$ falls $\bar{t}=t$, $\mu_t=0$ sonst, verwenden.

Betrachten wir einmal ein Beispiel zum Begriff der Aktivierbarkeit von T-Vektoren und zur Schrittregel.

Beispiel 5 (Aktivierbarkeit von Schritten und Schrittregel)

Wir betrachten ein SNL-System FARBE, dessen Spezifikationsteil eine Sorte `farbe` mit zwei Konstanten `rot` und `grün` definiert, die wir trivialerweise über eine zweielementige Menge mit den Elementen `rot` und `grün` realisiert denken. Das Netz selbst habe die Gestalt, wie sie das folgende Bild verdeutlicht.

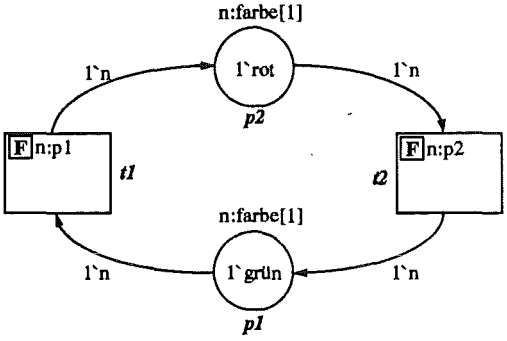


Abb. 1 SNL-System FARBE mit Anfangsmarkierung M_0

Zum besseren Verständnis erläutern wir die Netzelemente in Abb. 1 am Beispiel einer Stelle und einer Transition. Die untere Stelle trägt den Namen „p1" und die Anfangsmarkierung „1`grün". Die Beschriftung „n:farbe[1]" gibt an, daß der Stelle eine Variable der Sorte n zugeordnet ist, und daß die Stelle höchstens ein Objekt der Sorte farbe aufnehmen kann. D.h. die Kapazität der Stelle p1 bezüglich der Sorte farbe ist 1. Die linke Transition heißt „t1". Das Schalten der Transition „t1" hängt von der freien Variablen n, die zu der Stelle „p1" gehört, ab.

In diesem System kann keine Transition schalten, da für jede Transition die Nachbedingung nicht erfüllt ist (Die Kapazität 1 einer der Ausgabestellen wird überschritten). Der T-Vektor $\mu=(1`\{n\rightarrow grün\}, 1`\{n\rightarrow rot\})^T$ ist aber aktivierbar, da beim Vertauschen der beiden Farbmarken während einer atomaren Operation die Kapazität der Stellen nicht überschritten wird. Nach dem Schalten dieses Schrittes haben wir dann das SNL-System mit der folgenden Markierung (s. Abb. 2).

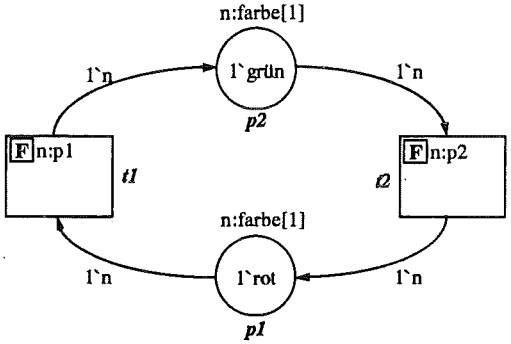


Abb. 2 SNL-System mit Markierung M_1

Wir wollen noch einige Begriffe einführen.

Definition 33 (Schrittvorgang, Schritt und Schrittfolge)

Sei *SNL-SYS*=(*NSPEC*,*A*) ein SNL-System und $M\in MARK(NSPEC,A)$ eine Markierung.

(a) Unter einem *Schrittvorgang* verstehen wir ein Tripel (M,μ,M') mit $M, M'\in MARK(NSPEC,A)$ und $\mu\in S(NSPEC,A)$, so daß μ unter M aktivierbar ist, und $M'=M+\Theta\cdot\mu$ die Folgemarkierung von M ist. μ nennen wir in diesem Fall den *Schritt*, der die Markierung M in die Markierung M' überführt.

Um suggestiv darzustellen, daß M' aus M durch Schalten des SNL-Systems im Modus μ entsteht, schreiben wir einen Schrittvorgang auch in der Form $M[_S\mu>M'$ oder, falls es nicht zu Mißverständissen führt, auch in der Form $M[\mu>M'$.

(b) Eine *Schrittfolge* ist eine Folge von Schrittvorgängen $(M_i,\mu_i,M_i')_{i=1,..,n}$ mit $M_i'=M_{i+1}$ für $i=1,..,n-1$.

Eine Schrittfolge schreiben wir suggestiv auch in der Form

$$M_1[\mu_1>M_2[\mu_2>.....M_n[\mu_n>M'_n = M_{n+1}$$

Wenn wir die Schrittsemantik eines SNL-Systems betrachten, interessieren uns in der Regel wiederum nicht alle möglichen Markierungen des SNL-Systems, sondern nur die, die von der Anfangsmarkierung aus durch Schalten von Schritten erreichbar sind. Hierzu führen wir die folgende Definition ein:

Definition 34 **(Schritterreichbarkeitsrelation und Schritterreichbarkeitsmenge)**

Sei *SNL-SYS*=(*NSPEC*,*A*) ein SNL-System und M_0 die zugehörige Anfangsmarkierung.

(a) Die *direkte Schritterreichbarkeitsrelation* $\Rightarrow_S$ sei auf *MARK*(*NSPEC*,*A*) wie folgt definiert:

$\forall M, M' \in MARK(NSPEC,A)$ gilt $M \Rightarrow_S M'$ genau dann, wenn ein T-Vektor $\mu \in S(NSPEC,A)$ existiert, so daß (M,μ,M') ein Schrittvorgang ist.

(b) Als *Schritterreichbarkeitsrelation* $\overset{*}{\Rightarrow}_S$ bezeichnen wir den transitiven und reflexiven Abschluß von $\Rightarrow_S$, d.h. $\forall M, M' \in MARK(NSPEC,A)$ gilt $M \overset{*}{\Rightarrow}_S M'$ genau dann, wenn $M=M'$ ist oder eine Schrittfolge $(M_i,\mu_i,M_i')_{i=1,\ldots,n}$ existiert, für die $M=M$ und $M'_n=M'$ ist.

(c) $[_SM\rangle := \{ M' \in MARK(NSPEC,A) \mid M \overset{*}{\Rightarrow}_S M' \}$

(d) $[_SM_0\rangle$ heißt *Schritterreichbarkeitsmenge* des SNL-Systems *SNL-SYS*.

Die für uns relevanten Zustände eines SNL-Systems bzgl. der Schrittsemantik sind also die Elemente der Schritterreichbarkeitsmenge $[_SM_0\rangle$. Entsprechend interessieren uns nur Schrittvorgänge, die von Markierungen aus $[_SM_0\rangle$ ausgehen können.

Definition 35

Sei *SNL-SYS*=(*NSPEC*,*A*) ein SNL-System und M_0 die zugehörige Anfangsmarkierung.

$SOCC(NSPEC,A) := \{ (M,\mu,M') \mid (M,\mu,M')$ ist ein Schrittvorgang und $M \in [_SM_0\rangle \}$.

Die Mengen $[_SM_0\rangle$ und *SOCC*(*NSPEC*,*A*) kann man wiederum als Knoten- und Kantenmenge eines Graphen, den wir *Schrittgraphen* nennen wollen, auffassen, der die dynamische Struktur eines SNL-Systems bzgl. der Schrittsemantik, die durch die Schrittregel gegeben ist, vollständig beschreibt. Der Schrittgraph wird ebenfalls später noch genauer beschrieben.

Bemerkung 8

Anstelle von *SOCC*(*NSPEC*,*A*) werden wir auch $\mathcal{S}_{N,M_0}$, anstelle von *TMOCC*(*NSPEC*,*A*) $\mathcal{TM}_{N,M_0}$ und anstelle von *TOCC*(*NSPEC*,*A*) $\mathcal{T}_{N,M_0}$ verwenden.

Für die Menge aller möglichen Markierungen *MARK*(*NSPEC*,*A*) werden wir meistens die Schreibweise $\mathcal{M}_N$ benutzen.

4. Erreichbarkeitsanalyse

Wir haben im vorherigen Kapitel SNL-Systeme beschrieben. Unser Ziel ist es nun, Methoden zur rechnergestützten Analyse dieser SNL-Systeme anzugeben. Wir unterteilen diese Analyse dabei in Erreichbarkeits- und Invariantenanalyse. Um eine rechnergestützte Analyse durchführen zu können, müssen wir zuerst die theoretischen Grundlagen für eine Analyse dieser SNL-Systeme schaffen. Wir stellen deshalb in diesem Kapitel die Grundlagen der Erreichbarkeitsanalyse und im nächsten Kapitel die Grundlagen der Invariantentheorie unserer SNL-Systeme vor.

4.1 Theoretische Grundlagen

Wir haben im vorherigen Kapitel das Schalten eines Schrittes bzw. einer Transition eingeführt. Über den Schaltbegriff haben wir die Erreichbarkeitsrelation definiert und dadurch die Erreichbarkeitsmenge eines SNL-Systems erhalten. Wir wollen nun mit Hilfe dieser Grundlagen den Erreichbarkeitsgraph, der für die Erreichbarkeitsanalyse von zentraler Bedeutung ist, einführen.

4.1.1 Der Erreichbarkeitsgraph

Als Beschreibungsmittel für das dynamische Verhalten eines SNL-Systems mit der Anfangsmarkierung M_0 bietet sich nun ein Graph an, der die von M_0 schritterreichbaren Markierungen $[_SM_0>$ als Knoten und die Schaltvorgänge $(M,\mu,M') \in \mathcal{S}_{N,M_0}$ als Kanten besitzt. Der betrachtete Graph ist dann ein Graph mit im allgemeinen mehreren gerichteten Kanten zwischen je zwei Knoten. Eine mögliche Definition für einen solchen Graph mit Mehrfachkanten ist:

Definition 36 **(gerichteter Graph)**

Ein 4-Tupel G=(KN,KA,a,e) heißt *gerichteter Graph*, wenn gilt:

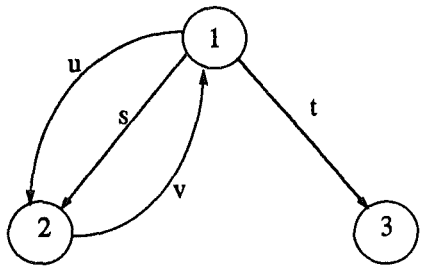
(a) KN und KA sind disjunkte Mengen, die wir Menge der *Knoten* und Menge der *Kanten* des Graphen nennen,

(b) a:KA→KN und e:KA→KN sind Abbildungen, die jeder Kante ka∈KA ihren Anfangspunkt a(ka)∈KN und ihren Endpunkt e(ka)∈KN zuordnen.

Wir wollen zu dieser Definition ein Beispiel betrachten.

Beispiel 6

Es sei KN={1,2,3} und KA={s,t,u,v}. Die Abbildungen a,e:KA→KN seien durch a(s)=1, a(t)=1, a(u)=1, a(v)=2 und e(s)=2, e(t)=3, e(u)=2, e(v)=1 gegeben. Den zugehörigen Graph können wir dann bildlich wie folgt darstellen:



Mit der obigen Definition eines Graphen können wir nun den Schrittgraph eines SNL-Systems wie folgt definieren:

Definition 37 **(Schrittgraph)**

Sei N=(NSPEC,A) ein SNL-System. Der allgemeine Schrittgraph $\mathcal{SG}_N$ von N ist definiert als der Graph $\mathcal{SG}_N=([_SM_0>,\mathcal{S}_{N,M_0},a,e)$, für den gilt:

(a) $a((M,\mu,M'))=M$ für alle $(M,\mu,M') \in \mathcal{S}_{N,M_0}$,

(b) $e((M,\mu,M'))=M'$ für alle $(M,\mu,M') \in \mathcal{S}_{N,M_0}$.

Wir wollen auch dazu wieder den Spezialfall des Schaltens von einzelnen Transitionen betrachten und den Erreichbarkeitsgraphen eines SNL-Systems wie folgt definieren:

Definition 38 (Erreichbarkeitsgraph)

Sei N=(NSPEC,A) ein SNL-System. Der allgemeine Erreichbarkeitsgraph (Reachability Graph) $\mathcal{RGM}_N$ von N ist definiert als der Graph $\mathcal{RGM}_N=([M_0>, \mathcal{TM}_{N,M_0}, a, e)$, für den gilt:

(a) $a((M,t{:}v,M'))=M$ für alle $(M,t{:}v,M') \in \mathcal{TM}_{N,M_0}$,

(b) $e((M,t{:}v,M'))=M'$ für alle $(M,t{:}v,M') \in \mathcal{TM}_{N,M_0}$.

Aus diesen beiden Definitionen kann man leicht sehen, daß der Schrittgraph eine Obermenge des Erreichbarkeitsgraphen ist, da auch jeder Schaltvorgang ein Schrittvorgang ist. Es stellt sich die Frage, ob der Schrittgraph auch eine echte Obermenge des Erreichbarkeitsgraphen sein kann, oder ob der Schrittgraph und der Erreichbarkeitsgraph immer identisch sind. Das folgende Beispiel, das wir im vorhergehenden Kapitel schon betrachtet haben, wird verdeutlichen, daß der Schrittgraph eine echte Obermenge des Erreichbarkeitsgraphen sein kann.

Beispiel 7

Wir betrachten ein SNL-System FARBE, dessen Spezifikationsteil eine Sorte `farbe` mit zwei Konstanten `rot` und `grün` definiert, die wir trivialerweise über eine zweielementige Menge mit den Elementen `rot` und `grün` realisiert denken. Das System selbst habe die Gestalt, wie sie das folgende Bild verdeutlicht.

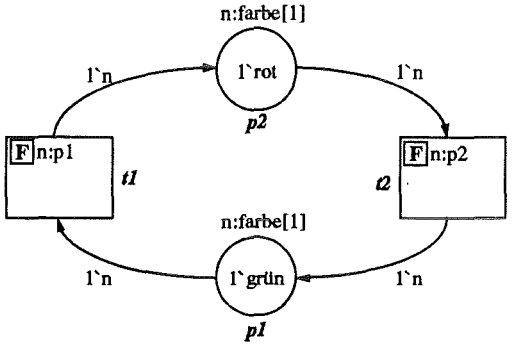


Abb. 3 SNL-System FARBE mit Anfangsmarkierung M_0

In diesem System kann keine Transition schalten, da für jede Transition die Nachbedingung nicht erfüllt ist (Die Kapazität 1 einer der Ausgabestellen wird überschritten).

Der T-Vektor $\mu=(1`\{n \rightarrow grün\}, 1`\{n \rightarrow rot\})^T$ ist aber aktivierbar, da beim Vertauschen der beiden Farbmarken während einer atomaren Operation die Kapazität der Stellen nicht überschritten wird. Nach dem Schalten dieses Schrittes haben wir dann das SNL-System mit der folgenden Markierung.

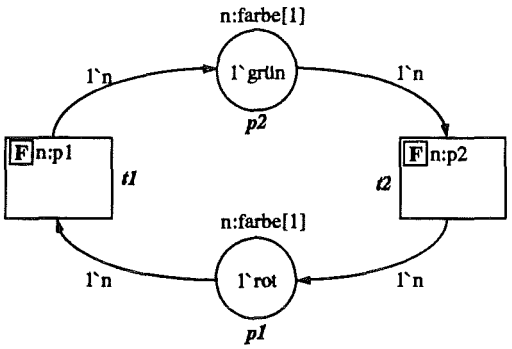
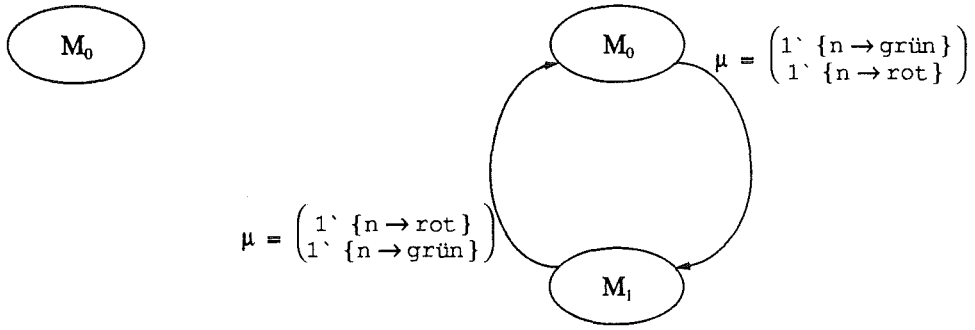


Abb. 4 SNL-System mit Markierung M_1

Durch Schalten des Schrittes μ=(1`{n→rot},1`{n→grün})^T tritt wieder die Anfangsmarkierung M_0 auf. Damit hat der Schrittgraph zwei Knoten M_0 und M_1 sowie zwei Kanten, während der Erreichbarkeitsgraph nur einen Knoten M_0 und keine Kanten hat. Das folgende Bild zeigt die beiden Graphen:



Erreichbarkeitsgraph **Schrittgraph**

Abb. 5 Vergleich zwischen Erreichbarkeitsgraph und Schrittgraph

Wenn auch die dargelegte Definition eines Graphen sehr anschaulich ist und sich besonders bei Problemstellungen, die bildlich dargestellt werden können, eignet, so läßt sie sich weniger gut für eine Algebraisierung von Problemen verwenden. Wir geben daher noch eine alternative, "mehr algebraische" Definition des Schritt- bzw. Erreichbarkeitsgraphen an, die auf der folgenden Definition eines Graphen basiert.

Definition 39 (alternative Definition eines Graphen)

Ein 3-Tupel *(V,dom,cod)* heißt *gerichteter Graph*, wenn gilt:

(a) V ist eine Menge, die wir Menge der Knoten und Kanten des Graphen nennen wollen.

(b) $dom,cod:V\to V$ sind Operationen auf V.

(c) Es gelten die Gleichungen

(1) $dom(dom(x))=cod(dom(x))=dom(x)$ für alle $x\in V$.

(2) $dom(cod(x))=cod(cod(x))=cod(x)$ für alle $x\in V$.

Wie können wir nun diese Definition als Graph im Sinne unserer ursprünglichen Definition sehen? Dazu müssen wir die Menge V zunächst in die Menge der Knoten und Kanten zerlegen. Ausgangspunkt hierfür ist, daß wir die Menge der Knoten als die Menge der Fixpunkte der beiden Operationen *dom* und *cod* definieren:

$$KN=\{x\in V \mid dom(x)=cod(x)=x\}$$

Die Menge der Kanten ist die Differenz KA=V\KN. Die Abbildungen a und e ergeben sich als Einschränkung von *dom* und *cod* auf die Menge der Kanten, d.h. $a=dom|_{KA}$ und $e=cod|_{KA}$. Da für alle ka∈KA ein y∈V existiert mit y=*dom*(ka) und aus Gleichung (1) der obigen Definition *dom*(y)=*dom*(*dom*(ka))=*dom*(ka)=y folgt, ist die oben definierte Einschränkung a offensichtlich eine Abbildung a:KA→KN, wie dies gewünscht ist. Aus (2) folgt analog, daß die als e definierte Abbildung wie gewünscht eine Abbildung e:KA→KN ist. Damit ist das 4-Tupel (KN,KA,a,e) wie oben definiert ein Graph in unserem ursprünglichen Sinne.

Wenn wir umgekehrt von einem Graphen (KN,KA,a,e) ausgehen, so läßt sich hieraus in einfacher Weise ein Graph in unserer alternativen Form definieren. Hierzu definieren wir V=KN∪KA und die Abbildungen *dom*:V→V und *cod*:V→V durch *dom*(x)=a(x), *cod*(x)=e(x) für alle x∈KA und *dom*(x)=*cod*(x)=x für alle x∈KN. Es ist dann leicht zu zeigen, daß (V,*dom*,*cod*) ein Graph gemäß unserer alternativen Definition ist. Man kann sich *dom* und *cod* ebenfalls als Funktionen vorstellen, die Elementen von V ihren Anfangs- bzw. Endpunkt zuordnen, wobei ein Knoten sich selbst als Anfangs- bzw. Endpunkt besitzt.

Unsere alternative Definition des Schrittgraphen ergibt sich damit zu:

Definition 40 **(Alternative Definition eines Schrittgraphen)**

Der Schrittgraph eines SNL-Systems N=(NSPEC,A) ist der Graph $\mathcal{SG}_N=(V,dom,cod)$, für den gilt:

(a) $V=[_SM_0>\cup\mathcal{S}_{N,M_0}$.

(b) dom:V→V ist definiert durch $dom(x)=M$ für alle $x=(M,\mu,M')\in\mathcal{S}_{N,M_0}$ und $dom(x)=x$ für alle $x\in[_SM_0>$.

(c) cod:V→V ist definiert durch $cod(x)=M'$ für alle $x=(M,\mu,M')\in\mathcal{S}_{N,M_0}$ und $cod(x)=x$ für alle $x\in[_SM_0>$.

Analog geben wir eine alternative Definition des Erreichbarkeitsgraphen an:

Definition 41 **(Alternative Definition eines Erreichbarkeitsgraphen)**

Der Erreichbarkeitsgraph eines SNL-Systems N=(NSPEC,A) ist der Graph $\mathcal{RGM}_N=(V,dom,cod)$, für den gilt:

(a) $V=[M_0>\cup\mathcal{TM}_{N,M_0}$.

(b) dom:V→V ist definiert durch $dom(x)=M$ für alle $x=(M,t{:}v,M')\in\mathcal{TM}_{N,M_0}$ und $dom(x)=x$ für alle $x\in[M_0>$.

(c) cod:V→V ist definiert durch $cod(x)=M'$ für alle $x=(M,t{:}v,M')\in\mathcal{TM}_{N,M_0}$ und $cod(x)=x$ für alle $x\in[M_0>$.

Schrittfolgen im SNL-System kann man im zugehörigen Schrittgraph als "Wege" interpretieren. Dazu müssen wir zunächst definieren, was ein Weg in einem Graph ist.

Definition 42 **(Weg in einem Graphen)**

Sei G=(KN,KA,a,e) ein Graph. Ein Wort $w\in KA^*$ heißt *Weg in G*, wenn gilt:
Falls $w=x_1..x_n$ ist, so ist $a(x_i)=e(x_{i-1})$ für alle i=2,..,n.
Ein Weg $w=x_1..x_n$ heißt *geschlossen*, wenn zusätzlich $a(x_1)=e(x_n)$ ist.

Aufgrund der Definition der Erreichbarkeitsrelation ist eine Markierung M' von der Anfangsmarkierung M_0 des SNL-Systems genau dann erreichbar, wenn es einen Weg im Schrittgraph von der Wurzel M_0 des Graphen zum Knoten M gibt. Wir haben also:

Bemerkung 9

Es sei N=(NSPEC,A) ein SNL-System und $M, M' \in [_SM_0>$. Die Markierung M' ist von der Markierung M genau dann schritterreichbar, wenn es einen Weg *w* im Schrittgraph von M nach M' gibt.

Im Schrittgraph ist jeder Knoten, da er per Definition eine von der Anfangsmarkierung M_0 aus schritterreichbare Markierung ist, über einen Weg vom Knoten M_0 aus erreichbar. Einen Graphen, der einen Knoten K besitzt, so daß alle Knoten von diesem aus über Wege erreichbar sind, nennt man auch Wurzelgraph mit Wurzel K. Es gilt daher:

Bemerkung 10

Der Schrittgraph ist ein Wurzelgraph mit M_0 als Wurzel.

Wir haben bisher immer außer dem Schrittgraph auch den Erreichbarkeitsgraph angegeben, der, wie wir gesehen haben im Schrittgraph, enthalten ist. Diese beiden Graphen drücken unterschiedliche Semantiken von Parallelität aus. Wenn uns auch die Semantik des Schaltens von Schritten sehr interessante Aspekte von Parallelität liefern kann, ist sie gegenüber der Semantik des Schaltens von Transitionen in der computergestützten Realisierung viel zeitaufwendiger. Man hat beim Schalten von Schritten das Problem, daß man von einer Markierung aus alle möglichen Schritte bilden und mit diesen Schritten dann das SNL-System schalten lassen muß. Beim Erreichbarkeitsgraph sind dagegen nur alle aktivierbaren Transitionen zu schalten. Wenn man sich veranschaulicht, daß beim Schrittgraph die Menge der möglichen Schritte gerade alle Kombinationen der aktivierbaren Transitionen sind, so wird deutlich, um wieviel umfangreicher die Berechnung des Schrittgraphen ist.

Deshalb beschreibt die Arbeit auch das theoretische Konzept auf Basis von Schrittsemantik, während das Werkzeug SMARAGD sich auf das Schalten von Transitionen beschränkt. Dies stellt sicherlich eine Einschränkung bezüglich den Möglichkeiten einer Modellierung von Systemen dar, beschränkt aber nicht die Menge der Systeme die modelliert werden können.

Die Beschränkung auf das Schalten von Transitionen ist eine allgemein übliche Vorgehensweise (siehe dazu z.B. auch [Lindquist89, Ochsenschläger88, Starke90]). Es wird jedoch meistens nicht darauf aufmerksam gemacht, daß Erreichbarkeitsgraph und Schrittgraph verschieden sein können.

Zur vereinfachten Analyse können wir den allgemeinen Erreichbarkeitsgraphen weiter reduzieren. Beim Erreichbarkeitsgraph können zwischen zwei Knoten mehrere Kanten existieren, je nachdem, ob eine Markierung aus der anderen durch Schalten von einer Transition in mehreren Modi entsteht. Die Erreichbarkeit einer Markierung von einer anderen Markierung eines SNL-Systems aus ist eine der vielen Eigenschaften von SNL-Systemen, die von den Markierungen des SNL-Systems und dem Änderungsverhalten der Markierungen, nicht aber davon, wie diese Änderung zustande kommt, abhängen. Insbesondere ist der Modus, mit dem eine Transition schaltet, für viele Eigenschaften von SNL-Systemen nicht relevant. Wir werden in solchen Fällen nur einen (reduzierten) Erreichbarkeitsgraphen betrachten, der aus dem allgemeinen Erreichbarkeitsgraphen dadurch entsteht, daß wir das Schalten von Transitionen nicht mehr nach der Art der Substitution unterscheiden.

Definition 43 **(reduzierter Erreichbarkeitsgraph)**

Sei N=(NSPEC,A) ein SNL-System. Der (reduzierte) Erreichbarkeitsgraph $\mathcal{RG}_N$ von N ist definiert als der Graph $\mathcal{RG}_N=([M_0>, \mathcal{T}_{N,M_0}, a, e)$, für den gilt:

(a) $a((M,t,M'))=M$ für alle $(M,t,M') \in \mathcal{T}_{N,M_0}$,

(b) $e((M,t,M'))=M$ für alle $(M,t,M') \in \mathcal{T}_{N,M_0}$.

Unsere Bemerkung über die Erreichbarkeit können wir jetzt so formulieren:

Bemerkung 11

Es sei N=(NSPEC,A) ein SNL-System und M, M' $\in [M_0>$. Die Markierung M' ist von der Markierung M genau dann *erreichbar*, wenn es einen Weg *w* im reduzierten Erreichbarkeitsgraphen von M nach M' gibt.

Wir haben in diesem Abschnitt den Erreichbarkeitsgraph eines SNL-Systems definiert. Wir werden später noch sehen, daß der Erreichbarkeitsgraph die wesentliche Grundlage der Erreichbarkeitsanalyse darstellt. Aus ihm wird der sogenannte Graph der starken Zusammenhangskomponenten generiert. Aus diesen beiden Graphen kann man dann sehr viele Eigenschaften des zugehörigen SNL-Systems ermitteln. Wir werden nun die starken Zusammenhangskomponenten und den daraus resultierenden Graph definieren und danach die Eigenschaften eines SNL-Systems behandeln, die man auf der Grundlage dieser beiden Graphen ermitteln kann.

4.1.2 Starke Zusammenhangskomponenten

Der reduzierte Erreichbarkeitsgraph entsteht aus dem allgemeinen Erreichbarkeitsgraph durch Identifizieren von Kanten. Ein weiterer für die Analyse wichtiger Graph entsteht aus dem Schritt- bzw. Erreichbarkeitsgraph durch Identifizierung von Knoten und Kanten. Hierzu wollen wir die Erreichbarkeitsrelation zunächst zu einer Äquivalenzrelation zwischen Markierungen ausdehnen, die uns die starken Zusammenhangskomponenten der Markierungen eines SNL-Systems liefert.

Definition 44 **(starke Schrittzusammenhangskomponenten)**

Sei N=(NSPEC,A) ein SNL-System und M_0 seine Anfangsmarkierung. Die Äquivalenzrelation $\Leftrightarrow_S \subseteq [_SM_0> \times [_SM_0>$ sei wie folgt definiert:

$$M \Leftrightarrow_S M' \text{ genau dann, wenn } M \Longrightarrow_S M' \text{ und } M' \Longrightarrow_S M \text{ gilt.}$$

Die Äquivalenzklassen von $\Leftrightarrow_S$ in $[_SM_0>$ nennt man *starke Schrittzusammenhangskomponenten* von N.

Es ist offensichtlich, daß $\Leftrightarrow_S$ eine Äquivalenzrelation ist. Man kann daher den Faktorgraph $\mathcal{SG}_N/\Leftrightarrow_S$ bilden, der dadurch entsteht, daß wir zunächst alle Knoten des Schrittgraphen identifizieren, die innerhalb derselben starken Schrittzusammenhangskomponente liegen, und dann Kanten nur noch zwischen den Schrittzusammenhangskomponenten einführen, für die ein Weg im Schrittgraph von einem Repräsentant der einen Schrittzusammenhangskomponente zu einem Repräsentant der anderen Schrittzusammenhangskomponente geht.

Definition 45 **(Graph der starken Schrittzusammenhangskomponenten)**

Sei N=(NSPEC,A) ein SNL-System und $\mathcal{SG}_N$ der zugehörige Schrittgraph. Der Graph der starken Schrittzusammenhangskomponenten von N (im Zeichen $SSCG_N$) ist der Graph $SSCG_N = (SSCC_N, KA, a, e)$, für den gilt:

(a) $SSCC_N = [_S M_0> / \Leftrightarrow_S$,

(b) $KA = \{(sscc_1, sscc_2) \in SSCC_N \times SSCC_N \mid \exists M \in sscc_1, M' \in sscc_2 \text{ mit: } M \Longrightarrow_S M'\}$,

(c) für $ka = (sscc_1, sscc_2) \in KA$ ist $a(ka) = sscc_1$ und $e(ka) = sscc_2$.

Dasselbe wollen wir wieder für unseren Spezialfall des Schaltens von Transitionen definieren.

Definition 46 **(starke Zusammenhangskomponenten)**

Sei N=(NSPEC,A) ein SNL-System und M_0 seine Anfangsmarkierung. Die Äquivalenzrelation $\Leftrightarrow \subseteq [M_0> \times [M_0>$ sei wie folgt definiert:

$$M \Leftrightarrow M' \text{ genau dann, wenn } M' \Longrightarrow M' \text{ und } M' \not\Longrightarrow M \text{ gilt.}$$

Die Äquivalenzklassen von $\Leftrightarrow$ in $[M_0>$ nennt man *starke Zusammenhangskomponenten (Strongly Connected Component (SCC))* von N.

Wir definieren nun den Graph der starken Zusammenhangskomponenten, wobei wir hier von dem reduzierten Erreichbarkeitsgraph ausgehen, da dieser weniger Kanten als der allgemeine Erreichbarkeitsgraph hat, aber zur Berechnung des Graphen der starken Zusammenhangskomponenten ausreicht.

Definition 47 **(Graph der starken Zusammenhangskomponenten)**

Sei N=(NSPEC,A) ein SNL-System und $\mathcal{RG}_N$ der zugehörige reduzierte Erreichbarkeitsgraph. Der Graph der starken Zusammenhangskomponenten von N (im Zeichen SCG_N) ist der Graph $SCG_N = (SCC_N, KA, a, e)$, für den gilt:

(a) $SCC_N = [M_0> / \Leftrightarrow$,

(b) $KA = \{(scc_1, scc_2) \in SCC_N \times SCC_N \mid \exists M \in scc_1, M' \in scc_2 \text{ mit: } M \Longrightarrow M'\}$,

(c) für $ka = (scc_1, scc_2) \in KA$ ist $a(ka) = scc_1$ und $e(ka) = scc_2$.

Wie der Schrittgraph ist auch der Graph der starken Schrittzusammenhangskomponenten ein Wurzelgraph. Auf Grund seiner Konstruktionsweise ist er jedoch azyklisch.

Bemerkung 12

Sei N=(NSPEC,A) ein SNL-System und M_0 die zugehörige Anfangsmarkierung.

(1) $SSCG_N$ ist ein azyklischer Wurzelgraph, dessen Wurzel diejenige Schrittzusammenhangskomponente ist, die die Anfangsmarkierung M_0 enthält.

(2) SCG_N ist ein azyklischer Wurzelgraph, dessen Wurzel diejenige Zusammenhangskomponente ist, die die Anfangsmarkierung M_0 enthält.

Wie bei der Definition der Erreichbarkeit kann man auch zwischen den starken Schrittzusammenhangskomponenten eines SNL-Systems eine Erreichbarkeitsrelation einführen.

Definition 48 **(Nachfolgerrelation zwischen starken Schrittzusammenhangskomponenten)**

Sei N=(NSPEC,A) ein SNL-System und $SSCC_N$ die Menge der starken Schrittzusammenhangskomponenten von N.

(a) $sscc_2 \in SSCC_N$ heißt direkter Nachfolger von $sscc_1 \in SSCC_N$, wenn $(sscc_1, sscc_2)$ eine Kante des Graphen der starken Schrittzusammenhangskomponenten $SSCG_N$ ist. Wir schreiben diese Relation im Zeichen mit $sscc_1 \Longrightarrow_S sscc_2$.

(b) Den reflexiven und transitiven Abschluß der unter (a) eingeführten Relation bezeichnen wir mit $\not\Longrightarrow_S$. Wir nennen diese Relation *Nachfolgerrelation* und sagen für $sscc_1 \not\Longrightarrow_S sscc_2$ auch, daß $sscc_2$ ein Nachfolger von $sscc_1$ ist.

(c) Elemente aus $SSCC_N$, die keine Nachfolger (außer sich selbst) besitzen, sind *Blätter* des Graphen $SSCG_N$.

Definition 49 **(Nachfolgerrelation zwischen starken Zusammenhangskomponenten)**

Sei N=(NSPEC,A) ein SNL-System und SCC_N die Menge der starken Zusammenhangskomponenten von N.

(a) $scc_2 \in SCC_N$ heißt direkter Nachfolger von $scc_1 \in SCC_N$, wenn (scc_1, scc_2) eine Kante des Graphen der starken Zusammenhangskomponenten SCG_N ist. Wir schreiben diese Relation im Zeichen mit $scc_1 \Rightarrow scc_2$.

(b) Den reflexiven und transitiven Abschluß der unter (a) eingeführten Relation bezeichnen wir mit $\hookrightarrow$. Wir nennen diese Relation *Nachfolgerrelation* und sagen für $scc_1 \hookrightarrow scc_2$ auch, daß scc_2 ein Nachfolger von scc_1 ist.

(c) Elemente aus SCC_N, die keine Nachfolger (außer sich selbst) besitzen, sind *Blätter* des Graphen SCG_N.

Bei der Erreichbarkeitsanalyse wollen wir Aussagen über Eigenschaften des SNL-Systems machen. Um solche Aussagen machen zu können, ist es nötig alle Zustände des SNL-Systems zu generieren. Im Erreichbarkeitsgraphen werden gerade alle Zustände des SNL-Systems und ihre Beziehungen festgehalten. Eine Reduzierung des Erreichbarkeitsgraphen ist der Graph der starken Zusammenhangskomponenten. In ihm werden alle Knoten mit gleicher Eigenschaft (wobei die Eigenschaft die wechselseitige Erreichbarkeit ist) zusammengefaßt. Diese beiden Graphen stellen die Grundlage für die Bestimmung von vielen Eigenschaften des SNL-Systems dar. Wir wollen im folgenden Abschnitt einige Eigenschaften, die aus dem Erreichbarkeitsgraph und dem Graph der starken Zusammenhangskomponenten gewonnen werden können, vorstellen.

4.2 Analysierbare SNL-Systemeigenschaften

Wir unterteilen die SNL-Systemeigenschaften in verschiedene Gruppen, wie z.B. statistische Eigenschaften oder Lebendigkeitseigenschaften, und erläutern diese Eigenschaften. Wir beginnen mit statistischen SNL-Systemeigenschaften, wie sie auch in anderen Analysewerkzeugen existieren, und die auch auf unser SNL-Modell übertragbar sind.

4.2.1 Statistische SNL-Systemeigenschaften

Unter statischen SNL-Systemeigenschaften verstehen wir z.B. die Anzahl der

- Stellen
- toten Stellen
- Transitionen
- toten Transitionen
- Knoten des Erreichbarkeitsgraphen ($|[M_0>|$)
- Kanten des Erreichbarkeitsgraphen ($|\mathcal{TM}_{N,M_0}|$)
- starken Zusammenhangskomponenten ($|SCC_N|$)

Wir müssen dazu noch den Begriff der toten Stellen definieren.

Definition 50 **(tote Stellen)**

Sei N=(NSPEC,A) ein SNL-System mit Anfangsmarkierung M_0, $\mathcal{RGM}_N$ der Erreichbarkeitsgraph und $\mathcal{TM}_{N,M_0}$ die Menge aller Schaltvorgänge von Transitionen in speziellen Modi von N. Eine Stelle $p \in P$ heißt *tote Stelle*, wenn gilt: $\forall (M, t_i, M') \in \mathcal{TM}_{N,M_0} : p \notin {}^\bullet t_i^\bullet$.

Mit weiteren statistischen Abfragen kann man sich Informationen über die starken Zusammenhangskomponenten beschaffen. Es sind dies:

- Anzahl der Knoten in einer starken Zusammenhangskomponente scc_i.
- Transitionen, die in scc_i hineinführen.
- Transitionen, die innerhalb von scc_i schalten.
- Transitionen, die aus scc_i herausführen.

Weitere bestimmbare Fähigkeiten von SNL-Systemen sind:

- Vorgänger eines Knotens des $\mathcal{RGM}_N$ oder SCG_N.
- Nachfolger eines Knotens des $\mathcal{RGM}_N$ oder SCG_N.
- gemeinsame Folgemarkierungen von Knoten des $\mathcal{RGM}_N$.
- Heimzustand eines Knotens des $\mathcal{RGM}_N$.

Heimzustände und Folgemarkierungen sind dabei wie folgt definiert:

Definition 51 (Heimzustand und Heimraum)

Sei N=(NSPEC,A) ein SNL-System mit Anfangsmarkierung M_0. Ein Heimzustand (Homestate) bzw. Heimraum (Homespace) ist diejenige Menge von Markierungen, die von allen Folgemarkierungen einer gegebenen Markierung bzw. Markierungsteilmenge aus erreichbar ist.

(1) Sei $M \in [M_0>$.

$$\text{homestate}(M) := \{M' \in [M_0> \mid \forall M'' \in [M> : M'' \Rightarrow M'\}.$$

(2) Sei $M \subseteq [M_0>$.

$$\text{homespace}(M) := \bigcap_{M \in M} \text{homestate}(M).$$

Definition 52 (Gemeinsame Folgemarkierungen)

Die Menge *cf* der gemeinsamen Folgemarkierungen (common followers) einer gegebenen Markierungsteilmenge $M \subseteq [M_0>$ ist definiert als

$$cf(M) := \bigcap_{M \in M} [M>$$

Nach den statistischen Eigenschaften eines SNL-Systems, die SMARAGD analysiert, werden wir im nächsten Abschnitt verschiedene Lebendigkeitseigenschaften des SNL-Systems betrachten. Wir werden später noch sehen, daß diese Lebendigkeitseigenschaften eine wichtige Rolle bei der Erreichbarkeitsanalyse spielen.

4.2.2 Lebendigkeitseigenschaften von SNL-Systemen

Jede Markierung, die nicht der Anfangsmarkierung eines SNL-Systems entspricht und unter der keine Schritte aktivierbar sind, nennt man eine (statische) Verklemmung.

Definition 53 (statische Verklemmung)

Sei N=(NSPEC,A) ein SNL-System mit der Anfangsmarkierung M_0 und $M \in [M_0>$ mit $M \neq M_0$.

(a) M heißt *(statische) Verklemmung des SNL-Systems N* genau dann, wenn keine Transition existiert, die unter M aktivierbar ist.

(b) N heißt *verklemmungsfrei* genau dann, wenn für alle Markierungen $M \in [M_0>$ gilt: M ist keine statische Verklemmung von N.

Der Begriff der (statischen) Verklemmung beschreibt somit Markierungen eines SNL-Systems, in denen der dynamische Ablauf des Tokenspiels sein Ende findet, da unter der betreffenden Markierung keine weiteren Transitionen mehr aktivierbar sind. Im Gegensatz hierzu beschreiben dynamische Verklemmungen Markierungen (bzw. Mengen von Markierungen), unter denen zwar noch Transitionen aktivierbar sind, aber die aus dem Schalten dieser Transitionen resultierenden Folgemarkierungen wieder zu der Ausgangsmarkierung (einem Element der Ausgangsmarkierungsmenge) führt.

Definition 54 (triviale dynamische Verklemmung)

Sei N=(NSPEC,A) ein SNL-System mit der Anfangsmarkierung M_0 und $M \in [M_0>$ mit $M \neq M_0$. M heißt *triviale dynamische Verklemmung* genau dann, wenn gilt:

(1) M ist keine statische Verklemmung.

(2) Für jeden Schaltvorgang (M,t,M') gilt: M'=M.

Bei einer trivialen dynamischen Verklemmung kann als Folgemarkierung also nur noch die Markierung M selbst auftreten. Der Zustand des Systems (beschrieben durch die Markierung) ist also ab einer trivialen dynamischen Verklemmung konstant.

Verklemmungen lassen sich sehr einfach bzgl. des Graphen der starken Zusammenhangskomponenten interpretieren. Die Bedingung, daß keine Folgemarkierung oder höchstens die Markierung selbst als Folgemarkierung auftritt, bedeutet, daß eine Verklemmung ein Blatt vom SCG_N sein muß, das genau ein Element enthält. Bei einer dynamischen Verklemmung muß zusätzlich eine Transition unter der besagten Markierung aktivierbar sein.

Bemerkung 13

$M \in [M_0>$ ist eine triviale dynamische Verklemmung genau dann, wenn ein Blatt $scc \in SCC_N$ des Graphen SCG_N existiert, für das gilt:

(1) $scc = \{M\}$,

(2) $M \neq M_0$,

(3) es gibt eine Transition, die unter M aktivierbar ist.

Wenn (1) und (2) nicht aber (3) erfüllt sind, handelt es sich offensichtlich um eine statische Verklemmung.

Die obige Bemerkung legt es nahe, daß man die Definition der dynamischen Verklemmung auf Blätter des Graphen der starken Zusammenhangskomponenten ausdehnt, die nicht nur aus einem Element bestehen.

Definition 55 (dynamische Verklemmung)

Sei N=(NSPEC,A) ein SNL-System mit der Anfangsmarkierung M_0. Eine Teilmenge $\mathcal{M} \subset [M_0>$ heißt *dynamische Verklemmung* genau dann, wenn gilt:

(1) Es gibt ein Blatt $scc \in SCC_N$ von SCG_N mit $scc = \mathcal{M}$.

(2) $M_0 \notin \mathcal{M}$ und $|\mathcal{M}| > 1$.

Es ist offensichtlich, daß gilt:

Bemerkung 14

(1) Statische Verklemmungen, triviale dynamische Verklemmungen und dynamische Verklemmungen sind wechselseitig disjunkte Blätter vom SCG_N.

(2) Blätter vom SCG_N, die nicht die Anfangsmarkierung enthalten, sind entweder dynamische, trivial dynamische oder statische Verklemmungen.

Wir wollen uns nun einzelnen Transitionen zuwenden und Eigenschaften dieser bzgl. ihrer Aktivierbarkeit untersuchen. Wir beginnen mit der folgenden Definition:

Definition 56 (Tote Transitionen und Lebendigkeit)

Sei N=(NSPEC,A) ein SNL-System und M_0 dessen Anfangsmarkierung.

(a) Eine Transition $t \in T$ heißt *tot*, wenn t unter keiner Markierung $M \in [M_0>$ aktivierbar ist. Andernfalls nennen wir t *schwach lebendig*.

(b) Das SNL-System N heißt *schwach lebendig*, wenn keine seiner Transitionen tot ist.

(c) Eine Transition $t \in T$ heißt *stark lebendig*, wenn zu jeder Markierung $M \in [M_0>$ eine Folgemarkierung $M' \in [M_0>$ existiert, so daß t unter M' aktiviert ist.

(d) Das SNL-System N heißt *stark lebendig*, wenn alle seine Transitionen stark lebendig sind.

Damit eine Transition $t_i \in T$ schwach lebendig ist, reicht es aus, wenn im zum SNL-System N gehörigen reduzierten Erreichbarkeitsgraph $\mathcal{RG}_N$ eine Kante der Gestalt (M, t_i, M') existiert. Damit eine Transition t_i stark lebendig ist, müssen in jeder starken Zusammenhangskomponente $scc \in SCC_N$ Markierungen $M,M' \in [M_0>$ existieren, so daß (M, t_i, M') ein Schaltvorgang von N ist. Weiter ist jede stark lebendige Transition sicher schwach lebendig und bei nur einer starken Zusammenhangskomponente ist jede schwach lebendige Transition schon stark lebendig. Wir haben also:

Bemerkung 15

Sei N=(NSPEC,A) ein SNL-System und M_0 dessen Anfangsmarkierung.

(1) Die Menge der stark lebendigen Transitionen ist genau die Menge aller innerhalb der Blattknoten des SCG_N schaltenden Transitionen.

(2) Jede stark lebendige Transition ist trivialer Weise schwach lebendig.

(3) Besitzt das SNL-System nur eine starke Zusammenhangskomponente, so ist jede schwach lebendige Transition bereits stark lebendig.

(4) Ein SNL-System, das stark lebendig ist, ist verklemmungsfrei.

Wenn eine Transition lebendig ist, so stellt sich weiter die Frage, wie oft diese Transition im dynamischen Ablauf des SNL-Systems schalten kann. Hier interessiert vor allem, ob die Transition nur endlich oft oder unendlich oft aktivierbar ist.

Definition 57 (Beschränkte Aktivierbarkeit)

Sei N=(NSPEC,A) ein SNL-System und M_0 die Anfangsmarkierung. Eine Transition $t \in T$ heißt *beschränkt aktivierbar* genau dann, wenn sie nicht tot ist, aber eine Konstante $n \in \mathbb{N}$ existiert, so daß $t \in T$ höchstens n-mal in einer beliebigen Schaltfolge $(M_i,t_i,M'_i)_{i \in I}$ auftreten kann.

Da wir es mit endlichen Erreichbarkeitsgraphen zu tun haben, kann eine Transition nur dann unbeschränkt aktivierbar sein, wenn sie in einem geschlossenen Weg im Erreichbarkeitsgraph vorkommt. Solche geschlossenen Wege werden aber durch die starke Zusammenhangsrelation gerade identifiziert. Daher gilt:

Bemerkung 16

Sei N=(NSPEC,A) ein SNL-System und M_0 die Anfangsmarkierung. Eine Transition $t \in T$ ist genau dann beschränkt aktivierbar, wenn sie nicht tot ist und nur in Übergängen zwischen starken Zusammenhangskomponenten schalten kann.

Wenn alle Transitionen nur beschränkt aktivierbar sind, kann der Erreichbarkeitsgraph keine geschlossenen Wege enthalten. Es gilt daher:

Bemerkung 17

Wenn alle Transitionen eines SNL-Systems N beschränkt aktivierbar sind, gibt es im System keine dynamischen Verklemmungen.

Neben dem Verhalten von Transitionen sind auch Fragen interessant, die die Reproduzierbarkeit von Markierungen betreffen. Hierzu führen wir die folgende Definition ein:

Definition 58 (Schwache Reproduzierbarkeit von Markierungen)

Sei N=(NSPEC,A) ein SNL-System und M_0 die Anfangsmarkierung von N. Eine Markierungsteilmenge $\mathcal{M} \subset [M_0>$ heißt *schwach reproduzierbar* genau dann, wenn jede Markierung $M \in \mathcal{M}$ nach ihrem Verlassen wieder erreicht werden kann.

Markierungen innerhalb einer starken Zusammenhangskomponente, die mehr als ein Element enthält, sind offensichtlich nach Verlassen der Markierung wieder erreichbar. Wir haben also:

Bemerkung 18

Jede starke Zusammenhangkomponente $scc \in SCC_N$, die mehr als ein Element enthält, ist schwach reproduzierbar.

Eine stärkere Version der schwachen Reproduzierbarkeit ist die starke Reproduzierbarkeit.

Definition 59 (Starke Reproduzierbarkeit von Markierungen)

Sei N=(NSPEC,A) ein SNL-System und M_0 die Anfangsmarkierung von N. Eine Markierungsteilmenge $\mathcal{M} \subset [M_0>$ heißt *stark reproduzierbar*, wenn jede Markierung $M \in \mathcal{M}$ von jeder Folgemarkierung $M' \in [M>$ mit $M \neq M'$ aus erreichbar ist.

Mit der Beobachtung, daß starke Zusammenhangskomponenten, die Blätter des Graphen der starken Zusammenhangskomponenten sind, nur noch Markierungen enthalten, von denen aus Schaltvorgänge die Zusammenhangskomponente nicht mehr verlassen können, können wir folgern:

Bemerkung 19

Starke Zusammenhangskomponenten, die Blätter des Graphen der starken Zusammenhangskomponenten sind und mehr als ein Element enthalten, sind stark reproduzierbar.

In SMARAGD gibt es die Möglichkeit, das SNL-System auf statische und dynamische Verklemmungen (*Deadlocks* und *Livelocks*) zu überprüfen. Hierbei faßt SMARAGD die Begriffe dynamische und triviale dynamische Verklemmung zu einem Begriff *Livelock* zusammen.

4.2.3 Wege und Knoten mit gleicher Eigenschaft

Für die Analyse eines SNL-Systems ist es nützlich, wenn man sich die kürzesten Wege von einem Knoten des Erreichbarkeitsgraphen zu einem anderen anschauen kann.

Eine weitere nützliche Fähigkeit ist die Möglichkeit sich alle Knoten, die bestimmte Eigenschaften haben, aufzulisten. Diese Eigenschaften können z.B. sein:

- Auf bestimmten Stellen liegt immer die gleiche Anzahl von Marken.
- Auf bestimmten Stellen liegt immer eine Mindestanzahl von Marken.
- Auf bestimmten Stellen liegt immer eine Maximalanzahl von Marken.
- Eine bestimmte Transition ist aktivierbar.
- Eine bestimmte Transition schaltet.

4.2.4 Konflikte zwischen Transitionen

Wir definieren im folgenden was wir unter einem Konflikt zwischen Transitionen verstehen. Wir unterscheiden dabei in Schalt- und Substitutionskonflikt.

Definition 60 (Schaltkonflikt)

Sei N=(NSPEC,A) ein SNL-System mit Anfangsmarkierung M_0. t_i und t_j seien zwei Transitionen von N, die in einer Markierung $M \in [M_0>$ aktivierbar sind. (M,t_i,M_i) und (M,t_j,M_j) seien zwei Schaltvorgänge. Zwischen t_i und t_j besteht ein Schaltkonflikt, falls t_i nicht in M_j und t_j nicht in M_i aktivierbar sind.

Definition 61 (Substitutionskonflikt)

Sei N=(NSPEC,A) ein SNL-System mit Anfangsmarkierung M_0. t sei eine Transition, die in den Modi v_i und v_j in einer Markierung $M \in [M_0>$ aktivierbar ist. $(M,t{:}v_i,M_i)$ und $(M,t{:}v_j,M_j)$ seien zwei Schaltvorgänge. Zwischen $t{:}v_i$ und $t{:}v_j$ besteht ein Substitutionskonflikt, falls $t{:}v_i$ nicht in M_j und $t{:}v_j$ nicht in M_i aktivierbar sind.

4.2.5 Gefrorene Marken

Unter gefrorenen Marken verstehen wir Marken, die nicht mehr von einer Stelle entfernt werden. Wir unterscheiden zwischen zwei Arten von gefrorenen Marken, den lokal und den global gefrorenen Marken. Unter global gefrorenen Marken verstehen wir dabei Marken ω einer Stelle p, die ab einer Markierung M in allen Markierungen $M' \in [M>$ auf der Stelle p liegen bleiben. Lokal gefrorene Marken sind Marken, die in einer Zusammenhangskomponente auf einer Stelle liegen. Wir wollen dazu die formalen Definitionen angeben:

Definition 62 (gefrorene Marken)

Sei N=(NSPEC,A) ein SNL-System und $M=(m_p)_{p\in P} \in [M_0>$ mit $m_p \in MULT^+_{fin}(A)$ eine Markierung von N. $\omega \in A$ heißt *(global) gefrorene Marke* der Stelle $p \in P$, wenn ein $d_\omega \in \mathbb{N}$ existiert, so daß für die Multimenge $\varpi = d_\omega \cdot` \omega$ gilt:

$$\forall\, M'=(m'_p)_{p\in P} \in [M> \text{ mit } m'_p \in MULT^+_{fin}(A) \text{ ist: } m'_p \cap \varpi = \varpi.$$

Definition 63 (lokal gefrorene Marken)

Sei N=(NSPEC,A) ein SNL-System.

$\omega \in A$ heißt *lokal gefrorene Marke* der Stelle $p \in P$ in der starken Zusammenhangskomponente $scc \in SCC_N$, wenn ein $d_\omega \in \mathbb{N}$ existiert, so daß für die Multimenge $\varpi = d_\omega \cdot` \omega$ gilt:

$$\forall\, M'=(m'_p)_{p\in P} \in scc \text{ mit } m'_p \in MULT^+_{fin}(A) \text{ ist: } m'_p \cap \varpi = \varpi.$$

Es erscheint wichtig, Algorithmen zur Entdeckung von gefrorenen Marken zu haben. Gefrorene Marken müssen natürlich nicht unbedingt eine nicht gewünschte Situation darstellen, werden es aber in vielen Fällen sein, worauf wir weiter unten noch näher eingehen werden.

Wir wollen nun noch den Begriff der gefrorenen Marken weiter eingrenzen. Unsere Definition von gefrorenen Marken läßt zu, daß Marken auf einer Stelle gefroren sind, aber diese Marken durch das Schalten einer markenproduzierenden Transition weiterhin durch das SNL-System fließen. Eine markenproduzierende Transition vermehrt beim Schalten Marken, d.h. nach dem Schalten liegen auf den Umgebungsstellen mehr Marken als vor dem Schalten. Es kann dabei vorkommen, daß eine Marke zwar nicht von einer Stelle abgezogen, diese Marke aber von einer Transition erzeugt und auf eine andere Stelle gelegt wird. Die Art der Kantenbeschriftungen in SNL-Systemen läßt solche Transitionen zu. Wir wollen deshalb den Begriff der flußgefrorenen Marken einführen.

Definition 64 (flußgefrorene Marken)

Sei N=(NSPEC,A) ein SNL-System. Eine (ab einer Markierung M) gefrorene Marke $\omega \in A$ einer Stelle $p \in P$ heißt *flußgefroren*, falls für alle Schaltvorgänge $(M',t,M'') \in TOCC(NSPEC,A)$ mit $M'=(m'_p)_{p\in P}, M''=(m''_p)_{p\in P} \in [M\rangle$ und $t \in {}^\bullet p^\bullet$ gilt:

$$\forall\, p' \in penv(t)\backslash\{p\} \text{ ist : } \omega \notin Tr(m''_{p'}).$$

Wir wollen nun für die vorgestellten Systemeigenschaften die Algorithmen angeben, mit Hilfe denen man diese Eigenschaften aus dem $\mathcal{RG}_N$ und SCG_N ermitteln kann.

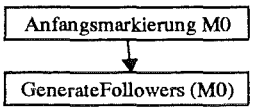
4.3 Algorithmen

Wir haben den Erreichbarkeitsgraph, den Graph der starken Zusammenhangskomponenten und verschiedene Eigenschaften des SNL-Systems beschrieben. Die Algorithmen zum Aufbau der Graphen und der Ermittlung der analysierbaren Eigenschaften werden in diesem Abschnitt beschrieben.

4.3.1 Der Aufbau des Erreichbarkeitsgraphen

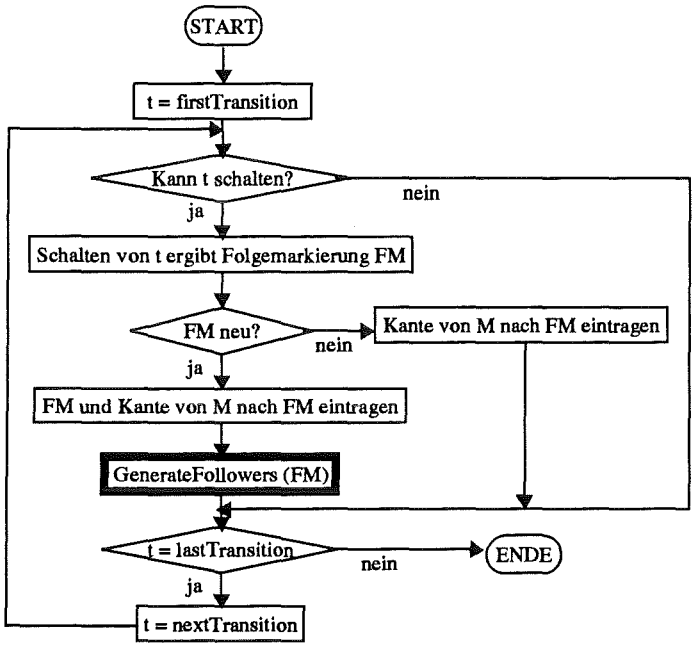
In den vorhergehenden Kapiteln wurden die Grundlagen für den Erreichbarkeitsgraphen und den Graphen der starken Zusammenhangskomponenten beschrieben. Wir wollen in diesem Kapitel einen Algorithmus für den Aufbau des Erreichbarkeitsgraphen angeben.

Das Prinzip, das dem Algorithmus zugrunde liegt, ist sehr einfach. Ausgehend von der Anfangsmarkierung werden zu jeder Markierung die direkten Folgemarkierungen, durch rekursives Aufrufen einer Funktion *GenerateFollowers*, bestimmt.



Ausgehend von der Anfangsmarkierung M0 wird durch *GenerateFollowers* der Erreichbarkeitsgraph aufgebaut, indem sich *GenerateFollowers* selbst rekursiv aufruft.

GenerateFollowers wird mit einer Markierung *M* als Parameter aufgerufen. *GenerateFollowers* untersucht zuerst, welche Transitionen in dieser Markierung *M* schalten können. Diese Transitionen werden der Reihe nach schalten gelassen, wodurch die direkten Folgemarkierungen generiert werden. Von einer Folgemarkierung wird überprüft, ob sie schon im Erreichbarkeitsgraph vorhanden war oder nicht. Falls nicht, werden von dieser Folgemarkierung ihrerseits die Folgemarkierungen durch Aufruf von *GenerateFollowers* berechnet. Übergänge von Markierungen, die durch das Schalten der Transitionen entstehen, werden als Kanten und neue Markierungen als Knoten in den Erreichbarkeitsgraph eingetragen.



Das Ablaufdiagramm zeigt die Abarbeitung eines Aufrufes von *GenerateFollowers(M)*, wobei M eine Markierung darstellt.

firstTransition, *nextTransition* und *lastTransition* sind Hilfsfunktionen, mit deren Hilfe die Transitionen eines SNL-Systems durchlaufen werden können.

GenerateFollowers ruft sich selbst rekursiv auf, was durch die dickere Umrandung des betreffenden Kastens hervorgehoben werden soll.

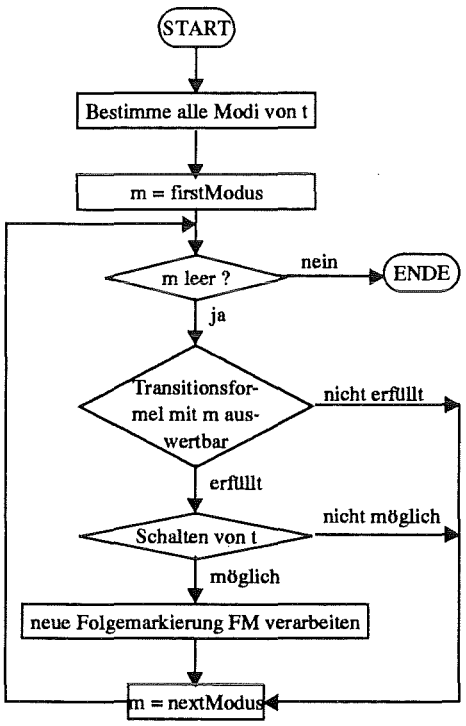
Abb. 6 Aufruf von *GenerateFollowers(M)*

Wichtige Operationen im obigen Algorithmus stellen die Teile *Kann t schalten?* und *Schalten von t ergibt Folgemarkierung FM*, die sich mit dem Schalten von Transitionen befassen, dar. Diese Operationen wollen wir im nächsten Abschnitt genauer erläutern.

Der Zeitaufwand für den Aufbau des Erreichbarkeitsgraphen ist O(n*t*m). Dabei ist n die Anzahl der Knoten des Erreichbarkeitsgraphen, t die Anzahl der Transitionen des zugehörigen SNL-Systems und m eine obere Schranke für die Zahl der möglichen Variablensubstitutionen beim Schalten einer Transition.

4.3.2 Das Schalten der Transitionen

Wir müssen von einer Markierung M aus alle möglichen direkten Folgemarkierungen berechnen. Ein direkte Folgemarkierung von M entsteht durch Schalten einer Transition in einem speziellen Modus. Um alle möglichen direkten Folgemarkierungen zu generieren, müssen wir deshalb nacheinander jede Transition des SNL-Systems hernehmen, alle möglichen Modi dieser Transition berechnen und danach mit jedem Modus untersuchen, ob die Transition schalten kann.



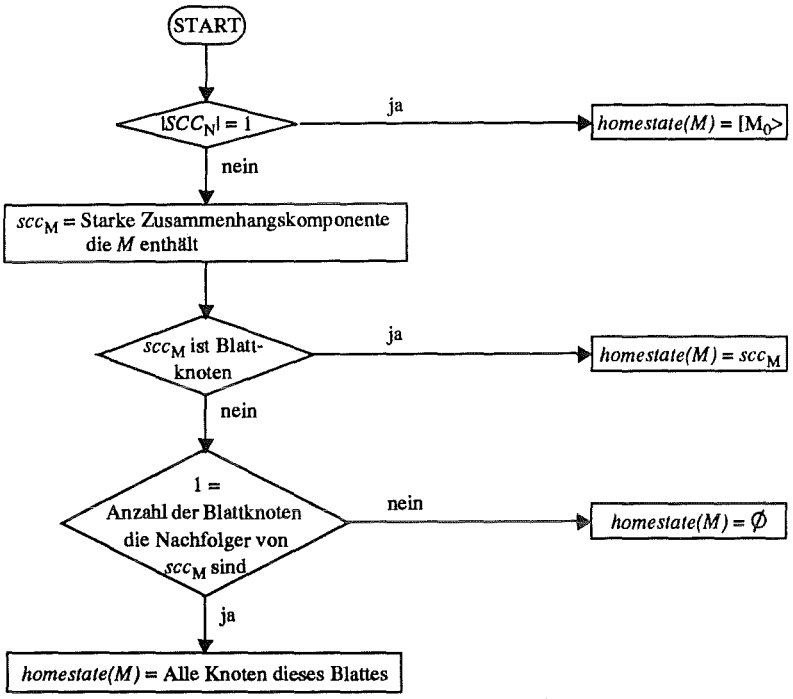
Das Ablaufdiagramm zeigt die Bestimmung aller Folgemarkierungen einer Markierung M, die durch Schalten einer Transition t in verschiedenen Modi entstehen. Als erstes wird die Menge aller verschiedenen Modi, d.h. die Menge aller möglichen Variablensubstitutionen der Transition t bestimmt. Mit *firstModus* wird der erste Modus (einer beliebigen aber festen Reihenfolge der Modi) ermittelt. Falls kein solcher Modus existiert, so konnte keine Folgemarkierung ermittelt werden und die Bearbeitung ist am Ende.

Existiert eine mögliche Variablensubstitution, so wird die Transitionsformel mit dem Modus m ausgewertet. Falls die Auswertung true ergibt, wird überprüft, ob die Transition schalten kann. Das Schalten ist nur dann möglich, wenn genügend Token auf den Inputstellen liegen und wenn nach dem Schalten alle Formeln der Umgebungsstellen von t erfüllt sind. Falls die Transition schalten konnte, so wird eine Folgemarkierung FM generiert, die gemäß Abb. 6 in den Erreichbarkeitsgraph eingetragen werden muß. Danach wird untersucht, ob die Transition auch mit den anderen Modi schalten kann.

Der Zeitaufwand zum Schalten einer Transition in allen ihren Modi ist O(m), wobei m eine obere Schranke für die Zahl der möglichen Variablensubstitutionen beim Schalten einer Transition ist.

4.3.3 Heimzustand- und Heimraumberechnung

Wir behandeln zuerst die Heimzustandberechnung, da die Heimräume auf den Heimzuständen basieren. Folgendes Diagramm veranschaulicht die Berechnung *homestate(M)*:



Die Berechnung des Heimzustandes *homestate(M)* eines Knotens *M* beginnt mit der Feststellung, wieviel starke Zusammenhangskomponenten der Erreichbarkeitsgraph $\mathcal{RG}_N$ hat. Hat der $\mathcal{RG}_N$ nur eine starke Zusammenhangskomponente, so ist *homestate(M)* = $[M_0>$, d.h. alle Knoten der Erreichbarkeitsmenge bilden den Heimzustand der Markierung M.

Bei mehreren starken Zusammenhangskomponenten schauen wir zuerst, in welcher starken Zusammenhangskomponente scc_M unsere Markierung *M* liegt. Ist scc_M ein Blattknoten des Graphen der starken Zusammenhangskomponenten SCG_N, so bildet dieser Blattknoten scc_M gerade den Heimzustand von *M*. Ansonsten müssen wir den Nachfolgebaum von scc_M betrachten. Unter dem Nachfolgebaum verstehen wir dabei denjenigen Teilbaum des SCG_N, der scc_M als Wurzel und alle Nachfolger von scc_M als Knoten hat. Die Kanten zwischen diesen Knoten sind dieselben wie im SCG_N. Existiert in diesem Teilbaum genau eine Verklemmung, dann bilden die Knoten dieser Verklemmung gerade *homestate(M)*. Hat der Teilbaum mehr als eine Verklemmung, so ist *homestate(M)* gleich der leeren Menge.

Der Zeitaufwand für die Berechnung des Heimzustandes ist O(v), wobei v die Anzahl der Knoten des Graphen der starken Zusammenhangskomponenten SCG_N ist.

homespace(M) einer Markierungsteilmenge $\mathcal{M}$ der Erreichbarkeitsmenge $[M_0>$ berechnen wir, indem wir für alle $M \in \mathcal{M}$ *homestate(M)* bestimmen und dann die Schnittmenge $\bigcap_{M \in \mathcal{M}}$ *homespace(M)* bilden.

4.3.4 Gefrorene Marken

Wir erläutern zuerst, wie wir lokalgefrorene Marken einer starken Zusammenhangskomponente *scc* ermitteln.

4.3.4.1 Lokal gefrorene Marken

Zur Bestimmung der lokal gefrorenen Marken einer starken Zusammenhangskomponente *scc* schneiden wir die einzelnen Markierungen miteinander. Der Schnitt von zwei Markierungen $M'=(m'_p)_{p\in P}$,$M''=(m''_p)_{p\in P}\in scc$ ist dabei gerade der Schnitt der Multimengen auf den einzelnen Stellen, d.h. $M'\cap M''=(m'_p\cap m''_p)_{p\in P}$.

Der Schnitt dieser Markierungen ergibt einen P-Vektor $\Omega=(\omega_p)_{p\in P}$. Ist eine dieser Multimengen ω_p ungleich der Multisumme $\bar{0}$, so sind die Elemente der Trägermenge $Tr(\omega_p)$ die gefrorenen Marken der starken Zusammenhangskomponente *scc* der Stelle $p\in P$.

Der Aufwand zur Berechnung der lokal gefrorenen Marken aller starken Zusamenhangskomponenten ist O(n*p). n ist die Anzahl der Knoten des Erreichbarkeitsgraphen und p die Anzahl der Stellen des zugehörigen SNL-Systems.

4.3.4.2 Global gefrorene Marken

Wir unterscheiden bei der Bestimmung der (global) gefrorenen Marken eines SNL-Systems zwei Fälle.

(1) **Nachweis, ob eine bestimmte Marke gefroren ist**

Wir wollen in diesem Fall nachweisen, daß eine bestimmte Marke ω ab der Markierung M in der Stelle p gefroren ist. Dazu müssen wir die Menge [M> aller Folgemarkierungen von M durchlaufen und dabei überprüfen, ob ω in allen Markierungen $M'=(m'_{p'})_{p'\in P}\in$ [M> in der Trägermenge $Tr(m'_p)$ vorkommt.

Wir versuchen dabei möglichst viel vorhandene Information auszunutzen. Solche vorhandene Information ist das Wissen, ob Marken in einer starken Zusammenhangskomponente gefroren sind. Genauso nützlich ist es, zu wissen, ob eine Marke in einer starken Zusammenhangskomponente nicht gefroren ist. Wir wollen wieder anhand eines Diagrammes den Nachweis, ob eine bestimmte Marke ω ab der Markierung M in der Stelle p gefroren ist, verdeutlichen.

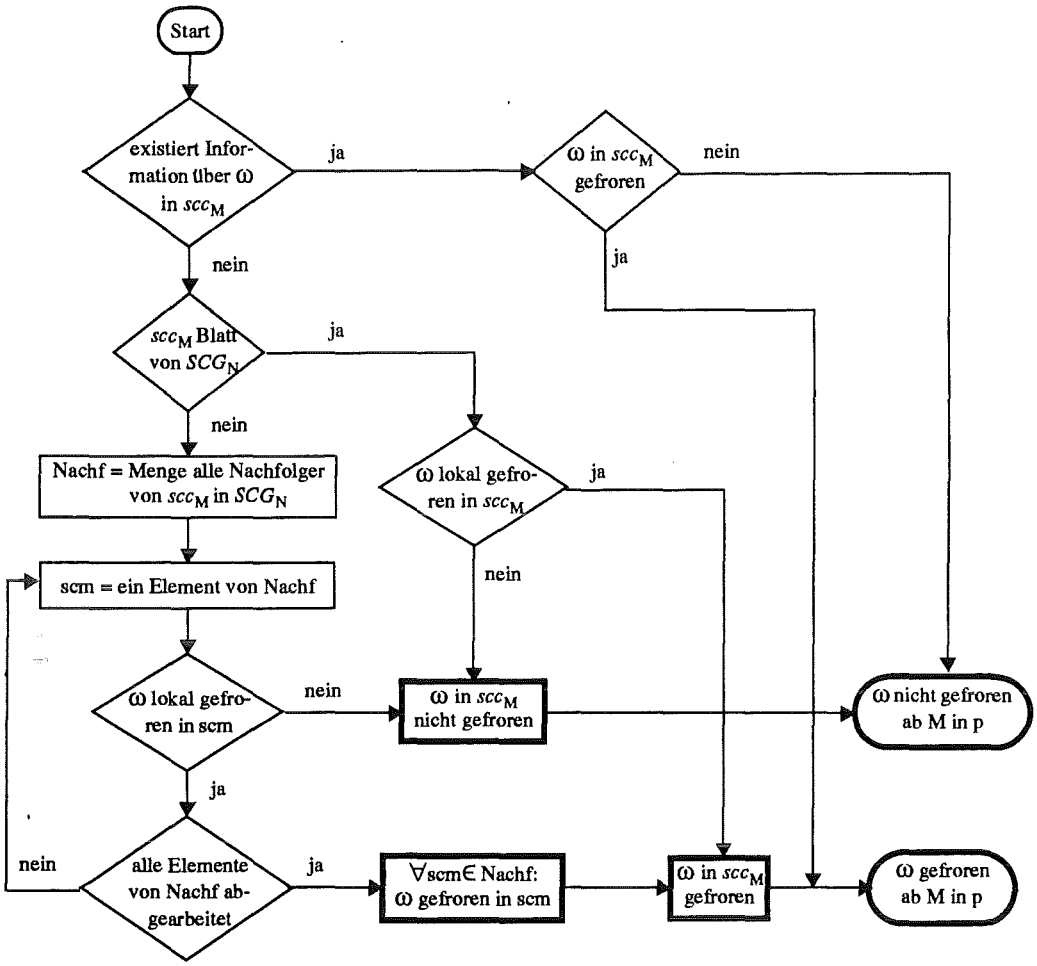
Zuerst überprüfen wir, ob irgendeine Information über Marken ω in der starken Zusammenhangskomponente scc_M, die M enthält, existiert. Diese Information kann entweder besagen, daß ω in scc_M gefroren ist oder nicht. Damit können wir sofort entscheiden, ob ω ab M in p gefroren ist oder nicht. Diese endgültige Entscheidung wird durch die ovalen, dick berandeten Kästen ausgedrückt.

Falls keine Information über ω in scc_M existiert, prüfen wir, ob scc_M ein Blattknoten ist. Ist scc_M ein Blattknoten, entscheiden wir, ob scc_M gefrorene Marken ω hat, halten diese Information für spätere Zwecke fest (dickumrandete Kästen) und verlassen das Diagramm über eine der Aussagen ω gefroren oder nicht gefroren ab M in p.

War scc_M kein Blattknoten, so bestimmen wir die Menge der Nachfolgeknoten von scc_M in SCG_N. Für alle diese starken Zusammenhangskomponenten entscheiden wir (durch Berechnung oder vorhandene Information), ob ω lokal gefroren ist, denn nur wenn ω in allen Nachfolgeknoten von scc_M im SCG_N lokal gefroren ist, ist ω gefroren ab M in p.

Der Zeitaufwand für den Nachweis, ob eine bestimmte Marke gefroren ist, ist O(n). Dabei ist n die Anzahl der Knoten des Erreichbarkeitsgraphen.

Das folgende Diagramm gibt eine schematische Übersicht über diesen Algorithmus:

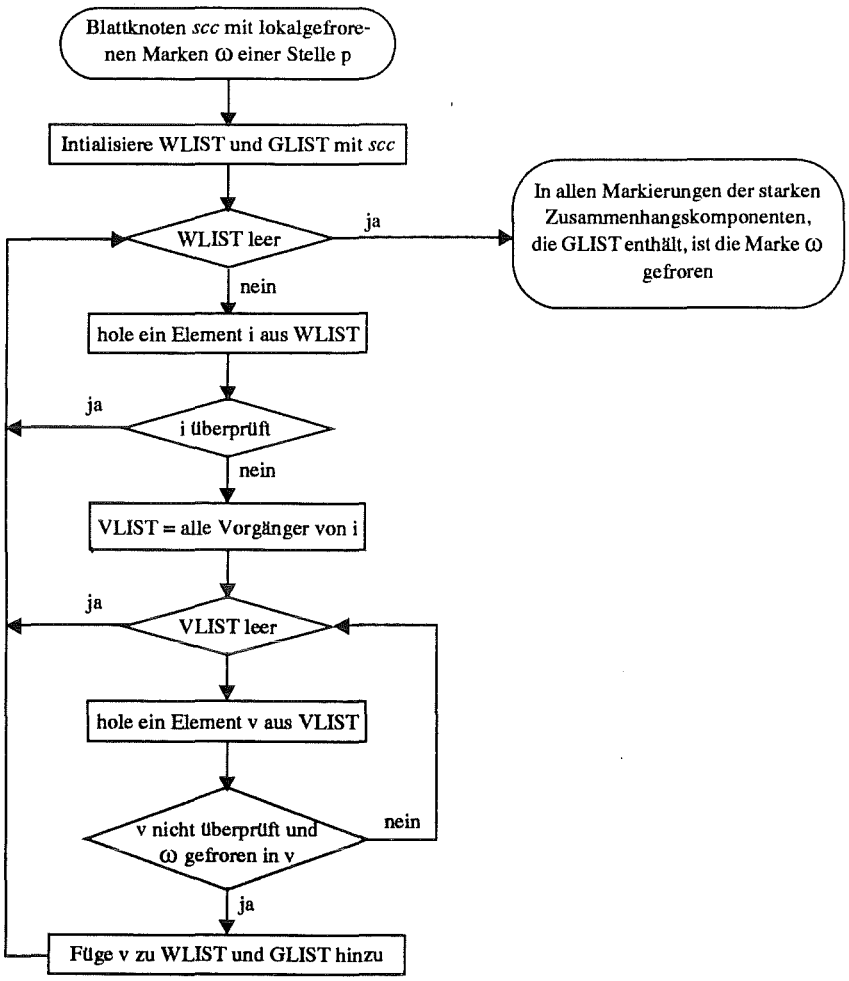


(2) Alle gefrorenen Marken eines SNL-Systems

Bei der Bestimmung aller gefrorener Marken eines SNL-Systems unterscheiden wir zwei Schritte.

(1) Zuerst werden die gefrorenen Marken ω der Blattknoten des SCG_N bestimmt. Diese Marken ω sind dann in allen Markierungen dieser starken Zusammenhangskomponente gefroren.

(2) Für alle diese gefrorenen Marken untersuchen wir, bei welchen Markierungen, die Vorgänger der Markierungen des Blattknotens sind, ebenfalls diese gefrorenen Marken auftreten. Einen Überblick darüber gibt das folgende Diagramm:



Dieser Algorithmus ermittelt, ob eine gefrorene Marke ω einer starken Zusammenhangskomponente *scc* auch schon ab vorhergehenden Markierungen gefroren ist.

Dazu werden zuerst die Listen WLIST und GLIST mit scc initialisiert. GLIST enthält die starken Zusammenhangskomponenten, in deren Markierungen die Marke ω in der Stelle p gefroren ist. In der Liste WLIST werden diejenigen starken Zusammenhangskomponenten gehalten, von denen noch die Vorgängerknoten bestimmt und auf gefrorene Marken überprüft werden müssen.

Der Algorithmus läuft solange, bis die Liste WLIST leer ist. Dazu wird immer ein Element i aus WLIST geholt. War i schon überprüft, so kann das nächste Element aus WLIST geholt werden. War i noch nicht überprüft, so werden alle Vorgängerknoten des SCG_N von i bestimmt. Für jeden Vorgängerknoten v wird überprüft, ob ω in allen Markierungen von v gefroren ist. War ω in allen Markierungen von v gefroren, so wird v in die Listen WLIST und GLIST eingetragen.

Am Ende der Abarbeitung stehen dann in GLIST alle starken Zusammenhangskomponenten, die Markierungen enthalten, in denen die Marke ω in der Stelle p gefroren ist.

Der Zeitaufwand für diesen Algorithmus ist O(n*g). n bezeichnet die Anzahl der Knoten des Erreichbarkeitsgraphen und g die maximal mögliche Anzahl von Marken in dem SNL-System.

4.3.4.3 Flußgefrorene Marken

Auch hierbei können wir wieder danach unterscheiden, ob wir alle flußgefrorenen Marken bestimmen oder ob wir von einer bestimmten Marke ihre Flußgefrorenheit nachweisen wollen. Im allgemeinen Fall müssen wir zuerst alle gefrorenen Marken ermitteln und dann von allen diesen gefrorenen Marken ermitteln, ob sie flußgefroren sind. Wenn wir von einer bestimmten Marke nachweisen wollen, ob sie flußgefroren ist, so müssen wir auch hier zuerst nachprüfen, ob die Marke gefroren ist. Damit müssen wir nur noch beschreiben, wie von einer gefrorenen Marke nachgewiesen wird, ob sie auch flußgefroren ist.

Diese Bestimmung, ob eine gefrorene Marke ω ab M in der Stelle p flußgefroren ist, ist zeitaufwendig, denn wir müssen alle Schaltvorgänge von Transitionen aus der Umgebung von p, die nach M kommen, untersuchen. Fließen in einem dieser Schaltvorgänge Marken auf eine andere Stelle, so ist die Marke ω nicht flußgefroren.

4.3.5 Einfache Algorithmen

Wir haben in vorhergehenden Abschnitten weitere analysierbare Eigenschaften, die das Verhalten des SNL-Systems beschreiben, eingeführt. Einige dieser Eigenschaften sind sehr einfach zu bestimmen, so daß wir diese einfachen Algorithmen gesammelt in diesem Abschnitt angeben.

- **Tote Stellen**

  Die Bestimmung der toten Stellen geschieht während des Aufbaus des Erreichbarkeitsgraphen. Wir versehen dazu die Speicherung jeder Stelle mit einer Flagge, die gesetzt wird, falls die Stelle einmal in der Umgebung einer schaltenden Transition vorkommt. Bleibt die Flagge ungesetzt, so ist die Stelle tot.

  Der Zeitaufwand dafür ist O(p), wobei p die Anzahl der Stellen des zugehörigen SNL-Systems ist.

- **Tote Transitionen**

  Hier gilt dasselbe wie bei toten Stellen, auch hierbei wird eine Flagge gesetzt, sobald eine Transition einmal geschaltet hat. Ist die Flagge nach dem Aufbau des Erreichbarkeitsgraphen gesetzt, so ist die Stelle mindestens schwach lebendig. Ist die Flagge ungesetzt, so ist die Transition tot.

  Der Zeitaufwand dafür ist O(t), wobei t die Anzahl der Transitionen des zugehörigen SNL-Systems ist.

- **Starke Lebendigkeit von Transitionen**

  Zur Bestimmung der stark lebendigen Transitionen müssen wir alle Blätter des Graphen der starken Zusammenhangskomponenten betrachten. Von jedem Blatt wird die Menge der Transitionen, die in dem Blatt schalten können, bestimmt. Von allen diesen Mengen bilden wir die Schnittmenge. Die Transitionen, die in dieser Schnittmenge liegen, sind die stark lebendigen Transitionen.

  Der Zeitaufwand dafür ist O(e), wobei e die Anzahl der Kanten des Erreichbarkeitsgraphen ist.

- **Beschränkte Aktivierbarkeit von Transitionen**

  Zur Bestimmung der beschränkten Aktivierbarkeit einer Transition überprüfen wir zuerst, ob die Transition nicht tot und nicht stark lebendig ist. Dann müssen wir überprüfen, ob sie in keiner starken Zusammenhangskomponente schalten kann.

  Der Zeitaufwand dafür ist O(e), wobei e die Anzahl der Kanten des Erreichbarkeitsgraphen ist.

- **Schwache Reproduzierbarkeit von Markierungen**

  Hat die starke Zusammenhangskomponente, in der die betreffende Markierung liegt, mehr als ein Element, so ist die Markierung schwach reproduzierbar. Der Zeitaufwand dafür ist konstant.

- **Starke Reproduzierbarkeit von Markierungen**

  Liegt die betreffende Markierung in einem Blattknoten des SCG_N, der mehr als eine Markierung enthält, so ist die Markierung stark reproduzierbar. Der Zeitaufwand dafür ist konstant.

- **Gemeinsame Folgemarkierungen**

  Für die Bestimmung der gemeinsamen Folgemarkierung einer Menge von Markierungen muß man die Erreichbarkeitsmengen der einzelnen Markierungen berechnen und von diesen Erreichbarkeitsmengen die Schnittmenge bilden. Der Zeitaufwand dafür ist $O(n^2)$, wobei n die Anzahl der Knoten des Erreichbarkeitsgraphen ist.

- **Substitutions- bzw. Schaltkonflikt**

  Substitutions- bzw. Schaltkonflikte werden über den Erreichbarkeitsgraph bestimmt. An allen Knoten des Erreichbarkeitsgraphen, von denen mehr als eine Kante ausgeht können Substitutions- bzw. Schaltkonflikte entstehen. Dies muß dann für jeden Knoten, von dem mehr als eine Kante abgeht, nachgeprüft werden.

  Der Zeitaufwand dafür ist O(n), wobei n die Anzahl der Knoten des Erreichbarkeitsgraphen ist.

4.3.6 Andere Algorithmen

Wie wir später noch sehen werden, verwenden wir bei der Implementierung der SMARAGD-Analyseeinheit eine Bibliothek von Algorithmen, die uns verschiedene Eigenschaften liefert. Wir wollen diese Algorithmen kurz nennen, ohne deren Struktur anzugeben.

- Algorithmus zur Bestimmung der starken Zusammenhangskomponenten
- Algorithmus zur Bildung des Graphen der starken Zusammenhangskomponenten
- Algorithmus für verschiedene Pfadprobleme im $\mathcal{RG}_N$ und SCG_N.

5. Invariantentheorie

Wir haben schon gesehen, daß unsere Analyse aus den Teilen Erreichbarkeits- und Invariantenanalyse besteht. Die Erreichbarkeitsanalyse haben wir im vorhergehenden Kapitel behandelt. In diesem Kapitel wollen wir zeigen, daß die für Netze übliche Invariantentheorie auf unser Netz-Modell übertragbar, eine automatische Invariantenberechnung jedoch nicht möglich ist. Dabei verstehen wir unter Invarianten nicht nur die klassischen P- und T-Invarianten, sondern wir reden auch von Invarianten, wenn gewisse Eigenschaften in allen Markierungen des Netzes auftreten.

Wir werden zuerst P-Invarianten und T-Invarianten auf unseren SNL-Systemen einführen, wobei wir einige Grundlagen aus der linearen Algebra benötigen. Danach werden wir einige anderen Invariantenformen einführen und dann zeigen, wie die Invarianten bei der Analyse eingesetzt werden können.

5.1 P-Invarianten und T-Invarianten

Wir zeigen in diesem Abschnitt, daß die Invariantentheorie (siehe z.B. [Lautenbach84b] und [Lautenbach85]) sich von elementaren Netzen auf unser Netz-Modell übertragen läßt. Dazu wiederholen wir einige linear algebraische Grundlagen des Netzes, die wir in Kapitel 3 eingeführt haben.

Die Kantenbeschriftungen $A_F(f)$ sind gemäß der Definition einer SNL-Netzspezifikation formale Summen mit den Randbedingungen $fvars(A_F(f)) \subseteq sorts(p)$ für $f=(p,t)$ bzw. (t,p) $t \in T$, $p \in P$. Die Realisierung so einer Kantenbeschriftung liefert eine Abbildung $\mathbf{R}_{FSM}(A_F(f))=A_F(f)_{\mathbf{R}} : VAL_t(NSPEC,A) \longrightarrow M_p(NSPEC,A)$. $A_F(f)_{\mathbf{R}}$ ist eine $\mathbf{Z}$-lineare Abbildung die Multimengen von Auswertungen der Umgebungsvariablen einer Transition auf Multimengen von Individuen abbildet.

Die Inzidenzmatrix $\Theta=(\Theta_{p,t})_{p \in P, t \in T}$ des Systemes ist dann als P×T Matrix von solchen $\mathbf{Z}$-linearen Abbildungen definiert. Dabei ist der $\mathbf{Z}$-Modulhomomorphismus $\Theta_{p,t}: S_t(NSPEC,A) \longrightarrow M_p(NSPEC,A)$ durch

$$\Theta_{p,t}= \begin{cases} 0 \in M_p(NSPEC,A) & \text{falls } (t,p) \text{ und } (p,t) \notin F \\ -\overline{A_F(f)_{\mathbf{R}}} & \text{falls } f=(p,t) \in F \text{ und } (t,p) \notin F \\ \overline{A_F(f)_{\mathbf{R}}} & \text{falls } (p,t) \notin F \text{ und } f=(t,p) \in F \\ -\overline{A_F((p,t))_{\mathbf{R}}} + \overline{A_F((t,p))_{\mathbf{R}}} & \text{falls } (p,t) \in F \text{ und } (t,p) \in F \end{cases}$$

definiert.

T-Vektoren haben wir in Definition 24 und Definition 25 eingeführt. Wir haben gesehen, daß ein T-Vektor μ als Komponenten Multimengen von Auswertungen der Umgebungsvariablen der jeweiligen Transition enthält. Die Anwendung einer Matrix auf einen T-Vektor (das Produkt einer Matrix mit einem T-Vektor) geschieht wie bei der üblichen Matrix-Vektor-Multiplikation, wobei das Produkt eines Matrixelementes mit einem Vektorelement allerdings die Anwendung der $\mathbf{Z}$-linearen Abbildung auf die jeweilige Multimenge von Auswertungen der Umgebungsvariablen ist. Aufgrund der $\mathbf{Z}$-Linearität der Matrixelemente übertragen sich dabei die üblichen Rechenregeln aus der linearen Algebra. (Siehe dazu auch Definition 20 und Definition 21).

Wir haben in Definition 32 gesehen, daß sich jede Markierung M aus der Anfangsmarkierung M_0 durch eine Gleichung der Gestalt $M=M_0+\Theta\mu$ ergibt, wobei μ ein T-Vektor von Multimengen von Auswertungen der jeweiligen Umgebungsvariablen ist. Damit definieren wir T-Invarianten wie folgt.

Definition 65 **(T-Invarianten)**

Sei $N=(NSPEC,A)$ ein SNL-System mit Inzidenzmatrix Θ und Anfangsmarkierung M_0.

(a) Ein T-Vektor μ heißt *T-Invariante*, falls $\Theta\mu=0$ gilt.

(b) Eine T-Invariante μ wird als *(mit Schrittvorgängen) realisierbare T-Invariante* bezeichnet, falls eine Markierung $M\in [_sM_0>$ und eine Zerlegung $\mu = \sum_{i=1}^{n} \mu_i$ existiert, mit $M[\mu_1>M_1 \ldots M_{n-1}[\mu_n>M$ (wobei $M_i\in [_sM_0>$ und μ_i ein Schrittvorgang).

(c) Eine T-Invariante μ wird als *(mit Schaltvorgängen) realisierbare T-Invariante* bezeichnet, falls eine Markierung $M\in [M_0>$ und eine Zerlegung $\mu = \sum_{i=1}^{n} t'_i$ existiert, mit $M[t'_1>M_1 \ldots M_{n-1}[t'_n>M$ (wobei $M_i\in [M_0>$ und t'_i ein Schaltvorgang).

Mit anderen Worten kann man sagen, daß zu einer mit Schrittvorgängen realisierbaren T-Invarianten mindestens eine Zerlegung existiert, die einen im Schrittgraph vorhandenen Zyklus beschreibt. Zu einer mit Schaltvorgängen realisierbaren T-Invarianten existiert mindestens eine Zerlegung, die einen im Erreichbarkeitsgraph vorhandenen Zyklus beschreibt. Es ist offensichtlich, daß jede mit Schaltvorgängen realisierbare T-Invariante auch mit Schrittvorgängen realisierbar ist, da der Erreichbarkeitsgraph eine Untermenge des Schrittgraphen und jeder Schaltvorgang auch ein Schrittvorgang ist.

Wir wollen nun P-Invarianten einführen. Dazu müssen wir zuerst definieren, welche Konstrukte überhaupt für eine P-Invariante in Frage kommen. Üblicherweise spricht man in der Invariantentheorie von P-Vektoren, die unter gewissen Bedingungen eine P-Invariante darstellen. P-Vektoren, wie wir sie definiert haben (siehe Definition 22), können keine P-Invarianten unseres Netzmodelles sein. P-Invarianten unseres Netzmodelles müssen P-indizierte Familien von Modulhomomorphismen sein. Wir nennen solche Konstrukte P-Zeilenvektoren. Wir führen solche P-Zeilenvektoren in der folgenden Definition ein.

Definition 66 **(P-Zeilenvektor)**

Sei $N=(NSPEC,A)$ ein SNL-System und B ein beliebiger $\mathbf{Z}$-Modul.

Wir nennen eine P-indizierte Familie $\lambda=(\lambda_p)_{p\in P}$ mit Modulhomomorphismen $\lambda_p:M_p(NSPEC,A)\rightarrow B$ $(\forall p\in P)$ einen P-Zeilenvektor (bzgl. B).

Bei einem P-Zeilenvektor handelt es sich um eine Matrix von Modulhomomorphismen, wie sie in Definition 19 eingeführt wurde, mit allerdings nur einer Zeile. Nach Definition 20 können wir solche P-Zeilenvektoren auf P-Vektoren anwenden und nach Definition 21 können wir das Produkt dieser P-Zeilenvektoren mit der Inzidenzmatrix bilden. Wir benötigen diese wichtigen Aussagen bei der Definition von P-Invarianten.

Definition 67 **(P-Invarianten)**

Sei $N=(NSPEC,A)$ ein SNL-System, M_0 die zugehörige Anfangsmarkierung und B ein beliebiger $\mathbf{Z}$-Modul.

Ein P-Zeilenvektor λ (bzgl. B) heißt P-Invariante (bzgl. B), falls gilt: $\forall M\in [M_0>$ ist $\lambda M=\lambda M_0$.

Wir haben weiter oben gesehen, daß wir jede Markierung M aus der Anfangsmarkierung M_0 durch die Gleichung $M=M_0+\Theta\mu$ erhalten können, wobei μ ein T-Vektor von Multimengen von Auswertungen der jeweiligen Umgebungsvariablen ist. Setzen wir diesen Sachverhalt in die obige Bedingung ein, so erhalten wir:

$$\begin{aligned}
& \lambda M = \lambda M_0 \\
\Leftrightarrow\quad & \lambda(M_0+\Theta\mu) = \lambda M_0 && \text{(siehe oben)} \\
\Leftrightarrow\quad & \lambda M_0+\lambda(\Theta\mu) = \lambda M_0 && \text{(siehe Bemerkung 5)} \\
\Leftrightarrow\quad & \lambda M_0+(\lambda\Theta)\mu = \lambda M_0 && \text{(siehe Bemerkung 6)} \\
\Leftrightarrow\quad & (\lambda\Theta)\mu = 0 \\
\Leftarrow\quad & \lambda\Theta = 0
\end{aligned}$$

Wir haben damit gezeigt, daß jeder P-Zeilenvektor λ mit $\lambda\Theta=0$ eine P-Invariante ist. Wir halten dies in der folgenden Bemerkung fest.

Bemerkung 20

Sei $N=(NSPEC,A)$ ein SNL-System, M_0 die zugehörige Anfangsmarkierung, Θ die Inzidenzmatrix von N und B ein beliebiger $\mathbf{Z}$-Modul.

Erfüllt ein P-Zeilenvektor λ (bzgl. B) die Bedingung $\lambda\Theta=0$, so ist λ eine P-Invariante (bzgl. B).

Wir wollen ein Beispiel für eine P-Invariante angeben. Dazu benutzen wir das SNL-System ERZEUGER-VERBRAUCHER aus Beispiel 3, wobei wir für den späteren Gebrauch das Beispiel leicht modifiziert haben.

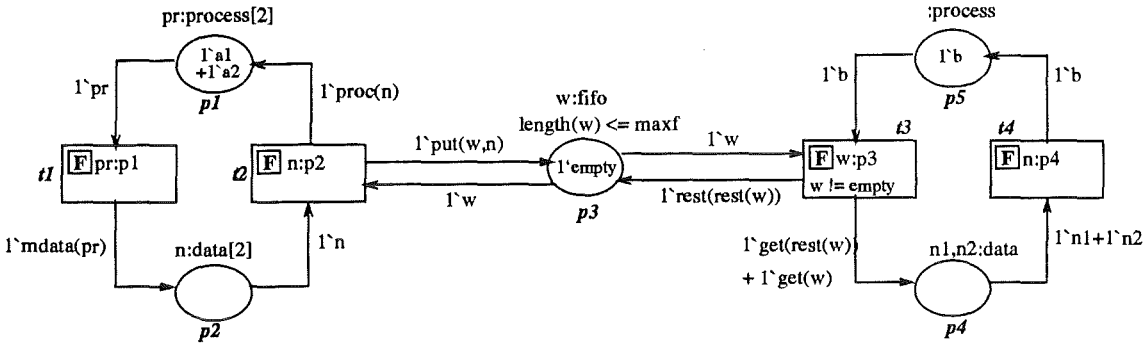


Abb. 7 Das SNL-System ERZEUGER-VERBRAUCHER

Die Inzidenzmatrix Θ dieses Systems ist die folgende:

$\Theta =$

	t_1	t_2	t_3	t_4
p_1	- 1`pr	1`proc(n)		
p_2	1`data(pr)	- 1`n		
p_3		- 1`w + 1`put(w,n)	- 1`w + 1`rest(rest(w))	
p_4			- 1`b	1`b
p_5			1`get(rest(w)) + 1`get(w)	- 1`n1 - 1`n2

Bevor wir eine Invariante angeben können, müssen wir noch einen Modul B angeben, bezüglich dessen wir die Invariante bilden. Bei unseren Definitionen sind wir immer von einem ganz beliebigen Modul ausgegangen. Wir werden in den folgenden zwei Beispielen den Modul B beliebig wählen und darstellen, wie die Wahl von B sich auf die Invariante auswirkt. Wir werden für B zuerst den Modul $M_p(NSPEC,A)$ und dann den Modul $\mathbf{Z}$ der ganzen Zahlen wählen.

Beispiel 8 **(B = $M_p(NSPEC,A)$)**

Sei $B=M_p(NSPEC,A)$ der Modul der Sorten des SNL-Systems ERZEUGER-VERBRAUCHER aus Abb. 7.

Die Operation proc über der Sorte data können wir uns in einfacher Art und Weise zu einer Operation $\overline{proc}$ auf Multimengen über der Sorte data als eindeutig bestimmte lineare Fortsetzung von proc denken. Es gelten dann z.B. die folgenden Regeln:

$$\overline{proc}(1`n) = 1`proc(n),$$
$$\overline{proc}(-1`n) = -\overline{proc}(1`n).$$

Eine P-Invariante (bzgl. $M_p(NSPEC,A)$) des Systems ist dann der P-Zeilenvektor $\lambda= (\bar{\xi}, \overline{proc}, 0, 0, 0)$, mit $\bar{\xi}$ als der identischen Abbildung in $M_p(NSPEC,A)$, denn für λ ist das Produkt $\lambda\Theta$ gleich dem Nullvektor, wie die folgende Berechnung zeigt:

$\lambda\Theta$ = $(\overline{\xi}(-1`pr) + \overline{proc}(1`data(pr)), \overline{\xi}(1`proc(n)) + \overline{proc}(-1`n), 0, 0, 0)$
= $(-1`pr + 1`proc(data(pr)), 1`proc(n) - 1`proc(n), 0, 0, 0)$
= $(-1`pr + 1`pr, 1`proc(n) - 1`proc(n), 0, 0, 0)$
= $(0, 0, 0, 0, 0)$

Hierbei haben wir ausgenützt, daß gemäß der Spezifikation des Erzeuger-Verbraucher Modells die Gleichung proc(data(pr))=pr gilt.

Die Bedeutung der Invariante ergibt sich aus der Gleichung $\lambda M=\lambda M_0$. Daraus folgt für unser Netz:

$$\overline{\xi}(M(p_1)) + \overline{proc}(M(p_2)) = 1`a1 + 1`a2$$

Das heißt, daß im Erzeugerzyklus keine Prozesse verloren gehen, sondern aus Daten, die auf p_2 liegen, und aus den Prozessen, die noch auf p_1 liegen, stets alle Prozesse im Erzeuger-Zyklus zurückgewonnen werden können.

In dem nächsten Beispiel wählen wir den Modul **Z** der ganzen Zahlen als unseren Modul B.

Beispiel 9 (B = Z)

Wir nehmen wieder das SNL-System ERZEUGER-VERBRAUCHER aus Abb. 7 und wählen B= **Z**.

Sei $\delta : M_p(NSPEC,A) \rightarrow \mathbf{Z}$ eine Abbildung, die die Anzahl der Elemente einer Multimenge bestimmt, wobei zum Beispiel gilt:

$$\delta(1`n) = 1,$$
$$\delta(-1`n) = -\delta(1`n).$$

Damit können wir den P-Zeilenvektor λ= (δ, δ, 0, 0, 0) bzgl. **Z** aufstellen. Wegen

$\lambda\Theta$ = $(\delta(-1`pr) + \delta(1`data(pr)), \delta(1`proc(n)) + \delta(-1`n), 0, 0, 0)$
= $(-\delta(1`pr) + 1, 1 - \delta(1`n), 0, 0, 0)$
= $(-1 + 1, 1 - 1, 0, 0, 0)$
= $(0, 0, 0, 0, 0)$

ist λ eine P-Invariante bzgl. **Z**.

Die Bedeutung der Invariante ergibt sich wieder aus der Gleichung $\lambda M=\lambda M_0$. Daraus folgt für unser Netz:

$$\delta(M(p_1)) + \delta(M(p_2)) = 1 + 1 = 2$$

Das heißt, daß im Erzeugerzyklus die Anzahl der Marken immer konstant gleich 2 bleibt.

Diese beiden Beispiel zeigen uns verschiedene Dinge. Zum einen, daß es durchaus Sinn macht, für B verschiedene Module zu wählen. Bei beiden Moduln erhalten wir sinnvolle Invarianten, die zwar verschieden sind, aber ähnliches Aussagen. In Beispiel 9 erfahren wir, daß die Markenanzahl im Erzeugerzyklus konstant bleibt, während die invariante Aussage in Beispiel 8 noch detaillierter angibt, daß die Prozesse im Erzeugerzyklus nicht verloren gehen.

In Bezug auf eine automatische Berechnung von Invarianten, ist die invariante Aussage aus Beispiel 9 interessant. Dazu beschriften wir das zum SNL-System ERZEUGER-VERBRAUCHER gehörige Netz N derart, daß wir die Marken nicht mehr nach ihrer Individualität unterscheiden, sondern nur die Anzahl der Marken zählen. Wir erhalten dann ein Platz/Transition-Netz, das die in Abb. 8 dargestellte Form hat. Wir nennen ein solches Netz das *Skelett* oder auch das *kanonische Netz* des SNL-Systems ERZEUGER-VERBRAUCHER. (Vergleiche hierzu auch [Vautherin87].)

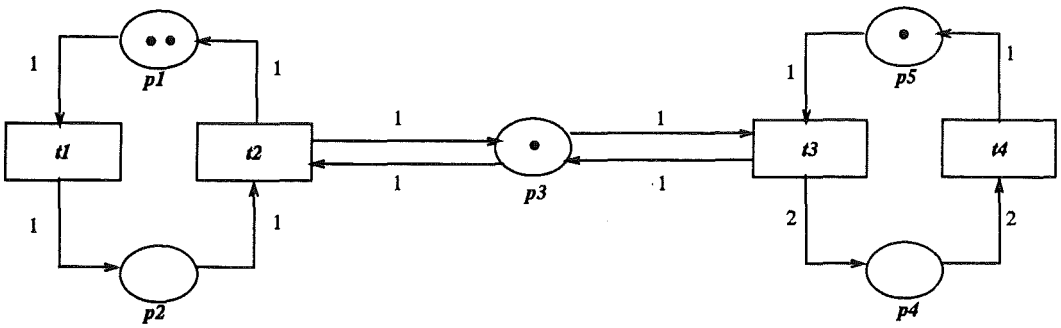


Abb. 8 Das kanonische Netz des SNL-Systems ERZEUGER-VERBRAUCHER

Die Inzidenzmatrix Θ dieses Netzes hat die folgende einfache Gestalt:

$\Theta =$

	t_1	t_2	t_3	t_4
p_1	- 1	1		
p_2	1	- 1		
p_3		0	0	
p_4			- 1	1
p_5			2	- 2

Die Matrixelemente, die nicht ausgefüllt sind, haben als Belegung eine 0. Die beiden eingetragenen Nullen sollen bedeuten, daß sie durch eine Eingangs- und eine Ausgangskante von der Transition zur entsprechenden Stelle entstanden sind. Bei P/T-Netzen können wir Invarianten durch Lösung des homogenen linearen Gleichungssystems $\lambda\Theta=0$ berechnen. Eine solche Lösung ist der Zeilenvektor (1, 1, 0, 0, 0), der als invariante Aussage liefert, daß die Anzahl der Marken im Erzeugerzyklus immer gleich 2 ist. Wir haben damit dieselbe invariante Aussage wie in Beispiel 9. Solche Parallelen bei den Invarianten eines SNL-Systems und des dazugehörigen kanonischen Platz/Transition-Netz erleichtern uns das Finden von P-Invarianten in SNL-Systemen. Wir können die P-Invarianten in dem kanonischen Netz berechnen und können dann nachprüfen, ob vergleichbare invariante Aussagen auch in dem SNL-System existieren.

5.2 Andere Invarianten

Auf eine ganz andere Art von Invarianten wollen wir in diesem Abschnitt eingeben. Diese Invarianten behandeln das minimale und maximale Vorkommen einer Marke sowie das Markenverhalten beim Schalten einer Transition.

5.2.1 Erhaltung von Marken

Es läßt sich eine Übersicht erstellen, welche Marken wie oft im Netz auftauchen. Damit ist gemeint, daß man das Vorkommen der Marken in den verschiedenen Markierungssituationen zählt und dann den minimalen und maximalen Wert des Vorkommens angibt. Wir wollen dazu die formale Definition angeben.

Definition 68 (minimales und maximales Vorkommen einer Marke)

Sei $N= (NSPEC,A)$ ein SNL-System.

Für $M=(m_p)_{p\in P} \in [M_0>$ bezeichnen wir mit $anz_{M,p}(\omega)$ die Anzahl der Marken $\omega\in A$ in der Markierung m_p einer Stelle $p\in P$.

Mit $vork_M(\omega) = \sum_{p\in P} anz_{M,p}(\omega)$ bezeichnen wir die Anzahl der Marken ω in einer Markierung M des SNL-Systems N.

Mit *minvork(ω)* bezeichnen wir die minimale Anzahl von Vorkommen von ω in einem SNL-System N. Wir definieren *minvork(ω)* wie folgt: $minvork(\omega) = min_{M\in [M_0>} vork_M(\omega)$.

Mit *maxvork(ω)* bezeichnen wir die maximale Anzahl von Vorkommen von ω in einem SNL-System N. Wir definieren *maxvork(ω)* wie folgt: $maxvork(\omega) = max_{M\in [M_0>} vork_M(\omega)$.

Wenn wir in dieser Definition (und auch in späteren Definitionen) von $\omega\in A$ sprechen, so meinen wir damit $\omega\in A= \bigcup_{s\in S} A_S(s)$, d.h. ω ist Element der Vereinigung der Trägermengen.

Den Nutzen von der Angabe des minimalen und maximalen Vorkommens kann man sich leicht vorstellen. Ein Modellierer macht sich gewisse Gedanken über den Markenfluß in einem Netz und kann sich durch *minvork* und *maxvork* sehr leicht anschauen, ob Marken einer gewissen Sorte verschwinden, konstant bleiben oder sich vielleicht sogar vermehren.

Oft fordert ein Benutzer von einem Netz, daß z.B. *maxvork(ω)< konst* ist. Dies ist aber gerade eine invariante Aussage über das Netz.

5.2.2 Markenbilanz einer Transition

Verschwinden bei einem Netz Marken oder verhält sich der Markenfluß nicht so wie erwartet, so möchte man die Ursache dafür finden. Dazu ist es sinnvoll, sehr schnell von einer Transition zu wissen, ob sie Marken produziert, vernichtet oder die Markenanzahl beim Schalten gleich bleibt.

Als praktischen Nutzen kann man sich folgende Situation vorstellen. Von gewissen Marken möchte man immer eine gewisse Anzahl im Netz haben. Wenn dies nicht eingehalten wird, so braucht man nur die Transitionen zu betrachten, die Marken vernichten. Das System kann vielleicht sogar selbstständig herausfinden, welche Transitionen fehlerhaft sind, in dem es zum Beispiel verfolgt, ob nach einer markenvernichtenden Transition diese Marken noch einmal benötigt werden, aber nun nicht mehr vorhanden sind. Auch darauf wollen wir zu einem späteren Zeitpunkt noch zurückkommen.

Definition 69 **(Markenverhalten beim Schalten einer Transition)**

Sei $N=(NSPEC,A)$ ein SNL-System und $\omega \in A$ eine Marke. Wir sagen:

Eine Transition t *vernichtet* Marken ω, falls gilt: Für alle $M,M' \in [M_0\rangle$ mit $M[t\rangle M'$ ist

$$\sum_{p \in {}^\bullet t^\bullet} anz_{M,p}(\omega) > \sum_{p \in {}^\bullet t^\bullet} anz_{M',p}(\omega) \quad .$$

Eine Transition t *erhält* Marken ω, falls gilt: Für alle $M,M' \in [M_0\rangle$ mit $M[t\rangle M'$ ist

$$\sum_{p \in {}^\bullet t^\bullet} anz_{M,p}(\omega) = \sum_{p \in {}^\bullet t^\bullet} anz_{M',p}(\omega) \quad .$$

Eine Transition t *produziert* Marken ω, falls gilt: Für alle $M,M' \in [M_0\rangle$ mit $M[t\rangle M'$ ist

$$\sum_{p \in {}^\bullet t^\bullet} anz_{M,p}(\omega) < \sum_{p \in {}^\bullet t^\bullet} anz_{M',p}(\omega) \quad .$$

In dem nun folgenden Abschnitt wollen wir ganz kurz anreißen, wie die vorgestellten Invarianten bestimmt werden können oder warum eine Bestimmung Schwierigkeiten macht.

5.3 Bestimmung von Invarianten

Wir haben in den vorhergehenden Abschnitten verschiedene Invarianten kennengelernt. Genauso verschieden wie diese Invarianten ist auch die Art und Weise, wie sie bestimmt werden.

Mit Schaltvorgängen realisierbare T-Invarianten können wir aus dem Erreichbarkeitsgraph bestimmen, denn eine mit Schaltvorgängen realisierbare T-Invariante läßt sich in mindestens eine Schaltfolge, die einen Zyklus im Erreichbarkeitsgraph darstellt, zerlegen. Solch ein Zyklus liegt aber in genau einer starken Zusammenhangskomponente. Damit lassen sich T-Invarianten aus den starken Zusammenhangskomponenten des Erreichbarkeitsgraphen bestimmen.

Zur Berechnung von P-Invarianten müssen wir das Gleichungssystem $\lambda\Theta = 0$ lösen. Wegen der Kompliziertheit der Matrix- und Vektorelemente ist eine automatische Berechnung nicht möglich, denn dazu müßten wir die Funktionen, die als Matrix- bzw. Vektorelemente auftreten, invertieren können. Da es nicht einmal sein muß, daß für jede Funktion eine inverse Funktion existiert, können wir schon gar nicht rechnergestützt diese Funktionen invertieren. Möglich ist aber eine computergestützte Verifikation von Invarianten. Damit ist gemeint, daß der Benutzer einen P-Zeilenvektor eingibt, von dem dann das System überprüft, ob dieser P-Zeilenvektor eine P-Invariante ist. Dieses Überprüfen wird durch multiplizieren des P-Zeilenvektors mit der Inzidenzmatrix gelöst. Diese Multiplikation ist möglich, da die Matrix- und Vektorelemente Konstrukte der SNL-Sprache sind, so daß wir sie miteinander verknüpfen können. Ein Problem ist noch das Finden eines P-Zeilenvektors, der eine P-Invariante sein könnte.

Eine Möglichkeit, eine P-Invariante zu finden, haben wir schon weiter oben angesprochen. Dazu berechnen wir die P-Invarianten des zu einem SNL-System gehörenden kanonischen Netzes. Ausgehend von diesen Invarianten stellen wir ähnliche P-Zeilenvektoren des SNL-Systems auf, von denen wir dann untersuchen, ob sie P-Invarianten sind.

Oft ist es auch so, daß der Benutzer schon beim Systementwurf gewisse Vorstellungen oder sogar zwingende Bedingungen hat, welche Aussagen im System invariant sein sollen. Aus diesen Vorstellungen kann der Benutzer dann einen P-Zeilenvektor formulieren und mit Hilfe des Rechners verifizieren.

Dieser Nachweis von invarianten Aussagen braucht nicht nur auf P-Invarianten beschränkt sein, sondern der Benutzer sollte auch weitere Eigenschaften mit invariantem Charakter nachweisen können. Eine solche Eigenschaft ist zum Beispiel, daß in einem Netz auf bestimmten Stellen immer eine Mindestanzahl von Marken einer Sorte vorhanden sein muß. In dem SNL-System ERZEUGER-VERBRAUCHER aus Abb. 7 ist zum Beispiel eine invariante Eigenschaft des Netzes, daß auf der Stelle p_3 immer eine Warteschlange liegt. Es wäre wünschenswert, daß ein System auch solche invarianten Eigenschaften verifizieren kann. Dazu muß der Benutzer invariante Eigenschaften in einer geeigneten Form eingeben können. Eine Möglichkeit ist, daß der Benutzer dem System Regeln mitteilt, die das System auf Einhaltung überprüft. Beispiele für solche möglichen Regeln sind:

(1) Die Anzahl einer Marke ω soll überall im Netz gleich sein, d.h. $minvork(\omega)=maxvork(\omega)$.

(2) Von einer Marke ω soll immer eine gewisse Mindestanzahl K vorhanden sein, d.h. $minvork(\omega)\geq K$.

(3) In einer Menge $P' \subseteq P$ von Stellen soll immer eine Marke ω sein, d.h. $min_{M \in [M_0>} \sum_{p \in P'} anz_{M,p}(t) > 0$

In dem existierenden 1. Prototyp von SMARAGD haben wir noch keine Möglichkeit, T-, P- oder andere Invarianten zu verifizieren. Dies wird eine Aufgabe für eine weitere Ausbaustufe des Werkzeuges sein.

6. Regelbasierte Auswertung von Analyseergebnissen

In den vorhergehenden Abschnitten wurden die verschiedenen Möglichkeiten der Analyse beschrieben. Aus diesen Analyseergebnissen kann der Benutzer die Ursache für ein Fehlverhalten des Netzes ermitteln. In vielen Fällen ist dies eine sehr komplexe Aufgabe. Für den Benutzer wäre es deshalb von Vorteil, wenn er eine rechnergestützte Interpretationshilfe für diese Analyseergebnisse hätte.

Als Möglichkeit kann man sich hierbei eine regelbasierte Entscheidungshilfe vorstellen. Die Notwendigkeit einer solchen regelbasierten Auswertung von Analyseergebnissen wird auch in [Starke90] festgestellt. Die Arbeitsweise einer solchen regelbasierten Entscheidungshilfe werden wir im folgenden vorstellen. Wir haben dabei eine Unterscheidung zwischen verschiedenen Fehlverhalten gemacht, deren mögliche Ursachenforschung wir beschreiben.

6.1 Gefrorene Marken

Wir haben bei der Definition von gefrorenen Marken schon angedeutet, daß gefrorene Marken nicht immer ein ungewünschter Zustand sein müssen. Z.B. kann ein Benutzer Schleifen (Nebenbedingungen) wie im folgenden Bild konstruieren, bei denen gefrorene Marken existieren, die aber gewünscht sind.

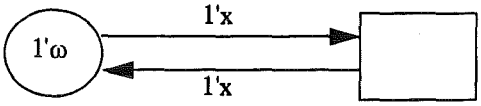


Abb. 9 Transition mit Nebenbedingung

Gefrorene Marken können jedoch auch Fehlerzustände sein, deren Behebung die Eigenschaften eines Netzes wesentlich verändern. Besonders flußgefrorene Marken werden häufig eine nicht gewünschte Situation darstellen. Die Ursachen von gefrorenen Marken können sehr vielfältig sein. Untersucht man gefrorene Marken näher, so liegt die Vermutung nahe, daß ihr Auftreten eng mit dem Vorkommen von toten Transitionen zusammenhängt. Es ist denkbar, daß Marken irgendwo gefroren werden, diese Marken aber an anderer Stelle zum Schalten einer Transition benötigt werden. Der nächste Abschnitt zeigt dazu näheres, wobei die Frage interessant wäre, ob man gewisse Abhängigkeiten zwischen toten Transitionen und gefrorenen Marken finden könnte. Dieselben Überlegungen lassen sich auch für Verklemmungen anstellen, was wir ebenfalls weiter unten näher erläutern wollen. Wir wollen in einem Diagramm (Abb. 10) einige mögliche Ursachen für gefrorene Marken zeigen:

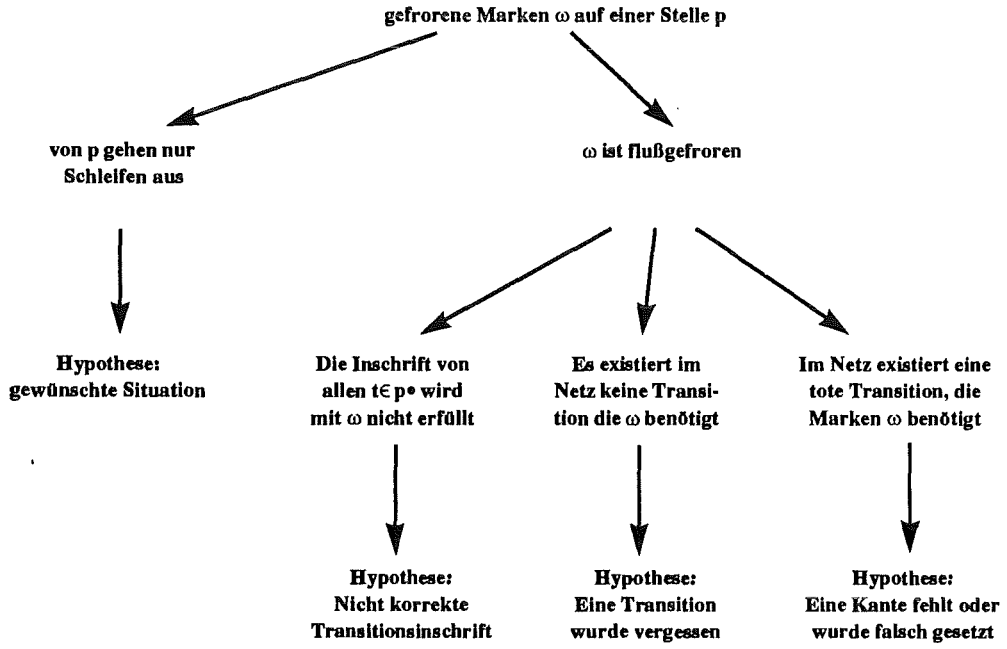


Abb. 10 Ursachen für gefrorene Marken

Wir wollen nun das Diagramm (Abb. 10) etwas erläutern. Eine gewünschte Situation kann z.B. vorliegen, wenn eine Schleife (siehe Abb. 9) auftritt. Zu jeder ausgehenden Kante einer Stelle existiert eine hineingehende, die dieselbe Marke wieder in die Stelle legt. Das Diagramm zeigt weiterhin drei Möglichkeiten, wie man zu nicht gewünschten gefrorenen Marken kommen kann. Eine dieser Möglichkeiten zeigt Abb. 11:

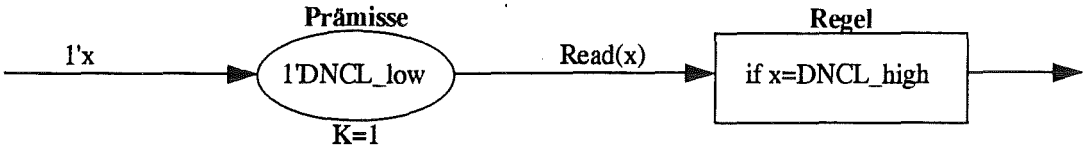


Abb. 11 Beispiel für eine gefrorene Marke

Dieser Ausschnitt eines Netzes soll das Modell einer Regel darstellen. Ist die Prämisse nicht erfüllt, kann die Regel nicht feuern. In unserem Fall haben wir als Prämisse den Wert DNCL_low, während die Regel nur mit der Prämisse DNCL_high feuern kann. Der Wert DNCL_low kann nicht mehr von der Stelle abgezogen werden, da keine Transition existiert, die diese Marke weitergibt. Das Bild stellt damit ein Beispiel für die Hypothese "Eine Transition wurde vergessen" dar. In Abb. 12 haben wir eine solche Transition (Alternative) hinzugefügt. Allerdings fehlt in diesem Bild eine Kante von Prämisse zur Alternative, so daß diese Transition Alternative tot ist. Wir haben damit ein Beispiel für die Hypothese "Eine Kante fehlt oder wurde falsch gesetzt".

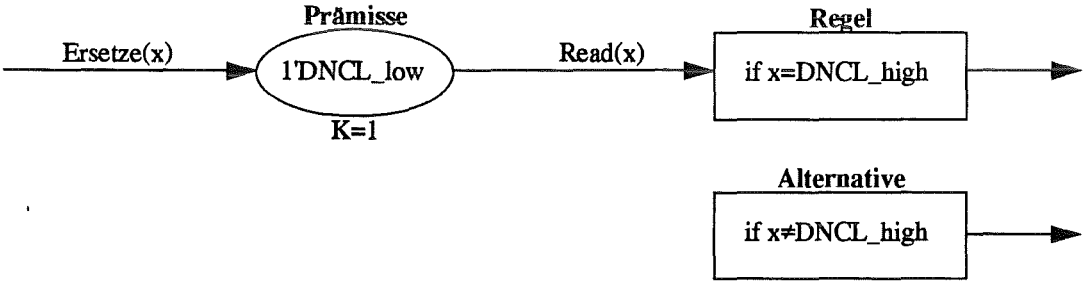


Abb. 12 Netzausschnitt mit fehlender Kante

Nach dem Hinzufügen dieser fehlenden Kante bekommen wir dann den korrekten Teilausschnitt des Netzes, der in Abb. 13 dargestellt ist:

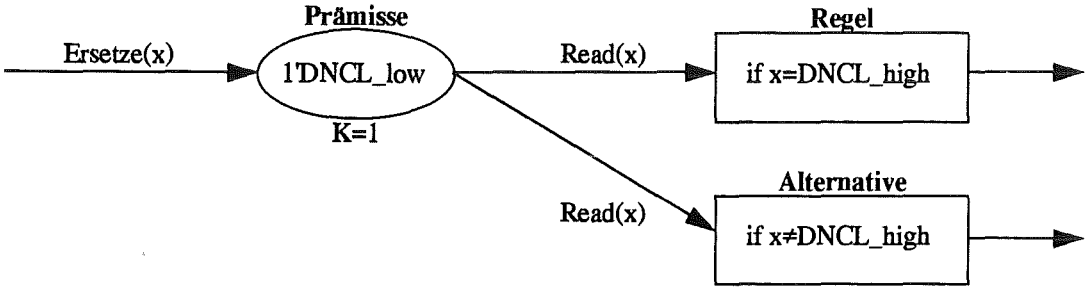


Abb. 13 Netzausschnitt mit hinzugefügter Kante

6.2 Tote Transitionen

Treten bei der Analyse eines Netzes tote Transitionen auf, so fragt man sich, warum kann diese Transition nie schalten. Wir wollen uns nun gewisse Vorgehensweisen überlegen, wie man einer solchen Frage nachgehen kann. Deshalb sei im folgenden N=(NSPEC,A) ein SNL-System. Wenn eine Transition $t \in T$ von N tot ist, so kann einer der folgenden Fälle vorliegen:

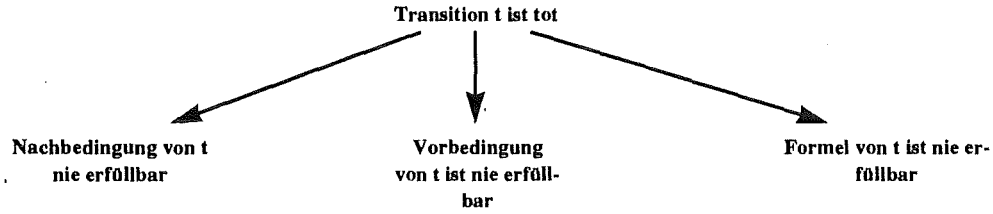


Abb. 14 Ursachen für tote Transitionen

Wir wollen nun die weitere Vorgehensweise bei diesen drei möglichen Fällen erläutern. Zu beachten ist dabei, daß diese drei Fälle nicht die einzig möglichen Fälle sind, denn bei diesen Fällen wurden angenommen, daß in allen Markierungen immer die Vorbedingung oder die Nachbedingung oder die Transitionsinschrift nicht erfüllt ist. Es wäre natürlich auch denkbar, daß in der einen Markierung die Vorbedingung, in der anderen die Nachbedingung und in einer dritten die Transitionsformel nicht erfüllt ist.

(1) Vorbedingung von t ist nie erfüllbar

Unter "Vorbedingung von t ist nie erfüllbar" wollen wir dabei die Situation verstehen, daß bei jeder von der Anfangsmarkierung M_0 aus erreichbaren Markierung M nicht genügend Token auf den Vorstellen von t liegen. Formal läßt sich dies in der folgenden Weise darstellen: Für alle $M \in [M_0>$ und alle Auswertungen ν existiert ein $p_M \in$ vor(t), so daß gilt: $(A_F(p_M, t))_{\mathbf{R}}(\nu) > m_{p_M}$.

Hierbei lassen sich wieder zwei Fälle unterscheiden, nämlich ob p_M für alle Markierungen $M \in [M_0>$ gleich oder verschieden ist. Ist $p_M=p$ für alle $M \in [M_0>$, so kann man untersuchen, welche Token erwartet werden. Durch eine Betrachtung, woher p die Token erwartet und wo solche Token vorkommen, kann man eventuell erkennen, warum die entsprechenden Token nicht bei p angekommen sind.

Diesen Fall kann man sich schematisch wie folgt darstellen:

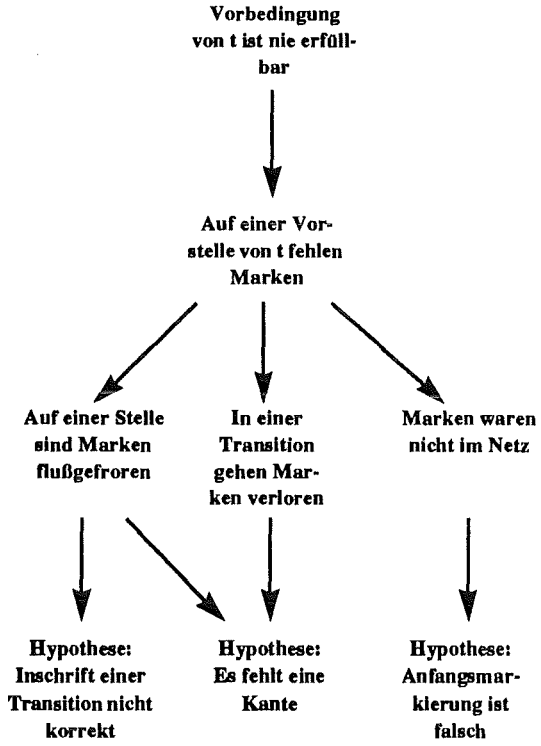


Abb. 15 Ursache für nicht erfüllte Vorbedingung

In diesem Schema haben wir außerdem noch einen zweiten Weg eingezeichnet, bei dem der Begriff der gefrorenen Marken auftaucht.Wir wollen nun den linken Ast des Schemas kurz erläutern. Nachdem man fehlende Marken ω einer Vorbedingung einer Transition bemerkt hat, versucht man eine Stelle zu finden, an der die Marken ω flußgefroren sind. Findet man eine solche Stelle, so kann man die Hypothese aufstellen, daß an einer Transition entsprechend Kanten fehlen, oder die Transition gibt diese Marken wegen etwas anderem nicht weiter, z.B. weil die Transitionsinschrift nicht der Intention entsprechend gewählt wurde, was durch einen Implementierungsfehler sehr leicht auftreten kann.

(2) Inschrift von t ist nie erfüllbar

Mit "Inschrift von t ist nie erfüllbar" ist gemeint, daß die Transitionsinschrift nie zu "true" ausgewertet werden kann. Dabei können mindestens zwei Fälle auftreten, nämlich, daß die Formel mit keiner Ersetzung der Variablen true wird, damit ist die Formel logisch nicht korrekt, oder die Formel kann zu true ausgewertet werden, allerdings wird in dem Netz keine Markierung angenommen, in der die Formel zu true wird. Es wäre dann wünschenswert, wenn man finden könnte, was bei den möglichen Auswertungen fehlt. Es könnte z.B. sein, daß man von einer Variablen gewisse Werte benötigt, die aber nie auf den Inputstellen auftreten. Damit wäre man in dem oben schon behandelten Fall, daß eine Marke fehlt.

Die Abb. 16 zeigt uns den Sachverhalt.

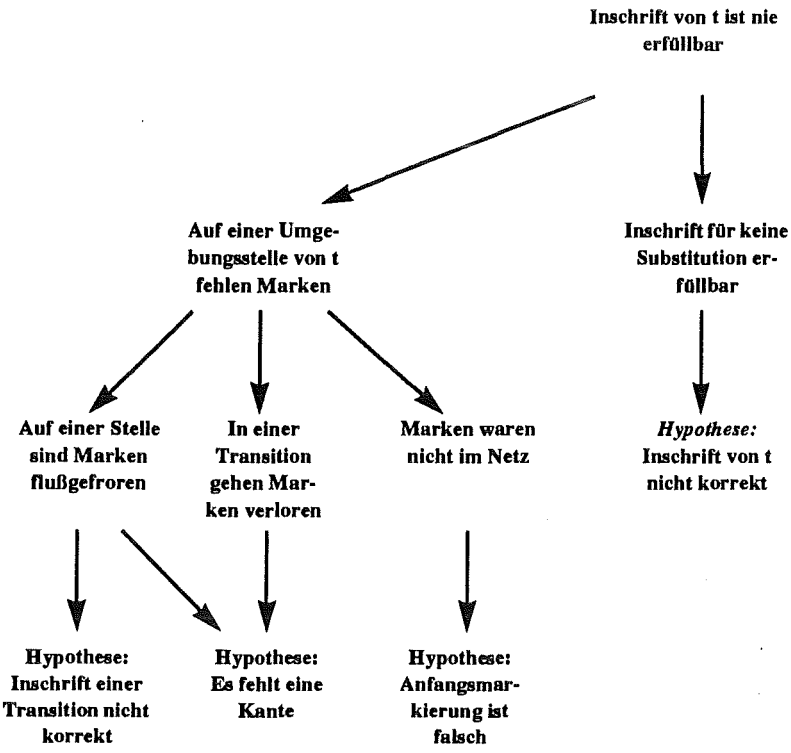


Abb. 16 Ursache für nicht erfüllte Inschrift

(3) Nachbedingung von t ist nie erfüllbar

Unter "Nachbedingung von t ist nie erfüllbar" wollen wir dabei die Situation verstehen, daß bei jeder von der Anfangsmarkierung M_0 aus erreichbaren Markierung M die Nachstellen von t nicht genügend Kapazität haben oder die Formeln nicht erfüllt sind. Formal läßt sich dies in der folgenden Weise darstellen: Für alle $M \in [M_0>$ und alle Auswertungen v existiert ein $p_M \in$ nach(t) und ein $s \in S$, so daß gilt:

$$card_s(m_{p_M}) - card_s(A_F(p_M, t)) + card_s(A_F(p_M, t)) > cap_{p_M}(s).$$

Hierbei lassen sich wieder zwei Fälle unterscheiden, nämlich ob p_M für alle Markierungen M∈ [M_0> gleich oder verschieden ist. Ist p_M=p für alle M∈ [M_0>, so kann es sein, daß auf p Marken liegen, die gefroren sind. Dann muß man bei diesen gefrorenen Marken untersuchen, warum sie gefroren sind. Damit ergibt sich folgendes Gesamtschema für die Untersuchung, warum eine Transition t tot ist.

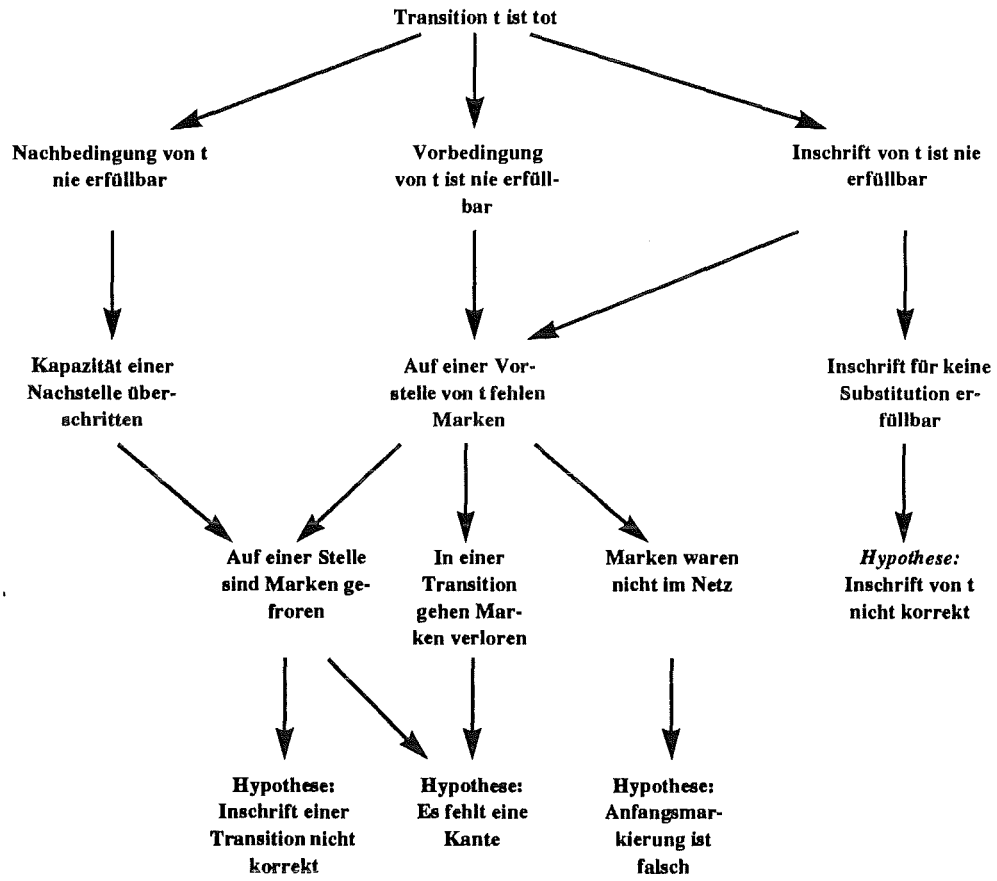


Abb. 17 Gesamtschema für die Ursachen einer toten Transition

6.3 Deadlocks

Hat man einen Deadlock entdeckt, der nicht gewünscht ist und den man sich nicht erklären kann, so erwartet man von dem System Hilfe bei der Analyse. Man könnte sich dabei die im folgenden Diagramm angedeuteten Ursachen für einen Deadlock vorstellen.

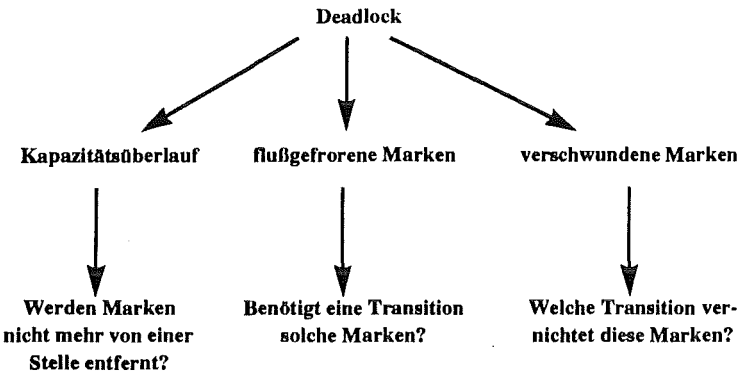


Abb. 18 Ursachen für Deadlocks

Wir wollen das Diagramm ein wenig näher erläutern. Wir haben das Ziel, bei Vorliegen eines Deadlocks die Ursache dafür zu finden. In diesem Diagramm sind einige Randerscheinungen aufgelistet, deren Auftreten vielleicht die Ursache für den Deadlock zeigt. Wir wollen diese Randerscheinungen näher erläutern.

(1) Kapazitätsüberlauf

In dem Deadlock kann es sein, daß die Vorbedingung und die Inschrift einer Transition t erfüllt sind, t aber nicht schalten kann, weil eine Nachbedingung nicht erfüllt ist. Solch eine Nachbedingung kann deshalb nicht erfüllt sein, weil die Kapazität einer Stelle erreicht ist. Dabei liegt die Vermutung nahe, daß von dieser Stelle Marken nicht korrekt abgezogen werden, ja sogar überhaupt nicht. Der Grund dafür könnte eine fehlende Kante zu einer "passenden" Transition sein. Es wäre auch denkbar, daß die Inschrift einer dieser "übergelaufenen" Stelle nachfolgenden Transition nicht korrekt ist. Das System sollte auf Transitionen, die Marken dieser Sorte benötigt, hinweisen und eine Hypothese abgeben, ob eine Kante fehlt oder eine Transition eine falsche Inschrift hat.

(2) Gefrorene Marken

Wie bei toten Transitionen kann man sich vorstellen, daß ein Deadlock eine Ursache in gefrorenen Marken hat. Wie wir schon in dem Abschnitt über gefrorene Marken gesehen haben, gibt es mehrere mögliche Ursachen. Man kann sich vorstellen, daß es eine Transition t gibt, die flußgefrorenen Marken ω einer Stelle p zum Schalten benötigt, aber keine Kante zwischen p und t existiert. Auch kann wieder die Inschrift einer Transition $t' \in p^\bullet$ nicht korrekt sein, wie wir dies schon öfters beschrieben haben.

(3) Verschwundene Marken

Der Grund für einen Deadlock könnte auch sein, daß eine Transition t deshalb nicht schalten kann, weil sie zum Schalten Marken ω benötigt, die aber überhaupt nicht im Netz vorkommen. Mit Hilfe von *maxvork*(ω) kann man prüfen, ob solche Marken überhaupt schon im Netz waren oder nicht. Falls nicht, d.h. ist *maxvork*(ω)=0, so wurden die Marken ω schon in der Anfangsmarkierung vergessen. Waren solche Marken schon vorhanden, d.h. war *maxvork*(ω)>0, so sind die Marken irgendwo "verloren" gegangen. Durch Betrachten derjenigen Transitionen, die Marken vernichten, findet man eventuell eine Transition, deren Inschrift falsch ist oder bei der eine Kante fehlt. Das System kann dabei eine Hypothese abgeben, indem es von der Transition t aus im Netz zurückgeht, bis es bei einer markenvernichtenden Transition ankommt. Die Wahrscheinlichkeit ist groß, daß diese Transition fehlerhaft ist.

6.4 Livelocks

Das Auftreten eines Livelocks kann von sehr vielen Ursachen abhängen. Wir wollen hier ein paar dieser Ursachen angeben und gehen dazu zuerst von sehr einfachen Fällen aus.

(1) Kleine Graphen der starken Zusammenhangskomponenten und ein Livelock

Wir wollen zuerst von Fällen ausgehen, bei denen wir einen sehr kleinen Graphen der starken Zusammenhangskomponenten und nur einen Livelock haben. Das folgende Schema zeigt einige Möglichkeiten.

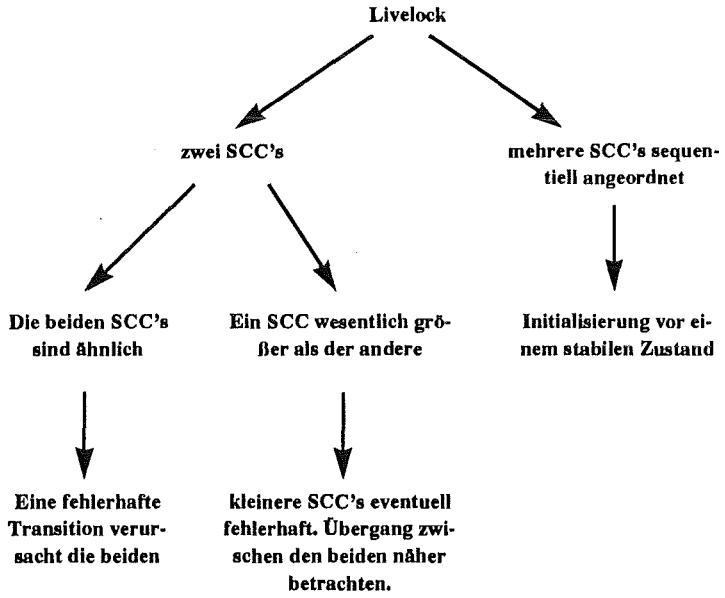


Abb. 19 Ursachen für Livelocks

Wir wollen das Schema näher erläutern. Der rechte Ast des Schemas zeigt eine Situation, zu der man sich den Graphen der starken Zusammenhangskomponenten wie folgt vorstellen kann.

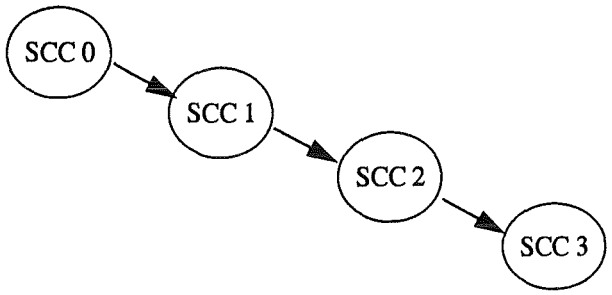


Abb. 20 Sequentiell angeordnete starke Zusammenhangskomponenten

Wir haben hierbei vier starke Zusammenhangskomponenten, die sequentiell hintereinander angeordnet sind. Sind SCC0, SCC1 und SCC2 außerdem noch relativ klein, bestehen vielleicht sogar nur aus einem Knoten, während der Livelock SCC3 umfangreich ist, so könnten SCC0, SCC1 und SCC2 Initialisierungsphasen sein, bevor das System in einen Zyklus hineinläuft, der ein gewünschtes Systemverhalten darstellt. Solche Initialisierungsphasen können durchaus gewünscht sein bzw. haben keinen Einfluß auf das Systemverhalten.

Haben wir im Graph der starken Zusammenhangskomponenten nur zwei starke Zusammenhangskomponenten, so ist es interessant, die Struktur dieser beiden Komponenten und die Übergänge zwischen ihnen zu betrachten. Es wäre möglich, daß die Komponenten sehr ähnlich sind, z.B. daß in beiden Komponenten dieselben Transitionen mit denselben Substitutionen schalten. Der Unterschied könnte darin bestehen, daß die Transition, die von der einen Komponente in die andere führt, Marken auf eine Stelle legt, die nicht mehr entfernt werden können. Wir hätten dann wieder gefrorene Marken, deren Beseitigung eventuell die beiden starken Zusammenhangskomponenten auf eine starke Zusammenhangskomponente reduziert.

Haben wir zwei starke Zusammenhangskomponenten, deren Größe sehr verschieden ist, so könnte es sein, daß die kleinere starke Zusammenhangskomponente nicht gewünscht ist. Man müßte nun die Transitionen, die den Übergang zwischen den Komponenten verursachen, genauer untersuchen. Eventuell kann man auch aus der Struktur dieser Komponente Aufschlüsse über sie erhalten.

(2) Mehrere Livelocks

Haben wir einen Graph der starken Zusammenhangskomponenten, in dem mehrere Livelocks auftreten, so kann man diesen Graphen vielleicht stückweise reduzieren. Wir wollen uns folgendes Beispiel eines Graphen der starken Zusammenhangskomponenten vorstellen.

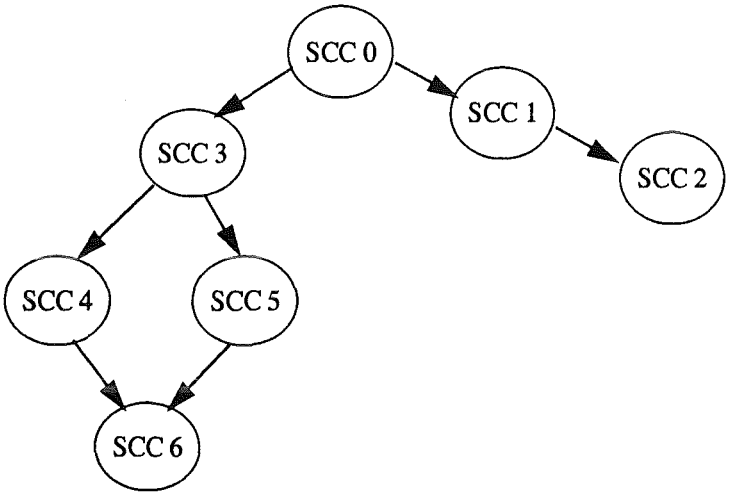


Abb. 21 Ein Graph von starken Zusammenhangskomponenten

Wir haben also in diesem Graphen zwei Livelocks SCC 2 und SCC6. Zu jedem dieser Livelocks kommt man über verschiedene Wege in dem Graphen. Vielleicht ist eine Reduzierung des Graphen dadurch möglich, daß man die starken Zusammenhangskomponten auf diesen beiden Wegen zusammenfaßt. Man kann diese Zusammenhangskomponenten nur dadurch zusammenfassen, daß man Fehler im Netz findet, die die starken Zusammenhangskomponenten reduzieren. In unserem Beispiel wäre ein erster Ansatzpunkt die Komponenten SCC 4 und SCC 5 zu reduzieren, indem man sich ihre Unterschiede anschaut. Vielleicht kann man den Graphen dann auf folgende Form bringen:

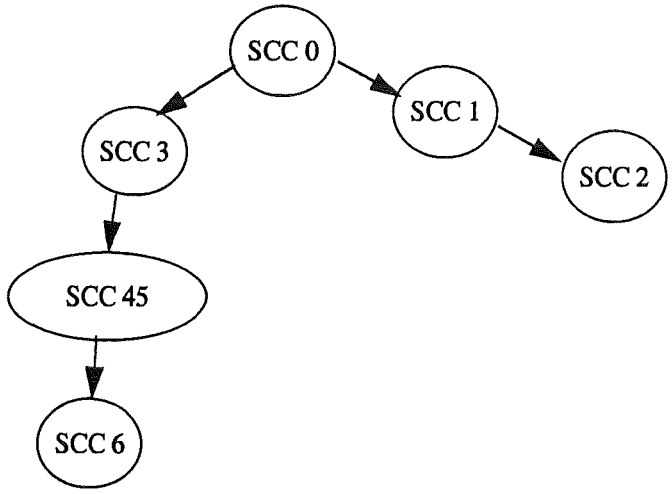


Abb. 22 Reduzierter Graph von starken Zusammenhangskomponenten

Findet man weitere Fehler im Netz, so kann man vielleicht den Graphen in eine noch einfachere Form bekommen, die dann die folgende Gestalt haben könnte.

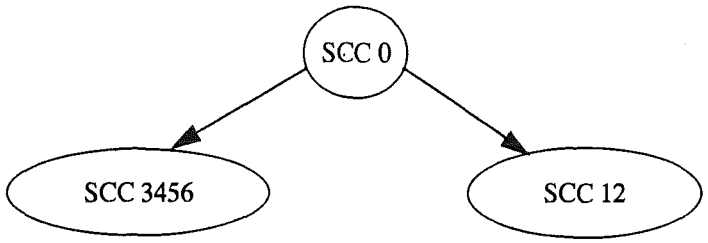


Abb. 23 Weitere Reduzierung des Graphen der starken Zusammenhangskomponenten

Nun verbleiben nur noch zwei Möglichkeiten, entweder die beiden Livelocks sind zwei Zyklen des Systems, die auch gewünscht sind, oder sie stellen einen Fehler dar, weil man nur eine starke Zusammenhangskomponente haben möchte. Falls man nur eine starke Zusammenhangskomponente haben möchte, so kann man vielleicht durch Strukturvergleiche der beiden Zusammenhangskomponenten oder durch Feststellung von gefrorenen Marken fehlerhaftes Verhalten des Netzes finden.

7. SMARAGD-Analyseeinheit

Wir beschreiben in diesem Kapitel die SMARAGD-Analyseeinheit. Dazu stellen wir zuerst das Konzept der Analyseeinheit vor und beschreiben dann die konkrete Realisierung eines ersten Prototyps der SMARAGD-Analyseeinheit. Die Analyseeinheit enthält noch nicht die regelbasierte Auswertung von Analyseergebnissen, denn bevor man etwas derart Neues integriert, sollten erst Erfahrungen über den praktischen Nutzen an Hand von Modellsituationen gesammelt werden.

7.1 Konzept der Analyseeinheit

Die Analyseeinheit von SMARAGD dient der Analyse von SNL-Systemen. Wir haben in den vorhergehenden Kapiteln die verschiedenen Analysefähigkeiten und -möglichkeiten dieser Systeme beschrieben. SMARAGD soll einmal alle diese Analysefähigkeiten bereitstellen. In diesem Abschnitt wollen wir das Gesamtkonzept der SMARAGD-Analyseeinheit vorstellen. Den Grundgedanken des Konzeptes können wir in einem Satz zusammenfassen:

Die Analyseeinheit ermittelt Eigenschaften eines SNL-Systems und erleichtert durch geeignete graphische Darstellungsweisen die Analyse dieses SNL-Systems.

Das Konzept, das wir vorstellen, beschreibt den bisher existierende Prototyp von SMARAGD. In diesem Prototyp ist nur die Erreichbarkeitsanalyse realisiert. Invariantenverifikation und regelbasierte Auswertung von Analyseergebnissen sind vorgesehen, werden jedoch erst in einer späteren Version von SMARAGD realisiert.

Es ist noch einmal zu betonen, daß die Erreichbarkeitsanalyse von SMARAGD auf dem Erreichbarkeitsgraph basiert. Schrittvorgänge sind deshalb in SMARAGD nicht zugelassen. Das hat praktische Gründe, denn die Erreichbarkeitsanalyse auf der Basis des Schrittgraphen erfordert einen sehr viel größeren Zeit- und Speicherplatzaufwand. Das bedeutet nicht, daß wir keine Parallelität im Netz haben, da die meisten Schrittvorgänge mit Schaltfolgen überdeckt werden können.

Wir beginnen die Beschreibung des Konzeptes mit einem Überblick über das komplette SMARAGD-System. SMARAGD ist in die Teile Editor, Simulator und Analyseeinheit (Analyser) gegliedert. Beim Aufruf von SMARAGD erhält deshalb der Benutzer das folgende Menü:

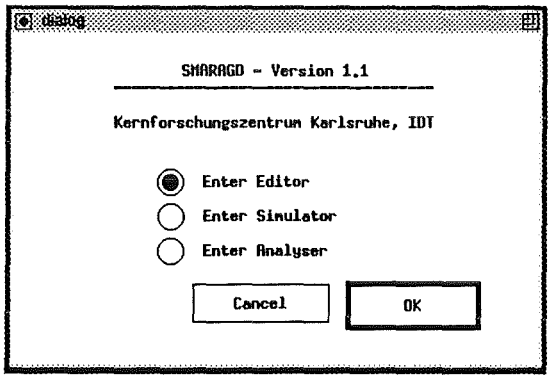


Abb. 24 Auswahlmenü von SMARAGD

Zu Beginn einer Sitzung kann der Benutzer entweder in den Editor oder die Analyseeinheit gehen. Um ein neues Netz zu erstellen oder ein vorhandenes Netz zu modifizieren, geht der Benutzer in den Editor. Wenn zu einem Netz der Erreichbarkeitsgraph bereits existiert, kann der Benutzer auch direkt in die Analyseeinheit gehen. Hat der Benutzer *Enter Editor* gewählt, schließt SMARAGD das Dialogfenster aus Abb. 24 und öffnet die Menüleiste des Editors (siehe Abb. 25).

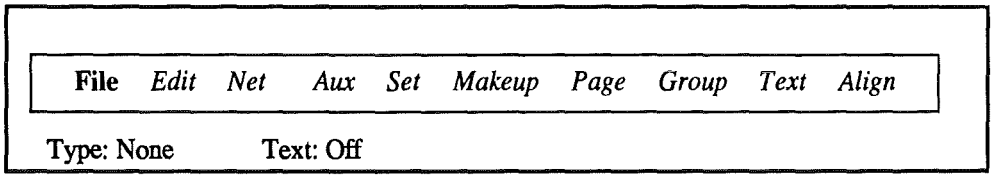


Abb. 25 Menüleiste des Editors

Durch Anklicken des Menüpunktes **File** wird ein Untermenü aktiviert (siehe Abb. 26), durch das ein Netz neu erstellt oder ein bestehendes Netz geladen werden kann.

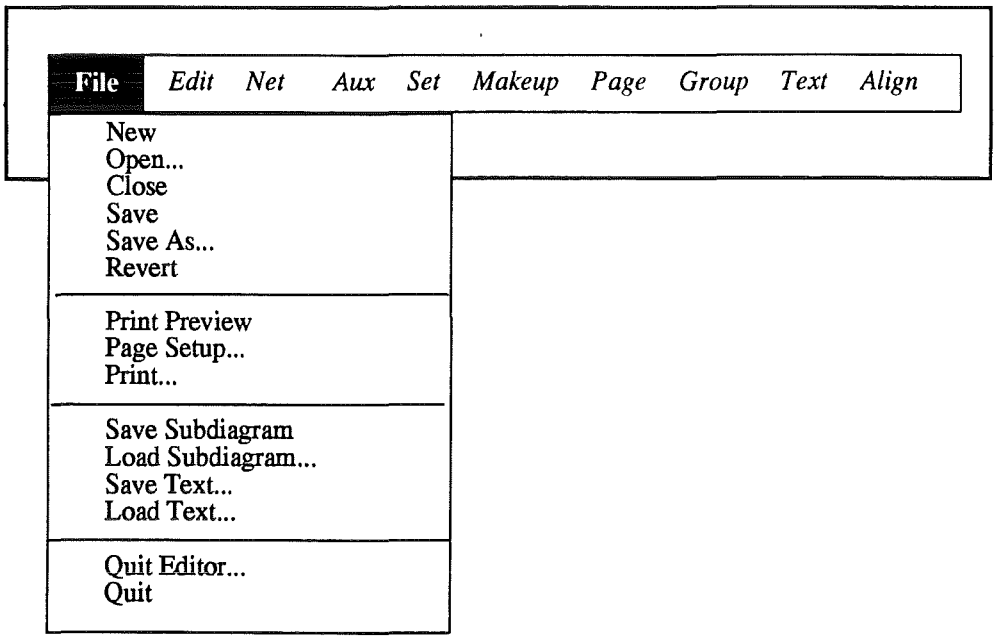


Abb. 26 Untermenü **File**

Möchten wir ein neues Netz generieren, so müssen wir die Menüpunkte des Untermenüs **Net** benutzen. Abb. 27 zeigt die Untermenüpunkte.

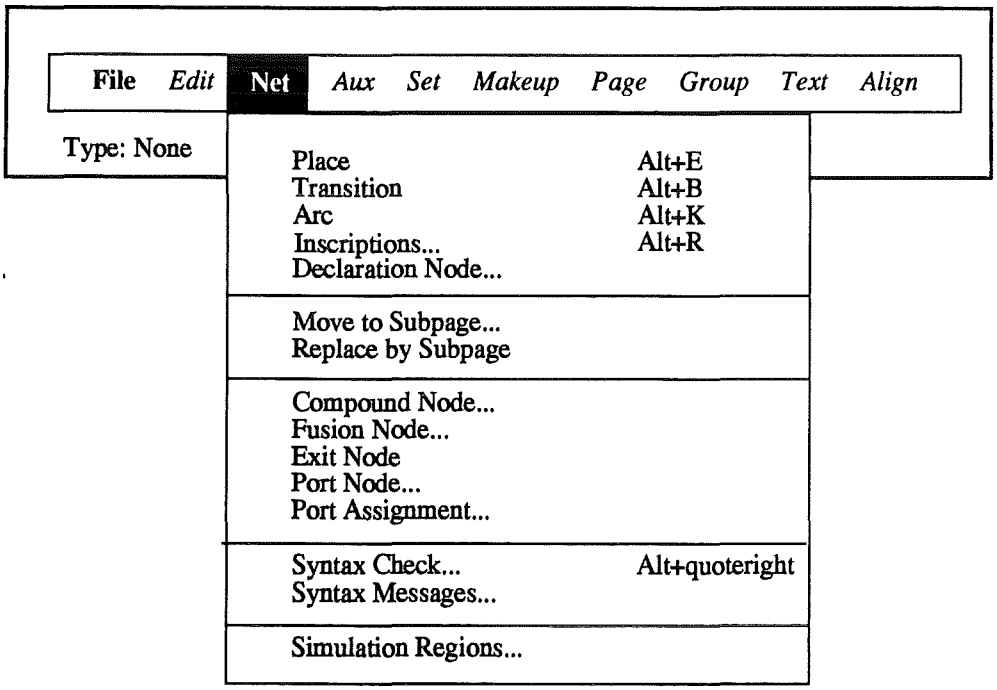


Abb. 27 Untermenü **Net**

Nach dem Öffnen bzw. Generieren eines Netzes haben wir auf dem Bildschirm die folgende Situation: Neben der Menüleiste aus Abb. 25 haben wir eine oder mehrere Seiten mit unserem Netzmodell. In Abb. 28 haben wir ein, aus einer Seite bestehendes, SMARAGD-Netz mit der Editormenüleiste dargestellt.

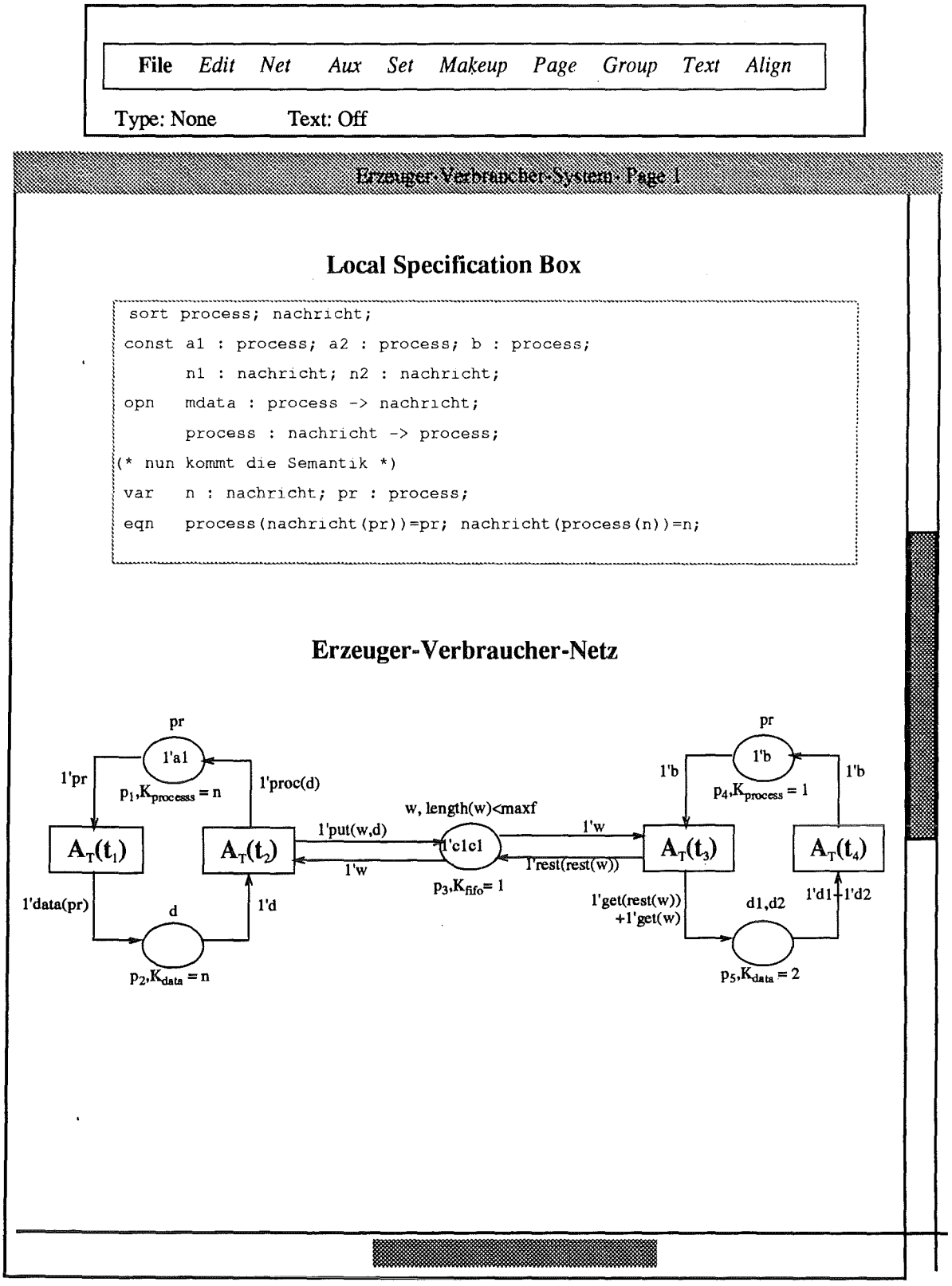


Abb. 28 Geöffnetes SMARAGD-Diagramm im Editor

Wollen wir nun dieses Netz analysieren, so wählen wir den Menüpunkt *Quit Editor ..* aus Abb. 26 an, wodurch wir wieder das Dialogfenster aus Abb. 24 erhalten. Hier können wir nun *Enter Analyser* auswählen, wodurch wir in die Analyseeinheit gelangen.

Man beachte dabei, daß es unterschiedlich ist ob wir direkt oder über den Editor in die Analyseeinheit gehen. Waren wir zuerst im Editor, so haben wir ein Diagramm geöffnet. Sind wir direkt in die Analyseeinheit gegangen, so haben wir kein offenes Diagramm.

Die Analyseeinheit hat eine Menüleiste, die fast wie die Menüleiste des Editors aussieht, nur ist der Menüpunkt **Net** durch den Menüpunkt **Analyse** ersetzt (siehe Abb. 29).

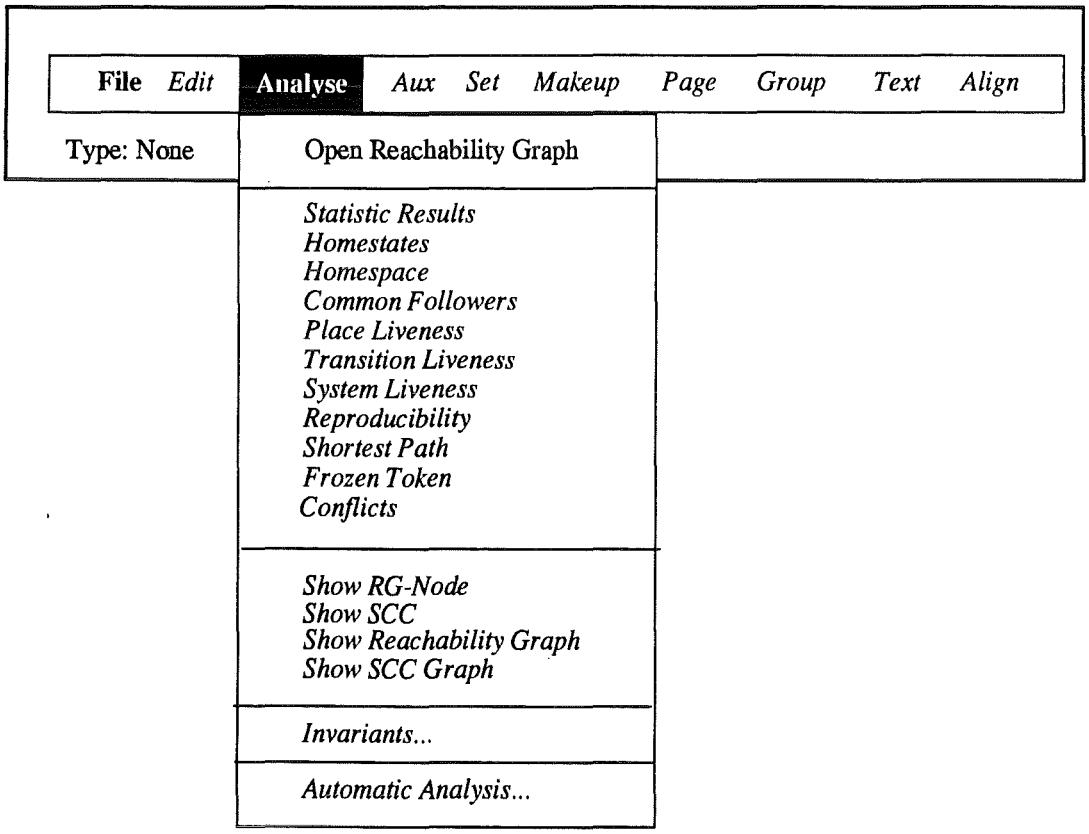


Abb. 29 Der Untermenüpunkt Analyse

Wir beginnen die Analyse eines SMARAGD-Diagramms, durch Auswahl des Untermenüpunktes *Open Reachability Graph* von **Analyse**. Wir können damit entweder zu einem geöffneten Diagramm den Erreichbarkeitsgraph generieren oder einen existierenden Erreichbarkeitsgraph einlesen. Nach dem Aufbau des Erreichbarkeitsgraphen erzeugt das Werkzeug automatisch den Graph der starken Zusammenhangskomponenten. Das Werkzeug hat die Möglichkeit diese beiden Graphen, oder Teile davon, graphisch darzustellen.

Mit dem Untermenüpunkt *Statistic Results* können wir eine Seite mit Namen *Results* öffnen, auf der die Ergebnisse der weiter vorne beschriebenen Analyseergebnisse textuell angegeben werden.

Eine wichtige Eigenschaft der SMARAGD-Analyseeinheit ist nun, daß die graphischen Darstellungen des Netzes und der beiden Graphen sowie die Ausgabeseite *Results* kombiniert werden können. Dadurch wird es für den Benutzer leichter in kurzer Zeit einen Überblick über das Verhalten des Netzes zu bekommen. Wir wollen im Folgenden prinzipiell vorstellen, wie diese Kombination aussieht. In der Beschreibung der konkreten Realisierung werden wir dann näher darauf eingehen.

Wir schauen uns zunächst ein typisches Bildschirmlayout des SMARAGD-Systems an.

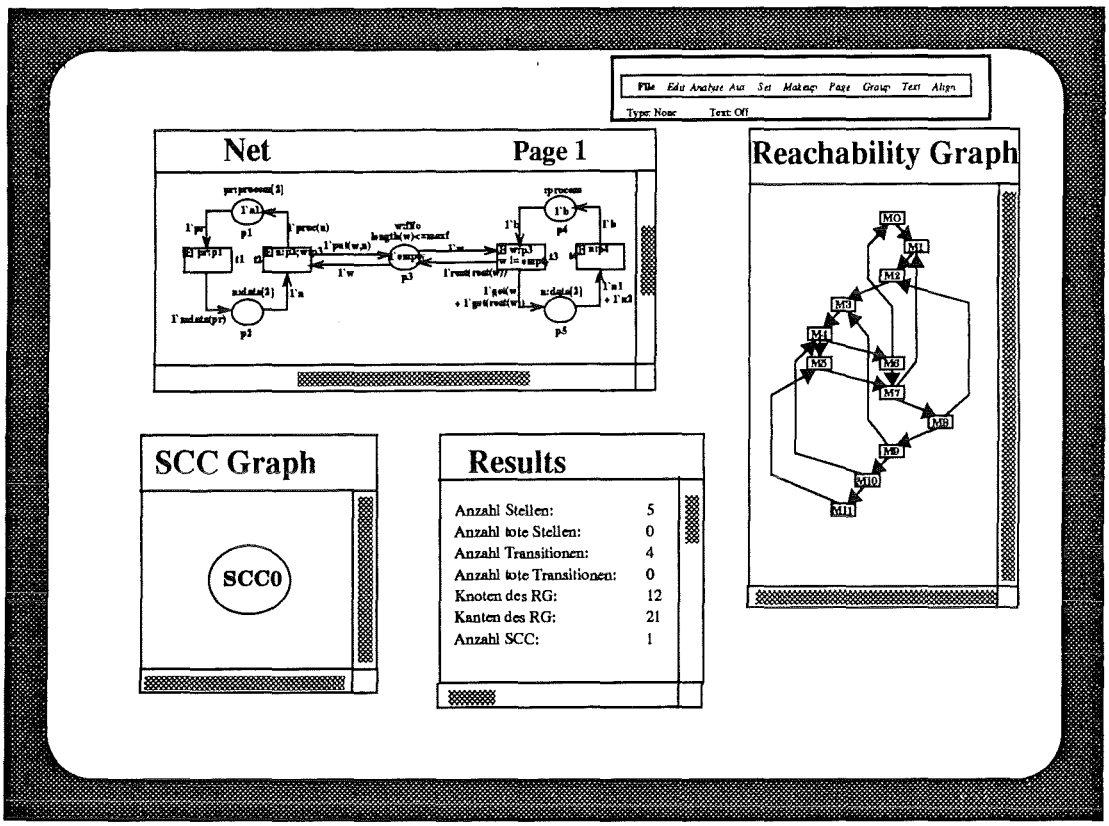


Abb. 30 Bildschirmlayout der SMARAGD-Analyseeinheit

Wir haben auf dem Bildschirm die Menüleiste der Analyseeinheit, ein SNL-Diagramm, den Erreichbarkeitsgraph, den Graph der starken Zusammenhangskomponenten und eine Seite *Results* mit den Analyseergebnissen. Die verschiedenen Darstellungen hängen wechselseitig zusammen, damit der Benutzer das Verhalten des Netzes besser überblicken kann.

Z.B. ist es möglich, einen Knoten des Erreichbarkeitsgraphen anzuklicken, wodurch sich die zu diesem Knoten gehörige Markierung im Netz-Modell und in *Results* darstellt. In Abb. 31 haben wir dies verdeutlicht. Im Erreichbarkeitsgraph ist die Markierung M4 angeklickt, wodurch im Netz das SNL-System mit dieser Markierung und in der Seite *Results* eine textuelle Darstellung der Markierung erscheint.

Man erhält die Markierung M4 nicht nur über den Erreichbarkeitsgraph, sondern man hat auch die Möglichkeit, sich über die Menüleiste den Untermenüpunkt *Show RG-Node* anzuwählen, wodurch man auswählen kann, welche Markierung im Netz und in der Seite *Results* gezeigt wird. Zu dieser Markierung wird dann automatisch auch ihre Stellung im Erreichbarkeitsgraph gezeigt. Haben wir kein offenes Diagramm, so entfällt die Darstellung einer Markierung im Netz. Eine genauere Beschreibung der Möglichkeiten von SMARAGD geben wir weiter unten.

Der Menüpunkt *Invariants ..* und *Automatical Analysis ..* stehen in der bisherigen Version von SMARAGD nicht zur Verfügung. Der Menüpunkt *Invariants ..* wird dem Benutzer die Möglichkeit bieten, Invarianten zu verifizieren. Mit dem Menüpunkt *Automatical Analysis ..* soll in einer späteren Version eine regelbasierte Auswertung von Analyseergebnissen gestartet werden.

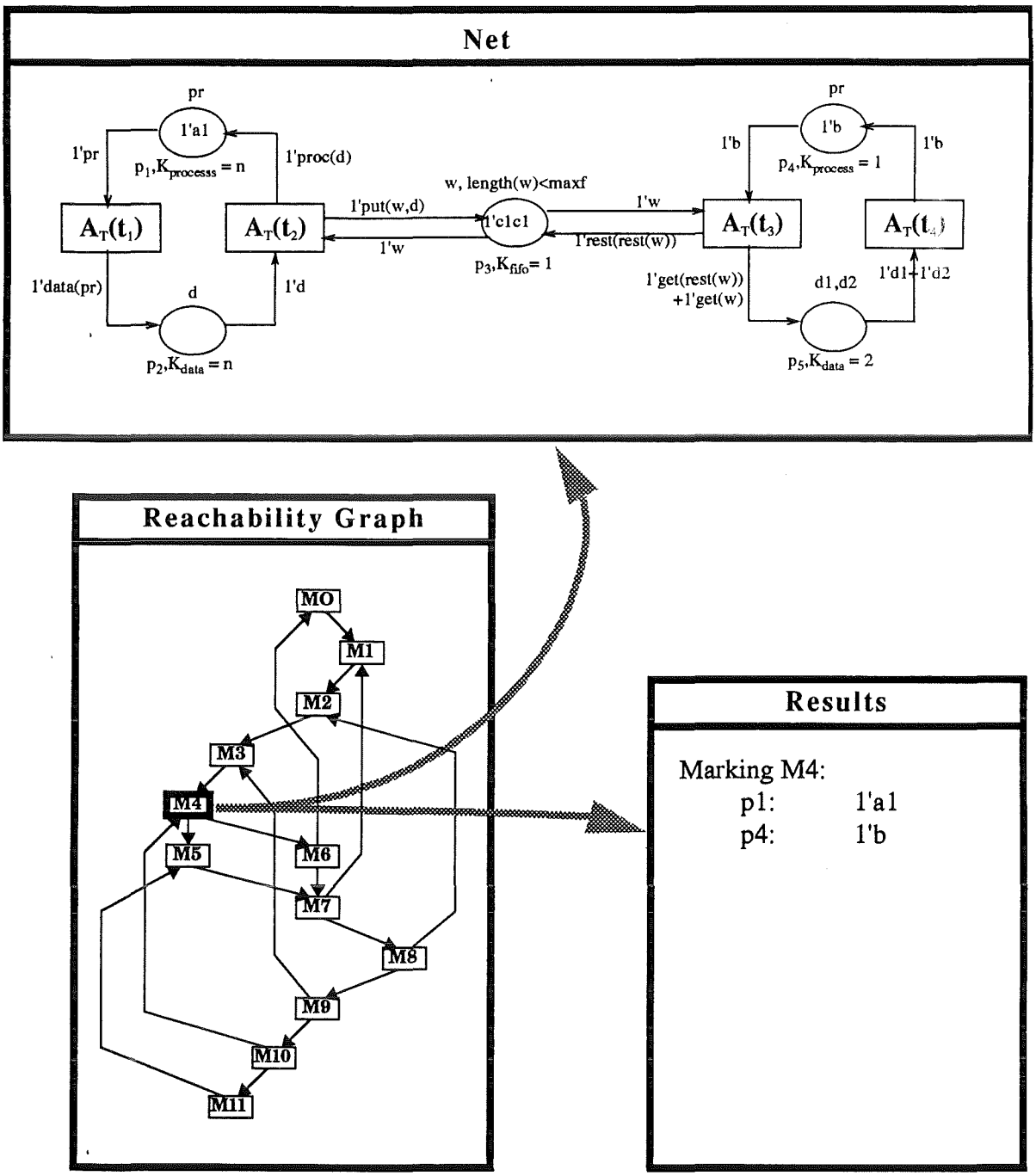


Abb. 31 Graphische Beziehung zwischen Netz und Erreichbarkeitsgraph

7.2 Realisierung der SMARAGD-Analyseeinheit

In diesem Abschnitt werden wir die konkrete Realisierung des 1. Prototyps der SMARAGD-Analyseeinheit beschreiben. Dazu beginnen wir mit der Beschreibung der Funktionalität des ersten Prototyps von SMARAGD. Nach ein paar Sätzen zur Umgebung von SMARAGD werden wir dann auf wichtige Details der Implementierung, wie etwa der Datenhaltung eingehen.

7.2.1 Funktionalität des ersten Prototyps von SMARAGD

Im ersten Prototyp der SMARAGD-Analyseeinheit ist die Erreichbarkeitsanalyse für unsere SNL-Systeme realisiert. Wir wollen diesen Prototyp im folgenden beschreiben.

Wir haben im vorherigen Abschnitt den Aufbau von SMARAGD mit den drei Teilen Editor, Simulator und Analyser angesprochen. Die Teile Editor und Simulator werden in [Düpmeier92] näher erläutert. Wir wollen hier die Analyseeinheit (Analyser) beschreiben. Wir haben schon gesehen, daß der wesentliche Teil der Analyseeinheit die Menüleiste ist. Wir wiederholen dazu die Darstellung der Menüleiste aus Abb. 29:

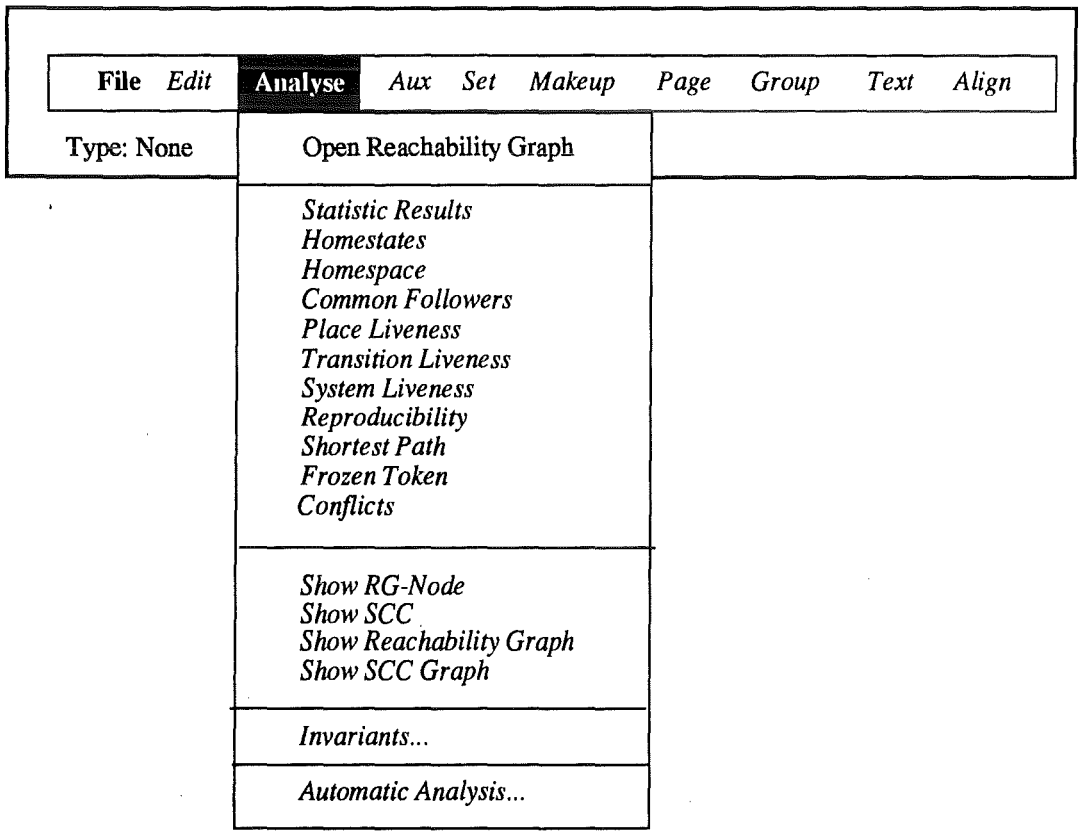


Abb. 32 Menüleiste der Analyseeinheit

Nach dem Öffnen der Analyseeinheit ist nur der Untermenüpunkt *Open Reachability Graph* von **Analyse** selektierbar. Die in kursiver Schrift geschriebenen Menüpunkte sind zu diesem Zeitpunkt nicht selektierbar, was sinnvoll ist, denn bevor wir auf den Erreichbarkeitsgraph zugreifen können, muß er erst existieren. Wählen wir *Open Reachability Graph* aus, so erscheint der in Abb. 33 dargestellte Dialog.

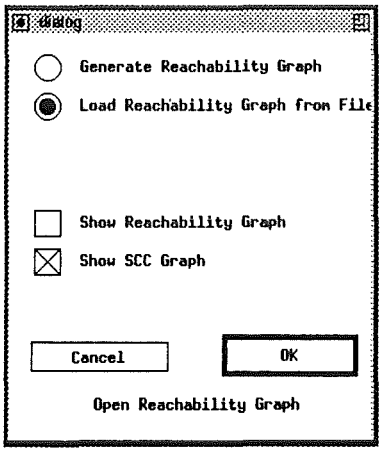


Abb. 33 Der Dialog *Open Reachability Graph*

Wir können bei diesem Dialog auswählen, ob wir den Erreichbarkeitsgraph zu einem geöffneten Diagramm generieren oder einen bereits erzeugten Erreichbarkeitsgraph von einer Datei einlesen wollen. Das System erzeugt zu dem Erreichbarkeitsgraphen dann automatisch den Graph der starken Zusammenhangskomponenten.

Wir können optional angeben, ob wir den Erreichbarkeitsgraph und bzw. oder den Graph der starken Zusammenhangskomponenten graphisch dargestellt haben wollen. Zum Einlesen eines existierenden Erreichbarkeitsgraphen ruft SMARAGD einen weiteren Dialog auf, der nach der Art eines "Directory Browsers" die Selektion eines Dateinamens erlaubt.

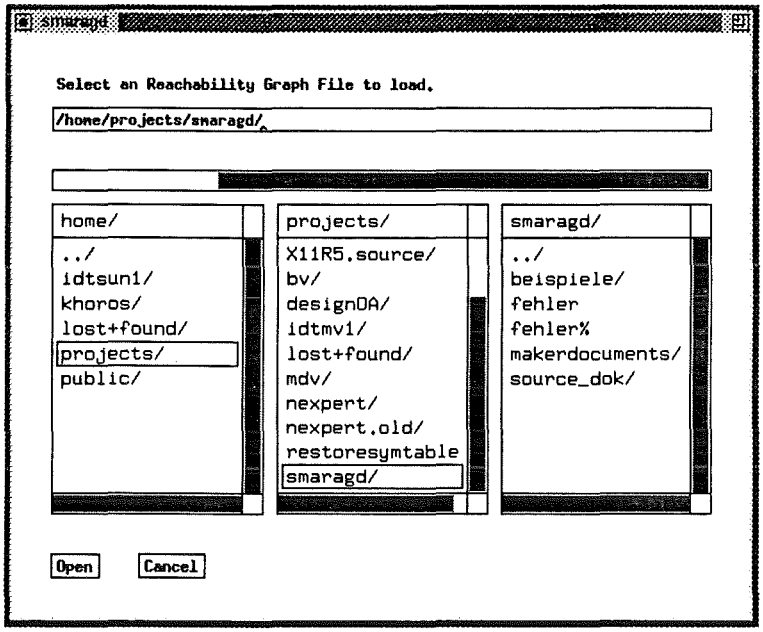


Abb. 34 Menü zum Auswählen eines Dateinamens

Nach dem Aufbau des Erreichbarkeitsgraphen sind in unserer Menüleiste die Untermenüpunkte *Statistic Results*, *Show Reachability Graph* und *Show SCC Graph* selektierbar. Die Menüpunkte *Show Reachability Graph* und *Show SCC Graph* dienen dazu den Erreichbarkeitsgraph bzw. den Graph der starken Zusammenhangskomponenten graphisch darzustellen, falls dies nicht schon im Dialogmenü *Open Reachability Graph* gewählt wurde.

Wir wollen auf diese graphische Darstellung näher eingehen. Das Layout der beiden Graphen muß auf dem Bildschirm automatisch erzeugt werden. Die Darstellung soll dabei möglichst übersichtlich sein. Dieses Problem wurde in zahlreichen Grapheneditoren schon gelöst.

An der Universität Karlsruhe wurde das Werkzeug EDGE (Extendible Directed Graph Editor) entwickelt, in dem verschiedene Layoutalgorithmen realisiert sind. Wir haben uns, aus den in EDGE vorhandenen Layoutalgorithmen den Sugiyama-Algorithmus ausgewählt und in SMARAGD inkorporiert. Die Auswahl des Sugiyama-Algorithmus hat im wesentlichen subjektive Gründe. Da alle in EDGE realisierten Algorithmen in etwa dasselbe leisteten, wurde der Algorithmus gewählt, der nach unserer Meinung den besten Überblick über den Graphen bietet. In späteren Ausbaustufen des Werkzeuges kann man sich überlegen, ob weitere Darstellungsformen für die Graphen angeboten werden sollten. Das Werkzeug EDGE wird in [Newbery90] beschrieben. Ein weiteres Werkzeug, das wir zu diesem Zweck näher untersucht haben ist XGRAB (siehe dazu [Barnes89]). Näheres über den Sugiyama-Algorithmus findet man in [Sugiyama81] und [Rowe87].

In mehreren, weiter vorne angegebenen Darstellungen eines Erreichbarkeitsgraphen haben wir dieses automatisch erzeugte Layout bereits benutzt ohne dies erwähnt zu haben. Wir wollen in der Abb. 35 noch einmal anhand des Erreichbarkeitsgraphen unseres Erzeuger-Verbraucher Modells ein automatisches Layout, das nach dem Sugiyama-Algorithmus erstellt wurde, veranschaulichen.

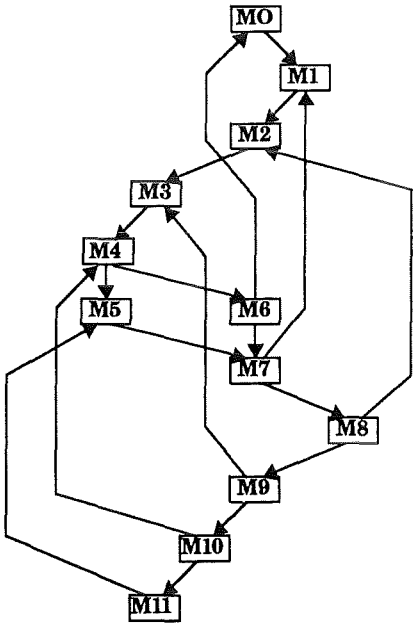


Abb. 35 Automatisches Layout des Erreichbarkeitsgraphen

Nach dem Aufbau der Graphen öffnen wir mit *Statistic Results* eine Seite mit Namen *Results*, auf der die verschiedenen Analyseergebnisse textuell angegeben werden. Beim Erzeugen hat diese Seite folgenden Aufbau:

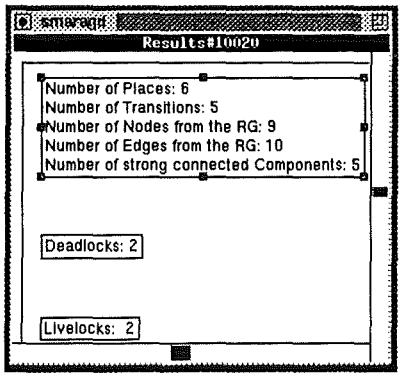


Abb. 36 Die Seite *Results*

Neben Informationen über die Anzahl der Stellen, Transitionen, Knoten, Kanten und starken Zusammenhangskomponenten haben wir angegeben, wieviel Deadlocks und Livelocks in dem Netz vorkommen. Diese Kästen, mit der Anzeige wieviel Deadlocks und Livelocks im Netz sind, sind maussensitiv. Durch Anklicken kann man nähere Informationen über diese Dead- bzw. Livelocks erhalten.

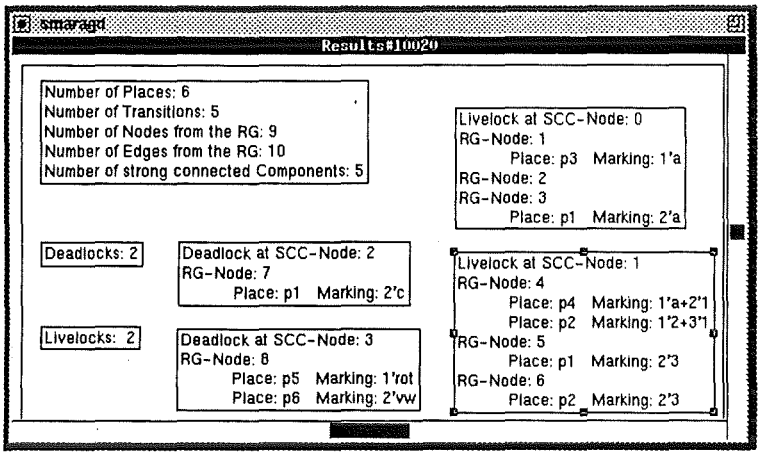


Abb. 37 Die Seite *Results* mit näherer Information zu den Verklemmungen

Klickt man diese Kästen mit den näheren Informationen noch einmal an, so erhält man die zugehörigen Markierungen im Erreichbarkeitsgraph oder im Graph der starken Zusammenhangskomponenten.

Nach dem Ausführen von *Statistic Results* sind alle Untermenüpunkte des Menüs **Analyse** selektierbar, außer *Invariants ..* und *Automatic Analysis ..*, die in dieser Version von SMARAGD noch nicht verfügbar sind.

Wir erläutern als nächstes wie wir die Heimzustände einer Markierung M berechnen. Haben wir eine Markierung im Erreichbarkeitsgraph selektiert, so wird durch Auswählen des Untermenüpunktes *Homestate* von dieser selektierten Markierung der Heimzustand berechnet. Haben wir keine Markierung selektiert, so kommt ein Dialogfenster hoch, in dem wir eine Markierung angeben können, von der dann der Heimzustand berechnet werden.

Das Ergebnis der Heimzustandberechnung wird auf der Seite *Results* angegeben und im Graph der starken Zusammenhangskomponenten, falls dieser geöffnet ist, angezeigt. Nach demselben Prinzip arbeiten auch die Menüpunkte *Homespace* und *Common Followers*. Wir betrachten noch einmal die Darstellung der Seite *Results*, dieses Mal mit Anzeige der Heimzustände und Heimräume.

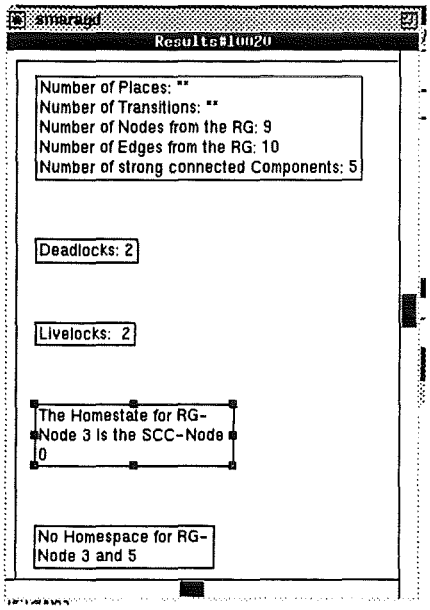


Abb. 38 Die Seite *Results* mit Angabe von Heimzuständen und Heimräumen

Auch hierbei sind die Kästen für die Heimzustände und Heimräume maussensitiv, so daß man nähere Informationen in der Seite *Results* und im Graph der starken Zusammenhangskomponenten erhält.

Mit dem Kommando *Place Liveness* ermitteln wir, ob eine Stelle tot ist. Wir können dazu die Stelle im Netz selektieren oder über einen Dialog den Namen der Stelle eingeben. Nach demselben Muster wird auch *Transition Liveness* ausgeführt, wobei von einer Transition bestimmt wird, ob sie tot, schwach lebendig, beschränkt aktivierbar oder stark lebendig ist. *System Liveness* stellt fest, ob das SNL-System stark oder schwach lebendig ist.

Reproducibility ermittelt die starke oder schwache Reproduzierbarkeit von Markierungen. Auch hierbei können wir die Markierungen entweder selektieren (im Erreichbarkeitsgraph) oder in einem Dialog angeben. Die Suche nach lokal, global oder flußgefrorenen Marken wird mit *Frozen Token* gestartet.

Durch Anwahl des Untermenüpunktes *Conflicts* werden im Netz Konflikte zwischen Transitionen graphisch dargestellt. Das folgende Bild zeigt eine derartige Möglichkeit. Das Netz in dem Bild soll dabei nur den prinzipiellen Sachverhalt ausdrücken und kein korrektes Netz sein. In dem Bild sind alle aktivierten Transitionen grau unterlegt. Die Transitionen mit Substitutionskonflikt haben zu der grauen Unterlegung eine dickere Umrandung. Transitionen, die in einem Schaltkonflikt zueinander stehen, sind mit gleichen Nummern K_i versehen.

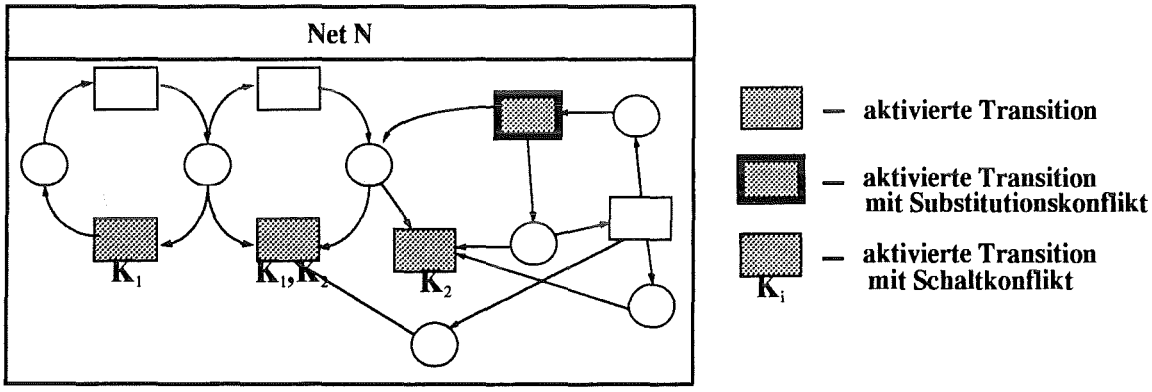


Abb. 39 Aktivierte Transitionen mit Konflikten

Mit *Show RG-Node* und *Show SCC* wird eine Markierung bzw. eine starke Zusammenhangskomponente textuell auf *Results* und graphisch im Erreichbarkeitsgraph bzw. Graph der starken Zusammenhangskomponenten gezeigt (siehe Abb. 40).

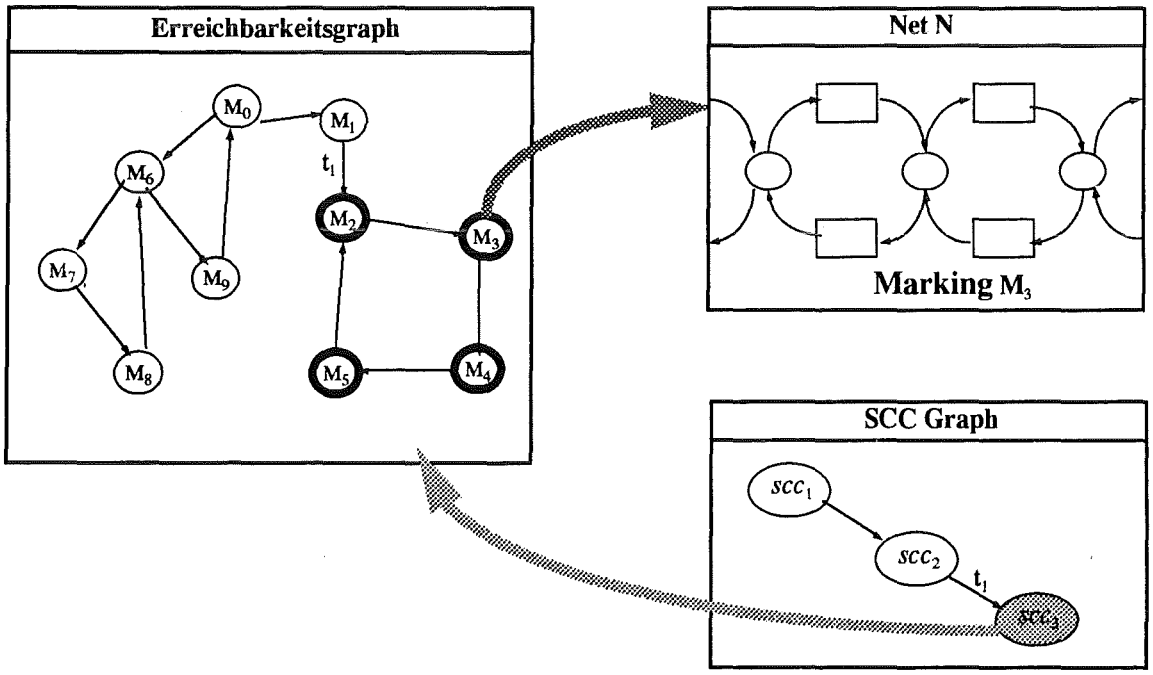


Abb. 40 Mehrere Fenster zur besseren Analyse eines Netzes

Nicht nur mit *Show RG-Node* und *Show SCC* erhält man Informationen über Markierungen und starke Zusammenhangskomponenten, sondern auch durch Anklicken. Klickt man eine starke Zusammenhangskomponente an, so erhält man die Markierungen dieser starken Zusammenhangskomponente im Erreichbarkeitsgraph geeignet markiert. Klickt man eine Markierung im Erreichbarkeitsgraph an, so erscheint das zugehörige SNL-System mit dieser Markierung.

Wir wollen in den folgenden Abschnitten ein wenig auf die Implementierung der SMARAGD-Analyseeinheit eingehen. Dabei werden zuerst ein paar Worte zu der Implementierung der graphischen Oberfläche sagen und dann die Datenstrukturen für die beiden Graphen vorstellen. Die Datenstrukturen für ein SNL-System werden in [Düpmeier92] vorgestellt.

7.2.2 Die Umgebung von SMARAGD

Wie bei jeder Systementwicklung, so stand auch am Anfang der Implementierung von SMARAGD die Frage in welcher Umgebung SMARAGD entwickelt wird. Rechner, Betriebssystem und Windowsystem waren vorgegeben und sind Sparcstations von der Firma Sun Microsystems, Betriebssystem Unix und das X-Windowsystem. Durch diese Randbedingungen lag es nahe, als Implementierungssprache die Sprache C bzw. C++ zu wählen. Blieb noch die Entscheidung, wie die graphische Oberfläche zu implementieren sei. Zur Wahl stand die direkte X-Windowprogrammierung oder die Benutzung eines Hilfsmittels, mit der man auf einem höheren Level die X-Windowprogrammierung realisieren konnte.

Wir haben uns für die Benutzung eines Hilfsmittels entschieden und dabei das Werkzeug *Design/OA* der Firma Meta Software Corporation ausgewählt. *Design/OA* stellt eine Bibliothek von verschiedenen Funktionen bereit, mit denen eine graphische Applikation relativ schnell realisiert werden kann. Ein wesentlicher Grund, warum wir uns für *Design/OA* entschieden haben, war, daß eine *Design/OA* Applikation namens *Design/CPN* existiert. *Design/CPN* ist ein Werkzeug zur Erstellung und Simulation von einer speziellen Form von Höheren Petri-Netzen, den sogenannten *Coloured Petri Nets*. Wir haben diese *Design/OA* Applikation als Grundgerüst für SMARAGD verwendet. Was dabei von *Design/CPN* benutzt wird, wird in [Düpmeier92] vorgestellt.

Für die Analyseeinheit haben wir von *Design/CPN* fast nichts verwenden können, sondern sie verwendet im wesentlichen *Design/OA*-Funktionen. Mit diesen *Design/OA*-Funktionen werden die einzelnen graphischen Darstellungen realisiert. Nähere Informationen zu *Design/OA* und *Design/CPN* sind in [OA90] und [CPN90] zu finden.

7.2.3 Die Datenstrukturen des Erreichbarkeitsgraphen

Dieser Abschnitt beschreibt die Datenstrukturen des Erreichbarkeitsgraphen. Wir spezifizieren dazu zuerst die Anforderungen an einen Erreichbarkeitsgraphen. Wie wir schon in Definition 38 gesehen haben, ist der Erreichbarkeitsgraph $\mathcal{RGM}_N=([M_0>, \mathcal{TM}_{N,M_0}, a, e)$ ein gerichteter Graph mit der Erreichbarkeitsmenge $[M_0>$ als Menge der Knoten und der Menge $\mathcal{TM}_{N,M_0}$ aller Schaltvorgänge von Transitionen in speziellen Modi als Menge der Kanten. Für den $\mathcal{RGM}_N$ gilt:

(i) $a((M,t:v,M'))=M$ für alle $(M,t:v,M') \in \mathcal{TM}_{N,M_0}$,

(ii) $e((M,t:v,M'))=M'$ für alle $(M,t:v,M') \in \mathcal{TM}_{N,M_0}$.

Eine geeignete Datenstruktur dieses gerichteten Graphen muß mehrere Anforderungen erfüllen, die wichtigsten davon sind:

(1) Leichtes Eintragen eines neuen Knotens oder einer neuen Kante in den Graph.
(2) Möglichst geringer Aufwand für die Entscheidung, ob eine Markierung $M \in [M_0>$ bereits im Graph enthalten ist oder nicht.
(3) Effiziente Speicherung des Graphen.
(4) Leichte Bestimmung der Vorgänger und Nachfolger eines Knotens.

Im folgenden werden wir zuerst einen Datentyp *REACHGRAPH* für den Erreichbarkeitsgraphen spezifizieren und danach eine geeignete Implementierung zu dieser Spezifikation angeben.

7.2.3.1 Spezifizierung eines Datentyps *REACHGRAPH*

Wir geben die Spezifikation des Datentyps *REACHGRAPH* für einen Erreichbarkeitsgraphen in C++ ähnlicher Notation an. Der Erreichbarkeitsgraph besitzt Knoten und Kanten, wobei wir zwischen Index und Information eines Knotens bzw. einer Kante unterscheiden. Der Index dient zur Referenzierung eines Knotens bzw. einer Kante. Ein Knoten des Erreichbarkeitsgraphen hat einen Index vom Typ *nodeindex* und eine Kante hat einen Index vom Typ *edgeindex*.

Die Information eines Knotens enthält eine Markierung des Netzes. Eine Kante des Erreichbarkeitsgraphen enthält die Information, daß eine Markierung M durch Schalten einer Transition t im Modus *v* in eine Markierung M' übergegangen ist. Für die Markierungen des Netzes benutzen wir einen Typ *MARKING* und für die Übergänge in einem Netz einen Datentyp *TRANSMODUS*, der eine Transition in einem speziellen Modus enthält.

Wir werden nun informell Operationen, die wir für einen Datentyp *REACHGRAPH* benötigen, beschreiben. Ausgehend von dieser qualitativen Beschreibung der Funktionalität eines solchen Datentyps werden wir eine genauere Spezifikation des Datentyps entwickeln.

bool insertNode (MARKING M, nodeindex n)

insertNode fügt einen neuen Knoten mit der Knoteninformation M in den Erreichbarkeitsgraphen ein. Diese Funktion muß überprüfen, ob ein Knoten mit Knoteninformation M schon existiert hat oder nicht, und darf nur dann einen neuen Knoten anlegen, wenn ein solcher Knoten noch nicht existiert hat. Wir benötigen damit in dem Datentyp einen Mechanismus, mit dem wir relativ schnell feststellen können, ob ein Knoten mit derselben Knoteninformation M bereits im Erreichbarkeitsgraphen existiert.
Als Rückgabewert gibt die Funktion an, ob der Knoten existierte oder nicht. Über das Argument *n* wird der Knotenindex zurückgegeben.

edgeindex insertEdge (nodeindex n1, nodeindex n2, TRANSMODUS T)

insertEdge fügt eine neue Kante in den Erreichbarkeitsgraphen ein. Diese Kante beginnt bei dem Knoten *n1* und endet bei dem Knoten *n2*, was als Ergebnis vom Typ *edgeindex* zurückgegeben wird. Als Information hat die Kante ein Element T des Datentyps *TRANSMODUS*, das eine Transition in einem speziellen Modus enthält. Durch das Schalten dieser Transition *T* in ihrem speziellen Modus geht das Netz von der Markierung, die die Information des Knotens *n1* darstellt, in die Markierung des Netzes, die die Information des Knotens *n2* darstellt, über. Der Algorithmus für den Aufbau des Erreichbarkeitsgraphen wurde dabei so konstruiert, daß ein Übergang in einem Netz nur genau einmal behandelt wird. Wir müssen deshalb keinen Mechanismus haben, mit dem wir vergleichen können, ob Kanten mit identischen Informationen vorliegen.

int count_of_nodes()
count_of_nodes gibt die Anzahl der Knoten im Erreichbarkeitsgraph an.

int count_of_edges()
count_of_edges gibt die Anzahl der Kanten im Erreichbarkeitsgraph an.

int SCC_number (nodeindex n)
Wie wir schon gesehen haben, sind die starken Zusammenhangskomponenten für die Analyse von großer Wichtigkeit. *SCC_number* gibt an, in welcher starken Zusammenhangskomponente des Erreichbarkeitsgraphen der Knoten mit dem Index *n* liegt. Dazu müssen die starken Zusammenhangskomponenten generiert werden. Dies geschieht beim ersten Aufruf dieser Funktion.

int number_of_SCC()
number_of_SCC gibt die Anzahl der starken Zusammenhangskomponenten im Erreichbarkeitsgraphen an. Auch hierbei müssen die starken Zusammenhangskomponenten vorher generiert worden sein. Falls nicht, müssen sie generiert werden.

list_of_nodes node_followers (nodeindex n)
node_followers erzeugt eine Liste mit den Knoten des Erreichbarkeitsgraphen, die direkte Nachfolger des Knoten *n* sind.

list_of_edges edge_followers (nodeindex n)
edge_followers erzeugt eine Liste mit den Kanten des Erreichbarkeitsgraphen, die vom Knoten *n* ausgehen.

Wir wollen nun in C++ ähnlicher Notation eine Klassendefinition des Datentyps *REACHGRAPH* machen, der die oben beschriebenen Operationen als Schnittstellen nach außen enthält. Die intern verwendeten Operationen und Typen (in C++ als private und protected bezeichnet) wollen wir erst später angeben.

```
class REACHGRAPH {
    private:
    protected:
    public:
        REACHGRAPH();
        ~REACHGRAPH();
        bool   insertNode(MARKING M);
        void   insertEdge(nodeindex n1, nodeindex n2, TRANSMODUS T);
        int    number_of_SCC();
        int    SCC_number(nodeindex n);
        int    count_of_nodes();
        int    count_of_edges();
        list_of_nodes node_followers(nodeindex n);
        list_of_edges edge_followers(nodeindex n);
}
```

Wir haben damit die äußere Schnittstelle des Datentyps spezifiziert. Wir wollen nun den internen Datentyp näher spezifizieren. Bei der Beschreibung der Funktion *insertNode* haben wir gesehen, daß im Datentyp *REACHGRAPH* Knoten und Kanten mit der jeweiligen Information enthalten sein müssen. In diesem Graph müssen wir sehr häufig nach einem Knoten mit einer gewissen Knoteninformation suchen, weshalb diese Operation möglichst wenig Aufwand bereiten sollte. Um diesen Aufwand zu begrenzen, bilden wir eine geeignete Struktur für *REACHGRAPH* aus zwei Teilen. Der eine Teil dient der Speicherung des Graphen. Wir wollen diesen Teil mit *RGRAPH* bezeichnen. Der andere Teil ist eine Hashtabelle, die ein schnelles Suchen innerhalb des Graphen ermöglicht. Wir wollen diese beiden Teile näher erläutern.

7.2.3.2 Die Klasse *RGRAPH*

Die Datenstruktur *RGRAPH* nimmt die Struktur des Erreichbarkeitsgraphen auf. Dazu benötigen wir Strukturen für die Knoten und die Kanten, die wir im folgenden *Knotenpool* und *Kantenpool* bezeichnen wollen.

Im Knotenpool unterscheiden wir zwischen Index und Information des Knotens. In der Information des Knotens halten wir eine Markierung des Netzes. Der Index eines Knotens wird beim Erzeugen eines neuen Knotens des Erreichbarkeitsgraphen generiert. Dieser Index erlaubt einen schnellen Zugriff auf einen Knoten des Erreichbarkeitsgraphen. Wir benötigen diesen schnellen Zugriff im Kantenpool und in der Hashtabelle.

Den Knotenpool kann man sich als eine Art Liste vorstellen, in der jedes Listenelement über den Index angesprochen werden kann. Der Inhalt dieses Listenelements beinhaltet die Information des Knotens. Die Indizierung der Knoten geschieht am einfachsten über natürliche Zahlen, in der Art, daß der Index eines neuen Knotens von der Anzahl der Knoten im Erreichbarkeitsgraphen abhängig ist. Ist die Anzahl der Knoten des Erreichbarkeitsgraphen 0, so erhält der erste Knoten, der eingetragen wird, den Index 1. Ist die Anzahl der Knoten des Erreichbarkeitsgraphen n, so erhält der neue Knoten den Index n+1.

In Abb. 41 wollen wir eine grobe Übersicht über den Knotenpool angeben.

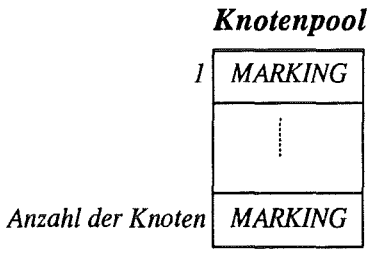


Abb. 41 Der Knotenpool des Erreichbarkeitsgraphen

Für den Knotenpool wollen wir einen Datentyp *NODES* in einer C++-ähnlichen Notation spezifizieren, wobei wir einen Typ *List_of_MARKING* für die Liste von Markierungen und einen Typ *nodeindex* = *int* für den Index eines Knotens verwenden wollen.

```
class NODES {
    friend class EDGES
    protected:
        int count_of_nodes;
        List_of_MARKING mlist;
    public:
        NODES();
        ~NODES();
        MARKING    get_Marking (nodeindex n);
        nodeindex  input_Marking(MARKING M);
        int        generate_SCC();
        int        get_SCC_number(nodeindex n);
        int        get_Scount_of_nodes() { return (count_of_nodes) };
}
```

Wir wollen die Funktionen dieser Klasse näher erläutern:

get_Marking
: *get_Marking* holt zu einem Index eines Knotens die zugehörige Knoteninformation vom Typ MARKING.

input_Marking
: *input_Marking* generiert einen neuen Knoten mit Knoteninformation vom Typ MARKING. Dabei wird *count_of_nodes* um eins erhöht und als Knotenindex zurückgegeben.

generate_SCC
: *generate_SCC* generiert die starken Zusammenhangskomponenten und gibt die Anzahl der starken Zusammenhangskomponenten zurück. In *mlist* wird außerdem zu jedem Knoten die zugehörige starke Zusammenhangskomponente festgehalten. *generate_SCC* benötigt auch die Kanten des Erreichbarkeitsgraphen und muß deshalb auch auf dem Kantenpool operieren können.

get_SCC_number

Zu einem Knoten mit Index *n* wird die Nummer der starken Zusammenhangskomponente zurückgegeben.

get_count_of_nodes

get_count_of_nodes gibt den aktuellen Wert von *count_of_nodes* zurück.

Auch im Kantenpool unterscheiden wir zwischen Index und Information einer Kante. Der Index einer Kante besteht aus einem Paar (i,j), wobei i der Index des Anfangsknotens und j der Index des Endknotens der Kante ist. Die Kanteninformation ist vom Typ *TRANSMODUS* und beinhaltet eine Transition mit einem speziellen Modus. Eine grobe Veranschaulichung des Kantenpools gibt Abb. 42.

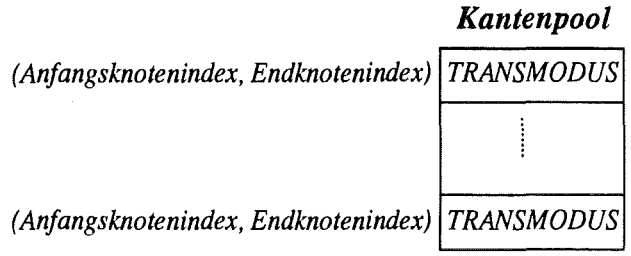


Abb. 42 Der Kantenpool des Erreichbarkeitsgraphen

Um den Kantenpool in C++-Notation spezifizieren zu können, müssen wir ihn noch weiter verfeinern. Die Indizierung des Kantenpool zerlegen wir in zwei Listen. Eine Liste, die wir *SOURCELIST* nennen wollen, nimmt die Anfangsknoten einer Kante auf. Sie hat dieselbe Indizierung wie der Knotenpool und beinhaltet einen Eintrag für jeden Knoten des Knotenpools. Jedes Element j dieser *SOURCELIST* hat als Element wieder eine Liste, die wir *FOLLOWERLIST* nennen, in der alle Indizes der Folgeknoten von j sowie alle Kanteninformationen (TRANSMODUS) der von j abgehenden Kanten enthalten sind.

Die Abb. 43 soll die Verfeinerung des Kantenpools näher verdeutlichen.

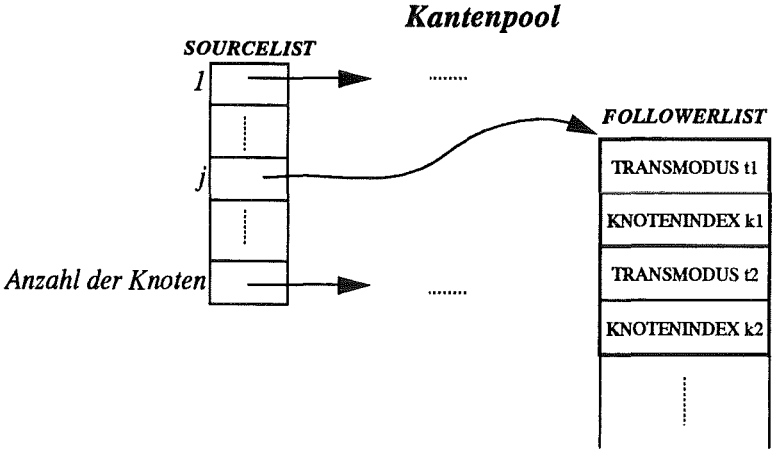


Abb. 43 Eine Verfeinerung des Kantenpools

In Abb. 43 haben wir zu einem Knoten mit Index j einen Nachfolgerknoten mit Index k1. Die Kanteninformation der Kante (j,k1) ist t1. Als weitere Kante haben wir eine Kante mit Index (j,k2) und Kanteninformation t2 angegeben. Durch die Verfeinerung des Kantenpools ist es für uns leichter, alle Nachfolgerknoten und alle Nachfolgekanten eines Knotens zu bestimmen. Wir wollen eine Klasse *EDGES* für unseren Kantenpool spezifizieren. Wir benutzen dazu Datentypen *edgeindex=int* × *int*, *FOLLOWERLIST* als Datentyp für die Nachfolgeknoten und -kanten und *SOURCELIST(FOLLOWERLIST)* als Liste mit Elementen vom Typ *FOLLOWERLIST*.

list_of_nodes und *list_of_edges* geben Listen von Knoten- bzw. Kantenindizes an.

```
class EDGES {
    friend class NODES;
    protected:
        int count_of_edges;
        SOURCELIST(FOLLOWERLIST) flist;
    public:
        EDGES();
        ~EDGES();
        edgeindex  input_EDGE(nodeindex s, nodeindex t, TRANSMODUS T)
        TRANSMODUS     get_TransModus(edgeindex e);
        list_of_nodes get_follow_nodes(nodeindex n);
        list_of_edges get_follow_edges(nodeindex n);
}
```

Wir erläutern die Operationen dieser Klasse:

input_EDGE
: *input_EDGE* trägt eine Kante mit Index (s,t) und Kanteninformation T in den Kantenpool ein.

get_TransModus
: *get_transModus* ermittelt zu einer Kante mit Index *e* die Kanteninformation.

get_follow_nodes
: *get_follow_nodes* ermittelt zu einem Knoten mit Index *n* die Liste aller Nachfolgeknoten.

get_follow_edges
: *get_follow_edges* ermittelt zu einem Knoten mit Index *n* die Liste aller abgehenden Kanten.

Damit können wir eine Klasse *RGRAPH* definieren:

```
class RGRAPH {
    protected:
        NODES Nodes;
        EDGES Edges;
    public:
        RGRAPH();
        ~RGRAPH();
        nodeindex  insertMarking(MARKING M);
        edgeindex  insertTransModus(nodeindex s,t, TRANSMODUS T);
        TRANSMODUS     get_TransModus(edgeindex e);
        list_of_nodes get_follow_nodes(nodeindex n);
        list_of_edges get_follow_edges(nodeindex n);
        int     generate_SCC();
        int     get_SCC_number(MARKING M);
        int     get_SCC_number(nodeindex n);
        int     count_of_nodes();
        int     count_of_edges();
}
```

In einer vereinfachten Darstellung (siehe Abb. 44) wollen wir *RGRAPH* verdeutlichen.

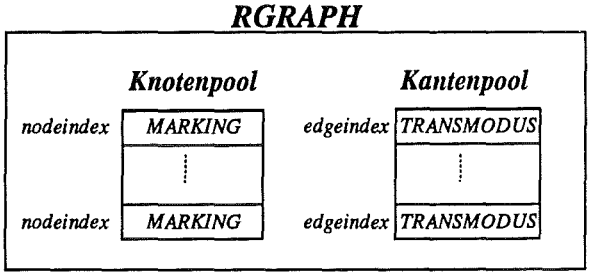


Abb. 44 Die Klasse *RGRAPH*

Damit können wir unsere Klasse *REACHGRAPH* weiter spezifizieren.

```
class REACHGRAPH {
    protected:
        RGRAPH rgraph
    public:
        REACHGRAPH();
        ~REACHGRAPH();
        bool    insertNode(MARKING M);
        void    insertEdge(nodeindex n1, nodeindex n2, TRANSMODUS T);
        int     number_of_SCC();
        int     SCC_number(nodeindex n);
        int     count_of_nodes();
        int     count_of_edges();
        list_of_nodes node_followers(nodeindex n);
        list_of_edges edge_followers(nodeindex n);
}
```

Wir haben damit unsere ursprüngliche Definition von *REACHGRAPH* um eine innere Struktur *rgraph* zur Aufnahme des Graphen erweitert. Es fehlt nun noch ein Hashverfahren zur schnellen Suche eines Knotens.

7.2.3.3 Hashverfahren zur Knotensuche

Für unseren Erreichbarkeitsgraphen ist die schnelle Suche eines Knotens im Erreichbarkeitsgraphen von großer Bedeutung, denn dadurch kann die zum Eintragen eines Knotens benötigte Zeit verkürzt werden. In unserem Fall haben wir eine Markierung M und wollen entscheiden, ob ein Knoten mit Knoteninformation M im Erreichbarkeitsgraph existiert. Wir verwenden für diese schnelle Suche ein Hashverfahren, wie dies z.B. in [Horowitz81] und [Wirth75] beschrieben wird.

Das Prinzip eines Hashverfahrens beruht darauf, daß von der Menge aller möglichen Markierungen eine Klassifizierung gebildet wird. Jede neue Markierung ordnen wir nun ihrer Klasse eindeutig zu und überprüfen, ob in der Klasse die Markierung schon aufgetreten ist. Durch die Reduzierung der Markierungen auf Klassen reduziert sich auch die Anzahl der Vergleiche. Zu dem Hashverfahren benötigen wir eine Hashfunktion, die die Markierungen klassifiziert, und eine Hashtabelle, die die Inhalte der einzelnen Klassen speichert.

Eine Hashfunktion *fHash: MARKING* $\rightarrow$ *HashValues* ist eine Abbildung, die die Menge der Markierungen *MARKING* in eine endliche Menge von Hashwerten *HashValues* abbildet. Das haben wir oben als eine Klassifizierung der Markierungen bezeichnet. In der Hashtabelle *hashtab* haben wir für jeden Hashwert $h \in$ *HashValues* einen Eintrag in Form einer Liste *nodeList*, welche die Referenzen auf die Knoten des Erreichbarkeitsgraphen enthält, die einen gleichen Hashwert *fHash(M)* besitzen. Diese Referenzen sind vom Typ *nodeindex*.

Wollen wir einen neuen Knoten mit Knoteninformation M in den Erreichbarkeitsgraph eintragen, so müssen wir zuerst überprüfen, ob nicht bereits ein Knoten existiert, der M als Knoteninformation besitzt. Dazu bilden wir den zu M gehörenden Hashwert *fHash(M)* und holen in der Hashtabelle die Liste *nodeList* aller Knoten mit demgleichen Hashwert. *nodeList* wird solange sequentiell durchlaufen, bis die Liste abgearbeitet oder die Markierung *M* gefunden ist. Eine Markierung finden heißt dabei, daß von jedem referenzierten Knoten die Knoteninformation ermittelt und mit M verglichen wird. Eine grobe Darstellung einer Hashtabelle zeigt Abb. 45, wobei als Vereinfachung für die Menge *HashValues* das Intervall *[1..maxhval]* betrachtet wird.

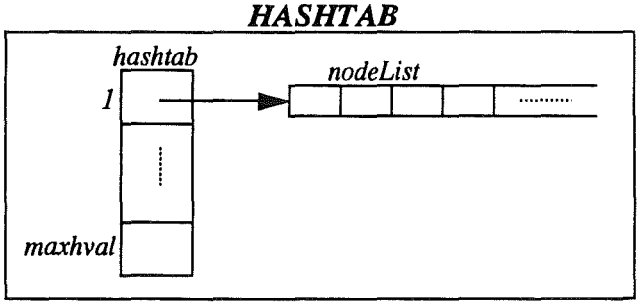


Abb. 45 Darstellung einer Hashtabelle

Ein Hashverfahren, wie in Abb. 45 dargestellt, wird als *Hashverfahren mit offener Adressierung* bezeichnet (siehe z.B. [Wirth75]).Wir integrieren ein Hashverfahren in unsere Klasse *REACHGRAPH* auf die folgende Weise:

```
class REACHGRAPH {
    private:
        HashValues fHASH(MARKING);
    protected:
        RGRAPH rgraph;
        list_of_nodes hashtab[maxhval];
    public:
        REACHGRAPH();
        ~REACHGRAPH();
        bool   insertNode(MARKING M);
        void   insertEdge(nodeindex n1, nodeindex n2, TRANSMODUS T);
        int    number_of_SCC();
        int    SCC_number(nodeindex n);
        int    count_of_nodes();
        int    count_of_edges();
        list_of_nodes node_followers(nodeindex n);
        list_of_edges edge_followers(nodeindex n);
}
```

Wir geben eine graphische Übersicht der Klasse *REACHGRAPH* an.

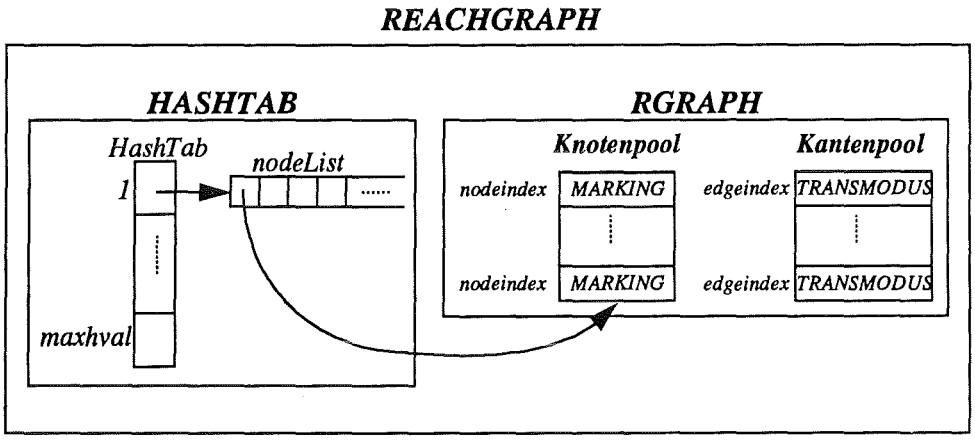


Abb. 46 Die Klasse *REACHGRAPH*

Wir haben unserem Erreichbarkeitsgraphen eine Hashfunktion *fHash* und eine Hashtabelle *hashtab* hinzugefügt. Die Hashfunktion *fHash* ist nur eine interne Funktion der Klasse, jedoch hat *fHash* eine große Bedeutung. Die Hashfunktion muß auf ihrem Wertebereich möglichst gleichverteilt sein, und ihre Berechnung muß mit möglichst geringem Aufwand erfolgen.

7.2.3.4 Implementierung des Datentyps *REACHGRAPH*

Wir haben gesehen, daß *REACHGRAPH* auf eine Klasse *RGRAPH* zurückgreift. *RGRAPH* stellte einen gerichteten Graphen dar. Die Implementierung eines gerichteten Graphen ist ein in der Informatik schon oft gelöstes Problem, weshalb wir auf eine existierende Lösung für die Implementierung eines gerichteten Graphen zurückgreifen wollen.

Wir werden dazu Datentypen benutzen, die in der Bibliothek LEDA (Library of Efficient Data Types and Algorithms) enthalten sind. LEDA ist eine in C++ implementierte Bibliothek und stellt effiziente Datentypen und Algorithmen in Form einer erweiterbaren Bibliothek zur Verfügung. LEDA entstand in einem gleichnamigen Projekt der Universität des Saarlandes. Eine genaue Beschreibung von LEDA bietet [Näher90]. Wir werden zeigen, daß wir mit LEDA sehr einfach die Klasse *RGRAPH* aufbauen können.

Wir beschreiben kurz die Hilfsmittel, die wir aus LEDA verwenden wollen, und zeigen dann, wie wir daraus *RGRAPH* aufbauen können.

LEDA enthält einen Grunddatentyp *graph*, der die Grobstruktur eines gerichteten Graphen darstellt. Eine Instanz G des Datentyps *graph* besteht aus einer Menge von Knoten V und einer Menge von Kanten E. Jede Kante $e \in E$ ist ein Paar von Knoten $(v,w) \in V \times V$, wobei v *source* von e und w *target* von e genannt wird. Zu jedem Knoten v wird die Liste seiner angrenzenden Kanten definiert als $adj\_list(v) = \{e \in E / source(e) = v\}$, d.h. in dieser Liste stehen die von v abgehenden Kanten. V und E beinhalten keinerlei Information über die Knoten und Kanten, sondern stellen lediglich die Struktur des Graphen dar. V und E werden in LEDA mit *node* und *edge* bezeichnet. Die wichtigsten, in LEDA vordefinierten Operationen des Datentyps *graph* sind:

Zugriffsoperationen

int	*G.indeg(node v)*	Anzahl der eingehenden Kanten von v (indegree)
int	*G.outdeg(node v)*	Anzahl der abgehenden Kanten von v (outdegree)
node	*G.source(edge e)*	Source-Knoten der Kante e
node	*G.target(edge e)*	Target-Knoten der Kante e
int	*G.number_of_nodes()*	Anzahl der Knoten in G
int	*G.number_of_edges()*	Anzahl der Kanten in G
list(*node*)	*G.all_nodes()*	Alle Knoten von G
list(*edge*)	*G.all_edges()*	Alle Kanten von G
list(*edge*)	*G.adj_edges(node v)*	Alle abgehenden Kanten von v
list(*node*)	*G.adj_nodes(node v)*	Alle Nachfolgerknoten von v
edge	*G.first_adj_edge(node v)*	Erste Kante von *adj_list(v)*
edge	*G.last_adj_edge(node v)*	Letzte Kante von *adj_list(v)*
edge	*G.adj_succ(edge e)*	Nachfolger von e in *adj_list(source(e))*
edge	*G.adj_pred(edge e)*	Vorgänger von e in *adj_list(source(e))*

Strukturoperationen

node	*G.new_node()*	fügt G einen neuen Knoten hinzu
edge	*G.new_edge(node v,w)*	fügt G eine neue Kante durch Anhängen an *adj_list(v)* hinzu

Man kann an den Operationen *G.new_node* und *G.new_edge* erkennen, daß die Knoten und Kanten keine speziellen Informationen tragen, sondern nur Knoten- und Kantenzähler sind, die beim Generieren eines neuen Knotens bzw. einer neuen Kante erzeugt werden. Da die Knoten und Kanten in unserem Erreichbarkeitsgraphen spezielle Informationen tragen, läßt sich unser Erreichbarkeitsgraph nicht mit dem Datentyp *graph* implementieren.

Es gibt jedoch in LEDA eine Fortsetzung *GRAPH(vtype,etype)* von *graph*, die als parametrisierter Graph bezeichnet wird und die unseren Ansprüchen genügt. *GRAPH(vtype,etype)* basiert auf dem Datentyp *graph*, wobei alle Operationen von *graph* auf *GRAPH(vtype,etype)* vererbt werden. Als Zusatz haben wir in *GRAPH(vtype,etype)* die Möglichkeit, zu den Knoten und Kanten spezielle Informationen vom Typ *vtype* und *etype* anzugeben.

Für eine Instanz G von *GRAPH(vtype,etype)* haben wir in LEDA die folgenden vordefinierten Operationen, wobei die in *graph* definierten Operationen auch in *GRAPH(vtype,etype)* verfügbar sind:

Operationen auf einem GRAPH G

vtype	*G.inf(node v)*	Information des Knotens v
etype	*G.inf(edge e)*	Information der Kante e
void	*G.assign(node v, vtype x)*	Macht x zur Information von v
void	*G.assign(edge e, etype y)*	Macht y zur Information von e
node	*G.new_node(vtype x)*	fügt einen neuen Knoten mit der Information x in G ein
edge	*G.new_edge(node v, w, etype y)*	fügt eine neue Kante mit der Information y in G ein

LEDA beinhaltet außerdem eine Funktion STRONG_COMPONENTS, die die starken Zusammenhangskomponenten eines gerichteten Graphen generiert. STRONG_COMPONENTS ist definiert als:

int *STRONG_COMPONENTS(graph& G, node_array(int)& compnum)*

Die Funktion nimmt einen gerichteten Graph *G* als Argument und berechnet für jeden Knoten *v*∈ *node* eine ganze Zahl *compnum(v)* aus dem Intervall [0, c-1], wobei c die Anzahl der starken Zusammenhangskomponenten von *G* ist. Falls *compnum(v)* = *i* ist, gehört *v* zu der i-ten starken Zusammenhangskomponente. *STRONG_COMPONENTS* gibt als Funktionswert die Anzahl c der starken Zusammenhangskomponenten zurück. Eine Instanz *compnum* von *node_array(int)* ist eine Abbildung von der Knotenmenge *node* des Graphen *G* in die Menge *int*. Mit diesen Hilfsmitteln können wir unseren Datentyp *RGRAPH* sehr leicht generieren, indem wir die folgenden Ersetzungen machen:

- *vtype* wird ersetzt durch *MARKING**,
- *etype* wird ersetzt durch *TRANSMODUS**.

Wir verwenden dabei Pointer auf die Strukturen *MARKING* und *TRANSMODUS*, um nicht unnötig Zeit und Speicherplatz zu verbrauchen. Für die Knoten- und Kantenindizierung, die wir *nodeindex* und *edgeindex* bezeichnet haben, verwenden wir die in LEDA vorgegebenen Indextypen *node* und *edge*.

Eine weitere Möglichkeit, die LEDA bietet, ist die Generierung einer Liste über einem beliebigen Elementtyp. Wir bilden damit für unsere Typen *list_of_nodes* und *list_of_edges* die Typen *list(node)* und *list(edge)*.

Wir erhalten den Datentyp *REACHGRAPH* durch die folgende Implementierung:

```
class REACHGRAPH {
    private:
        HashValues fHASH(MARKING);
    protected:
        GRAPH(MARKING*,TRANSMODUS*)rgraph;
        node_array(int)compnum;
        list(node)     hashtab[maxhval];
    public:
        REACHGRAPH();
        ~REACHGRAPH();
        bool   insertNode(MARKING* M);
        void   insertEdge(node n1, node n2, TRANSMODUS T);
        int    number_of_SCC();
        int    SCC_number(node n);
        int    count_of_nodes();
        int    count_of_edges();
        list(node)     node_followers(node n);
        list(edge)     edge_followers(node n);
}
```

Hierbei verhalten sich die unter *public* angegebenen Funktionen wie zu Beginn des Abschnittes angegeben.

7.2.4 Der Graph der starken Zusammenhangskomponenten

Wie wir schon in vorhergehenden Kapiteln gesehen haben, benötigen wir für die Analyse neben dem Erreichbarkeitsgraph auch den Graph der starken Zusammenhangskomponenten. Wir entwickeln den Graph der starken Zusammenhangskomponenten, indem wir den Erreichbarkeitsgraph vergröbern. Diese Vergröberung entsteht dadurch, daß wir alle Knoten des Erreichbarkeitsgraphen, die wechselseitig erreichbar sind, zu einem Knoten zusammenfassen. Alle Knoten des Erreichbarkeitsgraphen, die wechselseitig erreichbar sind, bilden eine starke Zusammenhangskomponente.

Wir haben den Graphen SCG_N der starken Zusammenhangskomponenten schon in Definition 47 eingeführt. SCG_N ist ein gerichteter Graph mit starken Zusammenhangskomponenten $scc_i \in SCC_N$ als Knoten und Übergängen (scc_i, scc_j) zwischen starken Zusammenhangskomponenten als Kanten.

Im folgenden werden wir einen Datentyp *SCGRAPH* für den Graph der starken Zusammenhangskomponenten spezifizieren und danach eine geeignete Implementierung zu dieser Spezifikation angeben.

7.2.4.1 Spezifikation des Datentyps *SCGRAPH*

Die Grundstruktur des Graphen der starken Zusammenhangskomponenten ist ein gerichteter Graph mit Knoten und Kanten. Dazu benötigen wir wieder Strukturen für die Knoten und die Kanten, die wir wieder *Knotenpool* und *Kantenpool* nennen.

Im Knotenpool unterscheiden wir zwischen Index und Information des Knotens. Als Knotenindex wählen wir zu jedem Knoten die Nummer der starken Zusammenhangskomponente, die er repräsentiert, und als Knoteninformation die Menge der Knoten des Erreichbarkeitsgraphen, die zu der betreffenden starken Zusammenhangskomponente gehören.

Den Knotenpool kann man sich als eine Art Liste vorstellen, in der jedes Listenelement über den Index angesprochen werden kann. Die Indizierung der Knoten geschieht über natürliche Zahlen, die die Nummern der starken Zusammenhangskomponente repräsentieren. Der Inhalt eines Listenelements beinhaltet die Information des Knotens. Diese Knoteninformation ist eine Menge von Knotenindizes, die auf Knoten im Erreichbarkeitsgraphen zeigen. Wir wollen für diese Menge später einen Datentyp definieren, den wir List_of_Nodes nennen werden.

In Abb. 41 wollen wir eine grobe Übersicht über den Knotenpool angeben.

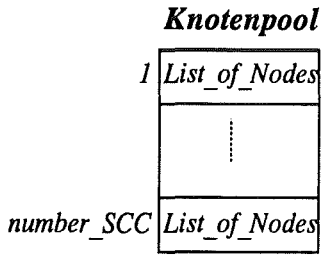


Abb. 47 Der Knotenpool des Graphen der starken Zusammenhangskomponenten

number_SCC ist die Anzahl der starken Zusammenhangskomponenten des Erreichbarkeitsgraphen. Für den Knotenpool spezifizieren wir einen Datentyp *SCNODES* in einer C++ ähnlichen Notation, wobei wir einen Typ *scnodeindex = [1..number_SCC]* für den Index eines Knotens verwenden werden.

```
class SCNODES {
        protected:
             int        number_SCC
             List_of_Nodes*          sccList[number_SCC];
        public:
             SCNODES();
             List_of_Nodes       get_SC_Nodes (scnodeindex i);
             scnodeindex*        get_Leafs ();
};
```

Wir wollen die Funktionen dieser Klasse näher erläutern:

SCNODES

Der Konstruktorfunktion dieser Klasse kommt eine sehr große Bedeutung zu. Durch sie werden die Knoten des Graphen der starken Zusammenhangskomponenten aus dem Erreichbarkeitsgraphen generiert.

get_SC_Nodes

Diese Funktion gibt die Menge der Knotenindizes der Knoten, die zu der starken Zusammenhangskomponente i gehören, zurück.

get_Leafs

Diese Funktion gibt die Liste aller starken Zusammenhangskomponenten, die Blätter des SCG_N sind, zurück.

Auch im Kantenpool unterscheiden wir zwischen Index und Information einer Kante. Eine Kante des Graphen der starken Zusammenhangskomponenten stellt einen Übergang von einer starken Zusammenhangskomponente in eine andere dar. Der Index einer Kante ist damit ein Paar (scc_i,scc_j), wobei scc_i der Index des Anfangsknotens und scc_j der Index des Endknotens der Kante ist. Als Kanteninformation halten wir die Menge derjenigen Kanten des Erreichbarkeitsgraphen, die von einem Knoten aus der starken Zusammenhangskomponente scc_i zu einem Knoten aus der starken Zusammenhangskomponente scc_j führen. Wir werden für die Kanteninformationen später einen Datentyp *List_of_Edges* definieren.

Eine grobe Veranschaulichung des Kantenpools gibt Abb. 42.

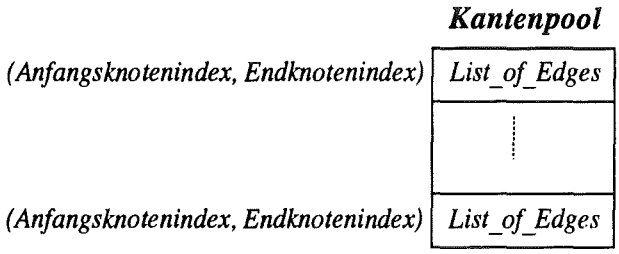


Abb. 48 Der Kantenpool des Graphen der starken Zusammenhangskomponenten

Um den Kantenpool in C++ Notation spezifizieren zu können, müssen wir ihn noch weiter verfeinern. Die Indizierung des Kantenpool zerlegen wir in zwei Listen. Eine Liste, die wir *SOURCELIST* nennen wollen, nimmt die Anfangsknoten einer Kante auf. Sie hat dieselbe Indizierung wie der Knotenpool und beinhaltet einen Eintrag für jeden Knoten des Knotenpools. Jedes Element j dieser *SOURCELIST* hat als Element wieder eine Liste, die wir *FOLLOWERLIST* nennen, in der alle Indizes der Folgeknoten von j sowie alle Kanteninformationen der von j abgehenden Kanten enthalten sind.

Die Abb. 49 soll die Verfeinerung des Kantenpools näher verdeutlichen.

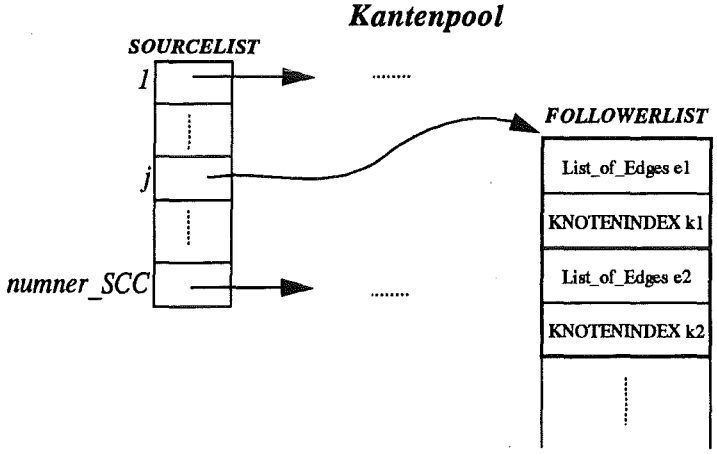


Abb. 49 Eine Verfeinerung des Kantenpools

In unserem Beispiel haben wir zu einem Knoten mit Index j (der starken Zusammenhangskomponente mit Nummer *j*) einen Nachfolgerknoten mit Index k1. Dieser Nachfolgerknoten stellt eine starke Zusammenhangskomponente k1 dar, die Nachfolger der starken Zusammenhangskomponenten j ist. Die Kanteninformation der Kante (j,k1) ist e1. Diese Kanteninformation e1 ist eine Liste mit allen Kanten des Erreichbarkeitsgraphen, die von Knoten der starken Zusammenhangskomponenten j nach Knoten der starken Zusammenhangskomponenten k1 gehen. Als weitere Kante haben wir eine Kante mit Index (j,k2) und Kanteninformation e2 angegeben.

Durch die Verfeinerung des Kantenpools ist es für uns leichter, alle Nachfolgerknoten und alle Nachfolgekanten eines Knotens zu bestimmen. Wir wollen eine Klasse *SCEDGES* für unseren Kantenpool spezifizieren. Wir benutzen dazu Datentypen *scedgeindex=scnodeindex* × *scnodeindex* und *FOLLOWERLIST* als Datentyp für die Nachfolgeknoten und -kanten.

```
class SCEDGES {
        protected:
             int count_of_edges;
             SOURCELIST(FOLLOWERLIST) flist;
        public:
             SCEDGES();
             List_of_Edges      get_all_edges (scedgeindex e);
             FOLLOWERLIST       get_Followers(scnodeindex i)
                                     {return flist(i) };
}
```

Wir erläutern die Operationen dieser Klasse:

SCEDGE

Wir generieren durch die Konstruktorfunktion die Kanten des Graphen der starken Zusammenhangskomponenten aus den Kanten des Erreichbarkeitsgraphen.

get_all_edges

get_all_edges ermittelt zu einer Kante mit Index *e* die Kanteninformation. Diese Kanteninformation stellt gerade alle Kanten des Erreichbarkeitsgraphen dar, die denselben Übergang zwischen starken Zusammenhangskomponenten bewirken.

get_Followers

get_Followers ermittelt zu einem Knoten mit Index i die Liste aller Nachfolgekanten und -knoten. Damit kann man zu einer starken Zusammenhangskomponente die Liste aller Nachfolgekomponenten ermitteln.

Wir können nun eine Klasse *SCGRAPH* definieren, indem wir *SCGRAPH* aus *SCNODES* und *SCEDGES* aufbauen.

```
class SCGRAPH {
        protected:
             REACHGRAPH*    RG;
             SCNODES SCNodes;
             SCEDGES SCEdges;
        public:
             SCGRAPH(REACHGRAPH*);
             ~SCGRAPH();
             int        count_of_nodes();
             int        count_of_edges();
             List_of_Nodes          get_RG_Nodes(scnodeindex n);
             List_of_Nodes          get_leafs();
}
```

Die Klasse *SCGRAPH* beinhaltet Knoten, Kanten, einen Zeiger auf den zugehörigen Erreichbarkeitsgraph und verschiedene Operationen. Die Notwendigkeit von Knoten und Kanten haben wir schon erläutert. Ein Zeiger auf den zugehörigen Erreichbarkeitsgraph ist aus verschiedenen Gründen nötig. Wir generieren unseren Graphen der starken Zusammenhangskomponenten aus dem Erreichbarkeitsgraph und halten als Knoten- bzw. Kanteninformation Indizes auf Knoten und Kanten des Erreichbarkeitsgraphen. Um an die Information dieser Knoten und Kanten des Erreichbarkeitsgraphen zu gelangen, müssen wir in *SCGRAPH* einen Zeiger auf den Erreichbarkeitsgraphen halten.

Der Konstruktorfunktion *SCGRAPH* kommt eine besondere Bedeutung zu. In ihr wird der Graph der starken Zusammenhangskomponenten aus dem Erreichbarkeitsgraph generiert.

In einer vereinfachten Darstellung wollen wir *SCGRAPH* verdeutlichen.

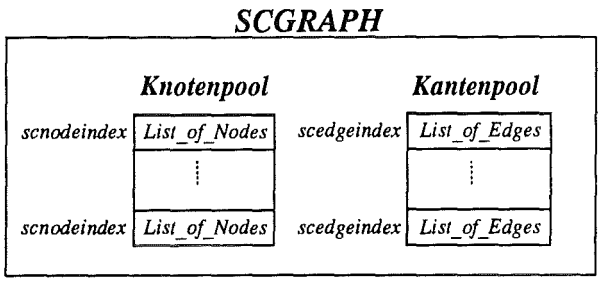


Abb. 50 Die Klasse *SCGRAPH*

7.2.4.2 Implementierung des Datentyps *SCGRAPH*

Wir erzeugen uns einen Datentyp *SCGRAPH* für einen Graphen der starken Zusammenhangskomponenten wieder dadurch, daß wir den Datentyp *GRAPH(vtype,etype)* aus der Bibliothek LEDA geeignet parametrisieren. Wir verwenden für die Indextypen *scnodeindex* und *scedgeindex* wieder die in LEDA vordefinierten Typen *node* und *edge*. *vtype* und *etype* ersetzen wir durch Listen von Knoten- und Kantenindizes *List_of_Nodes* und *List_of_Edges*.

Diese Typen bilden wir durch den in LEDA vordefinierten Typ einer parametrisierten Liste, aus dem wir zwei Typen *list(node)* und *list(edge)* bilden. *list(node)* beinhaltet die Indizes derjenigen Knoten des Erreichbarkeitsgraphen, die in der starken Zusammenhangskomponente liegen. *list(edge)* beinhaltet die Indizes von allen Kanten des Erreichbarkeitsgraphen, die von einer bestimmten starken Zusammenhangskomponente in eine andere führen.

Eine Implementierung der Klasse *SCGRAPH* hat die Gestalt:

```
class SCGRAPH {
        protected:
             int number_SCC;
             node* convert;
             GRAPH(nodelist_ptr, edgelist_ptr) scgraph;
             REACHGRAPH*   RG;
        public:
             SCGRAPH(REACHGRAPH*);
             int       count_of_nodes();
             int       count_of_edges();
             list(node)     get_RG_Nodes(scnodeindex n);
             list(node)     get_leafs();
}
```

Im Gegensatz zur Spezifikation sehen wir bei der Implementierung keine Datenstrukturen für die Knoten und Kanten. Wir unterscheiden natürlich auch bei der Implementierung zwischen Knoten und Kanten, jedoch wird diese Unterscheidung durch den vordefinierten Typ *GRAPH* verdeckt.

Zur Indizierung der Knoten verwendet LEDA eine bestimmte, nicht veränderbare Indizierung. Da wir auf einen Knoten des Graphen der starken Zusammenhangskomponenten auch über seine starke Zusammenhangskomponentennummer zugreifen wollen, haben wir zusätzlich ein Array *convert* eingeführt. Durch *convert* wird einer Integerzahl, die die Nummer einer starken Zusammenhangskomponente darstellt, der entsprechende Knoten des Graphen zugeordnet.

7.2.5 Beispiele

Wir wollen uns nun anhand des Erzeuger-Verbraucher-Modells und des dazugehörigen Erreichbarkeitsgraphen die Speicherung des Erreichbarkeitsgraphen verdeutlichen. Wir wiederholen dazu hier noch einmal die beiden Darstellungen.

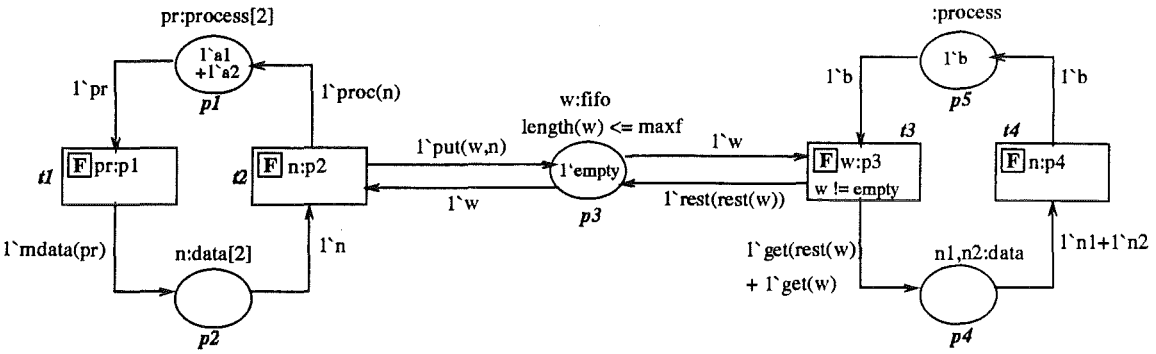


Abb. 51 Erzeuger-Verbraucher-Modell

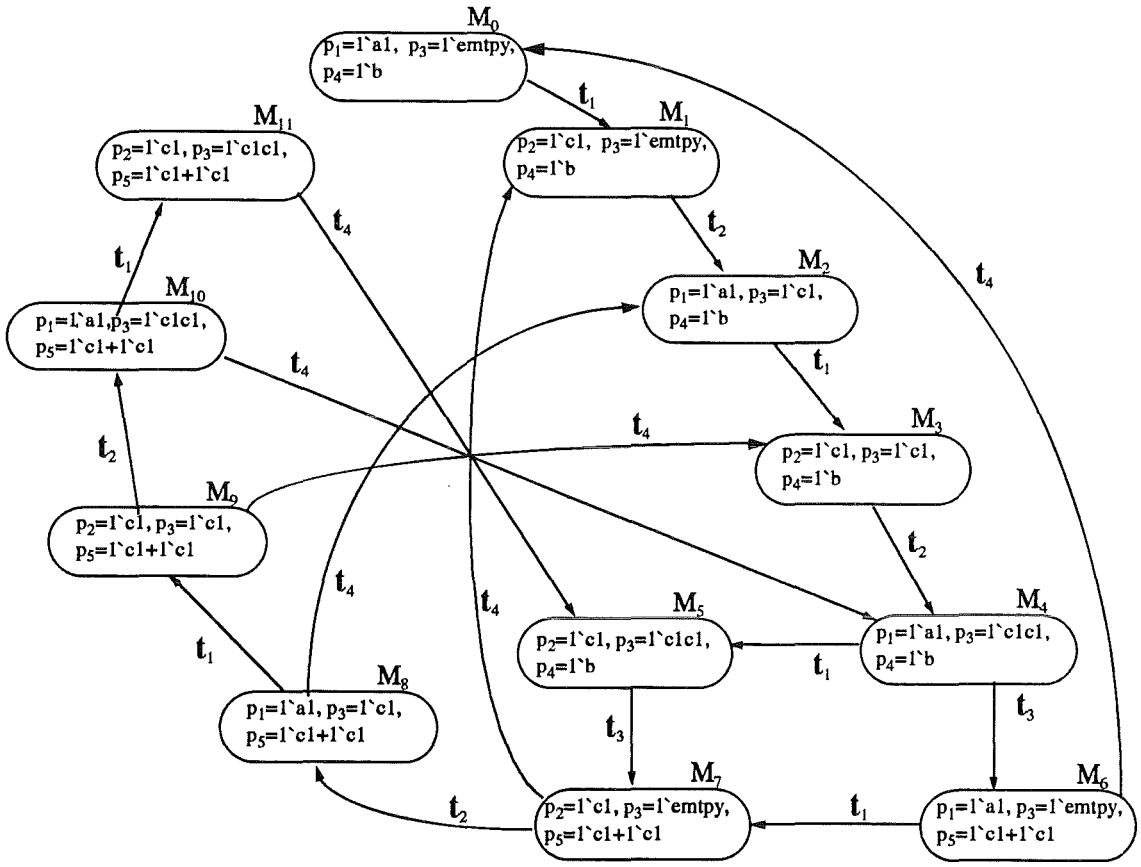


Abb. 52 Der Erreichbarkeitsgraph des Erzeuger-Verbraucher-Modells

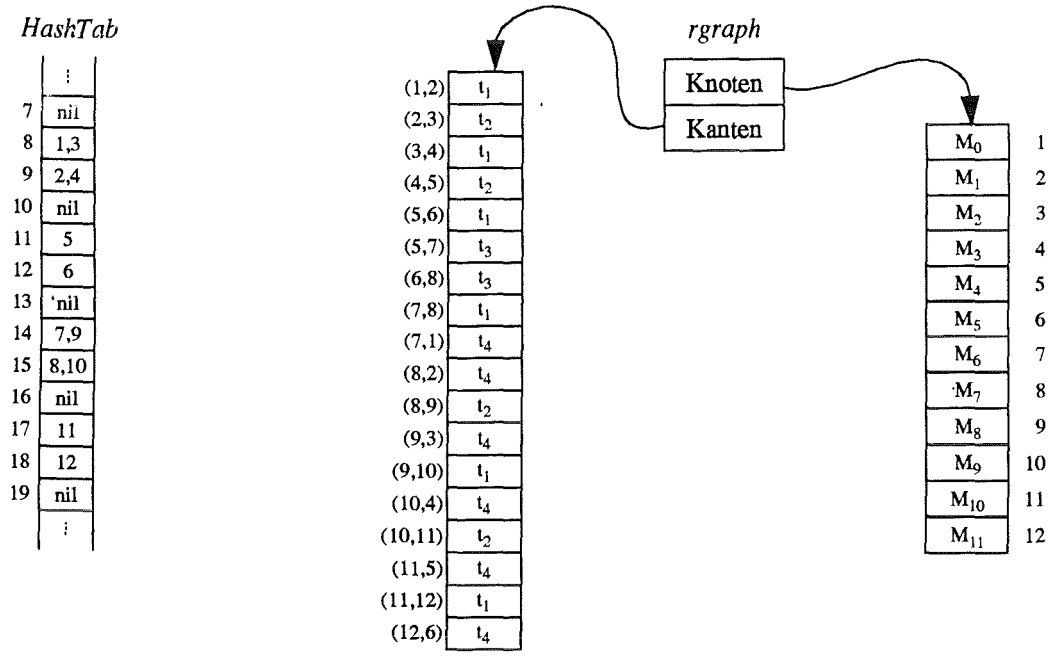


Abb. 53 Beispiel der Speicherung eines Erreichbarkeitsgraphen

Die verwendete Hashfunktion berechnet den Hashwert einer Markierung M nach der folgenden Formel:

$$Hashwert(M) = \sum_{StellenNummer=1}^{AnzahlderStellen} StellenNummer \times AnzahlderToken$$

Die Stellen sind dabei von 1 bis zur Gesamtanzahl aller Stellen durchnumeriert. Mit *AnzahlderToken* wird die Anzahl der Token, die auf der Stelle mit der Nummer *StellenNummer* liegen, gezählt, wobei wir in unserem Beispiel die Elemente der Warteschlange einzeln zählen.

Wir haben nur eine starke Zusammenhangskomponente, da alle Markierungen des Systems wechselseitig erreichbar sind. Deshalb ist es bei diesem Beispiel nicht sinnvoll, den Graph der starken Zusammenhangskomponenten zu bilden, da dieser nur einen Knoten und keine Kante hätte. Wir geben deshalb ein modifiziertes System an, dessen Erreichbarkeitsgraph mehrere starke Zusammenhangskomponenten enthält. Dieses System hat keine sinnvolle semantische Bedeutung, sondern dient nur zu Demonstrationszwecken.

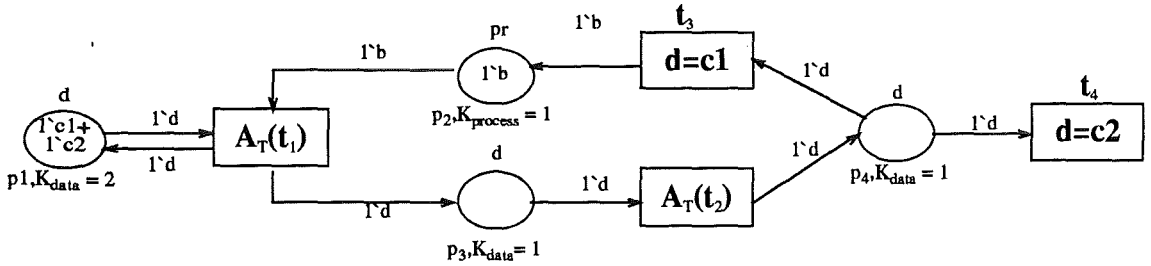


Abb. 54 Ein Beispielnetz

Abb. 54 beschreibt ein Netz mit einen kleinen Erreichbarkeitsgraph und mehreren starken Zusammenhangskomponenten. Der Erreichbarkeitsgraph besteht aus 6 Knoten, wobei die Markierungssituation M_3 ein Deadlock darstellt (siehe Abb. 55).

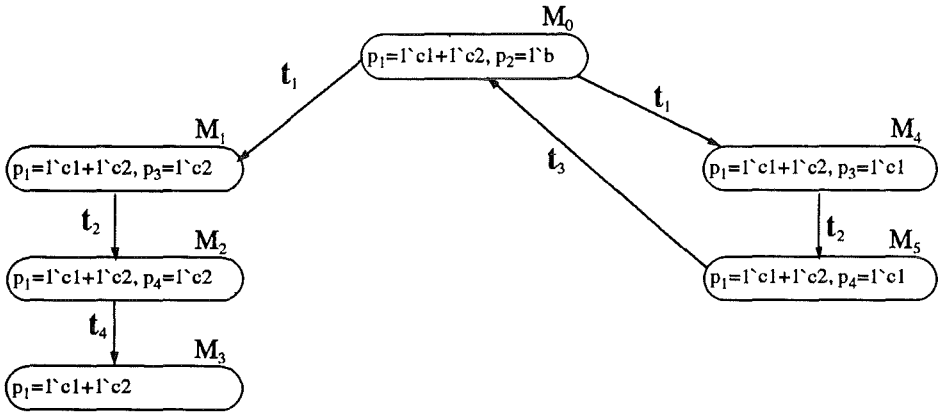


Abb. 55 Der Erreichbarkeitsgraph zu Abb. 54

Die Speicherung dieses Erreichbarkeitsgraphen hat die in Abb. 56 gezeigte Form.

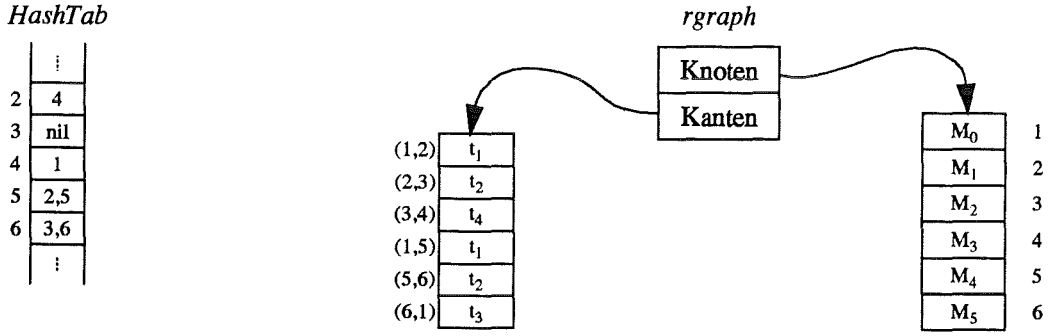


Abb. 56 Die Speicherung des Erreichbarkeitsgraphen aus Abb. 55

Dieser Erreichbarkeitsgraph enthält vier starke Zusammenhangskomponenten. Der zugehörige Graph der starken Zusammenhangskomponenten ist in Abb. 57 dargestellt. Die starken Zusammenhangskomponenten sind mit SCC_0, SCC_1, SCC_2 und SCC_3 bezeichnet, wobei in jeder starken Zusammenhangskomponente die zu ihr gehörenden Knoten des Erreichbarkeitsgraphen angegeben sind.

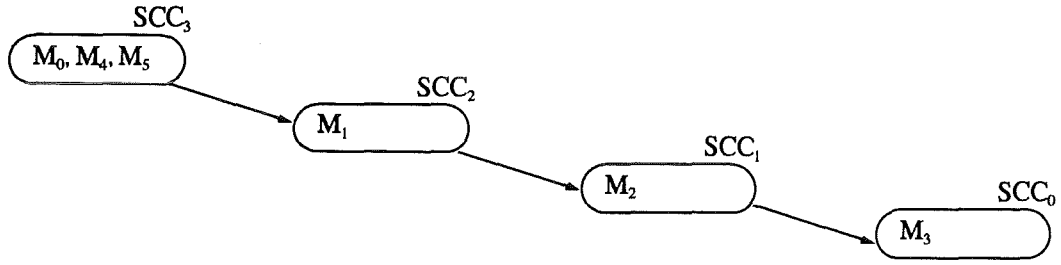


Abb. 57 Der Graph der starken Zusammenhangskomponenten zu Abb. 54

Die Speicherung des Graphen der starken Zusammenhangskomponenten sieht fast genauso aus wie die Speicherung des Erreichbarkeitsgraphen (siehe Abb. 58).

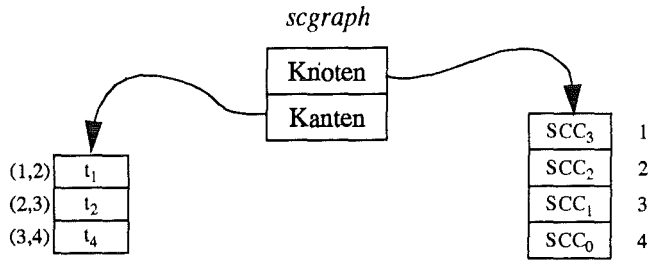


Abb. 58 Die Speicherung des Graphen der starken Zusammenhangskomponenten aus Abb. 57

Der Unterschied zu der Speicherung eines Erreichbarkeitsgraphen besteht darin, daß wir keine Hashtabelle benötigen und daß sich hinter SCC_i ein Verweis auf die Knoten des Erreichbarkeitsgraphen verbirgt, die in der starken Zusammenhangskomponente mit Nummer i liegen.

8. Vergleich mit anderen Petri-Netz-Werkzeugen

Wir wollen unser Petri-Netz-Werkzeug SMARAGD mit anderen Petri-Netz-Werkzeugen vergleichen. Für diesen Vergleich benötigen wir gewisse Kriterien, nach denen wir die Werkzeuge bewerten. In dieser Arbeit werden wir Petri-Netz-Werkzeuge nach ihren Analysefähigkeiten (einschließlich des Ausgabekomforts) bewerten. Die Bewertung der zugrundeliegenden Sprache und des Editors werden in [Düpmeier92] gemacht.

Grundlagen der Bewertung der verschiedenen Werkzeuge sind

- verschiedene allgemeine Überblicke über existierende Petri-Netz-Werkzeuge wie z.B. das Buch "Petri-Netz-Methoden und -Werkzeuge" von M. Leszak und H. Eggert ([Leszak88]) oder der Artikel "Petri Net Tool Overview 1989" von F. Feldbrugge ([Feldbrugge89]).
- Publikationen über einzelne Petri-Netz-Werkzeuge.
- Produktbeschreibungen der Herstellen von Petri-Netz-Werkzeugen.
- Persönliche Erfahrung mit einigen Petri-Netz-Werkzeugen.

Wie auch schon Starke ([Starke90]) festgestellt hat, ist die Beurteilung von Petri-Netz-Werkzeugen mit Vorsicht zu lesen, besonders wenn diese Beurteilung von jemandem gemacht ist, der selbst ein Petri-Netz-Werkzeug entwickelt hat. Unter diesem Aspekt ist auch diese Beurteilung zu lesen, wobei der Leser gebeten wird, eine eventuell zu subjektive Beurteilung zu entschuldigen. Wir wollen trotzdem nicht auf einen Vergleich mit anderen Werkzeugen verzichten, da eine Einordnung von SMARAGD in die große Konkurrenz der Petri-Netz-Werkzeuge, und sei sie auch noch so subjektiv, wichtig erscheint.

Es bleibt die Frage, nach welchen Kriterien wir die Werkzeuge bewerten. Jensen beschreibt in [Jensen87a] einige Anforderungen an ein Petri-Netz-Tool, aber auch bei Feldbrugge ([Feldbrugge89]) und Leszak/Eggert ([Leszak88]) werden Kriterien für eine Klassifizierung von Petri-Netzen angegeben. Daraus können wir Bewertungskriterien, nach denen wir in dieser Arbeit Petri-Netz-Werkzeuge beurteilen wollen, ableiten. Wir unterscheiden zwischen fünf Hauptpunkten:

- Simulation
- Erreichbarkeitsanalyse
- Invariantenanalyse
- Analyseunterstützung
- Bedienungsfreundlichkeit und Ausgabekomfort.

Im Simulationsteil eines Petri-Netz-Werkzeuges unterscheiden wir zwischen interaktiver oder automatischer Simulation. Bei der interaktiven Simulation kann der Benutzer selektieren, welche Transitionen bzw. Schritte schalten sollen. Bei der automatischen Simulation werden die Transitionen, die schalten, vom System ausgewählt. Als minimale Anforderung an die Simulation sollte sowohl bei der interaktiven als auch bei der automatischen Simulation der Benutzer jederzeit die Möglichkeit haben, die Simulation zu stoppen. Außerdem sollte es möglich sein, nach dem Stoppen der Simulation Schaltvorgänge rückgängig machen zu können, damit der Benutzer sehen kann, wie eine Markierung erreicht wurde.

Grundlage der Erreichbarkeitsanalyse ist der Erreichbarkeitsgraph. Die erste Frage ist deshalb, ob der Erreichbarkeitsgraph vollständig aufgebaut wird. Weiter ist die Handhabbarkeit des Erreichbarkeitsgraphen interessant. Darunter verstehen wir die Punkte:

- Welche Eigenschaften des Netzes werden aus dem Erreichbarkeitsgraph ermittelt?
- Wie sind die Visualisierungsmöglichkeiten des Erreichbarkeitsgraphen und der Eigenschaften des Netzes?
- Gibt es Reduktionsmöglichkeiten für den Erreichbarkeitsgraphen?

Ein weiterer, sehr wichtiger Punkt wäre ein Vergleich des Zeitaufwandes für die Berechnung und des Aufwandes für die Speicherung des Erreichbarkeitsgraphen. Ein solcher Vergleich kann nicht durchgeführt werden, weil nur wenige Werkzeuge überhaupt eine Erreichbarkeitsanalyse durchführen und dabei auch noch von völlig unterschiedlichen Netzmodellen ausgehen. Außerdem haben wir nicht auf alle Werkzeuge Zugriff, so daß wir auch keine Messungen durchführen könnten.

Bei der Invariantenanalyse unterscheiden wir, ob ein Werkzeug Invarianten berechnen oder verifizieren kann. Interessant ist dabei, welche Invarianten bearbeitet werden können und in welcher Form.

Die Idee der Analyseunterstützung ist noch nicht weit verbreitet. Insbesondere ist uns kein Werkzeug bekannt, in dem eine solche Analyseunterstützung existiert. Wir nehmen deshalb als Bewertungskriterium allein schon die Idee, eine solche regelbasierte Analyseunterstützung zu schaffen, auf.

Keine Analysefähigkeit, aber für die Benutzung eines Werkzeuges sehr wichtig, ist die Frage des Ausgabekomforts. Die Anforderungen an eine gute Ausgabe kann man eigentlich nur aus einer sehr subjektiven Sicht stellen. So wird jeder Benutzer eine andere Vorstellung einer guten Ausgabe haben. Die Beurteilung des Ausgabekomforts wird deshalb aus unserer subjektiven Sicht erfolgen. Sinnvoll erscheint eine Ausgabe, die graphisch unterstützt wird, so daß der Benutzer sehr schnell einen Überblick über die gelieferte Information erhält. Bei Bedarf sollte er dann die Möglichkeit haben, detailliertere Informationen zu erhalten. Ebenfalls sehr subjektiv ist die Bewertung der Bedienungsfreundlichkeit. Man kann nicht allgemein sagen, daß eine reine Bedienung über die Maus besser ist als eine Bedienung über die Tastatur, wenngleich die meisten Benutzer eine mausorientierte Benutzung vorziehen. Ideal ist deshalb, wenn ein Werkzeug beide Möglichkeiten anbietet.

Wir wollen nun von verschiedenen Werkzeugen die Eigenschaften herausarbeiten. Wir beginnen mit den Werkzeugen Design/CPN, das von der Firma Meta Software Corporation entwickelt wurde, und dem am IAI existierenden Werkzeug PROVER, weil diese Werkzeuge SMARAGD am meisten beeinflußt haben. Die weiteren Werkzeuge sind in alphabetischer Reihenfolge angeordnet.

8.1 Design/CPN

Wir beschreiben hier die Version 1.5 von Design/CPN. Design/CPN ist ein Werkzeug für sogenannte "Coloured Petri Netze (CP-Netze)". CP-Netze sind eine Form von Höheren Petri-Netzen. Sie werden z.B. in [Jensen85, Jensen87a und Jensen87b] beschrieben. Die Beschreibung dieses Werkzeuges basiert auf eigenen praktischen Erfahrungen und der Kurzbeschreibung in [Feldbrugge89].

Ein der wichtigsten Eigenschaften von Design/CPN ist die Möglichkeit der Hierarchisierung von CP-Netzen. Diese sogenannten hierarchischen CP-Netze sind CP-Netze, die eine Menge von Subnetze enthalten. Solche Subnetze werden in Design/CPN Seiten genannt. Der Benutzer kann Hierarchien durch einen Top-Down-Entwurf konstruieren, indem er neue Seiten bildet, oder durch einen Bottom-Up-Entwurf, indem er existierende Teilnetze verwendet. Design/CPN enthält einen Editor, in dem solche hierarchischen CP-Netze entwickelt werden können. Für die Benutzung von Design/CPN ist eine graphikfähige Workstation mit einem hochauflösenden Bildschirm und einer Maus nötig. Bei der Bearbeitung von CP-Netzen mit dem Design/CPN Editor kann der Benutzer jede Editieroperation direkt am Bildschirm, der immer ein exaktes Abbild des bearbeiteten Netzes enthält, verfolgen.

Der Design/CPN Editor kennt die Strukturen eines CP-Netzes. Nicht erlaubte Operationen auf einem CP-Netz, wie z.B. eine Kante zwischen zwei Transitionen, werden vom Editor verweigert. Bewegt oder entfernt der Benutzer eine Stelle oder eine Transition, so werden auch alle dazugehörigen Kanten mitbewegt oder entfernt. Durch eine große Anzahl von verschiedenen Attributen (Form, Größe, Farbe, Hintergrund, Schrifttyp etc.) kann jedes Objekt eines CP-Netzes (Stellen, Kanten, Transitionen, Markierungen, Beschriftungen etc.) individuell gestaltet werden.

Die Beschriftungen eines CP-Netzes gibt der Benutzer in der Sprache CPN ML an. CPN ML ist eine Erweiterung der funktionalen Programmiersprache Standard ML (SML), die an der Universität Edinburgh entwickelt wurde (siehe z. B. [Harper90 und ML90]). Der Editor hat eine große Anzahl von optionalen Syntaxüberprüfungen, ähnlich denen einer höheren Programmiersprache. Diese Syntaxüberprüfungen sind vor allem für die Simulation wichtig. Eine genauere Beschreibung dieses Editors erfolgt in [Düpmeier92].

Der Design/CPN Simulator erlaubt es dem Benutzer die mit dem Design/CPN Editor erstellten hierarchischen CP-Netze zu simulieren. Die internen Algorithmen für die Berechnung der Aktivierbarkeit und das Schalten von Transitionen sind ebenfalls in CPN ML geschrieben. Ein genauere Beschreibung des Design/CPN Simulators erfolgt weiter unten.

Nach dieser kurzen Einführung werden wir nun das Werkzeug nach den fünf Hauptpunkten Simulation, Erreichbarkeitsanalyse, Invariantenanalyse, Analyseunterstützung sowie Bedienungsfreundlichkeit und Ausgabekomfort beurteilen.

Simulation

Design/CPN hat eine sehr mächtige Simulationskomponente. Der Benutzer kann nach dem Erstellen eines Netzes vom Editor in den Simulator überwechseln. Das System führt dazu automatisch eine Syntaxüberprüfung des Netzes und der Beschriftung durch. Wurde kein Sytnaxfehler gefunden, befindet sich der Benutzer nach der Syntaxüberprüfung im Simulator, in dem sowohl interaktive als auch automatische Simulation durchgeführt werden kann. Für die Auswertung der Netzbeschriftung benutzt Design/CPN die Sprache CPN ML (siehe dazu auch [Harper86, Harper89, Harper90, MacQueen85, ML90]).

Die Simulation kann manuell (indem jeder Schritt durch den Benutzer angegeben wird) oder automatisch (indem jeder Schritt zufällig durch das System ausgewählt wird) ausgeführt werden. In beiden Fällen kann der Benutzer das Verhalten der Input- und Outputtoken am Bildschirm verfolgen, auf dem jede Seite eines CPN-Netzes durch ein eigenes Fenster dargestellt wird. Der Simulator erlaubt Schalt- und Schrittvorgänge.

Im manuellen (interaktiven) Modus hat der Benutzer volle Kontrolle über alle Simulationsschritte. Der Benutzer kann sich jedoch vom System unterstützen lassen. Er kann z.B. alle möglichen Variablensubstitutionen für eine bestimmte Transition berechnen lassen und dann eine gewünschte Variablensubstitution auswählen, mit der die Transition schalten soll. Ein weitere Möglichkeit ist alle in einer Markierung aktivierbaren Schritte zu bestimmen und dann den Schritt, der ausgeführt werden soll, auszuwählen.

Bei der automatischen Simulation kann der Benutzer "Breakpunkte" setzen, an denen die Simulation stoppt und nachfragt, ob die Simulation beendet oder fortgesetzt werden soll.

Die Veränderung eines Netzes ist im Simulator nicht möglich, sondern der Benutzer muß zum Ändern eines Netzes in den Editor gehen, dort die Änderung durchführen und wieder zurück in den Simulator wechseln.

Erreichbarkeitsanalyse

Eine Erreichbarkeitsanalyse ist in der Version 1.5 von Design/CPN noch nicht möglich, jedoch für spätere Versionen geplant.

Invariantenanalyse

Auch eine Invariantenanalyse ist erst für spätere Versionen von Design/CPN vorgesehen.

Analyseunterstützung

Eine regelbasierte Auswertung von Analyseergebnissen ist auch für spätere Versionen von Design/CPN nicht vorgesehen.

Bedienungsfreundlichkeit und Ausgabekomfort

Wie schon weiter oben bemerkt, ist für die Benutzung von Design/CPN eine graphikfähige Workstation mit einem hochauflösenden Bildschirm und einer Maus nötig. Die Bearbeitung von CP-Netzen mit dem Design/CPN Editor erfolgt über Menüs, die mit der Maus angewählt werden können. Jede Editieroperation kann der Benutzer direkt am Bildschirm verfolgen. Ein geübter Benutzer kann mit Design/CPN sehr schnell komplexe Modelle erstellen. Der ungeübte Benutzer muß zuerst die Sprache ML erlernen, damit er Netze beschriften kann. Design/CPN hat eine Vielzahl von Möglichkeiten, um Modellen eine individuelle Form zu geben.

Die Steuerung der Simulation erfolgt über die Maus und ist leicht erlern- und ausführbar. Die Ausgabe der Simulationsergebnisse erfolgt graphisch, wobei ein neuer Zustand so dargestellt wird, wie er auch vom Benutzer eingegeben worden wäre. Das Tokenspiel beim Schalten eines Schrittes wird vom System übersichtlich dargestellt, so daß der Benutzer die Tokenbewegung sehr leicht nachvollziehen kann.

8.2 PROVER

Am IAI existiert schon seit längerer Zeit das Petri-Netz-Werkzeug PROVER (**P**redicate/**T**ransition net **o**riented **V**erification system). PROVER wird z.B. in [Leszak88] und [Düpmeier89c] beschrieben. Wir wollen PROVER ganz kurz vorstellen und benutzen dazu einige Darstellungen und Formulierungen aus [Leszak88].

Das Petri-Netz-Werkzeug PROVER besteht aus drei kooperierenden Moduln (vergleiche Abb. 59):

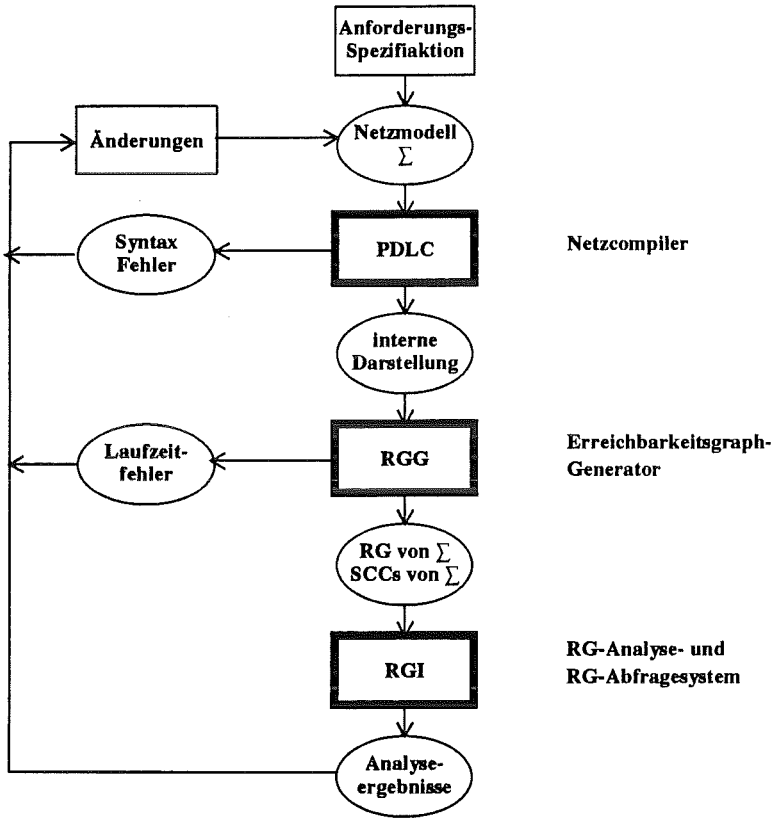


Abb. 59 Interne Struktur des Petri-Netz-Werkzeuges PROVER

(1) Der Pr/T-Netz-Compiler PDLC übersetzt ein in der PDL (predicate/transition net description language) erstelltes Netzmodell in eine interne Darstellung. Syntaktische und semantische Modellfehler werden erkannt.

(2) Der Erreichbarkeitsgraph-Generator RGG erzeugt daraus den vollständigen Erreichbarkeitsgraph.

(3) Der Erreichbarkeitsgraph-Interpreter RGI führt eine Vielzahl unterschiedlicher Verfahren durch, die alle auf graphen-theoretischer Erreichbarkeitsanalyse basieren.

Simulation

Eine Simulation ist bei PROVER nicht möglich.

Erreichbarkeitsanalyse

PROVER hat eine sehr gute Erreichbarkeitsanalyse mit z.B. folgenden Verfahren:

- Erkennung statischer und dynamischer Verklemmungen
- Erkennung toter, (un-)beschränkt oft aktivierbarer, (nicht-)lebendiger und (un)fairer Systemteile.
- Erkennung von Konflikt- und Kapazitätsüberlauf-Situationen
- Ermittlung minimaler, ggf. auch zyklischer Aktivierungsfolgen von Transitionen

Invariantenanalyse

PROVER unterstützt keine Invariantenanalyse.

Analyseunterstützung

Auch eine regelbasierte Auswertung von Analyseergebnissen ist bei PROVER nicht vorhanden.

Bedienungsfreundlichkeit und Ausgabekomfort

PROVER hat eine rein textuelle Benutzerschnittstelle. Sämtliche Ein- und Ausgaben von PROVER erfolgen textuell. Netze müssen in der Sprache PDL eingegeben werden. PDL ist von der Struktur her mit höheren Programmiersprachen wie z.B. PASCAL zu vergleichen. Da kein graphischer Editor für Netze existiert, muß der Benutzer seine Netze auf Papier erstellen und von Hand in PDL übertragen. Auch die Bedienung des Erreichbarkeitsgraph-Interpreters RGI erfolgt rein über die Tastatur. Die Abfragesprache des RGI ist jedoch so aufgebaut, daß man sehr schnell und auf vielfältige Weise an die Analyseergebnisse gelangt. Die Aufbereitung der Analyseergebisse leidet jedoch sehr stark darunter, daß die Anzeige der Analyseergebnisse textuell ist. Man erhält so sehr viele Informationen, die sehr schwer erfaß- und verarbeitbar sind.

8.3 GRASPIN

GRASPIN[1] war eines der ersten ESPRIT[2]-Projekte im Bereich Software Technologie. Das Ergebnis von GRASPIN waren zwei Prototypen einer Softwareentwicklungsumgebung, die die Konstruktion und Verifikation von verteilten, nicht sequentiellen Softwaresystemen unterstützt. Der auf Unix basierende Prototyp wurde von den italienischen Partnern des Projektes in der Sprache C implementiert. Der auf Lisp basierende Prototyp wurde von den deutschen Partnern entwickelt.

Wir stützen uns bei der kurzen Beschreibung von GRASPIN auf die in [GRASPIN90] vorgestellten Ergebnisse des Projektes.

Ähnlich wie bei unserem Ansatz wurde im GRASPIN-Projekt eine Spezifikationssprache (*SEGRAS*) entwickelt, die auf der Kombination von algebraischen Spezifikationen und Höheren Petri-Netzen basiert. Als Petri-Netze werden dabei sogeannnte PrE (Predicate/Event) - Netze verwendet. In [Düpmeier92] werden diese *SEGRAS*-Netze näher beschrieben.

Simulation

Die Simulation von *SEGRAS*-Netzen ist möglich. Ähnlich wie SMARAGD die Sprache ML für das Parsen und Analysieren der algebraischen Spezifikationen verwendet, verwendet GRASPIN das im Rahmen des Projektes entwickelte RRLab (Rewrite Rule Laboratory) für diesen Zweck.

Erreichbarkeitsanalyse

In [GRASPIN90] wird beschrieben, daß die Analyse von GRASPIN ein SEGRAS-Netz auf Lebendigkeit und Sicherheit untersucht. Diese Analyse ist jedoch keine Erreichbarkeitsanalyse in unserem Sinne, da kein Erreichbarkeitsgraph aufgebaut wird. Diese Analyseergebnisse werden aus der Simulation und mit "Rewrite"-Techniken gewonnen. Deshalb sind die Aussagen über die Lebendigkeit und Sicherheit des Netzes nicht vollständig abgesichert.

Invariantenanalyse

GRASPIN unterstützt keine Invariantenanalyse.

Analyseunterstützung

Auch ein regelbasierte Auswertung von Analyseergebnissen ist bei GRASPIN nicht vorhanden.

---

1. Personal Environment for Incremental Graphical Specification and Formal Implementation of Non-Sequential Systems
2. European Strategic Programs for Research and Development in Information Technology

Bedienungsfreundlichkeit und Ausgabekomfort

Da wir GRASPIN nicht aus der praktischen Erfahrung heraus kennen, können wir keine Aussagen über Bedienungsfreundlichkeit und Ausgabekomfort machen.

8.4 NET

Wir beschreiben in diesem Abschnitt das von der Firma PSI, Berlin entwickelte Netztool NET. Die Beschreibung erfolgt auf Grund von Produktbeschreibungen der Firma PSI (siehe [PSI89]). NET besteht aus den drei Teilen NET-Editor (NED), NET-Analysator (PAN) und NET-Simulator (SIM). Der Editor und Simulator erlaubt gefärbte Petri-Netze (Coloured Petri Nets), während der Analysator das Netz als Stellen/Transitions-Netz betrachtet, d.h. Ein- und Ausgangsbedingungen, Zeitbeschriftungen und Unterschiede zwischen den Marken ("Farben") werden nicht berücksichtigt. Hat der Benutzer ein System nicht als ein Stellen/Transitions-Netz, sondern als farbiges Netz modelliert, so muß er die Analyseergebnisse von dem Stellen/Transitions-Netz auf das gefärbte Netz übertragen und interpretieren.

Für die Untersuchung eines Systemmodells empfiehlt die Firma PSI folgende zweistufige Vorgehensweise: zuerst soviel Analyse wie möglich, um allgemeine Eigenschaften des Modells festzustellen, dann soviel Simulation wie nötig, um die Eigenschaften zu klären, für die es bisher noch keine Analysemethoden gibt.

Diese Vorgehensweise wird empfohlen, weil die Analyse nur Hinweise auf die Eigenschaften eines Netzes geben kann, da nicht die höheren Netze, sondern die zugehörigen Stellen/Transitions-Netze analysiert werden. Hat man gewisse Eigenschaften der Stellen/Transitions-Netze gefunden, so muß der Benutzer durch Simulation selbst herausfinden, ob entsprechende Eigenschaften auch für die höheren Netze gelten. Z.B. kann er im Falle eines Deadlocks in dem Stellen/Transitions-Netz durch Simulation des höheren Netzes eventuell auch einen Deadlock im höheren Netz finden.

Simulation

Der NET-Simulator SIM, der auch gefärbte Netze simuliert, hat vielfältige Möglichkeiten. Es wird sowohl interaktive als auch automatische Simulation unterstützt, die jederzeit über ein Interruptfunktion unterbrochen werden kann. Er benutzt eine Simulationssteuerdatei, in der die Definition von Monitoren, die Aktivierung von Interruptfunktionen, die Anfangsmarkierung, Protokollangaben, Statistikangaben und Abbruchbedingungen angegeben werden können. SIM besitzt sogenannte Statistik- und Snapshotfunktionen. Mit den Statistikfunktionen können verschiedene Daten während einer Simulation mitgeschnitten werden. Die Snapshotfunktion erlaubt es, einen während einer Simulation angenommenen Zustand "einzufrieren". Dieser eingefrorene Zustand kann dann später genauer untersucht oder eine neue Simulation von diesem Zustand aus gestartet werden.

Laut [PSI89] kann die Markierung eines Netzes und auch das Netz selbst im Simulator verändert werden.

Erreichbarkeitsanalyse

PAN kann keine gefärbten Netze analysieren, sondern ein höheres Netz wird als Stellen/Transitions-Netz betrachtet und dann analysiert. Wie schon weiter oben angesprochen, müssen die Analyseergebnisse dieses Stellen/Transitions-Netzes dann vom Benutzer auf das höhere Netz übertragen werden. Übertragen werden solche Analyseergebnisse des Stellen/Transitions-Netzes, indem der Benutzer durch Simulation versucht, entsprechende Eigenschaften auch für das höhere Netz nachzuweisen. Es ist offensichtlich, daß damit im allgemeinen das höhere Netz nicht vollständig analysiert werden kann.

Die Analyse für die Stellen/Transitions-Netze ist sehr mächtig. Es können verschiedene Beschränktheits-, Erreichbarkeits- und Lebendigkeitseigenschaften ermittelt werden.

Invariantenanalyse

NET erlaubt Invariantenanalyse unter derselben Einschränkung wie bei der Erreichbarkeitsanalyse, nämlich daß Invarianten nicht von einem höheren Netz, sondern von dem zugehörigen Stellen/Transitions-Netz berechnet werden. NET bestimmt sowohl S- als auch T-Invarianten. Außerdem erlaubt NET auch die Verifikation von durch den Benutzer vorgegebenen S- und T-Invarianten.

Analyseunterstützung

Eine regelbasierte Auswertung von Analyseergebnissen ist bei NET nicht vorhanden.

Bedienungsfreundlichkeit und Ausgabekomfort

Da wir NET nicht aus der praktischen Erfahrung heraus kennen, können wir keine Aussagen über Bedienungsfreundlichkeit und Ausgabekomfort machen.

8.5 PACE

PACE ist der Nachfolger des Werkzeuges SPECS, das an der ETH Zürich entwickelt wurde. PACE wird kommerziell durch Dr. J. Dähler vertrieben. Wir beziehen uns bei der Beschreibung von PACE auf [Feldbrugge89]. PACE hat folgende charakteristische Punkte:

- PACE basiert auf höheren Petri-Netzen.
- Die Tokenattribute sind Smalltalk Objekte.
- Die Transitionsinschriften sind Smalltalk-Ausdrücke, die komplexe Berechnungen ausführen oder die die Aktivität beeinflussen können.
- Hierarchische Strukturierung.
- Automatische Restrukturierungsmöglichkeiten.
- Schaltwahrscheinlichkeiten für Transitionen.
- Stellenkapazitäten.
- Subnetz-Bibliotheken und wiederverwendbare Netzmodule.
- Vollständig integrierter graphischer Editor.
- Simulator.
- Statistiktool.
- Automatische Generierung von C-Programmen.
- PostScript Dokument Generator.
- Online Help.
- Interaktive Syntax- und Konsistenzüberprüfung.
- Modernes Benutzerschnittstelle.

Simulation

Der Simulator erlaubt:

- das Schalten von Schritten,
- rückwarts und vorwärts Simulation,
- das Setzen von Breakpoints (d.h. Punkten, an denen die Simulation stoppt),
- die Modifizierung von Markierungen und des Netzes während der Simulation.

Erreichbarkeitsanalyse

PACE besitzt keine Erreichbarkeitsanalyse.

Invariantenanalyse

Auch eine Invariantenanalyse wird von PACE nicht unterstützt.

Analyseunterstützung

Eine regelbasierte Auswertung von Analyseergebnissen ist bei PACE nicht vorhanden.

Bedienungsfreundlichkeit und Ausgabekomfort

Da wir PACE nicht aus der praktischen Erfahrung heraus kennen, können wir keine Aussagen über Bedienungsfreundlichkeit und Ausgabekomfort machen. Laut [Feldbrugge89] hat PACE ein modernes Benutzerinterface, so daß man auf gute Benutzerfreundlichkeit und guten Ausgabekomfort schließen kann.

8.6 Fazit

Zum besseren Überblick fassen wir den Vergleich der verschiedenen Werkzeuge in Tabelle 1 zusammen.

	Höhere Petri-Netzform	Simulation	Erreichbarkeitsanalyse	Invariantenanalyse	regelbasierte Analysehilfe	Ausgabe und Bedienung
SMARAGD KfK / IAI, (1991)	++	in Planung	++	in Planung	in Planung	++
Design/CPN Meta Software Corporation, (1989)	++	++	in Planung	in Planung	-	++
PROVER KfK / IAI, (1987)	+	-	++	-	-	-
GRASPIN Esprit Projekt Nr. 125, (1990)	++	++	-	-	-	++
NET PSI, Berlin, (1989)	++	++	+	+	-	++
PACE Dr. J. Dähler, Zürich, (1989)	++	++	-	-	-	++

++ vorhanden + teilweise vorhanden - nicht oder ungenügend vorhanden

Tabelle 1 Vergleich von verschiedenen Petri-Netz-Werkzeugen

Wir unterscheiden in dieser Tabelle zwischen den drei Bewertungsstufen:

- ++ für vorhanden,
- + für teilweise vorhanden und
- - für ungenügend oder überhaupt nicht vorhanden.

Zu den Werkzeugen haben wir die Entwickler und das Jahr derjenigen Version auf die wir uns im Vergleich beziehen angegeben.

Die Beschreibung der verschiedenen Petri-Netz-Werkzeuge zeigt, daß man die Werkzeuge grob in zwei Klassen einteilen kann. Die erste Klasse besteht aus denjenigen Werkzeugen, die auf einem Höheren Petri-Netz-Modell basieren und im Wesentlichen nur Simulation und keine mit SMARAGD vergleichbare Analyse anbieten. In der zweiten Klasse sind diejenigen Werkzeuge, die auch eine umfangreiche Analyse anbieten. Diese Werkzeuge basieren meistens auf Platz/Transitions-Netzen. Von solchen Netzen ist, wegen der Einfachheit der Beschriftungen, eine Erreichbarkeits- bzw. Invariantenanalyse machbar.

Typische Vertreter der ersten Klasse sind z.B. Design/CPN, GRASPIN und PACE. Zu beiden Klassen kann man das Werkzeug NET zählen. Simuliert werden höhere Netze, analysiert können nur Platz/Transitions-Netze werden. Zur zweiten Klasse kann man PROVER rechnen, obwohl PROVER nicht auf typischen Platz/Transitions-Netzen jedoch auch nicht auf typischen Prädikat/Transitions-Netzen basiert.

SMARAGD vereint die Vorteile dieser beiden Klassen. SMARAGD basiert auf Höheren Petri-Netzen, wie die Werkzeuge der ersten Klasse, und ist analysefähig, wie die Werkzeuge der zweiten Klasse. Es stellt sich die Frage, warum ist bei SMARGAD Analyse möglich und bei anderen ebenfalls auf höheren Netzen basierenden Werkzeugen nicht? Der wesentliche Grund hierfür ist das in SMARAGD angewendete "Lokalitätsprinzip". Dadurch wird die Anzahl der möglichen Variablensubstitutionen, die der Rechner beim Schalten einer Transition bilden und verifizieren muß, minimal gehalten. Bei den anderen Werkzeugen tritt dagegen oft die Schwierigkeit auf, daß die Anzahl der Variablensubstitutionen sehr groß, je nach Wertebereich der Variablen sogar unendlich wird. Der Aufbau des Erreichbarkeitsgraphen wird deshalb sehr aufwendig oder sogar unmöglich.

Die für später vorgesehenen Erweiterungen von SMARAGD, Invariantenverifikation und Analyseunterstützung, sind ebenfalls für Werkzeuge, die auf höheren Netzen basieren, nicht üblich. Die Idee der regelbasierten Auswertung von Analyseergebnissen, die für SMARAGD angedacht ist, ist noch in keinem weiteren Werkzeug realisiert und auch für kein weiteres Werkzeug geplant. Eine unserem Konzept der Invariantenverifikation ähnliche Funktionalität besitzt nur das Werkzeug NET.

9. Zusammenfassung und Ausblick

In dieser Arbeit wurde die Analyse von SNL-Systemen, wie sie in [Düpmeier92] eingeführt wurden, vorgestellt. Nach den theoretischen Grundlagen der Analyse dieser SNL-Systeme haben wir eine Realisierung der Erreichbarkeitsanalyse in dem Werkzeug SMARAGD beschrieben. Die theoretischen Grundlagen gliederten sich in die drei Abschnitte Erreichbarkeitsanalyse, Invariantenverifikation und regelbasierte Auswertung von Analyseergebnissen.

Im Abschnitt Erreichbarkeitsanalyse (siehe Kapitel 4) haben wir den Erreichbarkeitsgraph und den Graph der starken Zusammenhangskomponenten eines SNL-Systems eingeführt und verschiedene Eigenschaften des Systems daraus abgeleitet. Solche Eigenschaften waren zum Beispiel Deadlocks, Livelocks, gemeinsame Folgemarkierungen, Heimzustände, Heimräume, verschiedene Lebendigkeitseigenschaften und gefrorene Marken.

Die mit Erreichbarkeitsanalyse ermittelbaren, in dieser Arbeit vorgestellten Eigenschaften sind in einem ersten Prototyp von SMARAGD realisiert. Diese Realisierung beinhaltet sehr viele graphische Elemente, so daß der Benutzer auf sehr anschauliche Art und Weise die verschiedenen Eigenschaften des Systems erhält. Dieser Prototyp wurde in Kapitel 7 beschrieben.

Die anderen vorgestellten Elemente der Analyse, Invariantenverifikation und regelbasierte Auswertung von Analyseergebnissen, sind in dem ersten Prototyp von SMARAGD noch nicht realisiert. Hier ist ein Ansatzpunkt für die weitere Entwicklung von SMARAGD. Eine Benutzerunterstützung durch das Werkzeug beim Finden und Verifizieren von Invarianten ist eine interessante Aufgabe für die weitere Entwicklung von SMARAGD. Die in Kapitel 5 entwickelten Konzepte bilden dazu eine wichtige Grundlage.

Wie auch in [Starke90] angesprochen, wird eine regelbasierte Auswertung von Analyseergebnissen in Zukunft eine immer größer werdende Rolle einnehmen. Starke nennt eine solche regelbasierte Auswertung von Analyseergebnissen *Expertsystem*. Durch die rasante Entwicklung auf dem Gebiet der künstlichen Intelligenz wird es möglich sein, den Benutzer bei der Analyse des Netzes sehr stark zu unterstützten. Diese Unterstützung wird notwendige Analysen ausführen und unnötige Analysen vermeiden. Durch Interpretation der analysierten Eigenschaften eines Netzes wird das System dem Benutzer Hypothesen über das Verhalten des Netzes und mögliche Korrekturhinweise geben. Erste Ansätze für eine solche regelbasierte Auswertung von Analyseergebnissen wurden in Kapitel 6 erarbeitet.

Neben der funktionalen Erweiterung der SMARAGD-Analyseeinheit um die beiden weiteren Komponenten der Analyse ist eine Erneuerung der Graphik von SMARAGD geplant. Wie schon angesprochen wurde SMARAGD aus Design/CPN unter Verwendung von Design/OA entwickelt. Dadurch ist eine Weitergabe von SMARAGD an eine andere Institution aus Lizenzgründen nicht ohne weiteres möglich. Deshalb wird der nächste Schritt in der Entwicklung von SMARAGD die Erneuerung der Graphik von SMARAGD sein. Die neue Graphik von SMARAGD soll durch direkte X-Windowprogrammierung erstellt werden, wobei nur Hilfsmittel benutzt werden, die bei einer späteren Vermarktung keine Lizenzschwierigkeiten ergeben. Ein solches Hilfsmittel wird GUIDE von der Firma Sun Microsystems sein, mit dessen Hilfe das Aussehen der Oberfläche von SMARAGD entwickelt wird.

Als weitere Ergänzung von SMARAGD ist eine Simulationskomponente geplant. Diese Simulationskomponente soll dem Benutzer interaktive und automatische Simulation erlauben. Mit Hilfe des Simulators kann der Benutzer schon während der Entwicklung eines Modells Teile der Funktionalität überprüfen, ohne den Erreichbarkeitsgraph aufbauen zu müssen.

Außer diesen für SMARAGD sehr wichtigen Erweiterungen kann man sich noch viele weitere Ergänzungen zu SMARAGD vorstellen. Solche Ergänzungen sind sehr stark vom Benutzer abhängig, so daß wohl jeder Benutzer eine eigene Wunschliste an Ergänzungen hat. Ein Beispiel einer solchen Wunschanforderung an SMARAGD, die bei der Benutzung von SMARAGD im IAI entstanden ist, betrifft das Schalten in SNL-Systemen. In unseren SNL-Systemen können alle aktivierbaren Schritte und Transitionen schalten. Eine Ergänzungsanforderung an SMARAGD war, das Schalten noch von anderen Faktoren abhängig zu machen. Zum Beispiel indem man einer bestimmten Transition verbietet, parallel zu anderen Transitionen zu schalten, oder indem man verlangt, daß eine Transition immer nur parallel zu anderen Transitionen schaltet. Man sieht, daß die Möglichkeiten der Erweiterung vielschichtig sind, und daß ein steter Dialog zwischen Anwender und Entwickler das Werkzeug immer weiter verbessern kann.

In [Düpmeier92] werden die Konstrukte Netz-Klasse und Netz-Modul eingeführt, mit denen wir eine Hierarchisierung der Netze erreichen. Solche Hierarchisierungen sind für die Entwicklung eines SNL-Modells sehr wichtig, denn dadurch kann man übersichtliche Modelle erstellen. Außerdem können häufig benötigte Netz-Klassen oder -Module in Bibliotheken gesammelt und so für andere Anwendungen wiederverwendbar gemacht werden. Bei der Analyse eines SNL-Modells entfalten wir diese Hierarchisierung, so daß wir ein Modell in nur einer Ebene haben. Eine interessante Frage für die Zukunft ist, ob man diese Netz-Klassen oder -Module im voraus analysieren und diese Ergebnisse auch auf das Modell, in dem eine Netz-Klasse oder ein Netz-Modul verwendet wird, übertragen kann. Diese Frage ist nicht einfach zu beantworten. Man vermutet zwar zuerst, daß man den Erreichbarkeitsgraph für ein Netz-Modul oder ein Netz-Klasse berechnen und diesen in den Erreichbarkeitsgraph des Gesamtmodells einsetzen kann. Doch dabei geht man von der falschen Voraussetzung aus, daß das Netz-Modul oder die Netz-Klasse vollständig schaltet, ehe eine Transition eines anderen Teiles des Gesamtmodells wieder schalten kann. Dies ist jedoch nur ein Spezialfall und nicht allgemein gültig.

Weitere Ansätze die Erreichbarkeitsanalyse zu reduzieren werden z.B. von Herrn Dr. Eggert aus dem IAI und Herrn Dr. Korczynski, Universität Warschau verfolgt. Ihre Idee ist die Entwicklung eines Modells durch die schrittweise Verfeinerung eines globalen Modells nach bestimmten eigenschaftserhaltenden Regeln. Die Einarbeitung solcher Regeln in das Werkzeug SMARAGD ist für spätere Versionen geplant.

Mit diesem kleinen Ausblick auf weitere Entwicklungsmöglichkeiten von SMARAGD wollen wir diese Arbeit schließen.

Glossar

$MULT(A)$	Menge aller Multimengen über der Menge A
$MULT^+(A)$	Teilmenge aller eigentlichen Multimengen von $MULT(A)$
$Tr(\mu)$	Trägermenge von μ
$MULT_{fin}(A)$	Teilmenge von $MULT(A)$ mit Multimengen, deren Trägermengen endlich sind
$MULT^+_{fin}(A)$	Teilmenge von $MULT^+(A)$ mit Multimengen, deren Trägermengen endlich sind
SIG	Signatur ($SIG=(S,\Omega,\Pi,typ_\Omega,typ_\Pi)$)
V_S	Familie von Variablen zu den Sorten $s\in S$ ($V_S=(V_S(s))_{s\in S}$)
$TERM(SIG,V_S)$	Menge der Terme über der Signatur SIG mit Variablen aus der Familie V_S
$FSM(SIG,V_S)$	Menge der formalen Summen über $TERM(SIG,V_S)$
$var(\mu)$	Menge der Variablen von μ
$SPEC(SIG,E)$	algebraische Spezifikation
N	Netz (Wird oft auch als Bezeichnung für ein SNL-System verwendet.)
A	Algebraisches System
$\bullet x$ bzw. $x\bullet$	Vor- bzw. Nachstellen der Transition x
penv(x)	Umgebungsstellen von x
$^<x$ bzw. $x^>$	Ein- bzw. Ausgabekanten von x
$^<x^>$	Menge der Umgebungskanten von x
M^f	(formale) Markierung eines beschrifteten Netzes
$NSPEC$	SNL-Netz-Spezifikation mit Anfangsmarkierung M^f_0 über der algebraischen Spezifikation $SPEC$ ($NSPEC=(N,A_N,M^f_0)$)
$sorts(p)$	Menge der Sorten einer Stelle p
$senv(x)$	Menge der Sorten in der Umgebung von x
$venv(x)$	Familie der Variablen in der Umgebung von x
$fvars(x)$	Familie der freien Variablen von x
$bvars(x)$	Familie der gebundenen Variablen von x
$vars(x)$	Familie der Variablen von x
$obj_{v,x}(M)$	Menge aller Objekte der Sorte $typ(v)$, die bzgl. der Familie M in der Umgebung von x liegen
$VAL_t(NSPEC,A)$	Menge der Auswertungen der freien Variablen von t in A
$\mathbf{R}_T$	Realistation von Termen
$\mathbf{R}_{EF}$	Realistation von atomaren Formeln
$\mathbf{R}_{FSM}$	Realistation der formalen Summen aus $FSM(SIG,V_S)$
$\mathbf{R}_N$	Realistation der Netzbeschriftung einer SNL-Netz-Spezifikation $NSPEC$
$\mathbf{R}$	Realistation der SNL-Netz-Spezifikation $NSPEC$ ($\mathbf{R}=(\mathbf{R}_T,\mathbf{R}_F,\mathbf{R}_{FSM},\mathbf{R}_N)$)
$NEQN(NSPEC)$	Menge der Formeln in der Netz-Beschriftung von $NSPEC$
$EQN(NSPEC)$	Menge der Formeln im Spezifikationsteil von $NSPEC$
$M \models_{\mathbf{R}} NSPEC$	M erfüllt die SNL-Netz-Spezifikation $NSPEC$ bzgl. der Realistation $\mathbf{R}$
$SNL\text{-}SYS$	SNL-System (oder SNL-Modell) ($SNL\text{-}SYS=(NSPEC,A)$)
$M(NSPEC,A)$	$\mathbb{Z}$-Modul, dessen Elemente wir P-Vektoren von $SNL\text{-}SYS$ nennen $M(NSPEC,A):=M_P(NSPEC,A):=\{m_p\in MULT_{fin}(A) \mid typ(m_p)\subseteq sorts(p)\}$
$M^+(NSPEC,A)$	P-Vektoren von $SNL\text{-}SYS$ mit $M^+(NSPEC,A)=M^+_P(NSPEC,A):=\{m_p\in MULT^+_{fin}(A) \mid typ(m_p)\subseteq sorts(p)\}$
$MARK(NSPEC,A)$	Menge aller möglichen Markierungen von $SNL\text{-}SYS$ $MARK(NSPEC,A)\subseteq M^+(NSPEC,A)\subseteq M(NSPEC,A)$
$\mathcal{M}_N$	andere Schreibweise für $MARK(NSPEC,A)$
$S(NSPEC,A)$	$\mathbb{Z}$-Modul, dessen Elemente wir T-Vektoren von $SNL\text{-}SYS$ nennen $S(NSPEC,A):=S_T(NSPEC,A):=MULT_{fin}(VAL_t(NSPEC,A))$
$S^+(NSPEC,A)$	T-Vektoren von $SNL\text{-}SYS$ mit $S^+(NSPEC,A):=S^+_T(NSPEC,A):=MULT^+_{fin}(VAL_t(NSPEC,A))$
$S(NSPEC,M)$	$\mathbb{Z}$-Modul, dessen Elemente wir T-Vektoren von $SNL\text{-}SYS$ unter der Markierung M nennen
$S^+(NSPEC,M)$	T-Vektoren von $SNL\text{-}SYS$ unter der Markierung M
$M \models_\mu NSPEC$	Der *T-Vektor* μ erfüllt die Transitionsformeln von $SNL\text{-}SYS$
(M,μ,M')	Schaltvorgang eines Schrittes μ
$M[_S\mu>M'$	Schaltvorgang eines Schrittes μ

(M,t,M')	Schaltvorgang einer Transition t
(M,t:*v*,M')	Schaltvorgang einer Transition t im Modus *v*
M[t>M'	Schaltvorgang einer Transition t
M[t:*v*>M'	Schaltvorgang einer Transition t im Modus *v*
$\Rightarrow_S$	direkte Schritterreichbarkeitsrelation
$\leadsto_S$	Schritterreichbarkeitsrelation
$\Rightarrow$	direkte Erreichbarkeitsrelation
$\leadsto$	Erreichbarkeitsrelation
$[_SM_0>$	Schritterreichbarkeitsmenge
$[M_0>$	Erreichbarkeitsmenge
$\mathcal{S}_{N,M_0}$	Menge aller Schaltvorgänge von Schritten (eine andere Schreibweise ist *SOCC(NSPEC,A)*)
$\mathcal{T}_{N,M_0}$	Menge aller Schaltvorgänge von Transitionen (eine andere Schreibweise ist *TOCC(NSPEC,A)*)
$\mathcal{TM}_{N,M_0}$	Menge aller Schaltvorgänge von Transitionen in speziellen Modi (eine andere Schreibweise ist *TMOCC(NSPEC,A)*)
$\mathcal{SG}_N$	allgemeiner Schrittgraph
$\mathcal{RGM}_N$	allgemeiner Erreichbarkeitsgraph
$\mathcal{RG}_N$	(reduzierter) Erreichbarkeitsgraph
$\Leftrightarrow_S$	starke Schrittzusammenhangsrelation
$SSCC_N$	Menge der starken Schrittzusammenhangskomponenten
$SSCG_N$	Graph der starken Schrittzusammenhangskomponenten
$\Leftrightarrow$	starke Zusammenhangsrelation
SCC_N	Menge der starken Zusammenhangskomponenten
SCG_N	Graph der starken Zusammenhangskomponenten
cf(M)	gemeinsame Folgemarkierung einer Markierungsteilmenge *M*
$anz_{M,p}(\omega)$	Anzahl der Marken ω in der Markierung M einer Stelle p∈P
$vork_M(\omega)$	Anzahl der Marken ω in einer Markierung M
minvork(ω)	minimale Anzahl von Vorkommen von ω in einem Netz N
maxvork(ω)	maximale Anzahl von Vorkommen von ω in einem Netz N

Literatur

Barnes89 BARNES, G.: *How To Use Xgrab (Version2.3)*, Tera Computer Company, Seattle (1989)

Budinas88 BUDINAS B.L.: *Decidability of the Petri Net Reachability Problem.* Translated from Avtomatika i Telemekhanika, No. 11, pp. 3-39, (1988)

CPN90 *Design/CPN User and Reference Manual*, Meta Software Corporation, Cambridge (MA)(1990)

Düpmeier89a DÜPMEIER, C.; SÜß, W.: *PDLC-Spezifikation*, Unveröffentlichter Bericht des Kernforschungszentrums Karlsruhe (1989)

Düpmeier89b DÜPMEIER, C.; SÜß, W.: *RGG-Spezifikation*, Unveröffentlichter Bericht des Kernforschungszentrums Karlsruhe (1989)

Düpmeier89c DÜPMEIER, C.; SÜß, W.: *PROVER-Benutzerbeschreibung*, Unveröffentlichter Bericht des Kernforschungszentrums Karlsruhe (1989)

Düpmeier90 DÜPMEIER, C.: *Spezifikation einer Höheren Petri-Netz-Sprache*, Unveröffentlichter Bericht des Kernforschungszentrums Karlsruhe (1990)

Düpmeier92 DÜPMEIER, C.: *Algebraische Spezifikation und Modellierung in Höheren Petri-Netzen mit zustandsabhängiger Schaltregel*, Dissertation, Universität Koblenz-Landau (1992)

Faltin88 FALTIN, U.; OCHSENSCHLÄGER, P.; PAULE, CH.: *Entwurf und Analyse von Produktnetzen.* Arbeitspapiere der GMD Nr. 332, (1988)

Feldbrugge86 FELDBRUGGE, F.: *Petri Net Tools.* Lecture Notes in Computer Science Vol. 222: Advances in Petri Nets 1985, Rozenberg, G. (eds.) -- Springer Verlag, pp. 203-223 (1986)

Feldbrugge89 FELDBRUGGE, F.: *Petri Net Tool Overview 1989.* Lecture Notes in Computer Science Vol. 424: Advances in Petri Nets 1989, Rozenberg, G. (eds.) -- Springer Verlag, pp. 151-178 (1989)

Flores77 FLORES, I: *Data Structure and Management.* Prentice Hall, New Jersey (1977)

Genrich78 GENRICH, H. J.; LAUTENBACH, K.: *Facts in Place/Transition-Nets.* In: Winkowski, J. (Ed.): Mathematical Foundations of Computer Science 1978, Springer Verlag, Berlin (1978)

Genrich79 GENRICH, H. J.; LAUTENBACH, K.: *The Analysis of Distributed Systems by Means of Predicate/Transition-Nets.* In: Goos, G. and Hartmanis, J.(Ed.): "Semantics of Concurrent Computation", Lecture Notes in Computer Science Vol. 70, Springer Verlag, Berlin, pp. 123-146 (1979)

Genrich80 GENRICH, H. J.; LAUTENBACH, K.; THIAGARAJAN, P. S.: *Elements of General Net Theory.* In: Brauer, W. (Ed.): "Net Theory and Applications", Lecture Notes in Computer Science Vol. 84, Springer Verlag, Berlin 1980, pp. 21-163

Genrich81 GENRICH, H. J.; LAUTENBACH, K.: *System Modelling with High-Level Petri Nets.* Theoretical Computer Science 13, pp. 109-136 (1981)

Genrich83 GENRICH, H. J.; LAUTENBACH, K.: *S-Invariance in Predicate/Transition-Nets.* In: Pagnoni, A.; Rozenberg, G. (Eds.): 3rd European Workshop on Application and Theory of Petri Nets, Informatik Fachberichte 66, Springer Verlag, Berlin 1983, pp. 98-111

Genrich86 GENRICH, H. J.: *Predicate/Transition Nets.* Lecture Notes in Computer Science Vol. 254: Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, September 1986 / Brauer, W.; Reisig, W.; Rozenberg, G. (eds.) -- Springer Verlag, pp. 207-247 (1987)

GRASPIN90 *GRASPIN: Towards the Next CASE Generation.* Final Report of Esprit Project 125, Itzfeldt, W. (Ed.), (1990)

Harper86 HARPER, R.; ET AL: *Standard ML.* LFCS Report (ECS-LFCS-86-2), Laboratory for Foundations of Computer Science, University of Edinburgh, Edinburgh 1986

Harper88 HARPER, R.: *Modules and Persistence in Standard ML.* In: Atkinson, M. P.; Bunemann, P.; Morrison, R. (Eds.): Data Types and Persistence. Topics in Information Systems, Springer Verlag, Berlin 1988

Harper89 HARPER, R.: *Introduction to Standard ML.* Laboratory for Foundations of Computer Science Report Series, University of Edinburgh, Edinburgh 1989

Harper90 HARPER, R.; MILNER, R.; TOFTE, M.: *The Definition of Standard ML.* MIT Press, London 1990

Horowitz76 HOROWITZ, E.; SAHNI, S: *Fundamentals of data structures.* Computer Science Press, Potomac, Maryland (1976)

Horowitz81 HOROWITZ, E.; SAHNI, S: *Algorithmen.* Springer Verlag, Berlin Heidelberg (1981)

Huber86 HUBER, P.; JENSEN, A.M.; JEPSEN L.O.; JENSEN, K.: *Reachability Trees for High-Level Petri Nets.* Theoretical Computer Science Vol 45 -- North-Holland, pp. 261-292 (1986)

Huber89 HUBER, P.; JENSEN, K.; SHAPIRO R.M.: *Hierarchies in Coloured Petri Nets.* Proceedings of the Tenth International Conference on Application and Theory of Petri Nets, pp. 192-209, Bonn (1989).

Jensen81a JENSEN, K.: *Coloured Petri Nets and the Invariant Method.* In: Theoretical Computer Science Vol. 14, pp. 317-336, North-Holland Verlag, 1981

Jensen81b JENSEN, K.: *How to Find Invariants for Coloured Petri Nets.* In: Mazurkiewicz, A. (Ed.): Mathematical Foundations of Computer Science, LNCS Vol. 118, pp. 327-338, Springer Verlag, Berlin 1981

Jensen85 JENSEN, K.: *An introduction to high-level Petri nets.* Proceedings of the 1985 International Symposium on Circuits and Systems, Kyoto 1985, IEEE, 723-726.

Jensen87a JENSEN, K.: *Coloured Petri nets. A way to describe and analyse real-world systems - without drowning in unnecessary details.* Proceedings ot hte Fifth International Conference on Systems Engineering, Dayton 1987, New York: IEEE, 395-401.

Jensen87b JENSEN, K.: *Coloured Petri nets.* Lecture Notes in Computer Science Vol. 254: Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, September 1986 / Brauer, W.; Reisig, W.; Rozenberg, G. (eds.) -- Springer Verlag, pp. 248-299 (1987)

Jensen87c JENSEN, K.: *Computer Tools for Construction, Modification and Analysis of Petri Nets.* Lecture Notes in Computer Science Vol. 255: Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, September 1986 / Brauer, W.; Reisig, W.; Rozenberg, G. (eds.) -- Springer Verlag, pp. 4-19 (1987)

Korczynski90 KORCZYNSKI, W.; DÜPMEIER, C.; SÜß, W.: *Eine Einführung in die Grundlagen der Theorie der Höheren Petri-Netze*, KfK-Bericht Nr. 4636 (1990)

Krämer85 KRÄMER, B.: *Stepwise Construction of Non-Sequential Software Systems Using a Net Based Specification Language.* In: Advances in Petri Nets 84, LNCS 188, Springer Verlag, Berlin 1985, pp. 307-330

Krämer86 KRÄMER, B.: *SEGRAS: The GRASPIN Specification Language* (Preliminary Reference Manual). GRASPIN TP GMD62, GMD, St. Augustin 1986, ISBN 3-88457-910-X

Krämer87 KRÄMER, B.; SCHMIDT, H. W.: *Types and Modules for Net Specifications.* In: Voss, K.; Genrich, H. J.; Rozenberg, G. (Eds.): "Concurrency and Nets", Springer Verlag, Berlin 1987, pp. 270-286

Krämer89 KRÄMER, B.: *Concepts, Syntax and Semantics of SEGRAS: A Specification Language for Distributed Systems.* GMD-Bericht Nr. 179, Oldenbourg Verlag, München 1989

Lautenbach73 LAUTENBACH, K.: *Exakte Bedingungen der Lebendigkeit für eine Klasse von Petri-Netzen.* GMD-Bericht Vol. 82, (1973)

Lautenbach84a LAUTENBACH, K.; PAGNONI, A.: *Liveness and Duality in marked-graph-like Predicate/Transition Nets.* In: Rozenberg, G. (Ed.): Advances in Petri Nets 1984, LNCS 188, pp. 331-352, Springer Verlag, Berlin 1984

Lautenbach84b LAUTENBACH, K.; PAGNONI, A.: *On the various high-level Petri-Nets and their Invariants.* Newsletter Vol. 16, Gesellschaft für Informatik, (1984)

Lautenbach85 LAUTENBACH, K.; PAGNONI, A.: *Invariance and Duality in Predicate/Transition Nets and in Coloured Nets.* Arbeitspapiere der GMD Vol. 132, (1985)

Lautenbach86a LAUTENBACH, K.: *Analysis Methods for Petri Nets Models.* Seminar "Applicability of Petri Nets to Operations Research", Universita Commerciale Luigi Bocconi, Mailand, (1986)

Lautenbach86b LAUTENBACH, K.: *Linear Algebraic Techniques for Place/Transition Nets.* In: Brauer, W.; Reisig, W.; Rozenberg, G. (Eds.): Advanced Course on Petri Nets, Bad Honnef, LNCS 254/255, Springer Verlag, Berlin 1986

Leszak88 LESZAK, M.; EGGERT, H.: *Petri-Netz-Methoden und -Werkzeuge*, Springer Verlag (1988)

Lindquist89 LINDQUIST, M.: *Parameterized Reachability Trees for Predicate/Transition Nets.* Acta Polytechnica Scandinavica, Mathematics and Computer Science Series No. 54, Helsinki (1989)

ML90 *Design/ML Reference Manual*, Meta Software Corporation, Cambrigde (MA)(1990)

MacQueen85 MACQUEEN, D. B.: *Modules in Standard ML.* In: Polymorphism II.2, October 1985

Martinez82 MARTINEZ, J.; SILVA, M.: *A simple and fast Algorithm to obtain all Invariants of a generalised Petri Net.* Informatik Fachbericht Nr. 52, Springer Verlag, Heidelberg (1982)

Mayr80 MAYR, E. W.: *Ein Algorithmus für das allgemeine Erreichbarkeitsproblem bei Petrinetzen und damit zusammenhängende Probleme.* Dissertation, ed. Fakultät für Mathematik der TU München (1980)

Memmi86 MEMMI, G.; VAUTHERIN, J.: *Analysing Nets by the Invariant Method.* Lecture Notes in Computer Science Vol. 254: Advances in Petri Nets 1986, Part I/ Rozenberg, G. (ed.) -- Springer Verlag, pp.300-336 (1987)

Milner87 MILNER, R.: *Changes to the Standard ML Core Language.* LFCS Report Series, University of Edinburgh, Edinburgh 1987

Näher90 NÄHER, S.: *LEDA User Manual*, Universität des Saarlandes, Saarbrücken (1990)

Newbery90 NEWBERY PAULISCH, F.; TICHY, W.: *EDGE: An Extendible Graph Editor,* Software - Practice and Experience 20, pp. 63-88 (1990)

Nitsche88 NITSCHE, U.: *Erreichbarkeitsgraphen von Produktnetzen und ihre Auswertung in PROLOG.* Arbeitspapiere der GMD Nr. 330, (1988)

OA90 *Design/OA User and Reference Manual*, Meta Software Corporation, Cambrigde (MA)(1990)

Ochsenschläger88 OCHSENSCHLÄGER, P: *Projektionen und reduzierte Erreichbarkeitsgraphen.* Arbeitspapiere der GMD Nr. 349, (1988)

Pascoletti86 PASCOLETTI, K-H.: *Diophantische Systeme und Lösungsmethoden zur Bestimmung aller Invarianten in Petri-Netzen.* GMD-Bericht Nr. 160, (1986)

PSI89 PSI GmbH Berlin: *Produktbeschreibungen und Release Notes zu NET-System 3.2,* Oktober 1989.

Razouk85 RAZOUK, R. R.; HIRSCHBERG, D. S.: *Tools for Efficient Analysis of Concurrent Software Systems.* Proc. SOFTAIR II (Software Development Tools, Techniques and Alternatives), San Francisco, (1985)

Reisig85 REISIG, W.: *Petri Nets with Individual Tokens.* In: Theoretical Computer Science Vol. 41, pp. 185-213, North-Holland Verlag, 1985

Reisig86	REISIG, W.: *Petrinetze -- Eine Einführung (2. überarbeitete Auflage).* Springer Verlag, Berlin 1986
Reisig91	REISIG, W.: *Petri nets and algebraic specifications.* In: Theoretical Computer Science Vol. 80, pp. 1-34, Elsevier Science Publishers, 1991
Rowe87	ROWE, L; DAVIS, M.; MESSINGER, E.; MEYER, C; SPIRAKIS, C; TUAN, A.: *A Browser for Directed Graphs,* Software - Practice and Experience 17, pp. 61-76 (1987)
Starke90	STARKE, P.: *Analyse von Petri-Netz-Modellen,* Teubner Verlag (1990)
Strack88	STRACK, V.: *Occurence Structures: A Model of Concurrency and Choice in System Behaviours.* Arbeitspapiere der GMD Nr. 328, (1988)
Sugiyama81	SUGIYAMA, K.; TAGAWA, S.; TODA, M.: *Methods for Visual Understanding of Hierarchical System Structures,* IEEE Transactions on Systems, Man and Cybernetics 11, pp. 109-125 (1981)
Süß90	SÜß, W.: *Konzept für die Analyse eines PrT-Netzes.* Unveröffentlichter Bericht des Kernforschungszentrums Karlsruhe, Karlsruhe 1990
Thieler88	THIELER-MEVISSEN, G.: *Korrektheit des Netzmodells für den CCR-Algorithmus - Entwurf eines Verfahrens zur parametrisierten Erreichbarkeitsanalyse.* Arbeitspapiere der GMD Nr. 301, (1988)
Trigila88	TRIGILA, S.; D'ANNA, M.: *Concurrent system analysis using Petri nets: an optimized algorithm for finding net invariants.* Computer Communications Vol. 11 No. 4 (1988)
Vautherin86	VAUTHERIN, J.: *Parallel Systems Specifications with colored Petri nets and algebraic abstract data types.* In: Proc. 7-th European Workshop on Applications and Theory of Petri Nets, Oxford 1986, pp. 5-23
Vautherin87	VAUTHERIN, J.: *Parallel Systems Specifications with Coloured Petri Nets and Algebraic Specifications.* Lecture Notes in Computer Science Vol. 266: Advances in Petri Nets 1987 / Rozenberg, G. (ed.) -- Springer Verlag, pp. 293-308 (1987)
Wikström87	WIKSTRÖM, A; *Functional Programming Using Standard ML.* Prentice Hall, London (1987)
Wirth75	WIRTH, N: *Algorithmen und Datenstrukturen.* B.G. Teubner Verlag, Stuttgart (1975)

Danksagung

Die vorliegende Arbeit wurde in den Jahren 1989 bis 1992 im Institut für Angewandte Informatik (IAI) des Kernforschungszentrums Karlsruhe erstellt.

Herrn Prof. Dr. Lautenbach von der Universität Koblenz-Landau möchte ich sehr herzlich für die Themenstellung und die fachliche Betreuung der Arbeit danken.

Herzlichen Dank sage ich Herrn Prof. Dr. Trauboth, der als Leiter des IAI diese Arbeit ermöglicht und unterstützt hat, und bereit war ein Korreferat zu übernehmen.

Mein herzlicher Dank geht auch an Herrn Dr. Korczynski, Universität Warschau, der das andere Korreferat übernahm und in zahlreichen Gesprächen wichtige Impulse für die Arbeit gab.

Mein ganz besonderer Dank gebührt Herrn Dr. Eggert, dem Leiter der Abteilung Mikrosysteminformatik des IAI, der durch seine vielfältige Unterstützung und die thematische Integration der Dissertation in die Projektarbeit die Voraussetzungen zum Gelingen dieser Arbeit schuf.

Meinem Kollegen Herrn Clemens Düpmeier danke ich für die hervorragende Zusammenarbeit und die zahlreichen wertvollen Diskussionen.

Herrn Dr. Voges gebührt Dank, da er durch wertvolle Hinweise den Fortgang der Arbeiten erleichtert hat.

Insbesondere danke ich meiner Frau Jutta, die viel Verständnis für meine Arbeit aufgebracht hat, und meinen Eltern, die mir durch ihre Unterstützung meine Ausbildung ermöglicht haben.